GSI Data Acquisition System V4.3: Release Notes

H.G. Essel, N. Kurz

January 9, 2004

GSI, Gesellschaft für Schwerionenforschung mbH
Planckstraße 1, D-64291 Darmstadt
Germany
Tel. (0 6159) 71-0
# Contents

1 MBS Release V4.3  
1.1 General Remarks .................................................. 3  
1.2 New Features in **MBS** Version 4.3 .............................. 4  
1.3 New GSI VME Trigger Module ........................................ 5  
1.4 GSI CAMAC Controller GTBV/GTBC .................................. 5  
1.5 Wiener CAMAC Controller VC32/CC32 ............................. 6  
1.6 New GSI VME Time Stamp Module TITRIS .......................... 6  
1.7 Gigabit Ethernet for RIO3 and LynxOS PC .......................... 7  
1.8 Static VME Mapping for RIO2 and RIO3 ........................... 8  
1.9 User Controlled Trigger Module “trigger clear” .................. 9  
1.10 RIO2 and RIO3 User Library for FASTBUS Controller NGF ...... 9  
1.11 RFIO Data Logging .................................................. 10  
1.12 Direct Mapping ..................................................... 11  
1.13 Delayed Event Building ............................................. 11  
  1.13.1 **MBS** Systems with a single readout processor ............ 13  
  1.13.2 **MBS** Systems with multiple readout processors ........... 14  
1.14 Stream Server Optimization ......................................... 15  
  1.14.1 Synchronous mode ............................................... 15  
  1.14.2 Asynchronous mode ............................................. 15  
      Keep mode without scale ....................................... 15  
      Scaled keep mode ............................................. 15  
1.15 MBS Euroball (RISING) Interface and Data Receiver m_rirec ....... 16  
1.16 Event Synchronisation with Time Stamps, m_to .................. 16  
1.17 Bugfixes and Miscellaneous ....................................... 18
Chapter 1

MBS Release V4.3

1.1 General Remarks

This Release Notes can be retrieved as pdf file from the web page
http://daq.gsi.de/document.htm They cover all changes for MBS version 3.x

Since January 1st, 2004, the MBS version 4.3 has become the new production version.
The new MBS and its commands can be used as before. To access and use the new production version a
few actions have to be taken. In case of troubles don’t hesitate to contact
N.Kurz, GSI Tel.: 2979, E-Mail: N.Kurz@gsi.de

1. Make sure that in the .login script in the home directory of your LynxOS account the
command source /mbs/prodlogin.com is present.

2. Make sure that this command is not followed by another command like mbslogin v40 (or v30, v41, v42),
source /mbs/develogin.com, or source /mbs/oldlogin.com.

3. If you had to change the .login script, logout and login again to make the changes active.
In any case you can issue the command mbslogin prod to set the production version and
mbsversion to check which version you are working with from the LynxOS prompt.

4. All user functions and programs (m_read_meb, m_collector) have to be recompiled and linked!
Your current make files are still valid and a make in your MBS working
directory is sufficient to rebuild your programs. You can issue make clean before make.

5. We strongly recommend to rename your main scripts to startup and and shutdown your
MBS to startup.scom and shutdown.scom. Otherwise the MBS graphical user interface
doesn’t work.
1.2 New Features in MBS Version 4.3

The new MBS version 4.3 provides several new features which will be described in further sections. Another source of information are the talks you will find on the web page http://daq.gsi.de/document.html

1. Support new GSI VME trigger module.
2. New GSI CAMAC controller GTBV/GTBC.
4. New GSI Time Stamp Module TITRIS.
5. Support of Gigabit ethernet for RIO3 and LynxOS PC.
6. Change in static VME page mapping for RIO3 VME processors.
7. User controlled trigger module “event clear”.
8. RIO2 and RIO3 user library for FASTBUS controller NGF.
9. RFIO support for direct data logging on the tape robots of GSI or on intermediate Linux data logger nodes.
10. Direct sub-event pipe mapping to speed up MBS throughput on RIO2 and RIO3 processors.
11. Delayed event building to make optimal use of macroscopic spill pauses.
12. New data receiver MBS process m_rece as interface to the RISING (EUROBALL) VXI data acquisition system.
13. New time sorter MBS process m_to allows to combine completely independent data acquisition systems in conjunction with the new GSI Time Stamp Module TITRIS.
14. Command arguments are no longer lower cased.
15. Bugfixes and smaller changes.

The new Makefiles of MBS v4 support the following platforms:

LynxOS 2.5: CVC, E6, E7, RIO2, PC
LynxOS 3.0: CVC, E7, RIO2, PC
LynxOS 3.1: RIO2, PC, RIO3

The current Platforms are:
LynxOS 2.5: CVC, E6, E7, RIO2, PC
LynxOS 3.1: RIO3
1.3 New GSI VME Trigger Module

The new GSI VME trigger module (blue front panel) is functional identical to the old one although based on completely different hardware. Even ECL input/output connectors are tilted as the old ones by 180 degrees. The main improvement is the new trigger bus for MBS systems with multiple trigger modules, which ranges longer and can take more modules (up to 170m with 7 trigger modules tested). Experience within FOPI and ALADiN proved stable running.

Unfortunately the time gap between sending a trigger to the input and the raise of the total dead time (TDT) output is now about 300 ns. Nevertheless no trigger is accepted in this gap. It is therefore recommended to split the trigger input signal, stretch one of it to about 1 µs and take the OR of this signal and the TDT from the trigger modul as “true” total dead time to generate busy signals. **Old and new type modules cannot be mixed!**

The setting of the multiple trigger module operation has changed slightly and is summarized below. The setting for a single trigger modul system remains unchanged.

**MBS trigger module setup command for old modules:**

\[
\text{mbs} > \text{set trig_mod} \quad ! \text{ master trigger module (single and/or multi)} \\
\text{mbs} > \text{set trig_mod -slave} \quad ! \text{ slave trigger module}
\]

**MBS trigger module setup command for new modules:**

\[
\text{mbs} > \text{set trig_mod} \quad ! \text{ master trigger module (single)} \\
\text{mbs} > \text{set trig_mod -multi} \quad ! \text{ master trigger module (multi)} \\
\text{mbs} > \text{set trig_mod -slave} \quad ! \text{ slave trigger module}
\]

**Note:** For the new trigger module as master (MT) to get a trigger output signal at least one of the sliders on the board (for setting the MT output signal length) must be enabled (set ON).

1.4 GSI CAMAC Controller GTBV/GTBC

The GTBV, a VSB to GTB converter and the GTBC, a CAMAC controller accessible via the GTB bus shall be the replacement for the “old” VSC/CBV combination. Although GTBV and GTBC make use of modern hardware they have the same user interface as the VSC/CBV. Especially no software or setup changes are needed within the MBS for running these controllers. As before up to 15 GTBC CAMAC controllers can be connected on a single GTBV in a chain.

The GTBV must be mounted (as before the VSC) on the backside of the VME crate at the position of a VME processor (i.e. E7, RIO2, RIO3) at the VME J2/P2. A GTB cable connects to the connector labelled “master” on the first GTBC. Each additional GTBC needs a GTB cable which connects the slave connector on the predecessor GTBC and the master connector of the next GTBC in the chain. Of course each GTBC needs a unique crate id, settable on a rotary
switch at the front panel. On the last GTBC in the chain the slave connector must be equipped with a terminator we also provide.

The GTB protocol uses less handshakes as the old VSB protocol and therefore the performance degradation for longer cables is much smaller. With a RIO2, a single CAMAC access needs 2.1 $\mu$s when using a 5 m GTB cable. A further improvement is the stability of the GTB bus. With the old differential VSB bus there have been many problems with connectors and cables especially when connecting many CBVs in a chain. In addition the GTB has a 10 Mbit ethernet interface, which allows “slow” CAMAC actions via the web.

1.5 Wiener CAMAC Controller VC32/CC32

The reason for another CAMAC controller is the fact that the VC32/CC32 supports the Fast CAMAC standard Level 1. This CAMAC mode operates with a maximum cycle frequency of 2.5 MHz (400 ns), compared to 1 MHz of standard CAMAC. The DGF module (Digital Gamma Finder) from the XIA company is up to now the only CAMAC module used at GSI, which supports Fast CAMAC Level 1. It should be stressed that the DGF can be fully operated with standard CAMAC but with less readout speed.

The VC32 is a VME module which is connected via a SCSI cable the CC32, the CAMAC controller. In contrary to the GSI CAMAC controllers the maximal cable length is about 5 m and only one CC32 can be connected to the VC32. The fastest measured cycle in the so called Auto Fast CAMAC mode needs 840 ns (1.2 MHz) with a RIO2 VME processor. This is actually the maximum speed of the RIO2 for single cycle VME accesses.

The VC32/CC32 can be easily used in the MBS by providing an A24 VME window. Multiple VC32s in a VME crate are possible. The VC32/CC32 uses like the GSI CAMAC controllers an address mapped CAMAC access. Unfortunately its developer decided to use a different encoding scheme of the CNAFs (Crate id C, Station number N, Sub-address A, Function code F), but a user readout function template with a new CNAF macro is provided.

What is missing is an update to the MBS ESONE server, to cope with the CNAF coding.

1.6 New GSI VME Time Stamp Module TITRIS

The GSI VME Time Stamp Module is available and allows the synchronisation of completely independent data acquisition systems. It has already been used successfully in the 2003 RISING campaign.

The module runs with a clock speed of 50 MHz and delivers 48 bit time stamps with a granularity of 20 ns. Although hardware identical, one arbitrary module must be configured as master, all others as slave modules. All time stamp modules in operation for a specific experimental setup are connected with a synchronisation bus. Via this bus the master module sends synchronisation pulses to all slaves to keep all modules on the same time base. Besides a control and status register each module has 3 time stamp registers for low, middle and high 16 bits of the 48 bit
counter, which covers more than 60 days.

To setup and operate the time stamp module a utility program and a template for MBS user readout functions are provided (N. Kurz). The utility program handles the setting of master and slave modules and the synchronisation cable delay calibration. The cable delay for the synchronisation pulses is then handled automatically by the slave modules. After setup, whenever a proper signal is sent to the input of an individual module, its time is latched to the time stamp registers. The module has no multi-event capability.

Test and in-beam measurements have shown consistently an effective resolution of 20 ns RMS. This includes continuous measurements with high rates over several weeks. The module was developed by J. Hoffmann from the DVEE department. See also the article in the GSI annual report 2002, page 224.

1.7 Gigabit Ethernet for RIO3 and LynxOS PC

Independent of the MBS version in use the RIO3 and PCs running LynxOS are prepared to use gigabit ethernet interfaces. The performance is in the range of 22-25 MB/s for the RIO3 and 35-50 MB/s for a PC. It increases substantially the data throughput in MBS systems with TCP/IP data transfer mode. For the RIO3 a gigabit PMC module (from CES) and for the PC a gigabit ethernet PCI card (from INTEL) has to be purchased.
1.8 Static VME Mapping for RIO2 and RIO3

Static Mapping of VME pages has changed slightly for RIO3. For completeness also the mapping for RIO2 is shown here. This section is independent of the MBS version in use.

The CES RIO2 and RIO3 provide static and dynamic mapping of VME pages for master read and write operations. With the CES user function calls “find_controller” and “return_controller” it is possible to map pages for all VME access modes of the VME specifications (see RIO2 and RIO3 manuals). For the sake of easy usage the RIO2 and RIO3 at GSI are configured to provide static VME access pages for the most important modes, namely A16, A24, and A32 non-piviledged data access.

To utilize the static VME mapping it is mandatory to set the most significant 4 bits (of normally 32 address bits) of the VME hardware base address on the VME slave modules to 0 (hex.: 0x00000000) This should be in most cases no problem since nearly all VME modules are equipped with hardware address switches.

RIO2 VME static mapping:

<table>
<thead>
<tr>
<th>Address Modifier</th>
<th>VME Hardware Address</th>
<th>Map Address</th>
<th>Map Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM: 0x09 (A32)</td>
<td>0x0</td>
<td>0xE0000000</td>
<td>0xE000000</td>
</tr>
<tr>
<td>AM: 0x39 (A24)</td>
<td>0x0</td>
<td>0xEE000000</td>
<td>0x1000000</td>
</tr>
<tr>
<td>AM: 0x29 (A16)</td>
<td>0x0</td>
<td>0xEFFE0000</td>
<td>0x10000</td>
</tr>
</tbody>
</table>

RIO3 VME static mapping:

<table>
<thead>
<tr>
<th>Address Modifier</th>
<th>VME Hardware Address</th>
<th>Map Address</th>
<th>Map Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM: 0x09 (A32)</td>
<td>0x0</td>
<td>0x50000000</td>
<td>0x100000000</td>
</tr>
<tr>
<td>AM: 0x39 (A24)</td>
<td>0x0</td>
<td>0x4F000000</td>
<td>0x1000000</td>
</tr>
<tr>
<td>AM: 0x29 (A16)</td>
<td>0x0</td>
<td>0x4E000000</td>
<td>0x10000</td>
</tr>
<tr>
<td>AM: 0x08 (BLT, D32 Block)</td>
<td>0x0</td>
<td>0x50000000</td>
<td>0x100000000</td>
</tr>
<tr>
<td>AM: 0x08 (MBLT, D64 Block)</td>
<td>0x0</td>
<td>0x60000000</td>
<td>0x100000000</td>
</tr>
</tbody>
</table>

Due to the different architecture of the RIO2 and RIO3 the map addresses for D32 and D64 block transfer modes are also shown for the RIO3. Block transfers on the RIO2 are specified with the VME hardware address of the VME slave modules.

To open a VME window the desired map address has to be used as input to the smem_create call and this function will return a virtual address for the VME access.

**Example:** To access from a RIO3 a VME slave module with VME base address set to 0x10000, for AM=39, A24/D32, the map address 0x4F010000 has to be used as input for the smem_create function call.

In the MBS setup files (setup.usf) also the map address has to be entered and the smem_create call is made by the MBS software (through the ia_shm interface).
Example: Setting the setup.usf parameter LOC_MEM_BASE to 0x52000000 will produce the virtual pointer pl_loc_hwacc which is passed to the user readout function for access of AM=9, A32/D32 VME accesses starting from VME address 0x2000000.

1.9 User Controlled Trigger Module “trigger clear”

In the standard MBS operation the trigger module is cleared after the sub-event readout by the MBS without user intervention. This is the appropriate behaviour for reading conventional digitizers. The utilisation of more modern devices like DSP boards or multi event digitizers could require a different treatment of the trigger module clear operation to get optimal performance. In many cases these devices are already prepared to take the next trigger (event) when the user readout function of the MBS is entered.

Up to now it was possible to make a user “trigger clear” only with some expert knowledge. Now a function named f_user_trigger_clear is provided to be used at any stage in the user readout function (f_user_readout) of the MBS. It takes care of all specialities like trigger 14 or 15.

Its usage is simple: First include sbs_def.h and f_user_trig_clear.h in the head of f_user.c. Second, call f_user_trig_clear (bh_trig_type) at any time in the functional part of f_user_readout. (The parameter bh_trig_type is the actual trigger type passed as parameter into f_user_readout.) At last before leaving the user readout function include the line: *l_read_stat = TRIG_CLEARED. A template function is provided as usual.

1.10 RIO2 and RIO3 User Library for FASTBUS Controller NGF

The FASTBUS controller NGF (Next Generation Fastbus) is supported by the MBS since a long time for the RIO2 and RIO3 VME processors. The supporting C structures and functions have now been moved from the “private” directory /nfs/groups/daq/usr/kurz/ngf_riox into the MBS directory structure. The old directory is obsolete and will be removed.

The NGF resource consists of the structures s_ngf and s_seq, which describe the NGF - and the FASTBUS sequencer on the NGF completely. RIOX.o contains the RIO2 and RIO3 dependent functions f_riox_map_slave_vme_ngf and f_riox_unmap_slave_vme_ngf and ngflib.o the NGF user functions f_ngf_get_ngf_ptr and f_ngf_init_fb_and_seq.

By using the environment variables MBSROOT and GSI_CPU_PLATFORM, which are defined on each GSI LynxOS node, the user can access the NGF resources independent of the MBS version and processor type. It is strongly recommended to use: $(MBSROOT)/ngf/src as path for the include files s_ngf.h and s_seq.h; and $(MBSROOT)/ngf/src/$(GSI_CPU_PLATFORM) as directory path for the RIOX.o and ngflib.o object files.

Template directories for RIO2 and RIO3 including Makefile, setup.usf and f_user.c provide full functioning NGF FASTBUS systems as starting point for user development.
1.11 RFIO Data Logging

RFIO (Remote File Input/Output) is a TCP based protocol and a library to write data directly to a remote mass storage device. The functions provided have the same functionality as the standard C disk IO calls. At GSI an MBS system can write list mode data directly to a data server.

The implementation of an RFIO server on the data movers for the GSI tape archive, which handles the incoming data for the tape robots is not yet finished at this date. As an intermediate solution it is possible to start an RFIO server on a Linux workstation. This server will write the data on its local disk. To start the RFIO server on Linux type:

\[ \text{> rawDispRFIO} \]

To file MBS data on this server use the MBS command:

\[ \text{mbs> open file node:filename -rfiodisk} \]

Node is the LINUX node name where rawDispRFIO had been started. Filename can be either a full path or a path relative to the one from which the server had been started. Of course you need write access to this path. See example:

\[ \text{mbs> open file lxi007:/tmp/kurz/test.lmd -rfiodisk} \]

The other open file options, like \text{-auto} for automatic tape file closing and opening, can also be used with the \text{rfiodisk} option. See example:

\[ \text{mbs> open file lxi007:/tmp/kurz/test.lmd size=200 first=1 -auto -rfiodisk} \]

**Important Note:**

Maximal 40 characters (in conjunction with the \text{-auto} option only 36!) for the node:file string are allowed. To reduce the file name length one should specify the filename in the open file command relativ to the directory where the rawDispRFIO server has been started.

Example: If rawDispRFIO has been started in \text{/u/kurz/test/RFIOfiles} on lxi014

\[ \text{mbs> open file lxi014:/testfile.lmd -rfiodisk} \]

will put the file \text{testfile.lmd} in the \text{/u/kurz/test/RFIOfiles} directory.

Performance measurements with one MBS event builder PC running LynxOS have shown a sustained data logging speed of 10-11 MB/sec, reaching the bandwidth of the fast ethernet interfaces in use. This is about a factor of two better than what we get with the DLT 8000 drives.

**Note:** The on-line data flow through the GSI network might create congestion for other network devices (depending, where sender and receiver is located).

**Note:** When writing data to a “private“ Linux server the data is not save against disk crashes. It must be moved per user command (admscli) to the final place in the robot.

The RFIO library functions implemented in the MBS and the rawDispRFIO server program have been provided by H.Goeringer. The MBS is prepared to write directly to the GSI tape archive, when the software development for the data movers has been finished.
1.12 Direct Mapping

Up to now sub-event pipes written by the MBS readout process and read by the MBS event builder have been mapped with the LynxOS system call smem_create (through f_shm interface). Memory which is mapped with this call is not cached and therefore only accessible with single address-data cycles. To utilize the speed of burst like data transfers (like it is done when data caches are updated) the complete DRAM of the VME processors RIO2 and RIO3 has been mapped to an extra address window where user processes can directly read and write.

A new setup parameter named LOC_PIPE_TYPE was introduced in the MBS setup.usf user setup file (usf) in order to distinguish between normal and direct mapping:

\[
\text{LOC\_PIPE\_TYPE} = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),\]

will create a direct mapped sub-event pipe for the processor in crate 0.

Omitting LOC_PIPE_TYPE from the setup.usf file or

\[
\text{LOC\_PIPE\_TYPE} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),\]

will create a conventional (smem_create) sub-event pipe for the processor in crate 0.

**RIO2**: DRAM address 0x0 directly mapped to 0x90000000

**RIO3**: DRAM address 0x0 directly mapped to 0xF0000000

Example for RIO3:

Direct Mapping: LOC\_PIPE\_TYPE 1
LOC\_PIPE\_BASE 0xF4000000

smem_create: LOC\_PIPE\_TYPE 0
LOC\_PIPE\_BASE 0x04000000

Performance measurements showed an improved data throughput of up to 20% for single cycle VME accesses in the user readout functions, compared to the conventional mapping with smem_create. When using VME block data transfer the MBS throughput increases up to 45%. The real gain in performance depends on the actions in the individual user readout function. In some cases it could be marginal.

Since the direct mapped RAM is not protected by the Memory Management Unit (MMU) you will not get a Segmentation Fault signal if you try to access memory outside the RAM address space. Most probably the processor will just hang and has to be rebooted.

1.13 Delayed Event Building

In the standard MBS setups the event builder processes (m\_collector, m\_dr, m\_to) try to get event data all the time if output buffer resources are available. Whenever events are ready in the sub-event pipes event data is transported. This is the case regardless if readout and event builder
processes run on the same or different processors and also regardless if an address mapped (i.e. RAM, VME/VSB memory) or message oriented transport protocol (TCP/IP) is used.

If an experiment runs with a DC kind trigger/event rate this is not an issue. The time to access the sub-event pipes has to be shared between reader and writer. In worst cases this could lead to a data throughput of only half of the full read/write speed, since all memories in use are single ported.

If the incoming triggers have a macroscopic spill structure (spill duration and pause larger than about a second) the described standard MBS behaviour is not optimal. An ideal solution would be the alternate processing of reader and writer: The event building stops when the spill goes on (and all pipes are rather empty) leaving the full resources to read the digitizers and fill the pipe. Event building is started when the spill goes off. Now the pipe writer is idle leaving the full resources for the event building. This mechanism in turn requires possibly large sub-event pipes, which should have enough space to take the full data of a complete spill. Ever larger DRAMs of our VME processors (RI02, RI03) fulfill this requirement. A new MBS mode named Delayed Event Building (DEB) has been developed and incorporated into the MBS.

At first it requires to inform the MBS system when spill is ON and when it is OFF. It has been decided to use and sacrifice trigger type 12 (spill goes ON) and 13 (spill is OFF) to signal the spill state to the MBS system. Spill ON trigger 12 shall be send about 100 ms before the spill starts and spill OFF trigger 13 about 100 ms after the spill ended. The electronic setup has to be provided by the user. If DEB is not enabled trigger type 12 and 13 are available for normal triggering.

As mentioned above, the sub-event pipes have to be configured as large as possible to get an optimal result. This has to be done as usual in the setup.usf file(s) with the PIPE_SEG_LEN parameter. In some cases RAM space for the operating system LynxOS may be cut and the freed space can be added to the sub-event pipe space. Ask N.Kurz how to do this. These two preparations are sufficient to utilize DEB.

To understand the behaviour and to make an optimal use of DEB a more detailed knowledge of the MBS sub-event pipe structure is needed:

The internal structure of the MBS sub-event pipe is defined by two setup parameters in setup.usf. PIPE_SEG_LEN defines its total memory size in bytes and PIPE_LEN specifies how many sub-events are at maximum disposed. (The third parameter for the pipe setup is its base address LOGPIPE_BASE, which does of course not affect the internals.)

With these two parameters the MBS prepares the sub-event pipe according to the following scheme: The first 356 bytes are reserved for status and control structures. For each sub-event slot defined in PIPE_LEN a 20 bytes control structure is prepared afterwards. The remaining memory space is then devided into a double data buffer (D0, D1) where both are 8 byte aligned in its base address and in its size. These double buffers are the place where the sub-event header and data is written in. Both reader and writer toggle between D0 and D1.

The rules for pipe writer and reader are the following: Pipe reader may read any time from the pipe if one event is ready, pipe writer must wait until a data buffer D0, D1 has been completely read by a pipe reader. Events are always readout in the same order as they have been written.
A sub-event pipe is blocked in two situations:
1. no sub-event slot (PIPE_LEN) is free,
2. one data buffer is full and the other one is not yet completely empty.

The user can easily avoid the first reason for pipe blocking by making the maximal number of sub-events reserved (PIPE_LEN) big enough.

If DEB is enabled the second reason for pipe blocking could lead to a deadlock if event building (pipe reading) is suspended and more than 50% of the data space is already occupied. In this case the spill OFF event (trigger 13) cannot be readout and event building will not resume. This implies that the space for the pipe (PIPE_SEG_LEN) shall be made at least twice as large as the total data size expected at maximum in a spill.

The new commands for enabling and disabling the Delayed Event Building are:
When using m_collector:

```
  mbs> enable del_eb_col  low_mark  high_mark
  mbs> disable del_eb_col
```

When using m_ds:

```
  mbs> enable delayed_eb  low_mark  high_mark
  mbs> disable delayed_eb
```

High and low mark can range from 1 to 49 and the high mark must be bigger than the low mark. This is checked by the MBS. Low and high mark are percentages of relative occupancy. The upper limit of 49 for the high marks avoids the deadlocks described above.

DEB can be used in combination with Direct Mapping to get optimal performance results.

### 1.13.1 MBS Systems with a single readout processor

If DEB is enabled in a MBS system with only one readout processor (and therefore only one sub-event pipe) the DEB mechanism works as following:
The m_collector (or m_ds) calculates after each sub-event the relative occupancy of the pipe in terms of occupied sub-event slots and in terms of occupied data space. If spill state is OFF (after trigger 13), this has no effect, event building goes on as usual. If spill state changes from OFF to ON event building is suspended until either the relative occupancy of sub-event slots, or of data space, crosses the high mark upwards. If this happens event building in spill resumes until both, the relative occupancy of sub-event slots, and of data space, crosses the low mark downwards.

**Remarks:**

1. If there is no special reason high mark shall be set to its maximum value of 49.
2. The low mark is also very important since it defines the event building work load to be done in spill OFF state.
3. If low mark and high mark are too close, event building may suspend and resume often during a single spill ON phase.

4. If sometimes a spill state change (trigger 12, 13 event) is missed it is not a problem. The performance will be at least as high as without DEB. If spill ON is missed the system behaves just like a conventional MBS without DEB. This is also the case if spill OFF is missed and the relative occupancy stays above the high mark.

1.13.2 **MBS** Systems with multiple readout processors

On systems with multiple sub-event pipes and m_collector as event builder process the explanation in the previous section is still valid. Since m_collector always reads the sub-event pipes in address mapped mode it can calculate the relative occupancies of all pipes to be read. Therefore it can control the DEB mechanism. In contrast to a single sub-event pipe system event building in spill resumes if any of the pipes in use cross the high mark upwards. Event building in spill is only suspended if the occupancies of all pipes are below the low mark.

In **MBS** systems, which make use of multiple pipes and utilize a message oriented data transfer mode (TCP/IP), the situation is different. In this system the data sender process m_ds runs on a readout processor. It reads a single pipe and sends the data to a remote event building processor. All m_ds in a system don’t know each other a-priori (the synchronisation is made by the trigger bus in hardware). Since data sending of all sub-events (of an event) always starts with the first m_ds (id=0) and the sequence is always the same for all m_ds, it is the m_ds with id=0 which needs the information of state changes from all other m_ds processes. This is necessary since (for example) any pipe which cross the high mark must bring the complete **MBS** system back to event building in spill.

Therefore, the user has to issue on all readout processors the `enable delayed_eb` command. When doing this the first time in an **MBS** session TCP/IP socket connections from all m_ds with ids ≠ 0 to the m_ds with id = 0 are established. The purpose of these sockets is to communicate only the change of in spill event building state (resume and suspend event building).

**Remarks:**

1. To guarantee a decent startup the “enable delayed_eb” command is only allowed in acquisition stopped state.

2. High and low marks set too closely might lead to fast state changes and therefore to some load on the TCP/IP control sockets. This should be avoided for performance reasons.

3. Tests and “real” experiments have shown substantial performance improvements for DEB operation compared to conventional **MBS** operation. The real gain, as always, is dependent on the special experimental conditions.

4. DEB is successfully in operation at the experiments FOPI and ALADiN.
1.14 Stream Server Optimization

The MBS stream server provides several operation modes for best performance in different setups. The modes are active only when a client is connected. In all modes the scale factor controls the DAQ performance. The default mode after startup is same as by following command:

```
  mbs> set stream_serv scale=2 -nosync -nokeep
```

This means, every 2nd stream is delivered to a client, if a request was there.

1.14.1 Synchronous mode

The new synchronisation mode is enabled by:

```
  mbs> set stream_serv scale=n -sync
```

This means, for every nth stream the server waits for a client’s request. With `scale=1` the client gets all streams and acquisition speed is defined by client.

1.14.2 Asynchronous mode

In asynchronous mode the stream server frees every full stream immediately when no request from a client is there. This means that a client will not get data when the DAQ does not produce events like in spill pauses. To avoid this the stream server can be told to keep full streams. There are two variants of `keep` modes:

**Keep mode without scale**

```
  mbs> set stream_serv scale=1 -keep
```

When `scale=1` the server keeps a stream as long as there are more than 3 free streams available. The effect is that all streams (except the three) are always full (except they are sent to a client). During a break in the event building (in spill pause or with DEB during spill) the client will get all these filled streams. The acquisition speed may slow down especially at high event rates but low data rates. The streams should be configured so big that the event builder has enough space in three streams to fill during one time slice, i.e. before the stream server can free streams. With `scale>1` keeping is switched off and the mode is like `nokeep`.

**Scaled keep mode**

```
  mbs> set stream_serv scale=n -scaled_keep
```

The server keeps every nth stream as long as there are more than 3 free streams available. The effect is that during a break in the event building the client will get streams on request, but
maybe not as many as in `-keep` mode. At the other hand in `-scaled_keep` mode the impact on the acquisition speed can be controlled. With `scale=1` this mode is the same as `-keep`.

The `show stream_serv` command has been enhanced and displays the mode and counters. The counters are only incremented if a client is connected. Counters are cleared by

```plaintext
mbs> show stream_serv -clear
```

To observe the filling status of the streams the `rate` command can be used:

```plaintext
lynx> rate -mbs
```

One should play with the modes to get the best compromise of DAQ performance and data transfer for monitoring.

### 1.15 MBS Euroball (RISING) Interface and Data Receiver `m_rirec`

Data produced in the Euroball VXI data acquisition system can now be read by MBS event builder nodes. It is based on a TCP/IP socket and was a joint effort between the Daresbury Euroball team (V.Pucknell), who provided the data sender process and GSI MBS responsible for the data receiver process. All data received is reformatted into the standard MBS listmode format. It is a pure data transporter and event triggering is still kept under control of the Euroball VXI system. It has been successfully used in the 2003 RISING campaign. See also article in GSI annual report 2002, page 224.

### 1.16 Event Synchronisation with Time Stamps, `m_to`

Triggered by the RISING project a new method to synchronize data in time, taken with independent MBS systems has been developed for the MBS. As mandatory requirement each daq system has to be equipped with the GSI time stamp module TITRIS explained in a previous section. Apart from that each system can be triggered independently and produces its local dead time. (Of course the user is also free to to feed identical triggers into several daq systems or combine the local dead time of several systems to a more global dead time.) Each event of the MBS in operation is required to deliver a unique daq identifier and 3 time stamp values (48 bit in total) with their appropriate identifiers as first data elements in the first sub-event of an event. An MBS user readout function template which fulfill this task is provided. The merging and sorting of data from the various DAQ systems is now made by an additional MBS system. The heart of it is the new MBS process `m_to`, which has to fullfill 3 main tasks.

1. Connect to the transport processes of the participating MBS systems.
2. Sort all events coming from the connected MBS systems according to their time stamp values.

3. Copy events from all input systems to a single output stream into the standard MBS format.

Data connection and disconnection can be made on the fly, regardless if other systems are already connected or the sorting process is active or halted. This gives the possibility to partition and change the MBS combinations very rapidly. If one data connection is established the time sorting process starts, although trivial, per default.

The time sorting algorithm requires at least one event ready from each connected system. This causes problems if one input line delivers events at a much lower rate as the others. In this case a slow rate pulser or calibration trigger for slow participating MBS systems is recommended. A rate of 0.1 - 1.0 Hz is in all cases sufficient.

It has to be stressed that the merging and time sorting of any data input stream to a single output stream does not compose new events. It is the task of the analysis to pick up out of the time sorted events those which have to be combined to the real “physics” event. This gives also the flexibility to use different criterias to compose physics events with identical data sets. This new approach has been successfully implemented in the RISING analysis based on the GO4 package from GSI.

The new commands for the time sorting MBS process are:

1. connect transport node name
   Connects to transport process to establish a data connection. node name is the node name where the transport runs.

2. disconnect transport node name
   Disconnect from transport process.

3. start output
   Starts the time sorting process. At startup this is done per default.

4. stop input
   Stops time sorting process and data output. This will also block data taking on all connected MBS systems, when their sub-event pipe and output buffers are full.

5. set toreceiver flush time in seconds
   With this command the maximum time is set until data is flushed out, although an output buffer stream was not completely filled. Its default value is 10 seconds, the command has the same functionality as set flush time for the m_collector process.

6. show input nodes
   Displays all input nodes connected.

Template directories to set up such a system can be provided. See also article in GSI annual report 2002, page 224.
1.17 Bugfixes and Miscellaneous

1. A bug in the MBS event server (m_event_serv) prevented the handling of big events. This bug has been fixed.

2. When closing a file the buffers of the last stream have not been written to the file. This is fixed.

3. It often happens that a terminal were an MBS session is running is killed before stopping the MBS. In this cases sometimes the MBS command prompter were permanently swamped by the same escape sequence. The MBS detects these as non valid commands and makes an entry into the logfile, which leads to huge and useless logfiles. The problem has been solved in such a way, that the MBS command interface logs only the first 50 of identical (invalid) commands.

4. The -auto option for the “open file” command was not working in conjunction with disk file writing (-disk option). This is now working.

5. The MBS prompter does not convert any more commands into lower case strings. This allows now for example, to specify UNIX file pathes containing upper and lower case letters.

6. The help command has been slightly changed. It works now inside MBS in the same way as in LynxOS:
   help -mbs will show all command keys available in the MBS.
   help -mbs key1 will show all command keys containing key1.
   help -mbs key1 key2 will show all command keys containing key1 and key2.
   help -mbs key1 key2 key3 will show all command keys containing key1, key2 and key3
   example: help -mbs open or help -mbs open file.
   Specifying all keys of a command will show the complete syntax. Useful is also the command help -mbs mbs which shows all MBS keywords, and for each keyword all commands.

   New in MBS prompts:
   ? displays a list of all MBS processes and state if they are running.
   ?? displays all MBS commands
   ?? string1 displays all MBS commands containing string1 in a command key.
   ??? displays all MBS commands with all parameters and options possible.
   ??? string1 displays all MBS commands containing string1 in a command key with all parameters and options.

7. The options for the rate utility has been enhanced substantially. With rate it is possible to monitor MBS systems from the LynxOS prompt. Try rate -h to see the various options or simply run rate without option to get the default running.
lynx> rate -h
All switches are additiv
-<u>ser # default selection
-<n|r>[s]t[reams] # number|rate of [server] streams
-<n|r>[s]bu[ffers] # number|rate of [server] buffers
-<n|r>[s]ev[ents] # number|rate of [server] events
-<n|r>[s]da[ta] # number|rate of [server] data
-e|f|k|st[reams] # empty|filled|kept streams
-s|f|r|ni|fi[le] # size|filled|rate|name|index of output file
The following are compound switches:
-<u>ser # default. Must be set if others shall be added
-<mbs # mbs part
-<fi[le] # all file switches
-<se[rver] # all server switches
-<ra[tes] # all rates
-<st[reams] # stream usage
RATENFLAG # value of this environment variable is added

Examples:
lynx> rate 5 -rfile -ifile
lynx> rate -user -t 5
lynx> rate -mbs -t 3 -rsdata
lynx> rate -rates

8. Reminder for working on Linux with LynxOS and MBS sources:
On the interactive GSI Linux nodes the LynxOS group directories are mounted. If one
wants to edit and/or manipulate LynxOS files with Linux editors or utilities login on Linux
with the same account name as on LynxOS and make cd /lynx/Lynx. In this directory one
will find all GSI group directories. If one, for example, wants to work as usr profi, which
is in the group frs cd to /lynx/Lynx/frs/usr/profi and you will find the LynxOS home
directory of usr profi. The user and group id mapping has been made identical on Linux
and LynxOS, so the permissions on both systems are the same. Compiling and linking
of programs must be made as usual on the LynxOS target machines. The directories
are automounted on Linux, so it could take some seconds until you see the contents of
/lynx/Lynx

9. Internal change in intertask communication: The message queues and shared memory mech-
anism has been changed. There are new commands to show the message queues and the
shared memory:

lynx> msg_sho
lynx> shm_sho
10. Reminder for VSB mapping on RIO2 and RIO3:
   Hardware prerequisite for utilizing the VSB bus is a VSB PMC piggy back module for RIO2 and RIO3. In contrast to the E7 the VSB bus has to be mapped on RIO2 and RIO3 by the user. It is agreed that the available VSB mapping utilities provided by CES can arise some confusion:

RIO2:
/bin/ces/vsbini F0000000 creates an 8 Mbyte window beginning from the VSB hardware base address 0xF0000000 mapped to the user space base address 0xd0000000. The user space address has to be used as input base address in the LynxOS smem_create call and as well in the MBS setup.usf files where VSB mapping is involved. (CBV and GTBC crate1: base address 0xd0380000, crate 2: d05800000 and so on.) The described mapping call covers a window for 3 CAMAC crates with (crate id 1-3) for the GSI VSC/GTB and GTBV/GTBC CAMAC controllers. This was not sufficient for some experiments and since the GSI controllers allow in principle 15 crate controllers connected in a single branch CES provided a new utility vsbini_new:

/bin/ces/vsbini_new F0000000 1 creates an 32 MB window sufficient to cover 15 CAMAC crates. Unfortunately it maps the VSB base address 0xF0000000 to the user space base address 0xd8000000 (in contrast to 0xd0000000 with vsbini). (CBV and GTBC crate 1: base address 0xd8380000, crate 2: d85800000 and so on.)

For simplicity reasons either vsbini or vsbini_new (as required by the DAQ coordinator) is executed during the booting phase. This means that one should first get the information if and which VSB initialisation utility was executed during booting before specifying VSB addresses in setup.usf. Another problem is that after calling vsbini vsbini_new does not work properly and vice versa. Our advice is therefore to move to vsbini_new whenever possible, which has no disadvantages compared to vsbini.

RIO3:
/bin/ces/vsbini F0000000 1 creates an 32 Mbyte window beginning from the VSB hardware base address 0xF0000000 to the user space base address 0xd0000000. (CBV and GTBC crate1: base address 0xd0380000, crate 2: d05800000 and so on.)

As mentioned above: In principle easy, but much room for confusion.
Contents