mvme - mesytec VME Data Acquisition

Release 1.21.0

mesytec GmbH & Co. KG <info@mesytec.com>

Aug 05, 2026

CONTENTS:

1 Introduction **1**
- 1.1 Features . . . 1
- 1.2 High-level overview . . . 2
  - 1.2.1 The VME side . . . 2
  - 1.2.2 Analysis . . . 3
  - 1.2.3 The DAQ process . . . 5

2 Installation **6**
- 2.1 System Requirements . . . 6
  - 2.1.1 Mesytec MVLC . . . 6
  - 2.1.2 WIENER VM-USB (Legacy) . . . 6
  - 2.1.3 Struck SIS3153 (Legacy) . . . 7
- 2.2 Installation Steps . . . 7
  - 2.2.1 Linux . . . 7
    - 2.2.1.1 MVLC_USB and VM-USB Device Permissions . . . 7
  - 2.2.2 Windows . . . 7
    - 2.2.2.1 MVLC USB Driver . . . 7
    - 2.2.2.2 VM-USB only: Driver Installation . . . 7
  - 2.2.3 Installation from source . . . 8
- 2.3 Ethernet DHCP/ARP setup . . . 8
  - 2.3.1 Using a manual ARP entry . . . 9

3 Quickstart Guide **10**
- 3.1 VME Setup . . . 11
- 3.2 Analysis Setup . . . 14
- 3.3 Starting the DAQ . . . 14
- 3.4 Event Counter readout . . . 16

4 Reference **17**
- 4.1 DAQ / Replay controls . . . 17
  - 4.1.1 Replaying data from listfiles . . . 18
- 4.2 VME configuration . . . 18
  - 4.2.1 Structure . . . 18
  - 4.2.2 Module and Event configuration . . . 19
    - 4.2.2.1 Variables . . . 19
  - 4.2.3 DAQ startup procedure . . . 20
  - 4.2.4 DAQ stop procedure . . . 21
  - 4.2.5 VME Controller specifics . . . 21
    - 4.2.5.1 Mesytec MVLC . . . 21
    - 4.2.5.2 SIS3153 . . . 21
  - 4.2.6 Custom VME Modules . . . 21
- 4.3 VME Scripts . . . 22
  - 4.3.1 Overview . . . 22
  - 4.3.2 Commands . . . 23

4.3.2.1 Writing . . . 23
4.3.2.2 Reading . . . 23
4.3.2.3 Block Transfers . . . 23
4.3.2.4 Miscellaneous . . . 24
4.3.2.5 MVLC special commands . . . 26
4.3.2.6 VMUSB specific . . . 28
4.3.3 Floating Point Values, Variables and Mathematical Expressions . . . 28
4.3.3.1 Variables . . . 28
4.3.3.2 Expressions . . . 29
4.3.4 Example Script . . . 29
4.3.5 VME block read command mappings . . . 30
4.4 MVLC Trigger I/O Module . . . 30
4.4.1 Introduction . . . 30
4.4.2 Structure and operation . . . 31
4.4.3 User Interface and integration into mvme . . . 32
4.4.4 Gate Generators . . . 32
4.4.5 I/O and logic units . . . 33
4.4.5.1 NIM I/Os . . . 33
4.4.5.2 LVDS outputs . . . 33
4.4.5.3 IRQ inputs . . . 33
4.4.5.4 Timers . . . 33
4.4.5.5 Trigger Resource Units . . . 34
4.4.5.6 Stack Busy . . . 34
4.4.5.7 Sysclk . . . 34
4.4.5.8 Lookup Tables (Levels 1 and 2) . . . 34
4.4.5.9 StackStart . . . 38
4.4.5.10 MasterTrigger . . . 38
4.4.5.11 Counters . . . 38
4.4.6 Digital Storage Oscilloscope . . . 38
4.4.6.1 Steps needed to acquire traces . . . 39
4.4.6.2 Notes . . . 39
4.4.7 Trigger IO Usage Example: Sysclk timestamp readout . . . 40
4.5 Analysis . . . 43
4.5.1 User Guide . . . 43
4.5.1.1 UI Overview . . . 43
4.5.1.2 Adding new objects . . . 44
4.5.1.3 Working with histograms . . . 45
4.5.1.4 Condition System . . . 50
4.5.1.5 Inspecting Dependencies . . . 52
4.5.2 System Details . . . 53
4.5.2.1 Parameter Arrays . . . 54
4.5.2.2 Connection types . . . 54
4.5.3 Data Sources . . . 55
4.5.3.1 Filter Extractor . . . 55
4.5.3.2 MultiHit Extractor . . . 57
4.5.4 Operators . . . 59
4.5.4.1 Calibration . . . 59
4.5.4.2 Previous Value . . . 60
4.5.4.3 Difference . . . 61
4.5.4.4 Sum / Mean . . . 61
4.5.4.5 Aggregate Operations . . . 62
4.5.4.6 Array Map . . . 62
4.5.4.7 1D Range Filter . . . 63
4.5.4.8 2D Rectangle Filter . . . 63
4.5.4.9 Condition Filter . . . 64
4.5.4.10 Expression Operator . . . 64
4.5.4.11 ScalerOverflow . . . 64
4.5.5 Data Sinks . . . 64

4.5.5.1 1D Histogram . . . 64
4.5.5.2 2D Histogram . . . 64
4.5.5.3 Export Sink . . . 65
4.5.5.4 Rate Monitor . . . 66
4.5.6 Conditions . . . 66
4.5.6.1 Interval Condition (1D Gates) . . . 66
4.5.6.2 Polygon Condition . . . 66
4.5.6.3 Expression Condition . . . 66
4.5.7 Readout Data Preprocessing . . . 66
4.5.7.1 Multi Event Processing . . . 66
4.5.7.2 Event Builder . . . 67
4.5.8 More UI structuring and interactions . . . 69
4.5.8.1 Directories . . . 69
4.5.8.2 Drag and Drop . . . 69
4.5.8.3 Plot Grid . . . 69
4.5.8.4 Multiselections . . . 70
4.5.8.5 Copy/Paste . . . 70
4.5.8.6 Import/Export . . . 70
4.5.8.7 Loading foreign analysis files . . . 70
4.6 JSON-RPC remote control support . . . 70
4.6.1 Methods . . . 71
4.6.1.1 getVersion . . . 71
4.6.1.2 getLogMessages . . . 71
4.6.1.3 getDAQStats . . . 71
4.6.1.4 getVMEControllerType . . . 71
4.6.1.5 getVMEControllerStats . . . 71
4.6.1.6 getVMEControllerState . . . 71
4.6.1.7 reconnectVMEController . . . 71
4.6.1.8 getSystemState . . . 72
4.6.1.9 getDAQState . . . 72
4.6.1.10 getAnalysisState . . . 72
4.6.1.11 startDAQ . . . 72
4.6.1.12 stopDAQ . . . 72
4.6.1.13 getGlobalMode . . . 72
4.6.1.14 loadAnalysis . . . 72
4.6.1.15 loadListfile . . . 72
4.6.1.16 closeListfile . . . 73
4.6.1.17 startReplay . . . 73
4.6.1.18 stopReplay . . . 73
4.6.2 Examples . . . 73
4.6.2.1 Raw JSON data . . . 73
4.6.2.2 mvme_jsonrpc_replay.py . . . 73
4.7 EventServer for data export . . . 74
4.7.1 Overview . . . 74
4.7.2 Using the EventServer . . . 75
4.7.3 Using the ROOT client . . . 76
4.8 Metrics . . . 76

5 How-To Guides **77**
5.1 Rate Estimation Setup . . . 77
5.1.1 Timestamp extraction . . . 77
5.1.2 Timestamp calibration . . . 77
5.1.3 Making the previous timestamp available . . . 78
5.1.4 Histogramming the timestamp difference . . . 79
5.2 VM-USB Firmware Update . . . 81

6 Changelog **82**
6.1 Version 1.21.0 . . . 82

6.2 Version 1.20.0 . . . 82
6.3 Version 1.19.0 . . . 82
6.4 Version 1.18.0 . . . 83
6.5 Version 1.17.5 . . . 83
6.6 Version 1.17.4 . . . 84
6.7 Version 1.17.3 . . . 84
6.8 Version 1.17.2 . . . 84
6.9 Version 1.17.1 . . . 84
6.10 Version 1.17.0 . . . 84
6.11 Version 1.16.3 . . . 85
6.12 Version 1.16.2 . . . 85
6.13 Version 1.16.1 . . . 86
6.14 Version 1.16.0 . . . 86
6.15 Version 1.15.2 . . . 87
6.16 Version 1.15.1 . . . 87
6.17 Version 1.15.0 . . . 87
6.18 Version 1.14.4 . . . 87
6.19 Version 1.14.3 . . . 87
6.20 Version 1.14.2 . . . 87
6.21 Version 1.14.1 . . . 88
6.22 Version 1.14.0 . . . 88
6.23 Version 1.13.0 . . . 88
6.24 Version 1.12.2 - works with MVLC FW0037 or earlier . . . 88
6.25 Version 1.11.0 - MVLC FW0037 or later only! . . . 88
6.26 Version 1.10.4 . . . 89
6.27 Version 1.10.3 . . . 89
6.28 Version 1.10.2 . . . 89
6.29 Version 1.10.1 . . . 89
6.30 Version 1.10.0 . . . 89
6.31 Version 1.9.2 . . . 90
6.32 Version 1.9.1 . . . 90
6.33 Version 1.9.0 . . . 90
6.34 Version 1.8.2 . . . 91
6.35 Version 1.8.1 . . . 91
6.36 Version 1.8.0 . . . 91
6.37 Version 1.7.2-1 . . . 91
6.38 Version 1.7.2 . . . 91
6.39 Version 1.7.1 . . . 92
6.40 Version 1.7.0 . . . 92
6.41 Version 1.6.3 . . . 93
6.42 Version 1.6.2 . . . 93
6.43 Version 1.6.1 . . . 93
6.44 Version 1.6.0 . . . 94
6.45 Version 1.5.0 . . . 94
6.46 Version 1.4.9.5 . . . 95
6.47 Version 1.4.9.4 . . . 95
6.48 Version 1.4.9.3 . . . 95
6.49 Version 1.4.9.2 . . . 95
6.50 Version 1.4.9 . . . 95
6.51 Version 1.4.8.2 . . . 96
6.52 Version 1.4.8.1 . . . 96
6.53 Version 1.4.8 . . . 96
6.54 Version 1.4.7 . . . 97
6.55 Version 1.4.6 . . . 97
6.56 Version 1.4.5 . . . 98
6.57 Version 1.4.4 . . . 98
6.58 Version 1.4.3 . . . 98
6.59 Version 1.4.2 . . . 98

6.60 Version 1.4.1 . . . 99
6.61 Version 1.4.0 . . . 99
6.62 Version 1.3.0 . . . 99
6.63 Version 1.2.1 . . . 100
6.64 Version 1.2.0 . . . 100
6.65 Version 1.1.0 . . . 101
6.66 Version 1.0.1 . . . 101
6.67 Version 1.0.0 . . . 101
6.68 Version 0.9.6 . . . 102
6.69 Version 0.9.5.5 . . . 102
6.70 Version 0.9.5.4 . . . 102
6.71 Version 0.9.5.3 . . . 102
6.72 Version 0.9.5.2 . . . 102
6.73 Version 0.9.5.1 . . . 103
6.74 Version 0.9.5 . . . 103
6.75 Version 0.9.4.1 . . . 104
6.76 Version 0.9.4 . . . 104
6.77 Version 0.9.3 . . . 104
6.78 Version 0.9.2 . . . 104
6.79 Version 0.9.1 . . . 105

Index **106**

CHAPTER

ONE

INTRODUCTION

mvme is a VME data acquisition solution by mesytec aimed at nuclear physics experiments involving a single VME controller. The goal of this project is to provide an easy to setup, easy to use, cross-platform data acquisition system with basic data visualization and analysis capabilities.

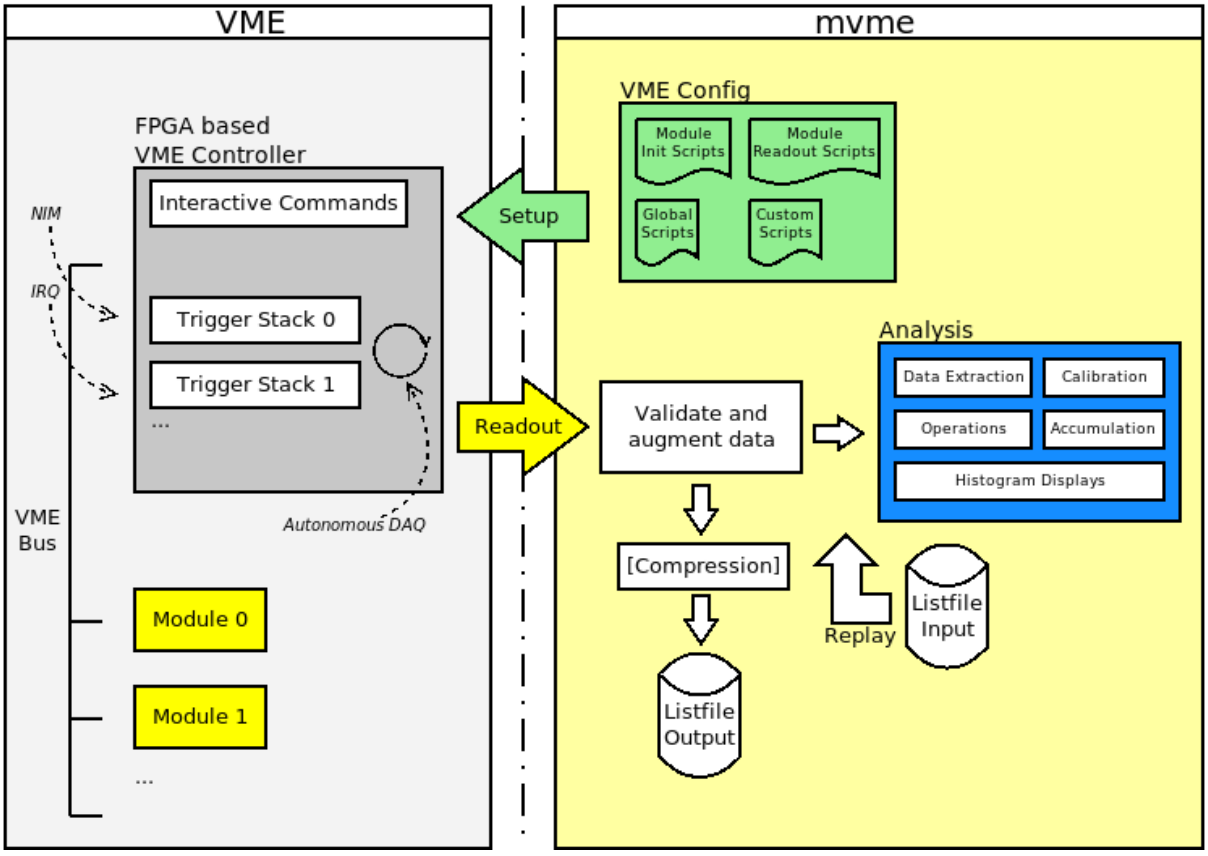


Fig. 1.1: mvme overview

1.1 Features

- High-rate, low-latency VME module readout
- Supports the following VME Controllers:
  - **Mesytec MVLC**
    - USB3, USB2 and Ethernet connectivity
    - Lowest-latency readout, lowest deadtime
    - Flexible Trigger and I/O logic module

    - Possibility to use optical fiber based Ethernet via onboard SFP cage
    - GBit/s Ethernet connection required if using Ethernet!
  - **WIENER VM-USB** (USB2, readout rates of up to 15 MB/s)
  - **Struck SIS3153** (Ethernet)
    - Readout rates of up to 35 MB/s from a single module.
    - Optimized setups can yield around 52 MB/s.
    - GBit/s Ethernet connection required!
- Easy creation and configuration of the VME setup
  - Multiple event triggers are possible (NIM, IRQ, periodic readout)
  - Multiple modules can be read out per trigger
  - Flexible VME module setup using configuration scripts
  - Not limited to mesytec VME modules
- Live histogramming of readout data (1D and 2D)
- Rate Monitoring of internal system rates and external rates generated from readout data (e.g. scaler modules, event counters).
- Flexible VME module data extraction using matching and filtering on the bit level
- Graphical analysis UI
- Optional compression of output listfiles
- Replays of recorded listfile data
- JSON-RPC remote control interface over TCP

  Allows to remotely start and stop DAQ runs and request status information.
- Export of readout data to ROOT object trees.
- The analysis side is detached from the readout side. This means analysis complexity does not have a negative effect on readout performance. Instead during a live DAQ run the analysis side will only get as much data as it can process.

1.2 High-level overview

1.2.1 The VME side

mvme achieves high data rate, low-latency VME readout by using the VME controllers **autonomous DAQ mode**. In this mode the controller executes lists of commands (**trigger stacks**) upon activation of a specific trigger condition. Data generated by the execution of trigger stacks is buffered and then sent over USB or Ethernet to the controlling computer.

In mvme the physical VME setup is described as a tree of objects with **events** as top-level nodes. Each event has a trigger condition (e.g. *Interrupt* or *NIM*) and contains the **modules** to be read out on every activation of the trigger.

A **module** in mvme has a collection of *VME Scripts*. These scripts contain the module configuration (module specific parameters, multicast setup, etc.) and the commands required to perform the readout. The readout commands of all modules belonging to the same event are combined to form the trigger stacks uploaded to the VME controller.

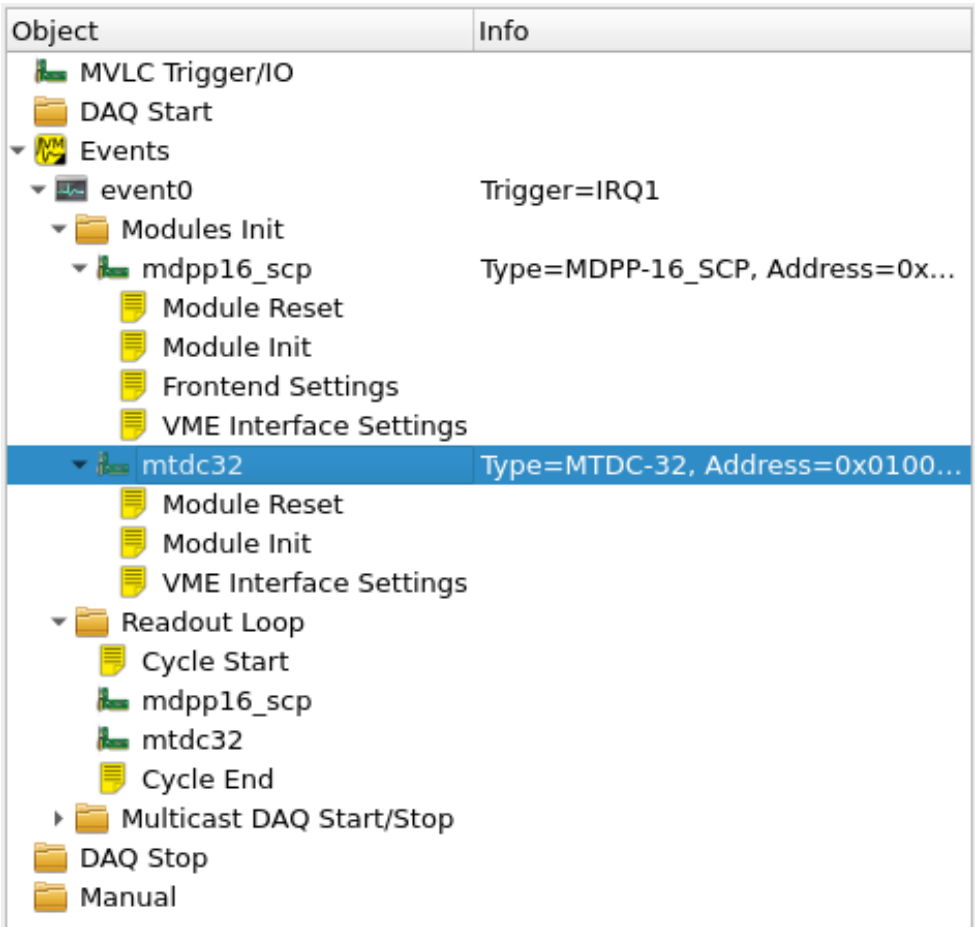


Fig. 1.2: VME setup with one event containing two modules

The VME setup also describes the structure of the data that is expected to be read out and thus allows the software to validate the received data stream.

For mesytec modules fully commented initialization files are bundled with mvme and can be loaded as templates.

1.2.2 Analysis

mvme contains an analysis system that allows parameter extraction (e.g. ADC values per channel), calibration, accumulation and visualization of data both during a DAQ run and while replaying from file. Additionally a set of built-in operators can be used to perform calculations and transformations on the data as it flows through the system.

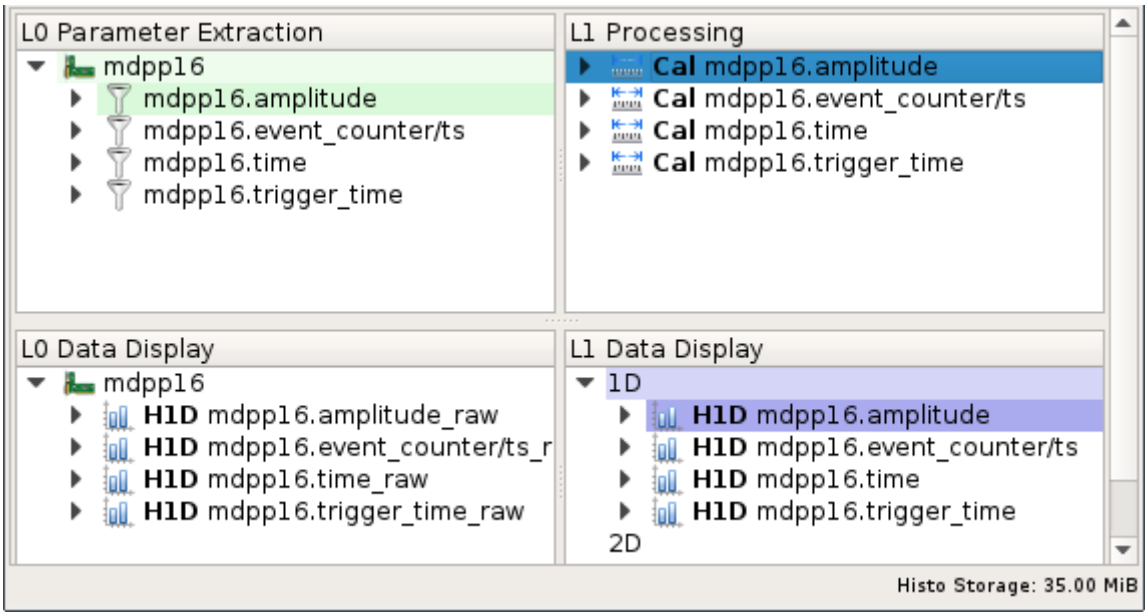


Fig. 1.3: Analysis UI with MDPP-16 default objects

The structure defined by the VME configuration is also present in the analysis: modules which are read out as a result of the same trigger condition are grouped together in an event.

The system itself models dataflow from **sources**, through **operators**, into **sinks**. Data is transported in the form of **parameter arrays** with each element carrying the parameters numeric value and additional meta information.

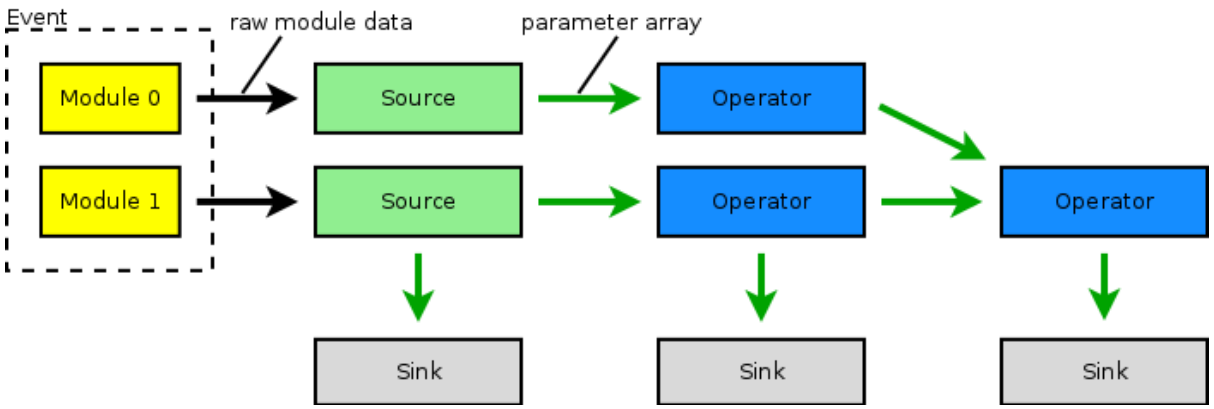


Fig. 1.4: Example analysis dataflow

Sources are data extractors that are directly attached to a VME module. A source receives each data word that was read out from the module in response to a trigger condition. Sources are used to split the data into logical parts, e.g *Amplitude* and *Time* data and to extract the corresponding raw values.

Operators are logic pieces used to perform calculations on the data (e.g. calibrate raw ADC values to voltage). Operators can have multiple inputs and produce a single output array.

Sinks are data accumulators that do not produce any output parameters. Currently 1D and 2D histograms, a rate monitor and a file exporter are implemented.

Output parameters of sources and operators can be inspected at runtime. Objects can be added, removed and modified even while the DAQ or a replay is running. Changes are effective immediately.

1.2.3 The DAQ process

When requested to start a data acquisition run mvme performs the following steps:

- Initialize the VME controller using information from the VME configuration
- Setup modules using the module VME scripts
- Switch the controller into autonomous DAQ mode
- Repeat until DAQ is stopped:
  - Read a data buffer from the VME controller
  - Validate the structure of the received data
  - Augment the data with mvme specific meta data
  - Write data to the listfile (optionally using compression)
  - Pass data to the *Analysis*
- Tell the controller to leave DAQ mode
- Close the listfile

Note

Data acquisition and writing the data to file have the highest priority in mvme. If during a DAQ run the analysis system cannot keep up with the incoming data rate some buffers will not be passed on to the analysis.

The fraction of *processed buffers / total buffers* is called the *analysis efficiency* and is shown in the bottom status bar of the analysis window. Hovering of the Efficiency number shows a tooltip with detailed buffer counts.

When replaying from file *all* buffers are passed to the analysis.

CHAPTER

TWO

INSTALLATION

2.1 System Requirements

- A recent 64-bit Linux distribution or a 64-bit version of Windows 7 or later.
- One of the supported VME Controllers:
  - mesytec MVLC (USB3/2, GBit/s Ethernet)
  - WIENER VM-USB (USB2)
  - Struck SIS3153 (GBit/s Ethernet)
- At least 4 GB RAM is recommended.
- A multicore processor is recommended as mvme itself can make use of multiple cores: readout, data compression, analysis and the user interface all run in separate threads.

2.1.1 Mesytec MVLC

- Firmware updates for the MVLC can be found here: https://mesytec.com/downloads/firmware%20updates/MVLC/
- When using the MVLC via USB the drivers for the FTDI USB Chip are usually shipped with the operating system. If your system does not recognize the device you can get the drivers from the FTDI website: https://www.ftdichip.com/Drivers/D3XX.htm.

  Direct link to the driver installer: https://www.ftdichip.com/Drivers/D3XX/FTD3XXDriver_WHQLCertified_v1.3.0.4_Installer.exe.zip
- Using the MVLC via Ethernet requires a GBit/s network connection.

2.1.2 WIENER VM-USB (Legacy)

- WIENER VM-USB VME Controller with a recent firmware

  The VM-USB firmware can be updated from within mvme. See *VM-USB Firmware Update* for a guide.
- Latest USB chipset driver for your system.

  Updating the driver is especially important for Windows versions prior to Windows 10 in combination with a NEC/Renesas chipset (frequently found in laptops). The driver shipped by Microsoft has a bug that prevents libusb from properly accessing devices. See the libusb wiki for more information.
- USB Driver: libusb-0.1 (Linux) / libusb-win32 (Windows)

  The windows installer can optionally run Zadig to handle the driver installation.

2.1.3 Struck SIS3153 (Legacy)

- The SIS3153 controller requires a GBit/s ethernet connection.

2.2 Installation Steps

2.2.1 Linux

The mvme archives for Linux include all required libraries. The only external dependency is the GNU C Library glibc. When using a modern Linux distribution no glibc versioning errors should occur.

To install mvme unpack the archive and source the mvme env script which will setup your `PATH` and `LD_LIBRARY_PATH`:

```
$ tar xf mvme-1.3.0-Linux-x64.tar.bz2
$ source ./mvme-1.3.0-Linux-x64/bin/initMVME
$ mvme
```

2.2.1.1 MVLC_USB and VM-USB Device Permissions

To be able to use the MVLC_USB or the VM-USB VME Controllers as a non-root user a udev rule to adjust the device permissions needs to be added to the system.

For the MVLC a udev rules file is contained in the installation directory under `extras/mvlc/51-ftd3xx.rules`. Copy this file to your udev rules directory (usually `/etc/udev/rules.d/`).

The rules file for the VMUSB can be found under `extras/vm-usb/999-wiener-vm_usb.rules`.

After copying the file reload udev using `service udev reload` or `/etc/init.d/udev reload` or `service systemd-udev reload` depending on your distribution or simply reboot the machine. The controller also has to be reconnected to your PC for the device permissions to update.

2.2.2 Windows

Run the supplied installer and follow the on screen instructions to install mvme.

2.2.2.1 MVLC USB Driver

If Windows does not recognize the MVLC when plugged in via the front USB and powered in a VME crate, your Windows installation might be missing the FTDI USB driver. The driver installer can be found on the FTDI website: https://www.ftdichip.com/Drivers/D3XX/FTD3XXDriver_WHQLCertified_v1.3.0.4_Installer.exe.zip.

A non-installer version is also available on this page: https://www.ftdichip.com/Drivers/D3XX.htm.

2.2.2.2 VM-USB only: Driver Installation

To be able to use the VM-USB VME Controller the *libusb-win32* driver needs to be installed and registered with the device. An easy way to install the driver is to use the Zadig USB Driver Installer which comes bundled with mvme. You can run Zadig at the end of the installation process or at a later time from the mvme installation directory.

In the Zadig UI the VM-USB will appear as *VM-USB VME CRATE CONTROLLER*. If it does not show up there's either a hardware issue or another driver is already registered to handle the VM-USB. Use *Options -> List All Devices* to get a list of all USB devices and look for the controller again.

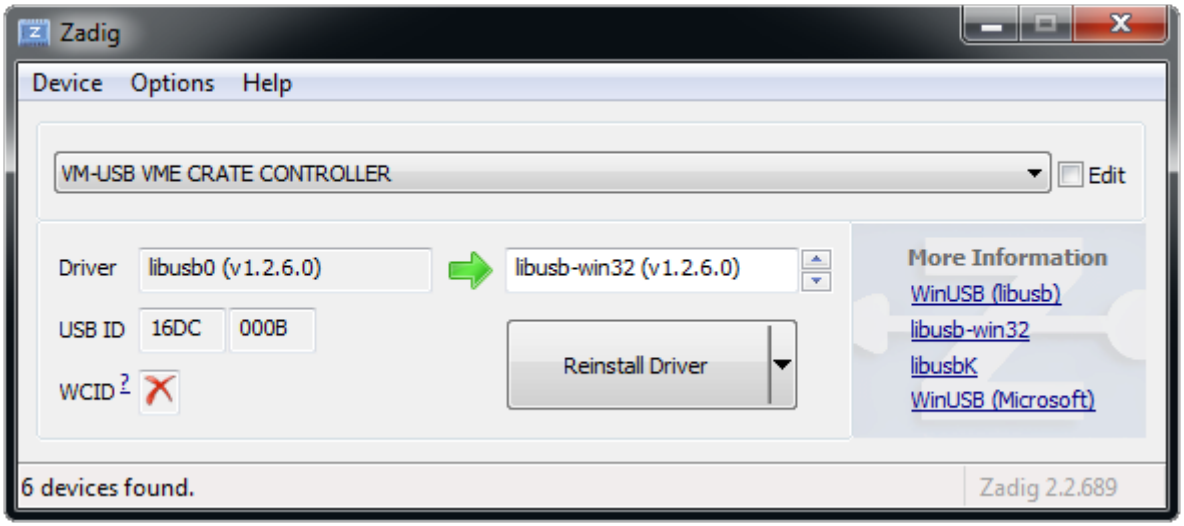


Fig. 2.1: Zadig with VM-USB and libusb-win32 selected

Make sure *libusb-win32* is selected as the driver to install, then click on *Install Driver*. Zadig will generate a self-signed certificate for the driver and start the installation process.

It is highly recommended to restart your system after driver installation, especially if you replaced an existing driver. Otherwise USB transfer errors can occur during VME data acquisition!

In case you want to manually install the driver a ZIP archive can be found here: libusb-win32.

2.2.3 Installation from source

The mvme sources are available on github: https://github.com/flueke/mvme

Refer to the README file for a list of required dependencies and build instructions for Linux, Windows and Mac OS X.

2.3 Ethernet DHCP/ARP setup

When using the MVLC via Ethernet or the SIS3153 controller some network setup has to be done.

The easiest way to get a working setup is if you are running a DHCP server on your network. Both of the controllers will request an IPv4-Address and a hostname via DHCP after powerup.

The MVLC will request the hostname `mvlc-NNNN` where `NNNN` is the serial number shown on the front-panel near the Ethernet port.

The SIS3153 requests a hostname of the form `sis3153-0DDD` where `DDD` is the decimal serial number as printed on the board.

After the DHCP phase the two controllers should be reachable via their hostnames. You can verify this by opening a command prompt and running

```
ping mvlc-0010
```

for the MVLC with serial number 10.

Note

The lease time offered by your local DHCP server must be set to some value **longer than 1 hour**. Since firmware version FW0038 the MVLC will renew its lease every hour.

2.3.1 Using a manual ARP entry

In case DHCP with hostname assignment should not or cannot be used an alternative approach is to manually associate the MAC-address of the controller with an IP-address.

- Obtaining the controllers MAC-address

  The first step is to figure out the controllers MAC-address. This is the serial-number dependent Ethernet address of the controller.

  For the MVLC the MAC-address is `04:85:46:d2:NN:NN` where the `NN:NN` is the serial number of the MVLC in decimal. So for MVLC-0015 the full MAC-address is `04:85:46:d2:00:15`.

  The MAC-address of the SIS3153 is `00:00:56:15:3x:xx` where `x:xx` is the serial number in hexadecimal. So for my development controller with S/N 42 the serial becomes `0x2a` and the resulting MAC-address is `00:00:56:15:30:2a`.

With the MAC-address at hand we can now create an IPv4-address to MAC-address mapping in the operating systems ARP table.

This step is specific to the operating system and will require root/admin permissions. The below examples associate the IP-address `192.168.100.42` with the controllers MAC-address. You have to change the IP-address to match your local network setup, otherwise the operating system does not know how to reach the controller.

- Creating the ARP entry under linux:

  With root permissions an ARP entry can be addded this way:

  ```
  arp -s 192.168.100.42 04:85:46:d2:00:15
  ```

  To make the entry permanent (at least on debian and ubuntu systems) the file /etc/ethers can be used. Add a line like this to the file:

  ```
  04:85:46:d2:00:15 192.168.100.42
  ```

  This will take effect on the next reboot (or when restarting the networking services I think).

- Creating the ARP entry under windows:

  Open a `cmd.exe` prompt with **administrator** permissions and use the following command to create the ARP entry:

  ```
  arp -s 192.168.100.42 04-85-46-d2-00-15
  ```

At this point mvme should be able to connect to the controller using the IP-address specified above.

To further diagnose connection issues the following commands can be used to ping the MVLC:

```
echo -ne '\xF3\x00\x00\x00' | nc -uq0 192.168.100.42 32768
ping 192.168.100.42
```

In the first line `echo` and `netcat` (`nc`) are used to send an empty but valid command frame to the MVLC. This is required to make the MVLC leave the DHCP phase and accept the supplied destination IP-address. The second command attempts to ping the controller. If everything is setup correctly the ping command should receive responses from the controller.

For the SIS3153 only the ping command is needed.

CHAPTER

THREE

QUICKSTART GUIDE

The quickstart guide explains how to create a simple setup using the mesytec MVLC VME controller and one mesytec VME module. The modules internal pulser is used to generate test data. Data readout is triggered by the module itself using IRQ1 on the VME bus.

Note

In this example an MDPP-16 with the SCP firmware is used but any **mesytec** VME module should work. For other modules the value written to the pulser register (0x6070) might need to be adjusted. Refer to the modules manual and the VME templates for details.

- Start mvme and create a new workspace directory using the file dialog that should open up. This directory will hold all configuration files, recorded listfiles, exported plots, etc.
- Three windows will open:
  - A main window containing DAQ controls, the VME configuration tree and a statistics area.
  - The analysis window. As there are no VME events and modules defined yet the window will be empty.
  - A log view where runtime messages will appear.

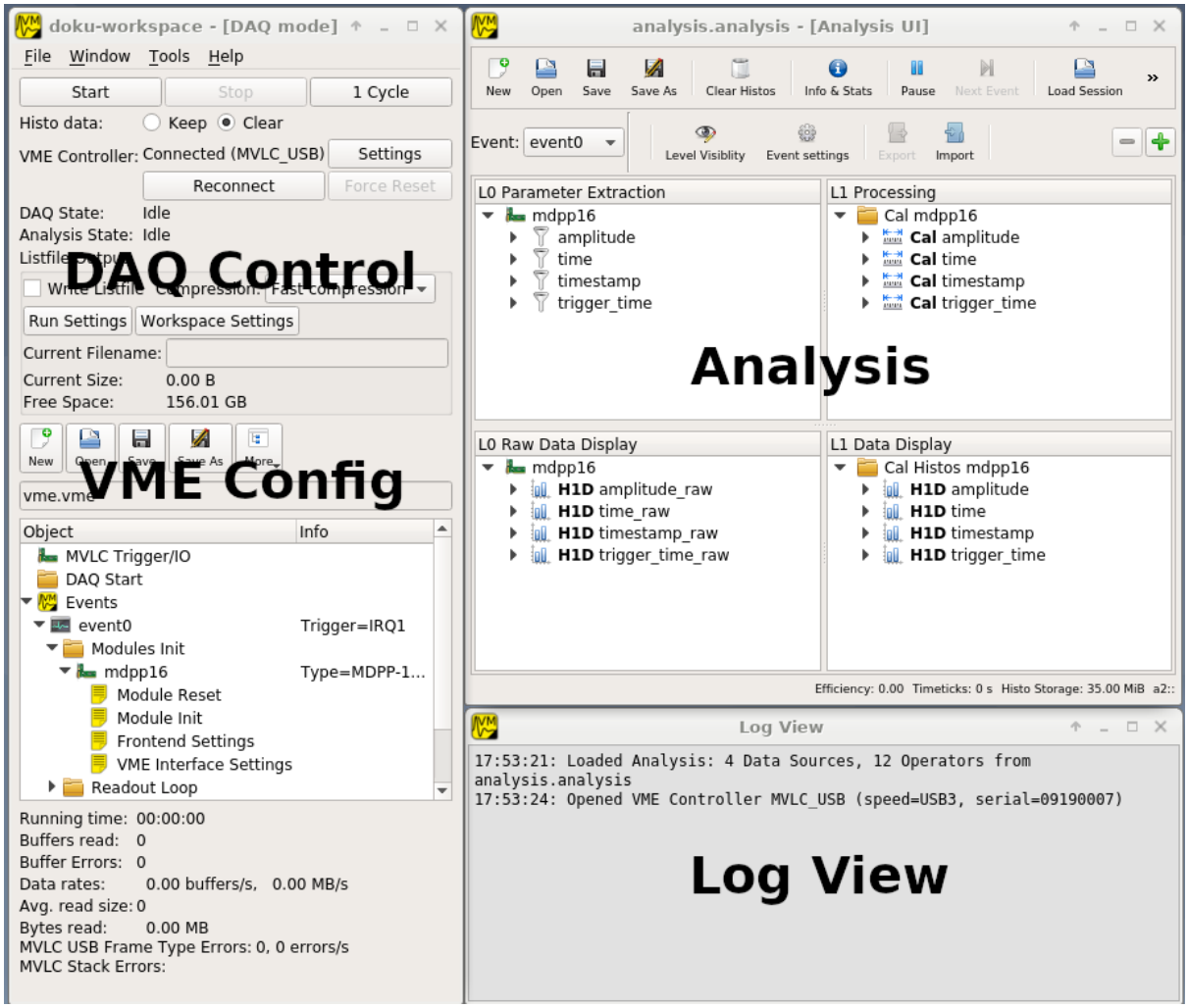


Fig. 3.1: GUI overview

Press the `Settings` button in the DAQ Control area and select the correct controller type (MVLC_USB in this example). Assuming the controller is connected and powered on mvme will automatically find and use it. Also a message containing information about the controller will appear in the Log View.

3.1 VME Setup

- Select the mvme main window (*Ctrl+1*) containing the *VME Config* area.
- Create a VME event:
  - Right-click the *Events* entry in the VME tree and select *Add Event*.
  - Select *Interrupt* in the *Condition* combobox. Keep the default of *IRQ Level = 1*.

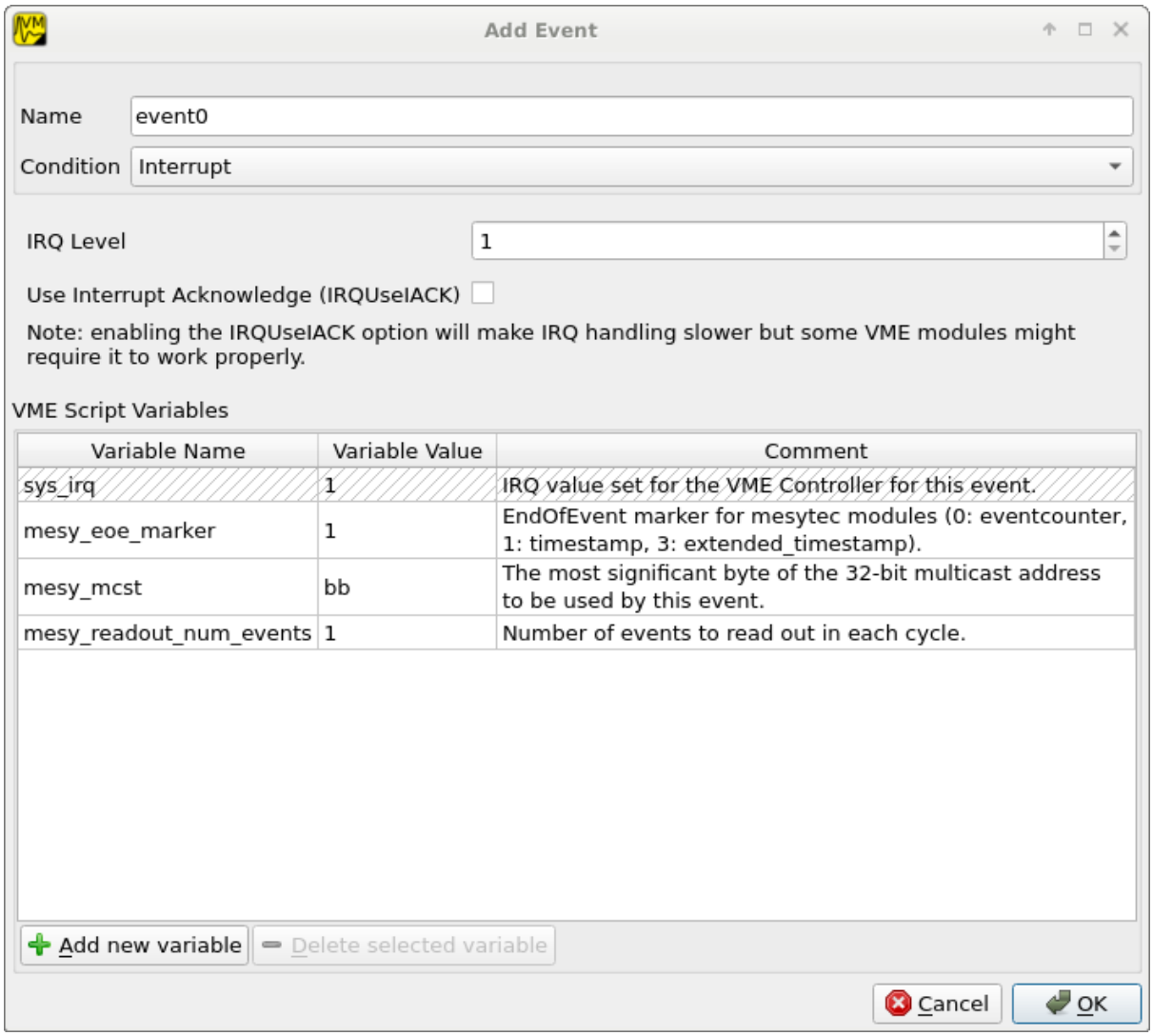


Fig. 3.2: Event Config Dialog

- Create a VME module:
  - Right-click the newly created event (called “event0” by default) and select *Add Module*.
  - Select *MDPP-16_SCP* from the module type list. If you changed the modules address encoders adjust the *Address* value accordingly (the address encoders modify the 4 most significant hex digits).

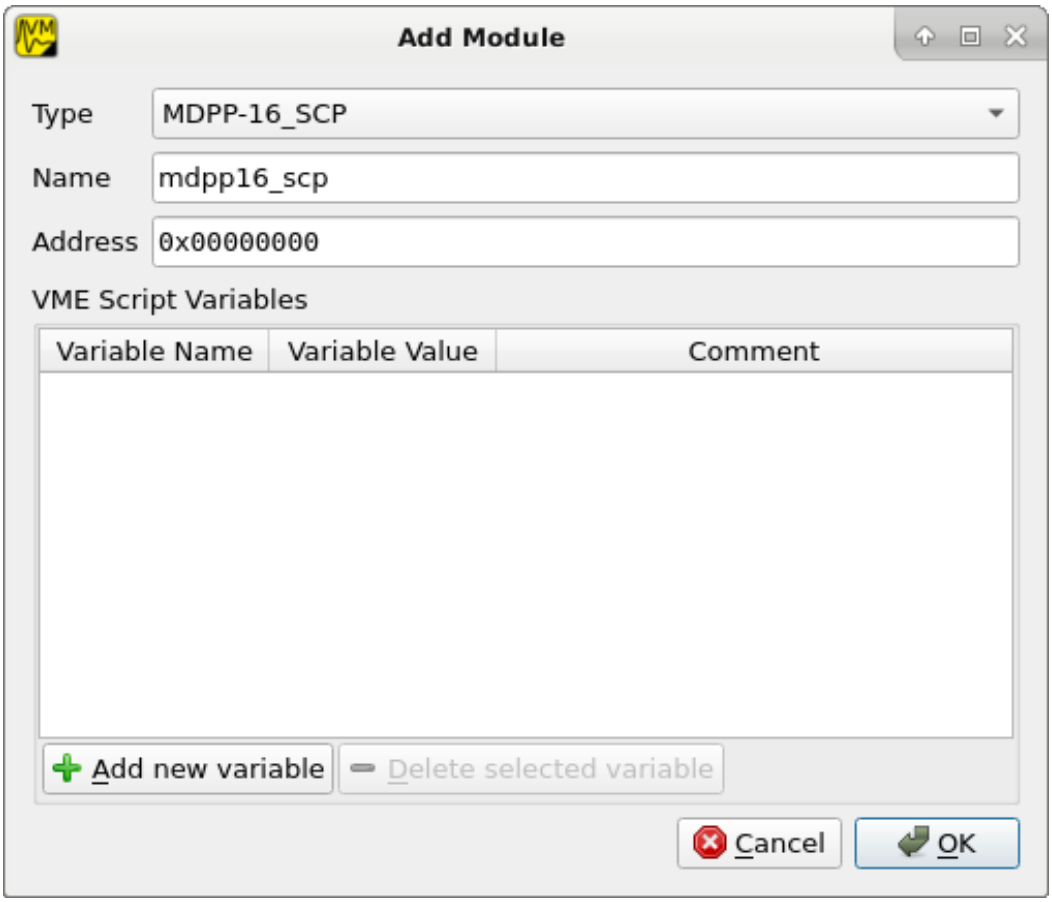


Fig. 3.3: Module Config Dialog

The VME Config should now look like similar to *VME Config tree*.

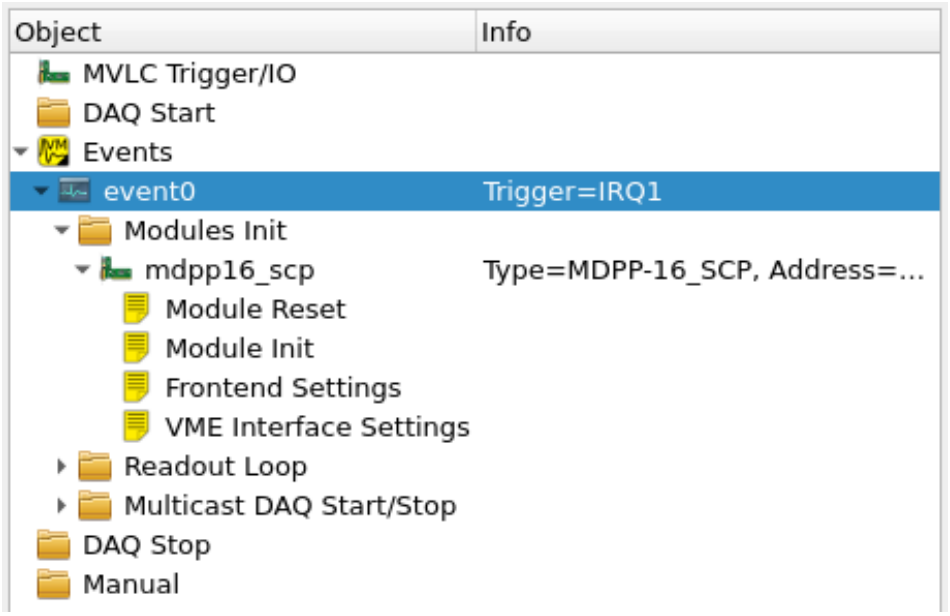


Fig. 3.4: VME Config tree

- Double-click the *Module Init* entry to open a VME Script Editor window. Scroll to the bottom of the editor window or use the search bar to search for `pulser` and adjust the register value for the modules internal pulser:

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

`0x6070 1`

This line tells mvme to write the value `1` to register address `0x6070`. The address is relative to the module base address.

- Click the *Apply* button on the editors toolbar to commit your changes to the VME configuration. Close the editor window.

3.2 Analysis Setup

- Activate the *Analysis UI* window (the shortcut is `Ctrl+2`). The event containing the module just created should be visible in the UI.
- Right-click the mdpp16 module and select *Generate default filters*. Choose *Yes* in the messagebox that pops up. This will generate a set of data extraction filters, calibration operators and histograms for the module.

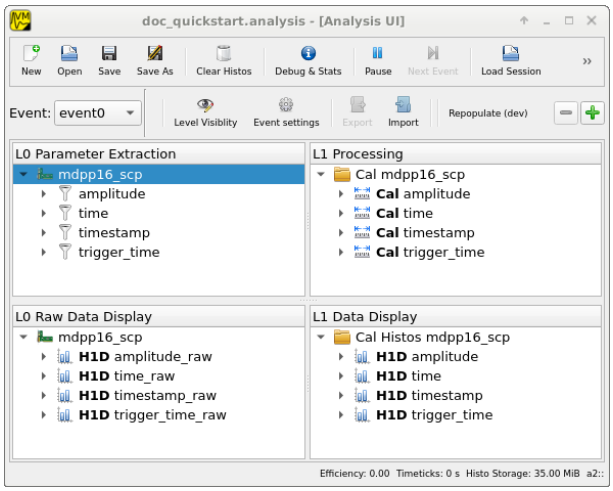


Fig. 3.5: Analysis UI with MDPP-16 default objects

3.3 Starting the DAQ

Activate the main window again (`Ctrl+1`). Make sure the *VME Controller* is shown as *Connected* in the top part of the window.

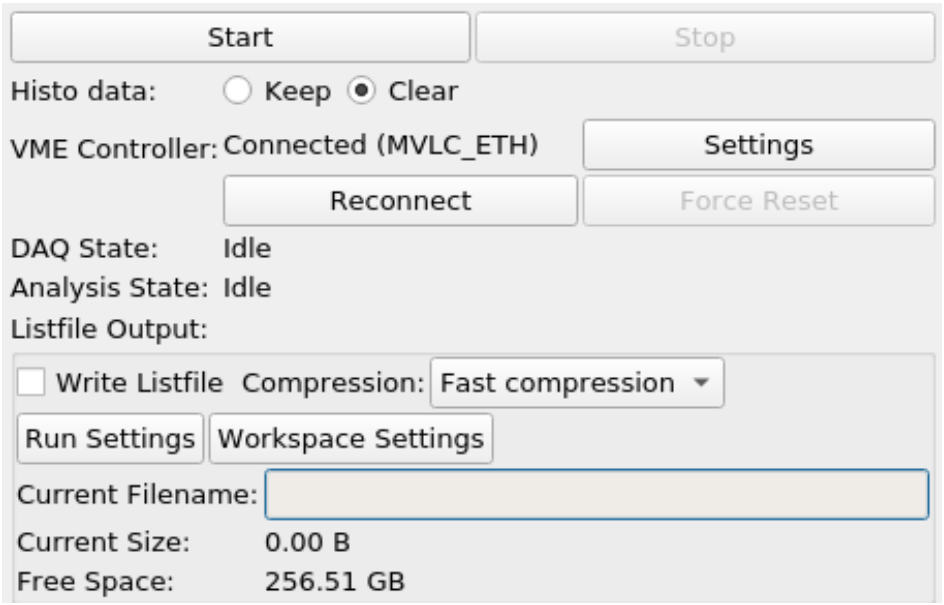


Fig. 3.6: DAQ control

Optionally uncheck the box titled *Write Listfile* to avoid writing the test data to disk. If left checked a listfile will be created for each newly started DAQ run. This listfile is placed in the workspace directory under `listfiles`. It is a standard ZIP archive containing the raw readout data and copies of the current analysis setup and the log buffer contents.

The naming scheme of the listfiles can be adjusted via the `Run Settings` dialog. Note that mvme will never overwrite existing listfiles even if you manually adjust the `Next Run Number` value.

Press the *Start* button to start the DAQ. Check the *Log View* (`Ctrl+3`) for warnings and errors.

In the *Analysis UI* double-click the histogram entry called *amplitude_raw* (bottom-left corner in the *L0 Data Display* tree) to open a histogram window.

If data acquisition and data extraction are working properly you should see new data appear in the histogram. Use the spinbox at the top right to cycle through the individual channels.

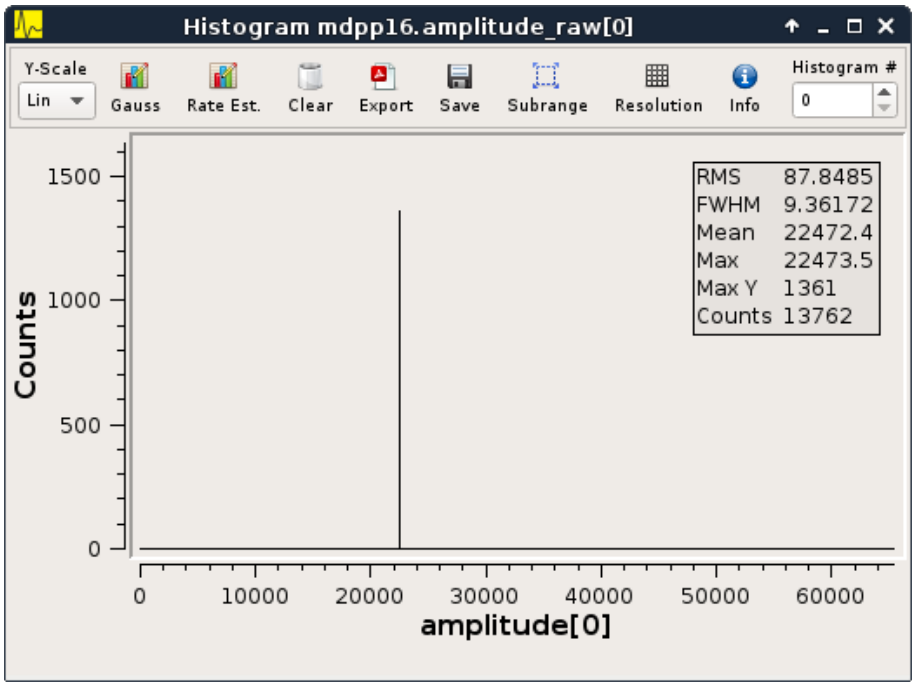


Fig. 3.7: Amplitude histogram

You can pause and/or stop the DAQ at any time using the corresponding buttons at the top of the main window.

3.4 Event Counter readout

Optionally a second event used to read out the modules event counter registers can be created. This event will be triggered periodically by the VME controller.

- Right-click *Events*, choose *Add Event*
- Set *Condition* to *Periodic* and the period to `1.0s`
- Right-click the newly created event, choose *Add Module*
- Select *MesytecCounter* as the module type
- Enter the same address as used for the MDPP-16 above

CHAPTER

FOUR

REFERENCE

4.1 DAQ / Replay controls

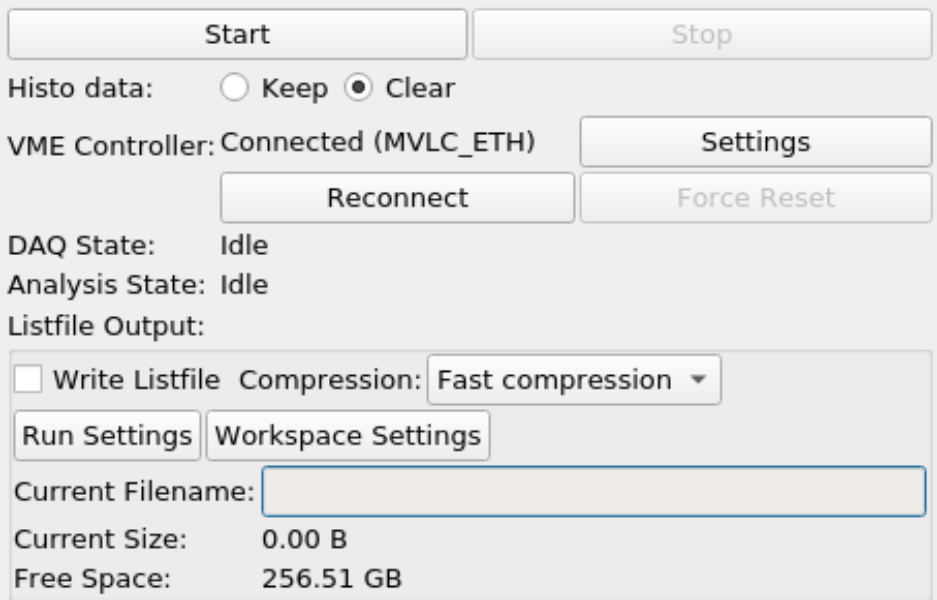


Fig. 4.1: DAQ controls

The effects of the buttons depend on the current mode - DAQ or replay - and the current state of the system:

Table 4.1: DAQ control actions

Action	DAQ mode	Replay mode
Start	• Run *DAQ startup procedure* • Open new listfile	Start replay from beginning of file
Stop	• Run *DAQ stop procedure* • Close listfile	Stop and rewind to beginning of file
Pause	• Leave DAQ mode • No special procedures are run	Pause replay

The **Start** button allows to choose what should happen with existing histogram data. Selecting **Clear** will clear all

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

histograms in the current analysis before accumulating new data. Using **Keep** allows to accumulate the data from multiple replays or DAQ runs into the same histograms.

The **Reconnect** button will attempt to reconnect to the current VME controller.

If **Write Listfile** is checked a new output listfile will be created when starting a DAQ run. The file will be created in the **listfiles** subdirectory of the current workspace. The filename is based off the current timestamp to make it unique.

Use the **Compression** drop down to switch between a fast compression mode (ZIP level 1) and no compression at all (ZIP level 0). Even without compression the raw readout data is still placed in a ZIP archive together with the analysis and log buffer contents.

Note: testing showed that higher compression levels did not yield a significant increase in data reduction but had a high performance impact so the higher level modes are not currently selectable via the GUI.

Use the **Run Settings** button to change settings regarding the naming scheme of the listfiles produced during DAQ runs.

The **Workspace Settings** dialog allows setting the experiment name (used by the 'Event Server' component) and enabling/disabling the JSON-RPC and Event Server components.

4.1.1 Replaying data from listfiles

To replay data from a single listfile use **File -> Open listfile** and choose a *.zip* or *.mvmelst* file.

When opening a listfile the VME config included inside the file is loaded and will replace the current config. The global mode will be switched to *Listfile*. To go back to DAQ mode use *File -> Close Listfile*.

There is also a dedicated 'Listfile Browser' window which can be opened via `Window -> Listfile Browser (Ctrl+4)`. In addition to replaying from single files the listfile browser allows to replay data from split listfiles (ending in `partNNN.zip`). Make sure `replay all parts` is checked before opening the first part of a split listfile, then start the replay normally. The partial listfile archive currently being read is printed to the log window. To the analysis it looks like the data stream is coming from a single file.

4.2 VME configuration

4.2.1 Structure

The VME configuration in mvme models the logical VME setup. *Modules* that should be read out as a result of the same trigger condition are grouped together in an *Event*:

```
Event0
    Module0.0
    Module0.1
    ...

Event1
    Module1.0
    Module1.1
    ...
```

The type of available trigger conditions depends on the VME controller in use. With the WIENER VM-USB the following triggers are available:

- NIM

  One external NIM input

- IRQ1-7

  The standard VME interrupts

- Periodic

  VM-USB supports one periodic trigger which is executed every `n * 0.5s` (*Period*) or on every `m-th` data event (*Frequency*). If both values are set both internal counters are reset on each activation of the trigger. Refer to section 3.4.3 of the VM-USB manual for details.

4.2.2 Module and Event configuration

The module and event configuration is done using *VME scripts* which contain the commands necessary to initialize and readout each module.

At the module level the following phases are defined:

- Reset

  Reset the module to a clean default state.

- Init

  Setup the module by writing specific registers.

- Readout

  The code needed to readout the module whenever the trigger condition fires.

The event level distinguishes between the following phases:

- Readout Cycle Start / End

  Inserted before / after the readout commands of the modules belonging to this event. The *Cycle Start* script is currently empty by default, the *Cycle End* script notifies the modules that readout has been performed. By default this is done by writing to the multicast address used for the event.

- DAQ Start / Stop

  Executed at the beginning / end of the a DAQ run. The purpose of the *DAQ Start* script is to reset module counters and tell each module to start data acquisition. *DAQ End* is used to tell the modules stop data acquisition. By default both scripts again use the multicast address of the corresponding event.

4.2.2.1 Variables

Since mvme-0.9.7 both event and module configs can contain a set of variables whose values can be used inside *VME scripts*.

Variables from the event scope are available inside the event scripts and all child module scripts. Variables defined at module scope are available to the module init and readout scripts. Module variables override variables defined at event scope.

When adding a new VME event a set of standard variables is created:

- sys_irq

  A system variable which is automatically set to the IRQ value used to trigger the event or 0 if the event is not IRQ triggered.

- mesy_mcst

  Contains the highest 8-bits of the VME multicast (MCST) address setup for the event. This is used to initialize the events member modules and to simultaneously write each of the modules *readout_reset* register at the end of each readout cycle.

- mesy_eoe_marker

  EndOfEvent marker for mesytec modules (register 0x6038).

- mesy_readout_num_events

  Number of events to read out from each module per readout cycle. This is used to set the values of the `irq_fifo_threshold (0x601E)` and `max_transfer_data (0x601A)` of mesytec module. By default `irq_fifo_threshold` is set to `mesy_readout_num_events + 1`.

The default VME templates shipped with mvme assume that the above variables are defined and contain valid values for mesytec modules.

In the GUI variables can be viewed, added and modified by editing the respective object (**Edit Event Settings** for events, **Edit Module Settings** for modules).

For event configs there is also a special variable editor available via the **Edit Variables** action button or context menu entry. This editor allows to edit variables during a DAQ run and automatically executes VME scripts that are affected by changes to variable values.

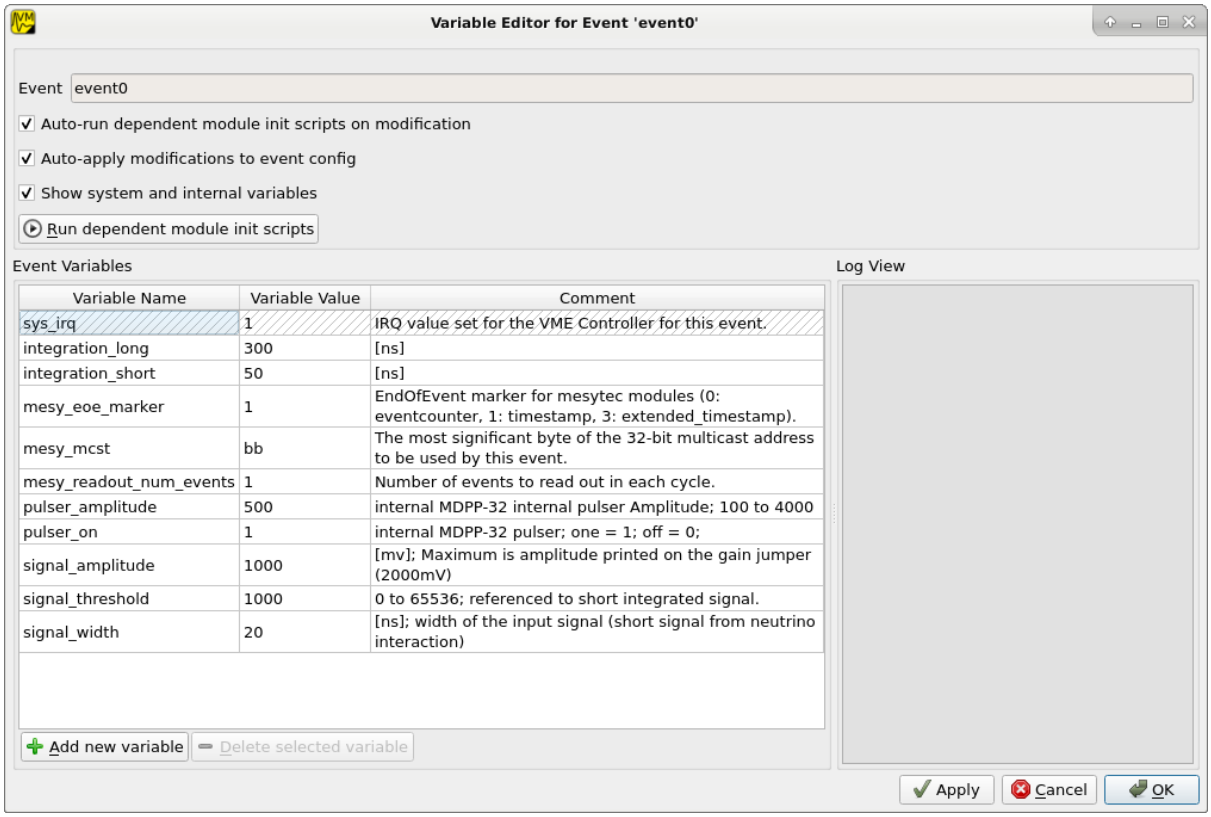


Fig. 4.2: Event Variable Editor with system and custom variables

4.2.3 DAQ startup procedure

- Reset and setup the VME controller
- Assemble readout code from configured Events

  For each Event do:

  - Add *Cycle Start* script
  - For each Module:
    - Add Module readout script
    - Add “Write EndMarker” command (0x87654321, not needed for the MVLC)
  - Add *Cycle End* script
- Upload the readout code to the controller and activate triggers
- Execute global *DAQ Start* scripts, including the MVLC Trigger/IO script
- Initialize Modules

  For each Event do:

  - For each Module do:
    * Run *Module Reset*
    * Run all *Module Init* scripts
  - Run the Events *Multicast DAQ Start* script
- Set the controller to autonomous DAQ mode

Control is handed to the VME controller. mvme is now reading and interpreting data returned from the controller.

4.2.4 DAQ stop procedure

- Tell the VME controller to leave autonomous DAQ mode
- Read leftover data from the VME controller
- Run the *DAQ Stop* script for each Event
- Execute global *DAQ Stop* scripts

4.2.5 VME Controller specifics

4.2.5.1 Mesytec MVLC

To implement periodic events on the MVLC the global *MVLC Trigger/IO* script is modified by mvme when starting a DAQ run: A *StackStart* unit is connected to a *Timers* unit which is setup with the parameters from the corresponding periodic event. The StackStart unit is then setup to start the command stack for the respective event. These changes are visible in the MVLC Trigger/IO gui immediately after starting the DAQ.

4.2.5.2 SIS3153

If using the SIS3153 VME controller additional commands which activate Lemo OUT2 during execution of the readout will be added to the script for the main readout event. The main event is considered to be the first non-periodic event defined in the VME configuration.

OUT1 and LED_A are activated prior to entering autonomous DAQ mode and deactivated after leaving DAQ mode.

4.2.6 Custom VME Modules

To create a template for a custom VME module start by adding an existing module type to the VME config as a base to start from (e.g. one of the UserModule_xx modules). Then edit the VME scripts and test the module until initialization and readout work correctly.

To save the module as a template right-click the module node and select `Save Module to file`.

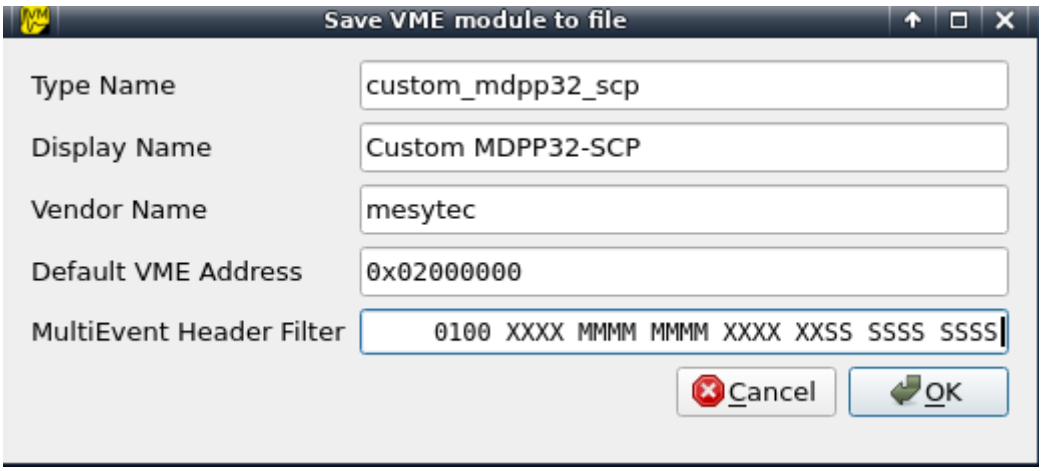


Fig. 4.3: Custom Module properties dialog

The dialog lets you specify the following module properties:

- Type Name: internal module indentifier. Must be unique across all modules defined in mvme. Also used as the base name for newly added module instances.

  E.g. `mdpp16_scp_custom`

- Display Name: User visible module type string, e.g. `MDPP16 SCP Custom`.
- Vendor Name: custom vendor name. Used to group modules together in the UI.
- Default VME Address: the full 32-bit VME address assigned to newly added instance of the module, e.g `0x00010000`
- MultiEvent Header Filter: in case the module supports reading out multiple buffered events using a single VME block transfer this property specifies the bit level filter used to split the mulievent data into separate events. Set a static bitmask to match the module headers and optionally use the character S to specify the bits used to extract the module data size in words. Setting the size bits speeds up the multievent splitting code a bit as it can jump from event to event instead of having to match each data word against the static bitmask.

Accept the dialog and chose a location and filename to save the file. The module file format is JSON so it can be hand-edited if needed.

To add an instance of a custom module to the VME config right-click an event node or the `Modules Init` node and select `Add Module from file`.

4.3 VME Scripts

4.3.1 Overview

VME Scripts are plain text files with one command per line. Comments may be started using the # character. They extend to the end of the line. Alternatively blocks can be commented out starting with `/*` and ending with `*/`.

Scripts belonging to a module (**Module Init Scripts**, **VME Interface Settings**, **Module Reset** and the readout code) will have the **module base address** added to most of the commands. This allows writing scripts containing module-relative addresses only. Exceptions are the *writeabs* and *readabs* commands which do not modify their address argument. The base address can also be temporarily replaced with a different value by using the *setbase* and *resetbase* commands.

The commands below accept the following values for VME address modes and data widths:

Table 4.2: VME Address Modes

Address Mode (<*amode*>)
a16
a24
a32

Additionally raw numeric amode values (e.g. 0x39) are also accepted.

Table 4.3: VME Data Widths

Data Width (<*dwidth*>)
d16
d32

The combination of amode, dwidth and BLT/MBLT/2eSST yields a VME address modifier to be sent over the bus. Internally these non-privileged (aka user) address modifiers will be used:

Table 4.4: VME address modifiers used by mvme

amode	single	BLT	MBLT
a16	0x29		
a24	0x39	0x3b	
a32	0x09	0x0b	0x08
cr	0x2f		

Numbers in the script (addresses, transfer counts, masks) may be specified in decimal, hex or floating point notation using the standard C prefixes (`0x` for hex). Additionally register values may be written in binary starting with a prefix of `0b` followed by `0`s and `1`s, optionally separated by `'` characters.

Example: `0b1010'0101'1100'0011` is equal to `0xa5c3`

CR/CSR addressing is implemented for the MVLC VME controller and recognized by all commands dealing with address modes, e.g. **read**, **write** and **mvlc_read_to_accu**.

4.3.2 Commands

4.3.2.1 Writing

- **write** <*amode*> <*dwidth*> <*address*> <*value*>
- **writeabs** <*amode*> <*dwidth*> <*address*> <*value*>

writeabs uses the given <*address*> unmodified, meaning the module base address will not be added.

There is a short syntax version of the write command: if a line consists of only two numbers separated by whitespace, a write using 32-bit addressing (a32) and 16-bit register width (d16) is assumed. The address is the first number, the value to be written is the second number.

Example: `0x6070 3` is the same as `write a32 d16 0x6070 3`

4.3.2.2 Reading

- **read** <*amode*> <*dwidth*> <*address*> [*'slow'|'late'*] [*'fifo', 'mem'*]
- **readabs** <*amode*> <*dwidth*> <*address*> [*'slow'|'late'*] [*'fifo', 'mem'*]

Reads a single value from the given <*address*>.

readabs uses the given <*address*> unmodified, meaning the module base address will not be added.

The optional argument 'slow'|'late' is implemented for the MVLC only. It causes the VME read to be carried out on the trailing edge of the modules DTACK signal instead of at the leading edge. This is a workaround for VME modules which do signal DTACK while not having the actual data ready yet.

The second optional argument 'fifo'|'mem' is also MVLC only. It controls if the read address is kept as is (fifo) or incremented (mem) for VME reads that are turned into block reads due to the MVLC stack accumulator being set.

During manual script execution the value read from the VME bus is stored in the script local software accumulator. The accumulator can be modified and tested using *accu_mask_rotate* and *accu_test*.

4.3.2.3 Block Transfers

mvme supports the following read-only block transfer commands:

- **blt** <*amode*> <*address*> <*count*>
- **bltfifo** <*amode*> <*address*> <*count*>
- **mblt** <*amode*> <*address*> <*count*>
- **mbltfifo** <*amode*> <*address*> <*count*>
- **mblts** <*amode*> <*address*> <*count*> (**MVLC only**, **word swapped**)

- **mbltsfifo** <*amode*> <*address*> <*count*> (**MVLC only**, **word swapped**)
- **2esst** <*address*> <*rate*> <*count*> (**MVLC only**, **FIFO mode**)
- **2esstfifo** <*address*> <*rate*> <*count*> (**MVLC only**, **FIFO mode**)
- **2esstmem** <*address*> <*rate*> <*count*> (**MVLC only**)
- **2essts** <*address*> <*rate*> <*count*> (**MVLC only**, **FIFO mode**, **word swapped**)
- **2esstsfifo** <*address*> <*rate*> <*count*> (**MVLC only**, **FIFO mode**, **word swapped**)
- **2esstsmem** <*address*> <*rate*> <*count*> (**MVLC only**, **word swapped**)
- **read** <*amode*> <*dwidth*> <*address*> *['slow'|'late'] ['fifo', 'mem']* (**MVLC only**, **used with the stack accu**)
- **readabs** <*amode*> <*dwidth*> <*address*> *['slow'|'late'] ['fifo', 'mem']* (**MVLC only**, **used with the stack accu**)

blt and **bltfifo** transfer <*count*> number of 32-bit words, **mblt** and **mbltfifo** transfer 64-bit words. **2esst** uses the 2eSST protocol for data transfer.

The **fifo** variants do not increment the given starting address.

mblts stands for *MBLT swapped* and is the same as MBLT but swaps the two 32-bit words in each transferred 64-bit word. It is only supported by the MVLC.

The **2esst** and **2essts** commands accept the following <*rate*> arguments:

Table 4.5: 2esst rates

160	160mb	0
276	276mb	1
320	320mb	2

2essts is the word swapped version of **2esst** like **mblts** for MBLT transfers.

The **read** and **readabs** commands usually return a single data word not contained in a block read structure. When used in combination with the MVLC stack accumulator these commands do produce block read structures. Example:

```
mvlc_set_accu 100
read a32 d16 0x0000 mem
```

Performs 100 reads from 0x0000 with incrementing read addresses. The data is yielded in a block read structure in the readout data stream just like, e.g. a **mblt** command would.

4.3.2.4 Miscellaneous

wait

- **wait** <*waitspec*>

Delays script execution for the given amount of time. <*waitspec*> is a number followed by one of `ns`, `ms` or `s` for nanoseconds, milliseconds and seconds respectively. If no suffix is given milliseconds are assumed.

Note

The wait command is only available when directly executing a script from within mvme. It is not supported in command stacks for the MVLC and SIS3153 controllers.

The VMUSB has limited support for the wait command in command stacks with a waitspec resolution of **200 ns** and the maximum possible delay being **51000 ns**.

Example: `wait 500ms # Delay script execution for 500ms`

marker

- **marker** <*marker_word*>

The marker command adds a 32-bit marker word into the data stream. This can be used to separate data from different modules.

setbase/resetbase

- **setbase** <*address*>
- **resetbase**

These commands can be used to temporarily replace the current base address with a different value. **setbase** sets a new base address, which will be effective for all following commands. Use **resetbase** to restore the original base address.

write_float_word

- **write_float_word** <*address_mode*> <*address*> <*part*> <*value*>

The write_float_word command is a helper function for dealing with VME modules using IEEE-754 floating point numbers internally (e.g. the ISEG VHS4030). The command writes a 16-bit part of a 32-bit float into the given register without performing any integer conversions.

Arguments:

- *address_mode*

  The VME address mode: a16, a24 or a32

- *address*

  Address of the register to write to.

- *part*

  One of **upper / 1** and **lower / 0**. The upper part contains the 16 most significant bits of the float, the lower part the 16 least significant bits.

- *value*

  The floating point value using a . as the decimal separator.

Example

```
write_float_word a16 0x0014 upper 3.14
write_float_word a16 0x0016 lower 3.14
```

Writes the 32-bit float value *3.14* to the two 16-bit registers 0x14 and 0x16.

print

Prints its arguments to the log output on a separate line. Arguments are separated by a space by default which means string quoting is not strictly required.

Example

```
print "Hello World!"
print Hello World!  # Quoting the string is optional
```

accu_set

- **accu_set** <*value*> Set the script local accumulator to a constant value.

accu_mask_rotate

- **accu_mask_rotate** <*mask*> <*rotate_amount*>

First applies the 32 bit <*mask*> value to the accu, then left rotates the accu by <*rotate_amount*> bits.

accu_test

- **accu_test** <*compare_op*> <*compare_value*> <*message*>

Compares the current accumulator value against the constant <*compare_value*> and prints a message containing the result. If the comparison fails an error is generated and the script execution is stopped.

Table 4.6: accu_test comparison operators

eq	==
neq	!=
lt	<
lte	<=
gt	>
gte	>=

accu_test_warn

- **accu_test_warn** <*compare_op*> <*compare_value*> <*message*>

Same as *accu_test* but does not stop script execution on failure.

accu_add

- **accu_add** <*value*>

Adds the given value to the script local accumulator.

accu_write

- **accu_write** <*amode*> <*dwidth*> <*address*> <*value*>

Same as *write* but writes the current value of the script local accumulator.

mvme_require_version

- **mvme_require_version** <*min version string*>

Runtime check for minimum mvme version required to run the current script.

- Example: `mvme_require_version 1.9.0`

  Will error out if the current mvme version is less than 1.9.0.

4.3.2.5 MVLC special commands

mvlc_wait

- **mvlc_wait** <*clocks*>

Delay execution of the following stack commands by the given number of clock cycles. The maximum value of the *clocks* argument is `(2**24) - 1`.

mvlc_signal_accu

- **mvlc_signal_accu**

Signal the internal accumulator of the current command stack.

mvlc_mask_shift_accu

- **mvlc_mask_shift_accu** <*mask*> <*shift*>

Set the stack accumulator *mask* and *shift* values. When the accumulator value is used for a VME block read, first the mask is applied, then the result is rotated left by the *shift* argument. The resulting value is the number of block read cycles to perform.

mvlc_set_accu

- **mvlc_set_accu** <*value*>

Set the stack accumulator to the given 32-bit value.

mvlc_read_to_accu

- **mvlc_read_to_accu** <*amode*> <*dwidth*> <*address*> [*'slow'*]

Same as *read* but reads into the stack accumulator instead of outputting to the data stream.

mvlc_compare_loop_accu

- **mvlc_compare_loop_accu** (*'eq'|'lt'|'gt'*) <*value*>

Repeatedly compares the value stored in the accumulator to the given *value* using the specified comparison operator. If the comparison fails jumps to the previous stack command, otherwise proceeds to the next command.

mvlc_writespecial

- **mvlc_writespecial** (*'timestamp'|'accu'|<numeric_value>*)

Writes the value of the specified internal MVLC variable to the current output.

mvlc_stack_begin/end

Allows to specify that a group of VME Script commands should be executed as a single MVLC command stack instead of running each command individually. This allows making use of features that require a stack context, e.g. the MVLC stack accumulator logic.

mvlc_stack_begin/end only has an effect when manually executing the VME Script, e.g. via *Run* in the VME Script Editor. During the DAQ initialization phase, if a script that is part of a MVLC readout stack (e.g. module readout) is processed, the mvlc_stack_begin/end lines are ignored and the block contents are used normally as part of the resulting readout stack.

mvlc_custom_begin/end

This is an advanced block command allowing to specify arbitrary stack data to upload and run on the MVLC. Example:

```
mvlc_custom_begin output_words=2
  0x140D0001   # read 16 bit word to accu
  0x00006030   #  from address 6030
  0xC5000000   # shift mask: no rotation
  0x0000FFFF   # mask lower 16 bits
  0xC1000001   # write_special_word = 1 (Accu)
mvlc_custom_end
```


mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

The *output_words* argument specifies the expected, fixed number of output words the execution of the custom stack will produce.

4.3.2.6 VMUSB specific

- **vmusb_write_reg** *(<register_address>|<register_name>) <value>*
- **vmusb_read_reg** *(<register_address>|<register_name>)*

These commands only work when using the WIENER VM-USB controller and allow read/write access to its internal registers. For details on the registers see the VM-USB manual section *3.4 - Internal Register File*.

Instead of using register addresses some registers are also accessible via name. The following name mappings are defined:

Table 4.7: VMUSB Register Names

Register Name	address
dev_src	0x10
dgg_a	0x14
dgg_b	0x18
dgg_ext	0x38
sclr_a	0x1c
sclr_b	0x20
daq_settings	0x08

4.3.3 Floating Point Values, Variables and Mathematical Expressions

Since mvme-0.9.7 VME scripts support evaluation of numerical expressions and can contain references to variables. Additionally floating point values can be used where previously only unsigned integers where allowed.

It is up to each specific command how floating point values are interpreted and what limits are imposed. The VME read and write commands use mathematical rounding and test that the resulting value fits in an unsigned 16 or 32 bit integer (depending on the commands data width argument). On the other hand the *write_float_word* command uses the floating point value directly without performing an integer conversion.

4.3.3.1 Variables

The variable system in VME Scripts is based on simple string replacement. Whenever a variable reference of the form `${varname}` is encountered the value stored under the name `varname` is looked up and is used to replace the variable reference. Variable expansion is currently not recursive so `${${foo}}` will try to look up the value of a variable named `${foo}`.

Variables are stored in lists of symbol tables with the variables from the first (innermost) table overriding those defined in the outer scopes.

Each object in the VME Config tree carries a symbol table: VME Events, VME Modules and VME Script objects each have a set of variables attached to them. When parsing a VME script the list symbol tables is assembled by traversing the VME Config tree upwards towards the root node. Each objects symbol table is appened to the list of tables. This way variables defined at script scope take precedence over those defined at module scope. The same is true for module and event scopes.

In addition to variables defined by VME Config objects variables can also be locally defined inside a VME Script using the `set` command. The variable will be entered into the most local symbol table and will override any other definition of a variable with the same name.

The mvme GUI currently contains a dedicated editor for variables defined at VME Event scope. Select an event in the VME Config tree and click the **Edit Variables** button above the tree. Module level variables can be accessed via **Edit Module Settings** from the context menu. A dedicated editor for Module and Script objects is going to be added in the future.

Example

```
set threshold 500
write a32 d16 0x1234 ${threshold}   # -> write a32 d16 0x1234 500

set addr 0x6789
set value 0b1010

write a32 d16 ${addr} ${value}      # -> write a32 d16 0x6789 0b1010
${addr} ${value}                    # same as above using the short form of the write␣
↪command
```

4.3.3.2 Expressions

Mathematical expressions in VME scripts are enclosed between `$(` and `)`. The enclosed string (including the outermost parentheses) is passed to the exprtk library for evaluation and the resulting value replaces the expression string before further parsing is done.

exprtk internally uses floating point arithmetic and the result of evaluating an expression is always a floating point value. It is up to the specific command of how the value is treated.

Variable references inside expressions are expanded before the expression is given to the exprtk library for evaluation.

Example

```
# From the MDPP-32-QDC init script: Window start = 16384  + delay[ns] / 1.56;
0x6050  $(16384 - 100 / 1.56)

# or using a local variable to hold the delay:
set my_delay -100
0x6050  $(16384 + ${my_delay} / 1.56)
```

4.3.4 Example Script

```
# BLT readout from VME address (module specific base address + 0x0000).
# Maximum number of transfers = 10000. Readout stops at BERR or when the
# number of transfers is reached.
bltfifo a32 0x0000 10000

# Write the value 3 to address 0x6070. If this appears in a module specific
# script (init, readout, reset) the module base address is added to the
# given address.
0x6070 3

# Same as above but explicitly using the write command.
write a32 d16 0x6070 3

# Set a different base address. This will replace the current base address
# until resetbase is used.
setbase 0xbb000000

# Results in an a32/d16 write to 0xbb006070.
0x6070 5

# Restore the original base address.
resetbase
```

(continues on next page)

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

(continued from previous page)

```
# Binary notation for the register value.
0x6070 0b0000'0101
```

4.3.5 VME block read command mappings

FIFO reads do not increment the read address, mem reads do. **VMEReadMem** and **VMEReadMemSwapped** are available since MVLC firmware `FW0036`.

Table 4.8: VME block read command mappings

VME-Script	MVLC YAML	MVLC command	Notes
blt	vme_block_read_mem	0x32 VMEReadMem	
bltfifo	vme_block_read	0x12 VMERead	
mblt	vme_block_read_mem	0x32 VMEReadMem	
mbltfifo	vme_block_read	0x12 VMERead	
mblts	vme_block_read_mem_swappe	0x33 VMEReadMem-Swapped	
mbltsfifo	vme_block_read_swapped	0x13 VMEReadSwapped	
2esst	vme_block_read	0x12 VMERead	for compatibility this is *fifo*, not *mem*
2esstfifo	vme_block_read	0x12 VMERead	same as `2esst`
2esstmem	vme_block_read_mem	0x32 VMEReadMem	explicit *mem* version
2essts	vme_block_read_swapped	0x13 VMEReadSwapped	for compatibility this is *fifo*, not *mem*
2esstsfifo	vme_block_read_swapped	0x13 VMEReadSwapped	same as `2essts`
2es-stsmem	vme_block_read_mem_swappe	0x33 VMEReadMem-Swapped	explicit *mem* version
read	vme_read	0x12 VMERead	MVLC stack accu + 'fifo' flag
read	vme_read_mem	0x32 VMEReadMem	MVLC stack accu + 'mem' flag

The mapping of the **read** command depends on whether the 'fifo' or 'mem' flags are specified. This is only used in combination with the MVLC stack accumulator.

4.4 MVLC Trigger I/O Module

4.4.1 Introduction

The MVLC VME controller contains a digitial logic module enabling flexible configuration of readout trigger logic and timings. The module includes setup of the front-panel NIM IOs and ECL outputs, internal logic functions, signal routing and access to utilities like timers, counters and the VME system clock.

The low-level setup of the Trigger I/O module is performed by writing to special registers via MVLCs internal VME interface at base address `0xffff0000`. This means standard VME write commands can be used to setup the module, create software triggers and read out counter values.

mvme contains a dedicated graphical interface representing the structure and internal connections of the logic module. This interface shields the user from the low-level details, allows to use custom signal names and simplifies creating complex setups. The low-level VME commands used to setup the logic module can still be viewed and manually edited if needed.

4.4.2 Structure and operation

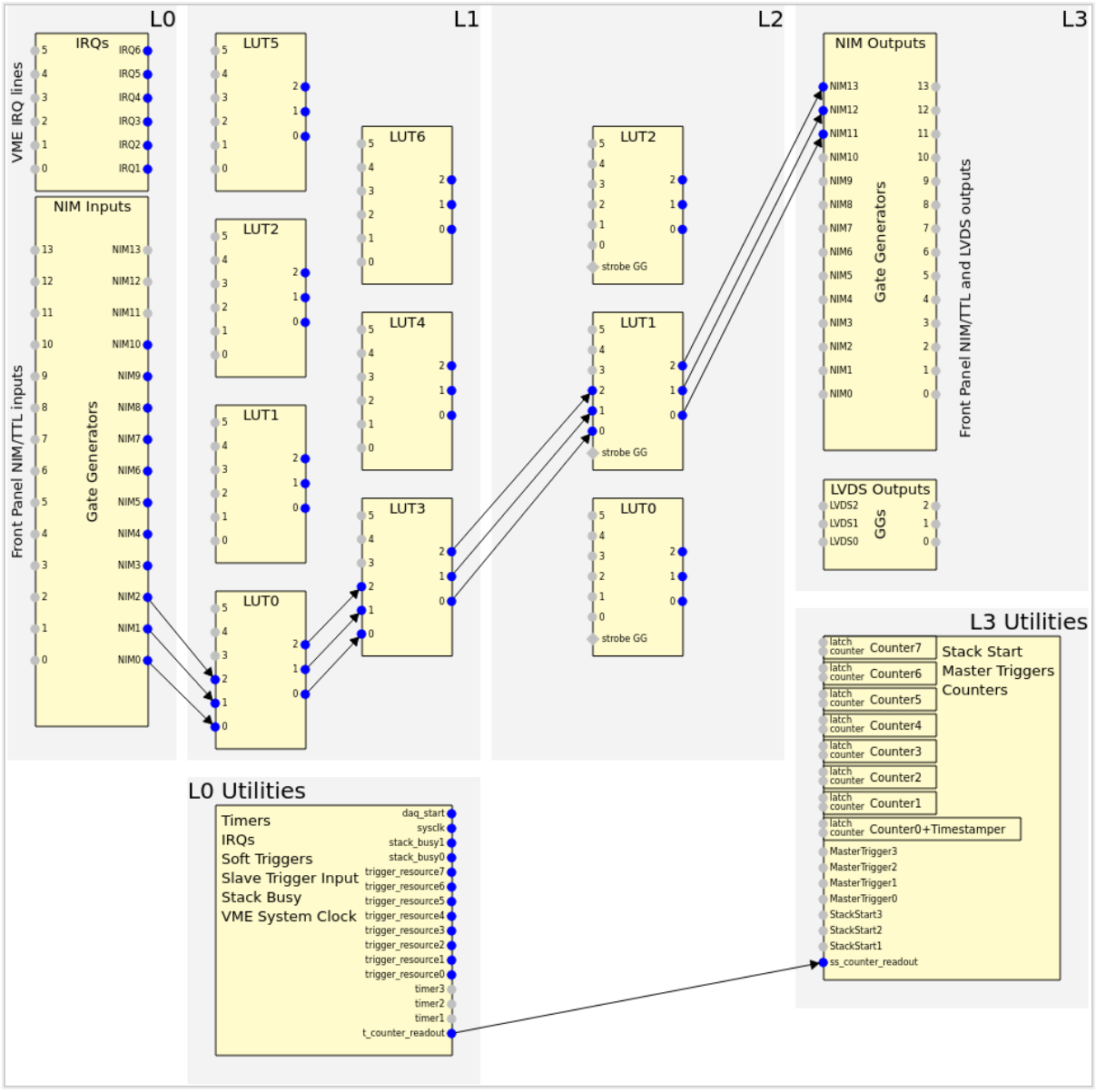


Fig. 4.4: Structure of the MVLC Trigger I/O module

Logically the MVLC Trigger I/O module is divided into 4 levels with signals originating on the lowest level and flowing through the system towards the higher levels.

Each of the levels contains a set of specific units representing functionality provided by the logic module. This includes the NIM I/Os, timers, lookup tables, counters, access to command stacks etc.

The units from higher levels connect back to units from lower levels. Some of these connections are hardwired while others can be dynamically selected from a set of options.

Level 0 contains signal-generating units like NIM inputs, the VME system clock and timers.

Levels 1 and 2 contain lookup tables which can be used to implement arbitrary boolean logic and route signals to higher levels.

Level 3 consists of signal-consuming units like NIM outputs, counters and units for executing command stacks.

The delay introduced by the logic module for a signal travelling from a NIM input to a NIM output is ~160 ns.

4.4.3 User Interface and integration into mvme

Inside mvme the MVLC Trigger I/O module will appear as the topmost item in the VME Config tree when one of the MVLC variants (USB or Ethernet) is selected as the VME Controller.

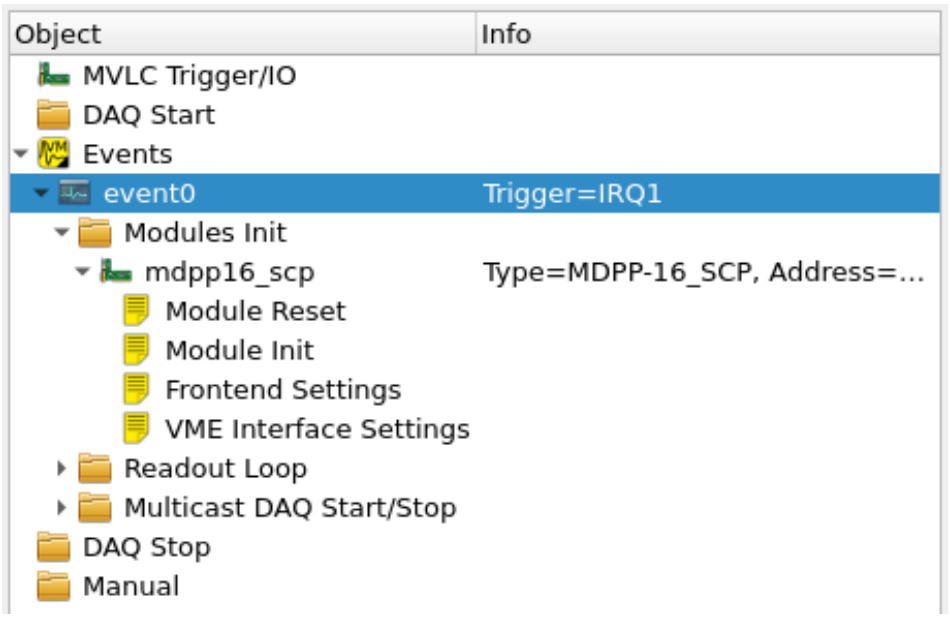


Fig. 4.5: VME Config tree with the MVLC Trigger I/O module at the top

Double-clicking this module will open a dedicated GUI representing the current configuration of the logic module. The view can be zoomed via the mouse wheel and panned by holding down the left mouse button and moving the cursor.

Each of the blocks can be double-clicked to open an editor window specific to the unit or groups of units. Input and output pins are represented by small circles near the edges of each block. Connections between pins are drawn as arrows from source to destination.

The user interface only draws arrows for connections it considers *active*. NIM and ECL units have their own activation flags present in the hardware while other units such as timers - which interally are always active - have a software-only activation flag. Lookup table inputs are only considered active if the corresponding input is actually used in the logic function implemented by the LUT.

Each of the specific editor windows allows editing the names of input/output pins which makes routing and connecting signals easier.

By default changes made in any of the editor windows are applied immediately upon closing the editor or pressing the `Apply` button as long as mvme is connected to an MVLC. This means the software representation of the Trigger I/O module is converted to a list of VME write commands targeting the MVLC and then this command list is directly executed. Use the `Autorun` button from the top toolbar to toggle this behaviour.

If you want to view or edit the VME write commands directly use the `View Script` button, make your changes inside the text editor that opens up and then press the `Reparse from script` button to update the user interface.

Descriptions of the available units and their corresponding GUI editors can be found in *I/O and logic units*.

When starting a new DAQ run the initialization procedure will apply the current logic setup to the MVLC before further initializing any modules and setting up the readout stacks.

4.4.4 Gate Generators

Some of the Trigger I/O units have builtin gate generators to influence the signals they produce. These units are the NIM I/Os, ECL outputs, VME IRQ inputs, LUT strobe inputs and the SlaveTrigger inputs. The gate generators share a common set of properties:

- Delay

  Delays generation of the output pulse by the specified time in nanoseconds.

  Minimum: 0 - no delay, maximum: 65535 ns

- Width

  The width of the generated pulse in nanoseconds. Setting the width to 0 disables the gate generator.

  Minimum: 8 ns, maximum: 65535 ns.

- Holdoff

  Holdoff sets the minimum time that must elapse before the next change to the output may occur.

  Minimum: 0 ns, maximum: 65535 ns.

- Invert

  By default output pulses are generated at the leading edge of the input signal. Setting the invert flag changes this to the trailing edge.

Each gate generator can be disabled by setting its `width` to 0. In this case the signal is passed through as is. This allows to make use of DC-level signals inside the logic (for example busy inputs from external hardware).

4.4.5 I/O and logic units

4.4.5.1 NIM I/Os

The front panel NIM connectors can be configured as either input or output. This means they are available both on the level0 input side and on the level3 output side.

Each of the NIM I/Os is driven by a *gate generator*. When a NIM is configured as an input the gate generator is used to generate the *internal* signal. If the NIM is configured as an output the gate generator affects the output signal of the NIM.

Note that it is possible to use a NIM as both input and output at the same time. In this case the NIM has to be configured as an output and the gate generator acts on the output signal only.

4.4.5.2 LVDS outputs

These are similar to the NIM output units. Each of the 3 outputs needs to be activated separately.

4.4.5.3 IRQ inputs

In addition to the front panel NIM inputs the Trigger I/O module also provides access to VME IRQs 1-6. Each IRQ unit is connected to a *gate generator* just like the NIM inputs. The output signal of the gate generator is available for further processing in a LUT on level 1.

4.4.5.4 Timers

Fixed frequency logic pulse generation. The generated output pulses have a fixed width of 8 ns. The minimum period between rising edges is 24 ns.

Settings

- Range

  The time unit the timer period refers to. One of *ns*, μs, *ms* or *s*.

- Period

  The period in units specified by Range.

  Minimum: 24 ns, maximum: 65535 s.

- Delay

  Delays generation of the output pulse by the specified time in nanoseconds.

  Minimum: 0 - no delay, maximum: 65535 ns

4.4.5.5 Trigger Resource Units

Each TriggerResource can be configured as one of IRQ, SoftTrigger or SlaveTrigger unit types.

IRQ Utility Units

These units generate a signal when one of the 7 available VME IRQs triggers. The only setting is the IRQ number (1-7) each unit should react to.

Soft Triggers

Software triggers which can either be permanently activated via the GUI editor or pulsed by executing one of the folowing VME Scripts:

Software triggers which can be pulsed or permanently activated via a VME write instruction.

Example script:

```
setbase 0xffff0000      # use the mvlc vme interface as the base address for the␣
↪following writes

set trigger_index 0                         # valid values: 0-7 (trigger_resource␣
↪units)

set trigger_unit $(4 + ${trigger_index})    # Level0 Units 4-11
0x0200 ${trigger_unit}                      # select the unit
0x0302 0                                    # write to the soft_trigger output␣
↪activation register
                                            # 0: generate a 8ns pulse, 1: set the␣
↪output to permanently high
```

The above script is available in mvme by right-clicking on the `Manual` directory in the VME Config Tree and choosing `Add Script from Library -> MVLC SoftTrigger`.

Slave Triggers

Activates when one of the slave triggers fires. This feature will be available in the future with a special multi-crate firmware and supporting software.

4.4.5.6 Stack Busy

The stack busy units are active while their corresponding VME command stack is being executed.

In the mvme user interface the command stack numbers are augmented with the event names defined in the VME config.

4.4.5.7 Sysclk

This unit provides access to the 16 MHz VMEbus system clock.

4.4.5.8 Lookup Tables (Levels 1 and 2)

The MVLC contains a set of lookup tables used to create logic functions and for signal routing. Each lookup table (LUT) maps 6 input bits to 3 output bits. This allows to e.g. implement 3 functions each mapping 6 input bits to one output bit or a single 6 to 3 bit function.

Most of the LUT inputs are hardwired while some LUTs have up to three variable inputs. Specifically level1.LUT2 can be connected to either NIM8-10 or IRQ1-3.

The LUTs on level2 connect back to the level1 LUTs and each has three variable inputs which can be connected to the level1 utility units or certain level1 LUT outputs. Additionally the level2 LUTs each have a strobe input which is used to synchronize the switching of the LUT outputs.

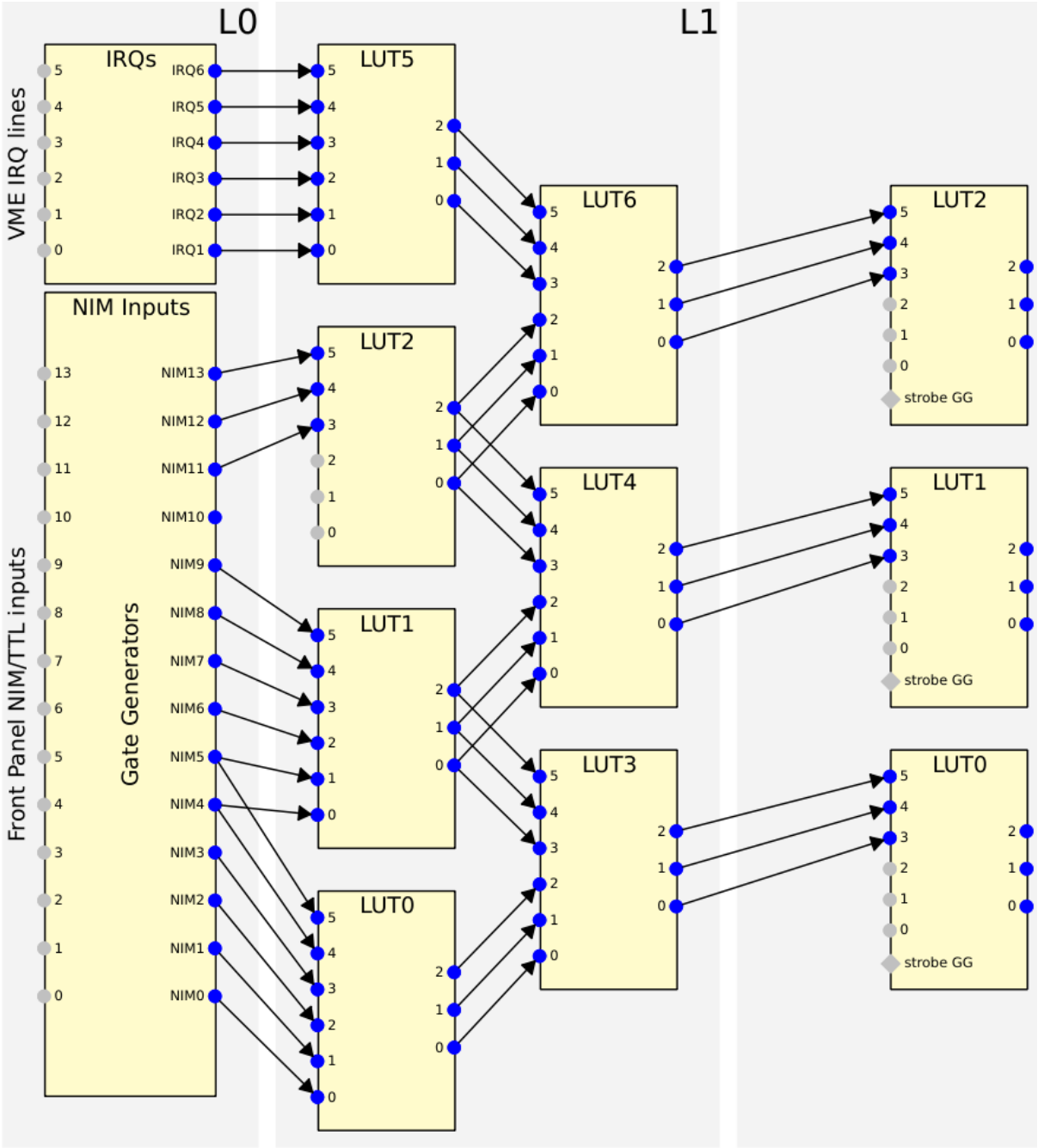


Fig. 4.6: The lookup tables with all hardwired connections active.

LUT Editor GUI

Editing of the LUT function is done via its own GUI:

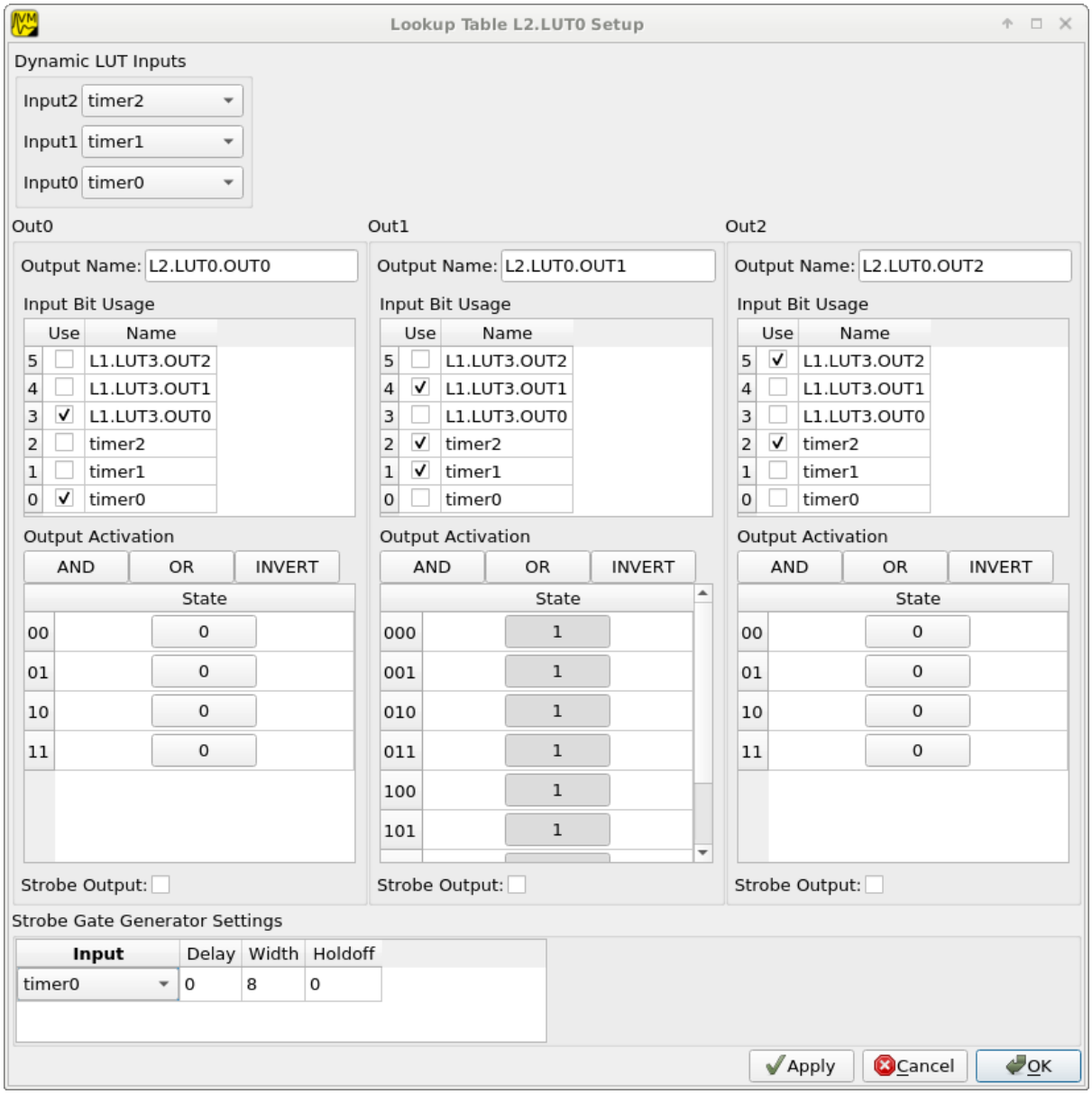


Fig. 4.7: Editor window for a LUT on Level2.

Elements from top to bottom:

- Dynamic input selection for the first three inputs.

  This only appears for LUTs on Level2. The drop down boxes are populated with the possible choices for each of the dynamic inputs.

- Three columns of LUT functions, one for each of the three LUT output signals.

  Each LUT maps 6 input bits to 3 output bits. This means a total of $2^6 = 64$ input combinations per LUT. To make editing easier only the combinations for selected input bits are shown.

  Select the inputs you want to use via the checkboxes under `Input Bit Usage`. This will populate the `Output Activation` table with the correct number of rows to represent each possible input combination.

  Each row of the `Output Activation` table represents the state of the output for the corresponding input

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

combination. The input bit combination is shown on each row header with the lowest bit taking the rightmost place. Click the button to toggle the output state for the corresponding input combination.

Using the `AND`, `OR` and `INVERT` buttons allows to quickly populate the table with the corresponding function or invert the current assignment.

- For Level2 LUTs only: strobe input selection and parameters.

  For Level2 LUTs an additional `Strobe Output` checkbox is visible below the function table. If set the corresponding output bit will be affected by the LUTs strobe input.

  The strobe input signal can be selected from a predefined list and its *gate generator* parameters can be set using the controls in `Strobe Gate Generator Settings`.

Note: mvme will attempt to minimize the boolean functions defined by each of the LUTs. This means that not all selected input bits will necessarily be selected again when next opening the editor window but the resulting function should be identical.

Example

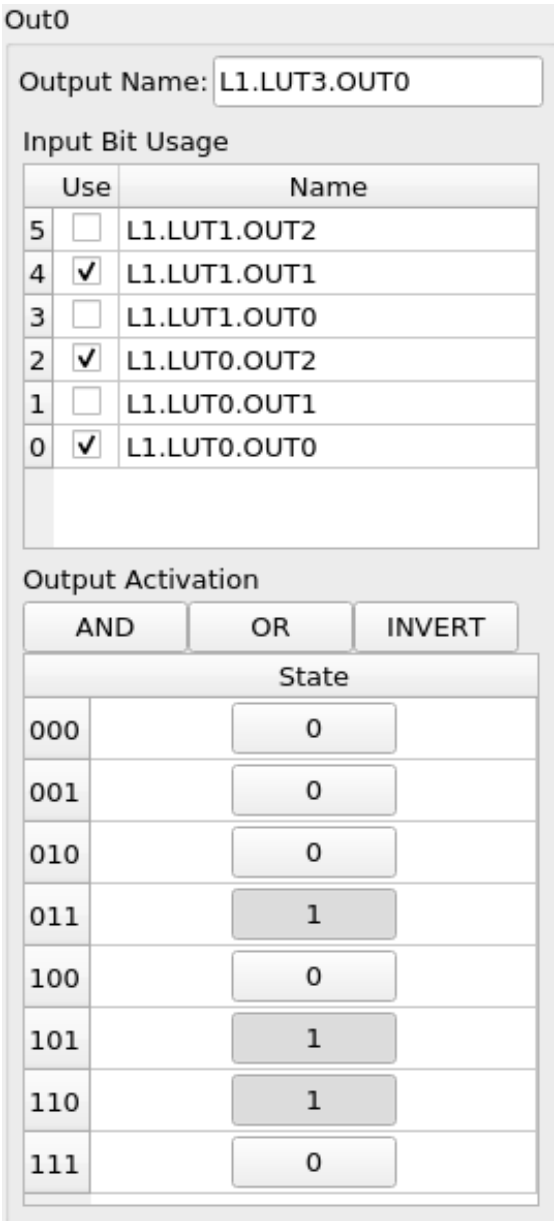


Fig. 4.8: Example LUT using input bits 0, 2 and 4. The output is activated if exactly two of the inputs are set.

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

4.4.5.9 StackStart

These units start the execution of one of the 7 MVLC command stacks.

Settings

- Index of the command stack to execute
- Delay: the delay in ns until the stack execution is started
- Activation flag

In the mvme user interface the command stack numbers are augmented with the event names defined in the VME config.

4.4.5.10 MasterTrigger

Generates a master trigger in multi-crate setups. This feature will be available in the future with a special multi-crate firmware and supporting software.

4.4.5.11 Counters

8 64-bit counter units incrementing by one each time the input rises. Each counter has an optional latch input which atomically transfers the current counter values to the counter registers. The latch can either be activated by the Trigger/IO module or by writing to a special latch register.

The counter units can be read out via MVCLs internal VME interface at base address `0xffff0000` using the following VME script:

```
setbase 0xffff0000

# counter0
0x0200 0x0308           # counter select
0x030a 1                # latch the counter (only needed if not done in the trigger_
↪io module)
read a32 d16 0x0300     # counter readout
read a32 d16 0x0302
read a32 d16 0x0304
read a32 d16 0x0306

# counter1
/*
0x0200 0x0309           # counter select
0x030a 1                # latch the counter (only needed if not done in the trigger_
↪io module)
read a32 d16 0x0300     # counter readout
read a32 d16 0x0302
read a32 d16 0x0304
read a32 d16 0x0306
*/
```

A dedicated VME module called `MVLC Timestamp/Counter` is provided by mvme to ease setting up a counter readout. Add an instance of this module to the VME Event where you want to read out the counter, edit the readout script (under `Readout Loop` in the user interface) and comment out all the counter blocks except for the one that should be read out.

4.4.6 Digital Storage Oscilloscope

Since Firmware FW0018 the MVLC contains a digital storage oscilloscope (DSO) allowing to acquire traces of the signals on the NIM, IRQ and Level1 utility units. Additionally parts of the internal logic of the Trigger IO module are simulated based on the sampled data.

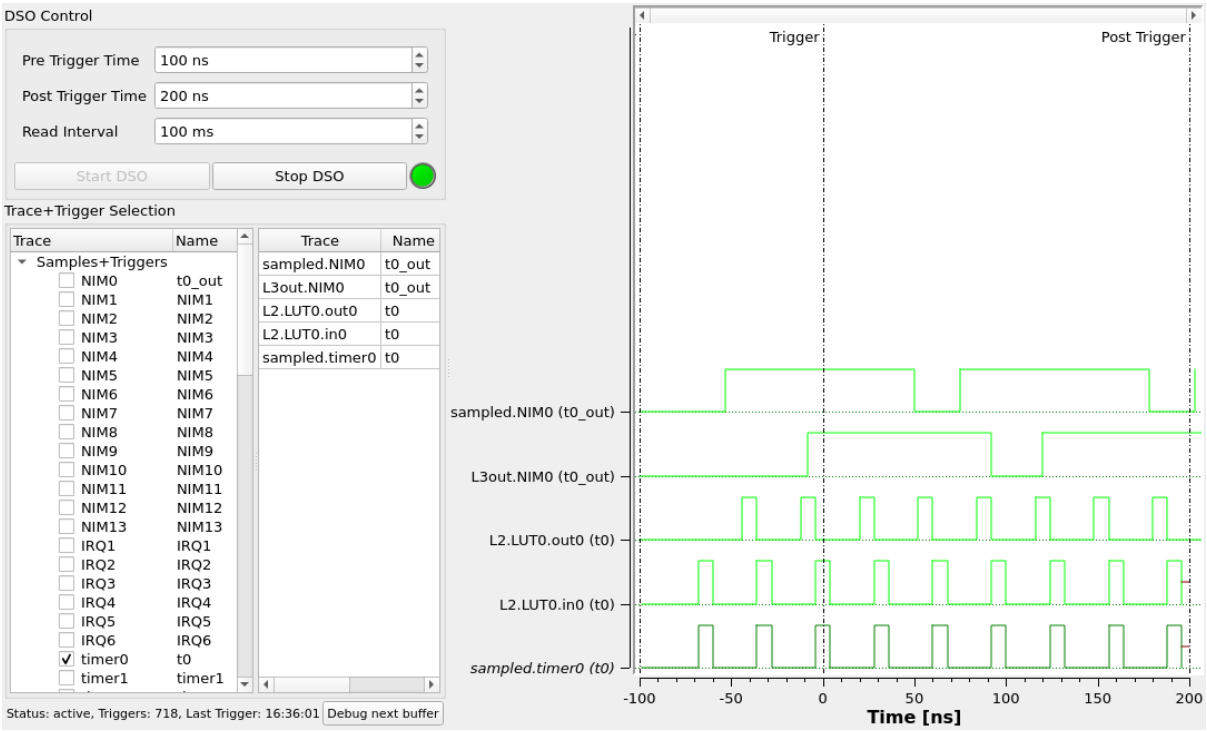


Fig. 4.9: DSO user interface

The user interface for the oscilloscope can be accessed using the `DSO` button in the Trigger IO editor. The left side of the window is used for controlling the DSO while the right side plots selected traces.

4.4.6.1 Steps needed to acquire traces

1. Press the start DSO button to activate the DSO. No data will arrive yet as no triggers are active.
2. In the trace selection tree open the `Samples+Triggers` node and use the checkboxes to select at least one of the signals as triggers for the DSO.
3. Drag traces from the tree to the right side list to make them visible in the plot view.
4. Optionally adjust the pre- and post trigger times and the read interval using the inputs in the top left. These parameters can be adjusted while the DSO is active.

4.4.6.2 Notes

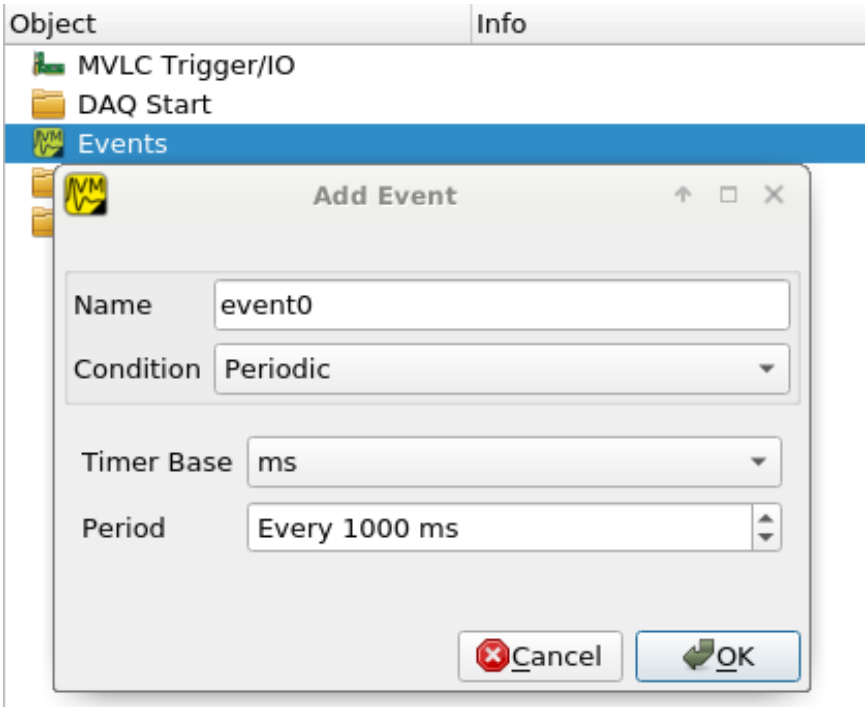
- The read interval is the software side polling interval. It implicitly affects the update frequency of the plot view. Setting the spinbox to the lowest value allows taking a single snapshot from the DSO by pressing the `Start DSO` button.
- The PreTrigger, PostTrigger and Trigger times are highlighted in the plot using vertical lines.
- Traces that are in the set of triggers are shown in *italics* in the plot view legend.
- Traces can be reordered via drag & drop in the trace list.
- Except for the traces under `Samples+Triggers` all other traces in the system are simulated based off the sampled data.
- The maximum number of samples per channel the MVLC can provide is limited. This means high frequency signals may be cut off and start after the PreTrigger time. The missing part of these traces is drawn using a red line signifying an *unknown* state. Simulation code also produces an *unknown* state if one of the inputs is *unknown*.

4.4.7 Trigger IO Usage Example: Sysclk timestamp readout

This example shows how to create a counter that increments with the VME system clock frequency and to read out the counter values by creating a periodically triggered readout event.

Only an MVLC is required for this setup to work.

- Start by creating new vme and analysis configs in mvme. Make sure the VME controller type is set to one of the MVLC variants and that mvme can succesfully establish the connection.
- In the VME Config tree right click the `Events` node and choose `Add Event`. Select `Periodic` as the condition and accept the dialog.



Creating the VME readout event

- Right-click the newly created event and select `Add Module`. Use the type drop-down and select `MVLC Timestamp/Counter`. Accept the dialog to create a module which will read out Counter0 of the Trigger I/O module.

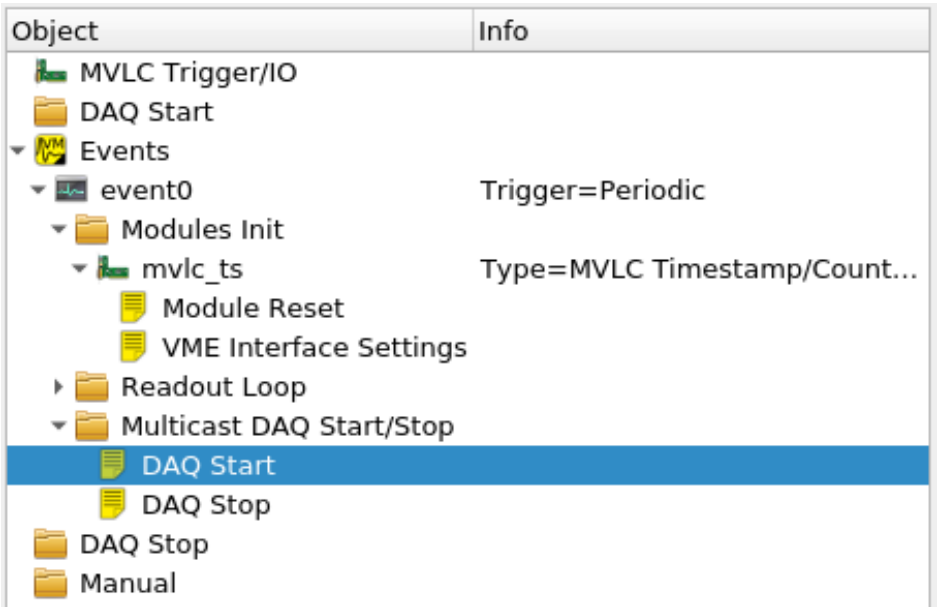


Fig. 4.10: VME Config Tree after creating the event and adding the mvlc timestamp module

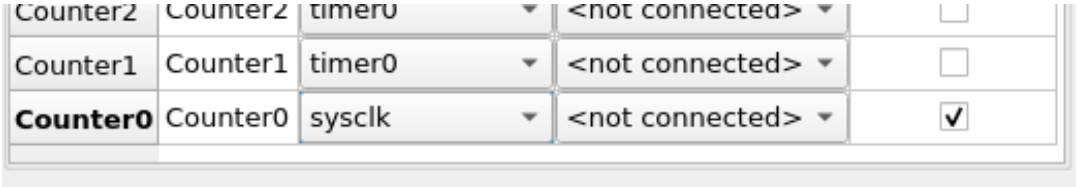
- Locate the `Multicast DAQ Start/Stop` node under the newly created event and double-click the `DAQ Start` script it to open an editor window.

  Add the following line to the script.

```
writeabs a32 d16 0xffff6090 1 # reset counters
```

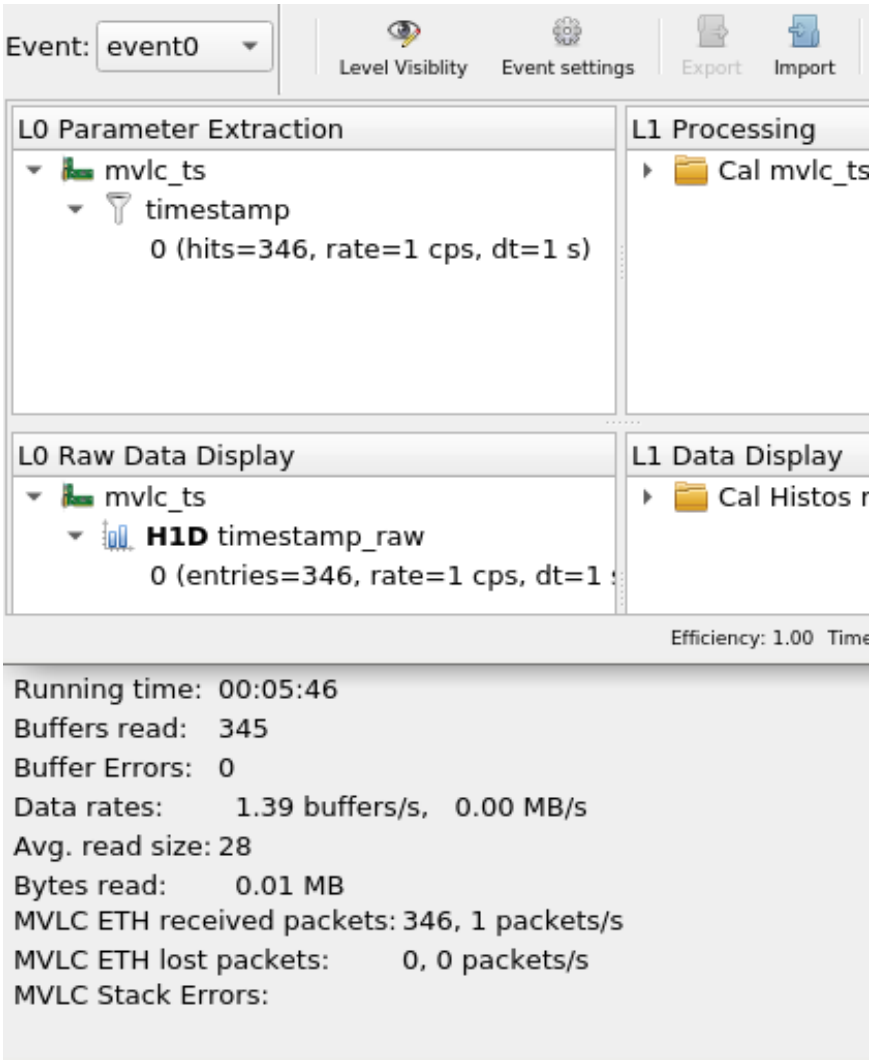
  This will make sure the counters are reset when starting a DAQ run.

- Double-click the `MVLC Trigger/IO` object in the VME Config tree to open the graphical editor.
- Double-click the `L3 Utilities` block. In the bottom-left select the input for Counter0 and set it to `sysclk`. Also check the `Soft Activate` checkbox.



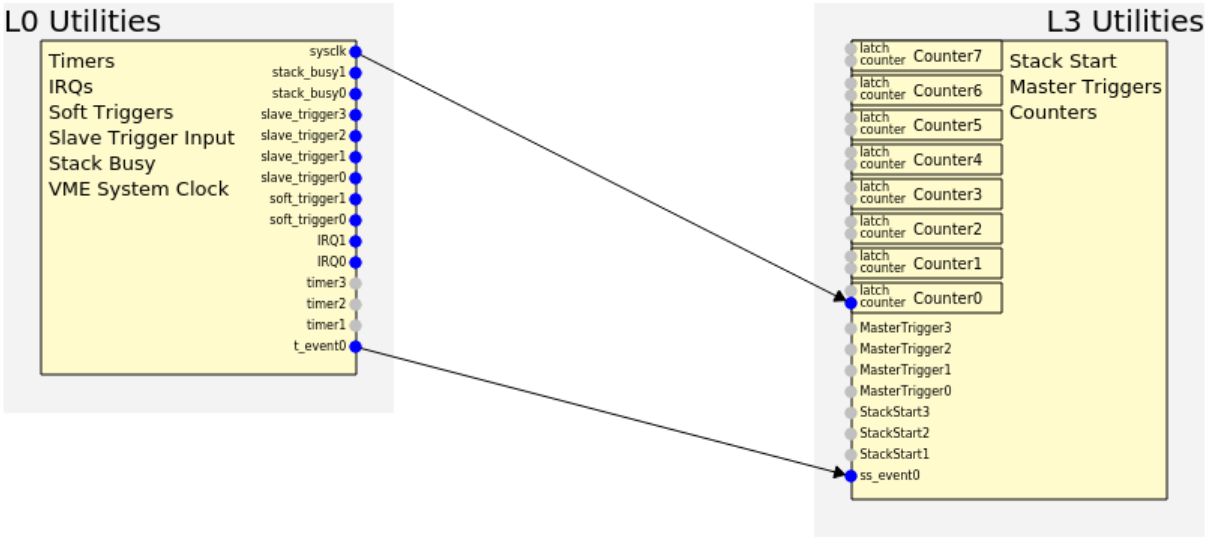
  Counter connected to sysclk and activated

- Now locate the `Analysis UI` window in mvme (Shortcut is *Ctrl-2*). event0 should show up in the Event drop-down and the mvlc_ts module should be visible. In the top area right-click the mvlc_ts module and select `Generate default filters`. Press ok to generate data extraction filters and histograms for the counter readout data.
- Use the Start button in the top-left area of the main window to start a DAQ run. If everything is setup correctly the DAQ should start successfully (*DAQ State: Running*) and an event rate of 1 count/s should be displayed in the Analysis window for the `mvlc_ts.timestamp` data source.



DAQ and analysis stats during a run

- You can reopen the `MVLC Trigger/IO` object again and verify that mvme used the first timer together with the first StackStart unit to implement the periodic readout for the event.



Internal Timer and StackStart usage by mvme

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

4.5 Analysis

4.5.1 User Guide

4.5.1.1 UI Overview

The Analysis system in mvme is designed to allow

- flexible parameter extraction from raw readout data.
- calibration and additional processing of extracted parameters.
- accumulation and visualization of processed data.

The user interface follows the structure of data flow in the system: the *Events* and *Modules* defined in the VME configuration are shown in the top-left tree.

The bottom-left tree contains the *raw histograms* used to accumulate the unmodified data extracted from the modules.

The next column (*Level 1*) contains calibration operators in the top view and calibrated histograms in the bottom view.

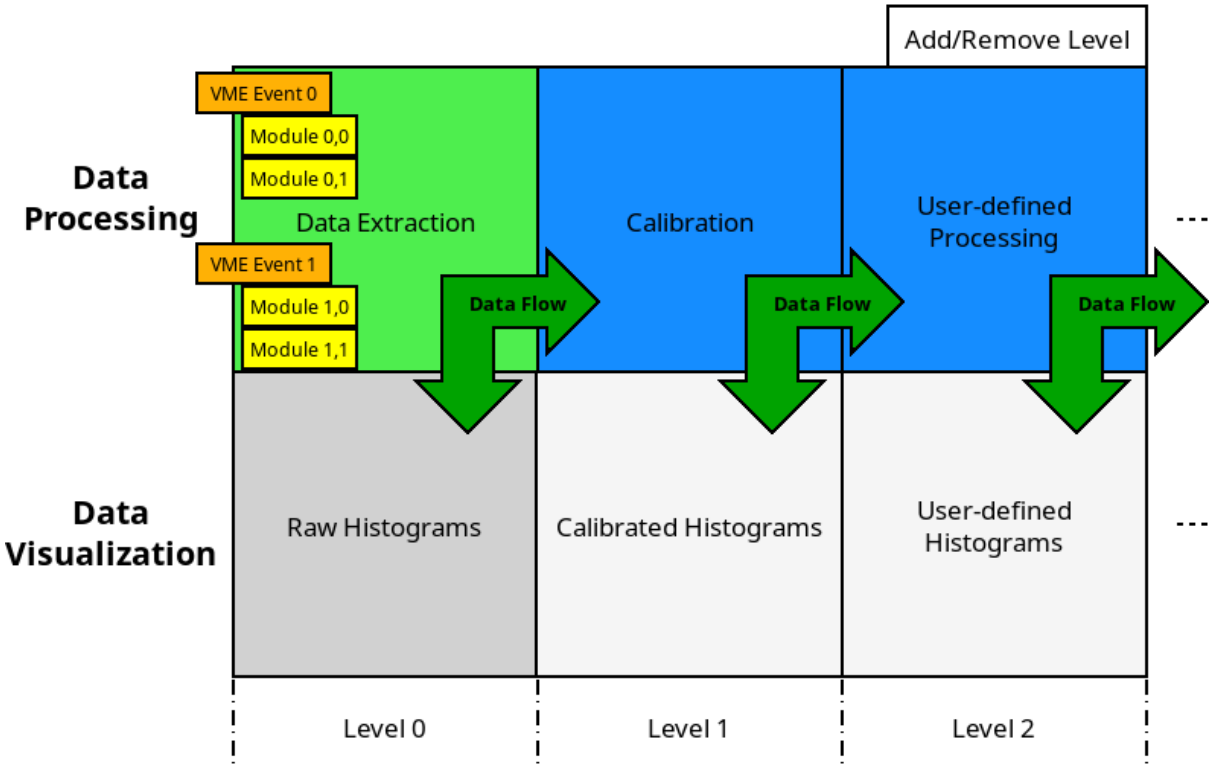


Fig. 4.11: Analysis UI Block Diagram

Note

To get a set of basic data extraction filters, calibration operators and histograms right-click on a module an select *Generate default filters*.

User Levels are used to structure the analysis and it's completely optional to have more than two of them. Additional levels can be added/removed using the + and - buttons at the top-right. Use the *Level Visibility* button to select which levels are shown/hidden.

The UI enforces the rule that operators can use inputs from levels less-than or equal to their own level. This means data should always flow from left to right. This restriction does not apply to data sinks in the bottom tree. These can be freely placed on any userlevel in the bottom tree.

Operators can be moved between levels by dragging and dropping them.

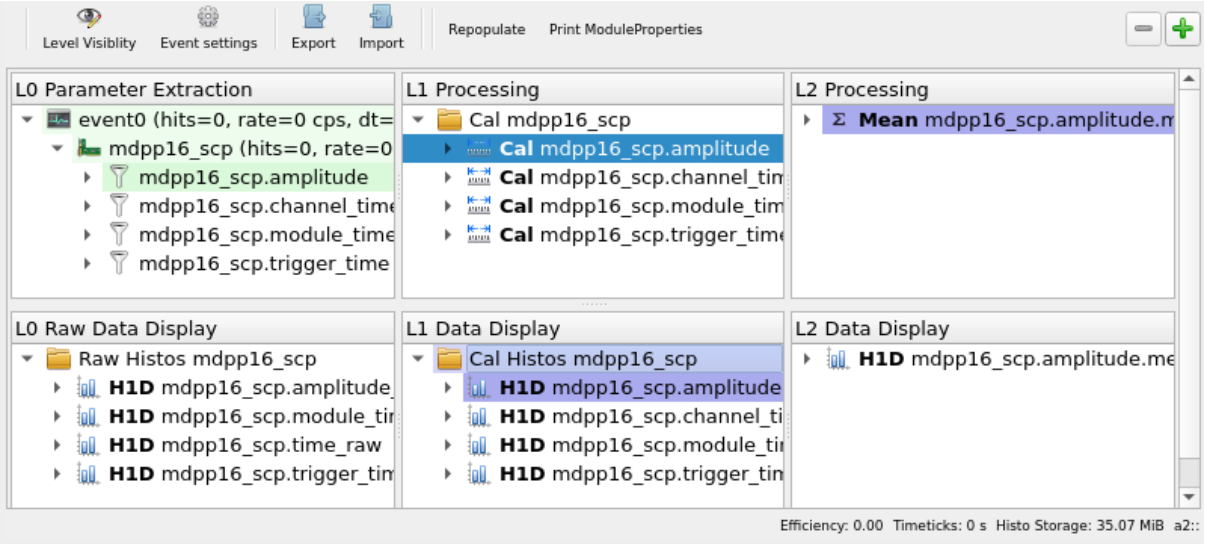


Fig. 4.12: Analysis UI Screenshot
The Calibration for *mdpp16.amplitude* is selected. Its input is shown in green. Operators using the calibrated data are shown in a light blueish color.

Selecting an object will highlight its input data sources in green and any operators using its output in blue.

4.5.1.2 Adding new objects

Right-click in any of the views and select *New* to add new operators and histograms. A dialog will pop up with input fields for operator specific settings and buttons to select the operators inputs.

Clicking any of the input buttons will make the user interface enter “input select mode”. In this mode valid outputs for the selected input are highlighted.

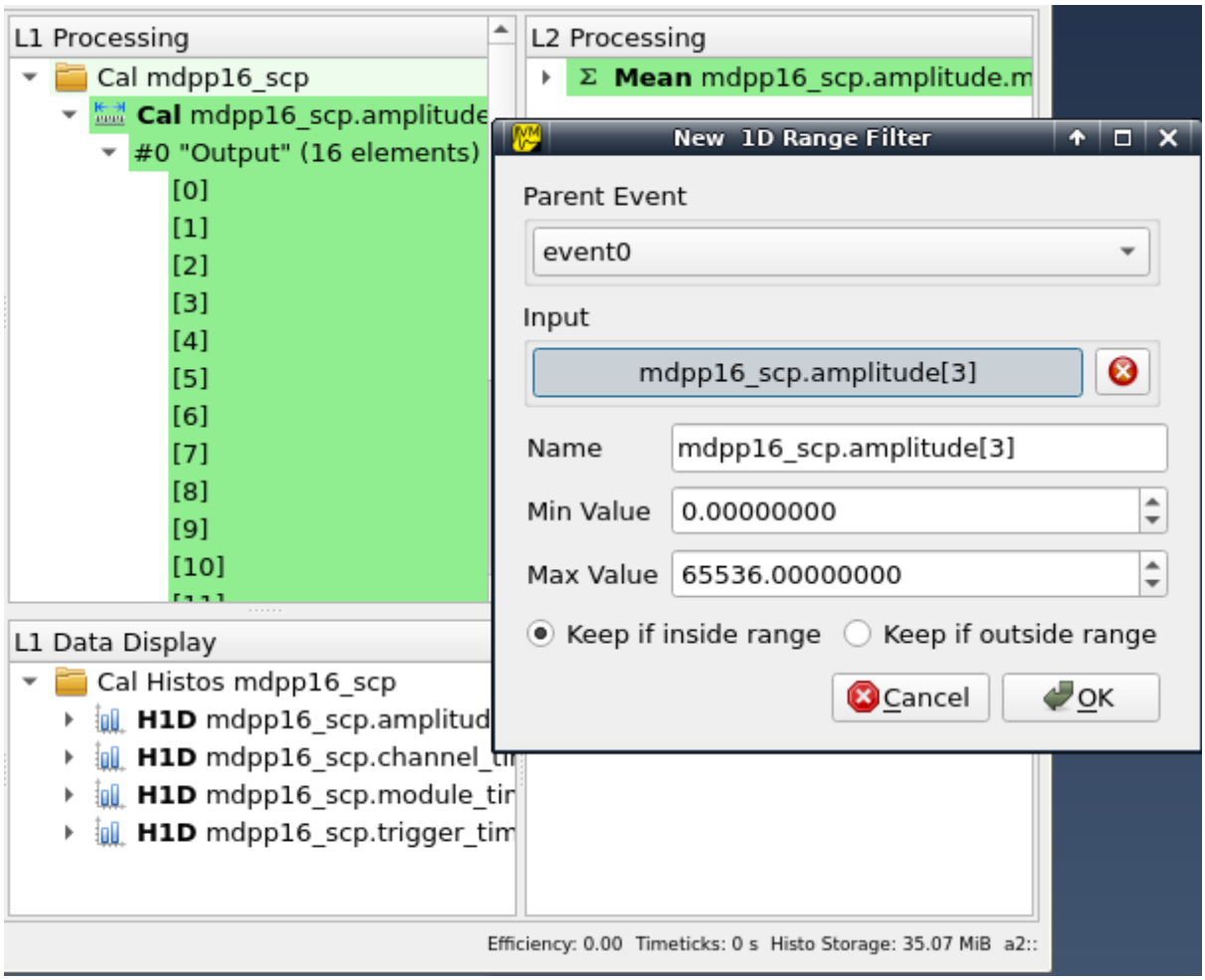


Fig. 4.13: Input select mode
Adding a *1D Range Filter* to Level 2. Valid inputs are highlighted in green.

Click an input node to use it for the new operator. The *Parent Event* specifies the event context in which the operator will be executed. The system tries to pick the correct event based on the input selected. Manually setting the parent event might be needed when doing cross-event analysis, e.g. creating an operator which takes inputs from two unrelated events.

If required fill in any additional operator specific data and accept the dialog. The operator will be added to the system and will immediately start processing data if a DAQ run or replay is active.

For details about data extraction refer to *Data Sources*. Descriptions of available operators can be found in *Operators*. For details about 1D and 2D histograms check the *Data Sinks* section.

4.5.1.3 Working with histograms

1D and 2D histograms are shown in the bottom row of the user interface. Raw 1D histograms are grouped by module in the bottom-left *L0 Raw Data Display* area. Higher level data displays are grouped by histogram type.

New histograms can be added by right-clicking in one of the data display areas, selecting *New* and choosing the histogram type.

1D

1D histograms can take full arrays as input parameters. Internally an array of histograms of the same size as the input array will be created.

Double-click on the *H1D* node to open the histogram array widget:

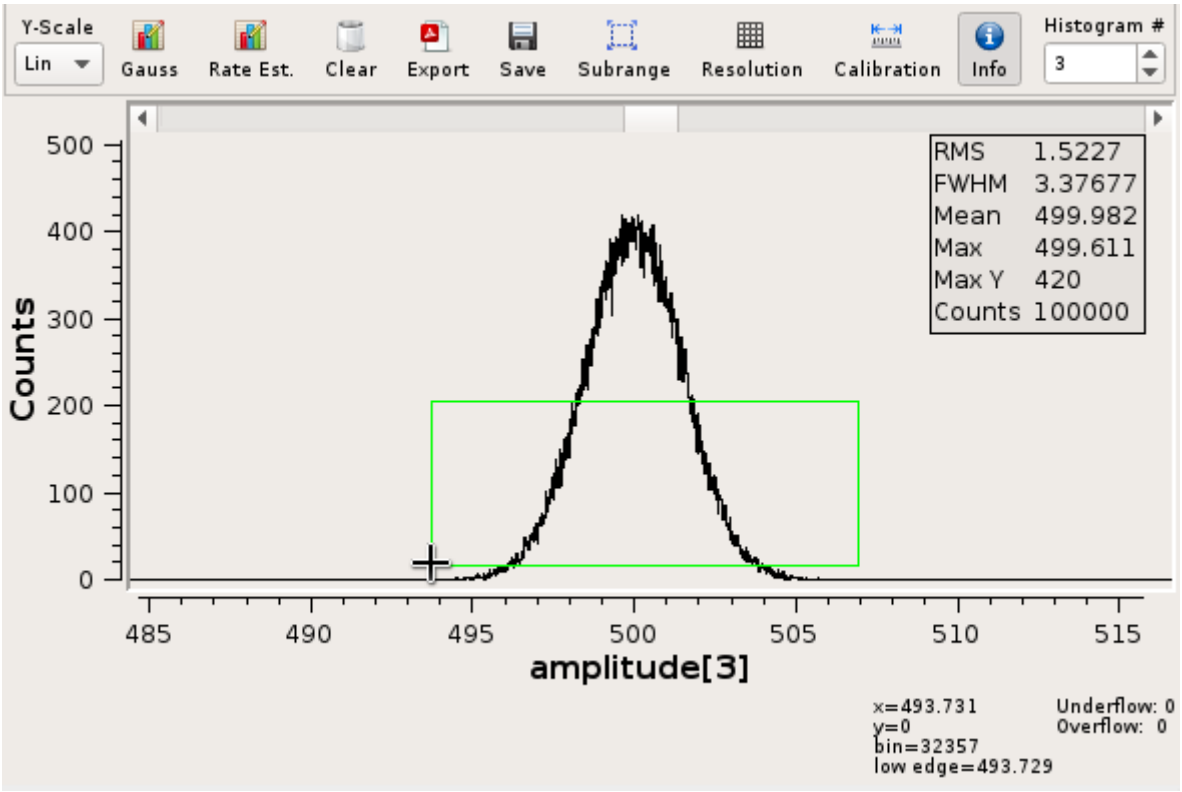


Fig. 4.14: 1D Histogram Array Widget

- The histogram index can be changed using the spinbox in the top-right corner.
- Zooming is achieved by dragging a rectangle using the left mouse button. Zoom levels are stacked. Click the right mouse button to zoom out one level.
- Press the *Info* button to enable an info display at the bottom-right of the window. This will show the current cursor coordinates and the corresponding bin number.
- Y-Scale

  Toggle between linear and logarithmic scales for the Y-Axis.
- Gauss

  Fit a gauss curve through the currently visible maximum value.
- Rate Est.

  Rate Estimation feature.

  Refer to *Rate Estimation Setup* for a how-to guide.
- Clear

  Clears the current histogram.
- Export

  Allows exporting to PDF and various image formats. Use the file type selection in the file dialog to choose the export format.
- Save

  Saves the histogram data to a flat text file.
- Subrange

  Allows limiting the range of data that's accumulated. Only input values falling within the specified interval will be accumulated.

This does not affect the histogram resolution: the full range of bins is still used with the limits given by the subrange.

- Resolution

  Change the resolution of the histogram in powers of two from 1 bit to 20 bits.

  This will not rebin existing data. Instead the histogram is cleared and new data is accumulated using the newly set resolution.

- Calibration

  This button is enabled if the histograms input is a *Calibration Operator* and allows to directly modify the calibration information from within the histogram:

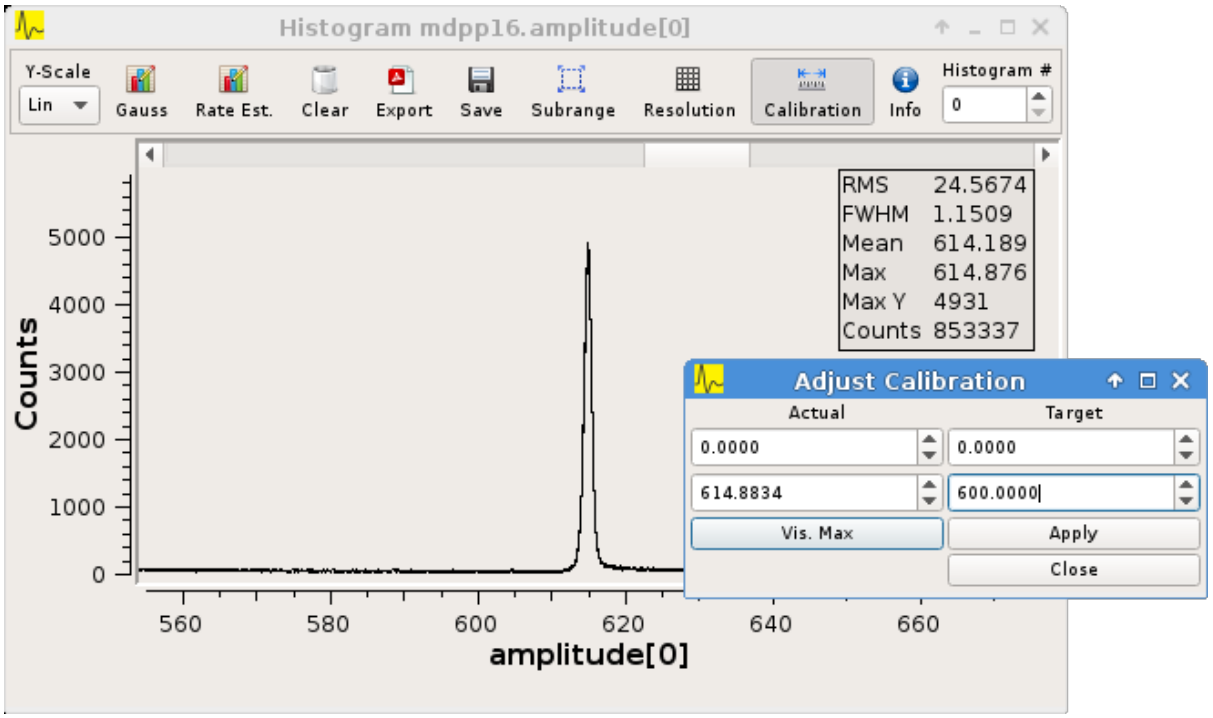


Fig. 4.15: Calibration adjustment from within the histogram display

The two inputs in the *Actual* column refer to the current x-axis scale. The inputs in the *Target* column are used to specify the desired x-axis values.

Click on one of the *Actual* inputs and then press the *Vis. Max* button to fill in the x-coordinate of the currently visible maximum value. Then enter the new x-coordinate value in the *Target* box and press *Apply*.

In the example above it is known that the peak should be at `x = 600.0`. The current x-coordinate of the peak was found using the *Vis. Max* button. Pressing *Apply* will modify the calibration for that particular histogram.

To see a list of calibration values for each channel open the Analysis UI (`Ctrl+2`), right-click the *Calibration Operator* and select *Edit*.

- 2D combined view

  A combined view of the histograms of an array of parameters can be opened by right-clicking a **H1D** node and selecting *Open 2D Combined View*. This option will open a 2D histogram with one column per 1D histogram in the array.

  The X-axes of the 1D histograms are plotted on the combined views Y-axis, the values of the histograms are plotted in Z.

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

This view allows to quickly see if any or all channels of a module are responding.

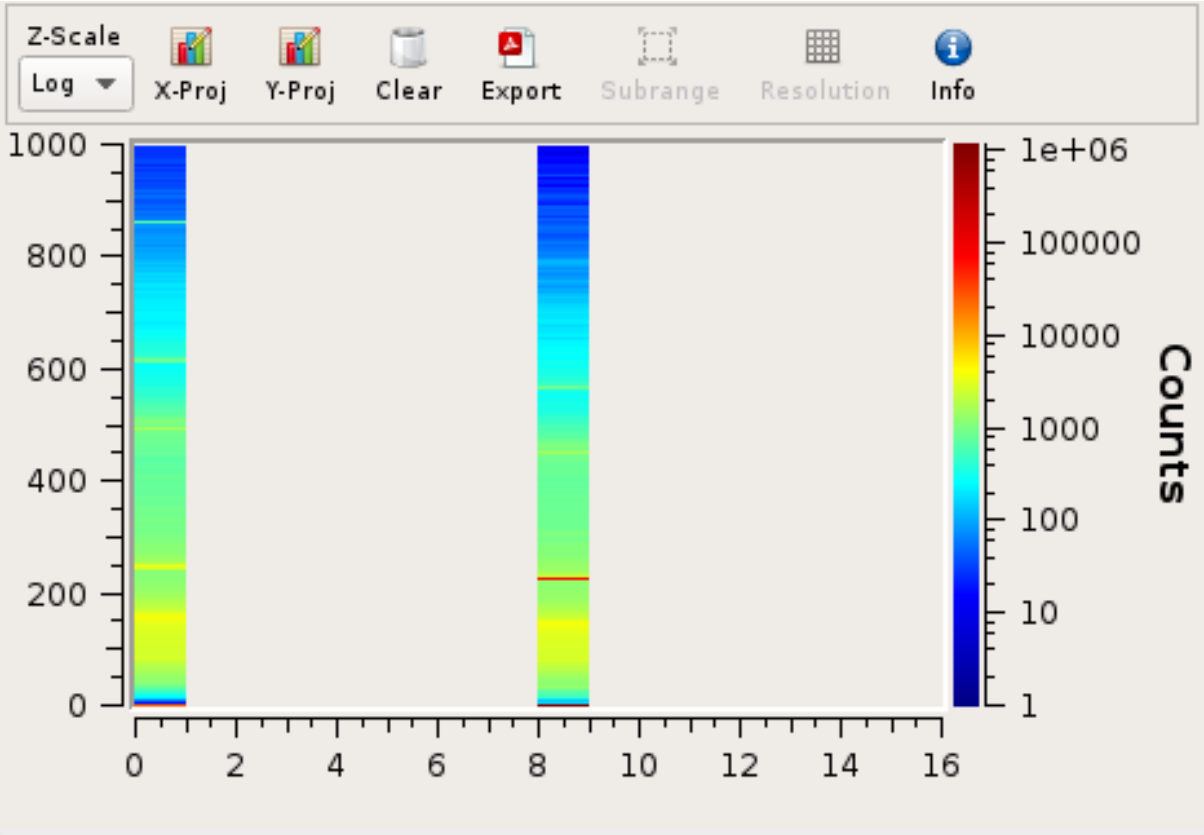


Fig. 4.16: 2D Combined View of MDPP-16_SCP amplitude values
Channels 0 and 8 are producing data with visible peaks at around 0 and 230.

2D

2D histograms take two single values as their inputs: the X and Y parameters to accumulate. When selecting the inputs you will need to expand other operators and select the desired index directly.

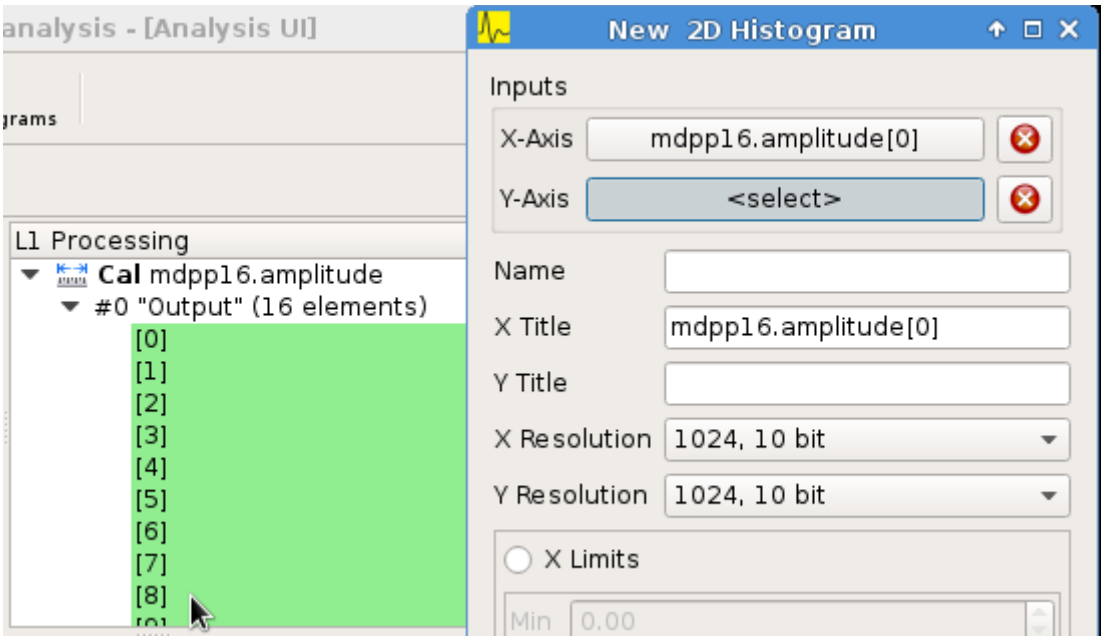


Fig. 4.17: Adding a 2D Histogram
Expand operator outputs and select individual indices for both axes.

Optional range limits can be specified for the axes. If enabled only values falling within the given interval will be accumulated.

Double-click on a *H2D* node to open the histogram widget:

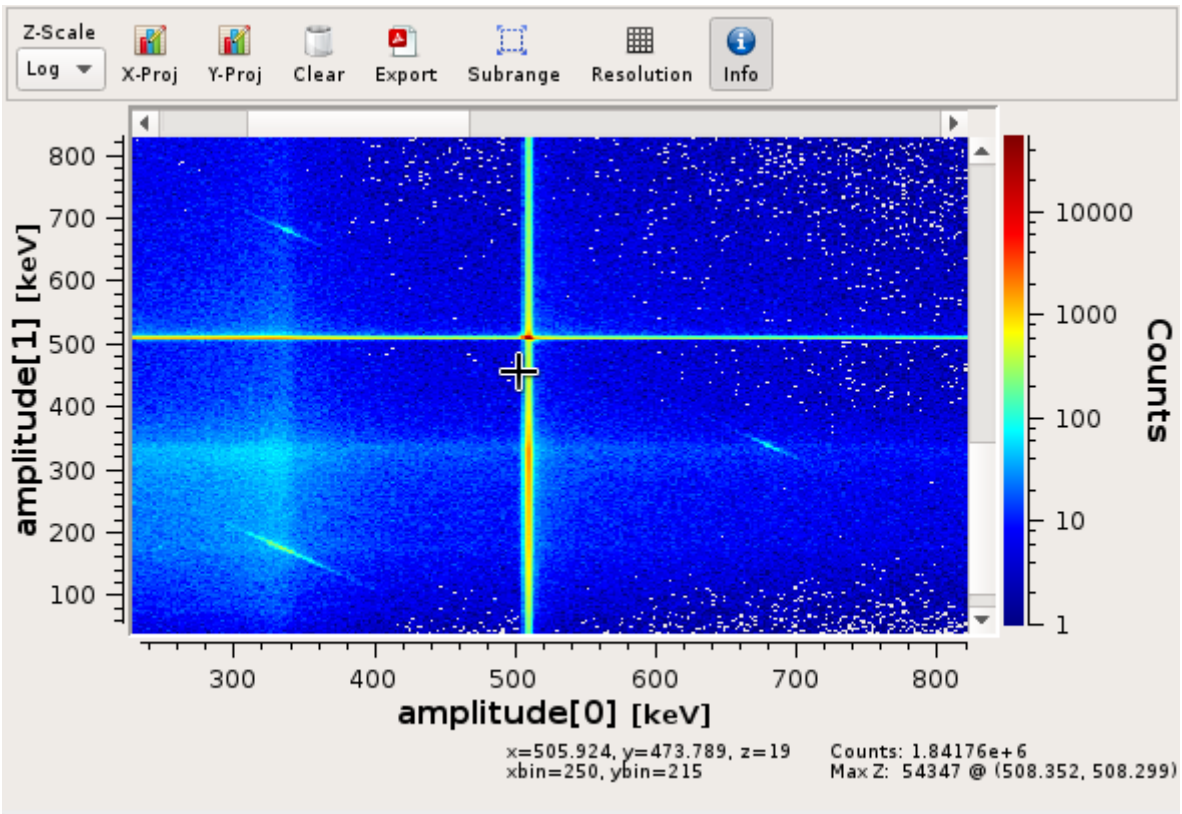


Fig. 4.18: 2D Histogram Widget

- Zooming is achieved by dragging a rectangle using the left mouse button. Zoom levels are stacked. Click

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

the right mouse button to zoom out one level.

- Press the *Info* button to show histo and cursor coordinate information at the bottom of the window.
- Z-Scale

  Toggle between linear and logarithmic scales for the Z-Axis.

- X- and Y-Proj

  Create the X/Y-Projection and open it in a new 1D histogram window. The projection will follow any zooming/scrolling done in the 2D histogram.

- Slice X / Slice Y

  Slices the histogram along the respective axis. The visible resolution of the chosen axis determines the number of slices. Slices are displayed in a 1D histogram window.

- Clear

  Clears the histogram.

- Export

  Allows exporting to PDF and various image formats. Use the file type selection in the file dialog to choose the export format.

- Subrange

  Allows limiting the range of data that's accumulated. Only input values falling within the specified interval will be accumulated.

  This does not affect the histogram resolution: the full range of bins is still used with the limits given by the subrange.

  Can optionally create a new histogram with the specified limits instead of modifying the current one. The newly created histogram will be added to the analysis.

- Resolution

  Change the resolution of the histograms axes in powers of two from 1 bit to 13 bits.

  This will not rebin existing data. Instead the histogram is cleared and new data is accumulated using the newly set resolution.

4.5.1.4 Condition System

Since mvme-1.5 the analysis system supports conditional evaluation/skipping of operators and data sinks.

Conditions are a special kind of operator producing only a single output value: the boolean result of evaluating the condition. Each condition can be applied to multiple operators and/or data sinks. Multiple conditions can be applied to the same object in which case the **logical AND** of all condition outputs is used to decide whether the operator should be run.

Currently *1d-interval*, *2d-polygon* and boolean *expression conditions* are implemented.

Creating and editing conditions

1d interval and *2d polygon* conditions can be created and edited from within histogram windows. Use the "Interval Conditions" or "Polygon Conditions" buttons to start the respective editor.

A new condition created this way will use the same input arrays as the histogram where the editor was started. The new condition will be placed in a common directory on the same userlevel as the histogram used to create the condition.

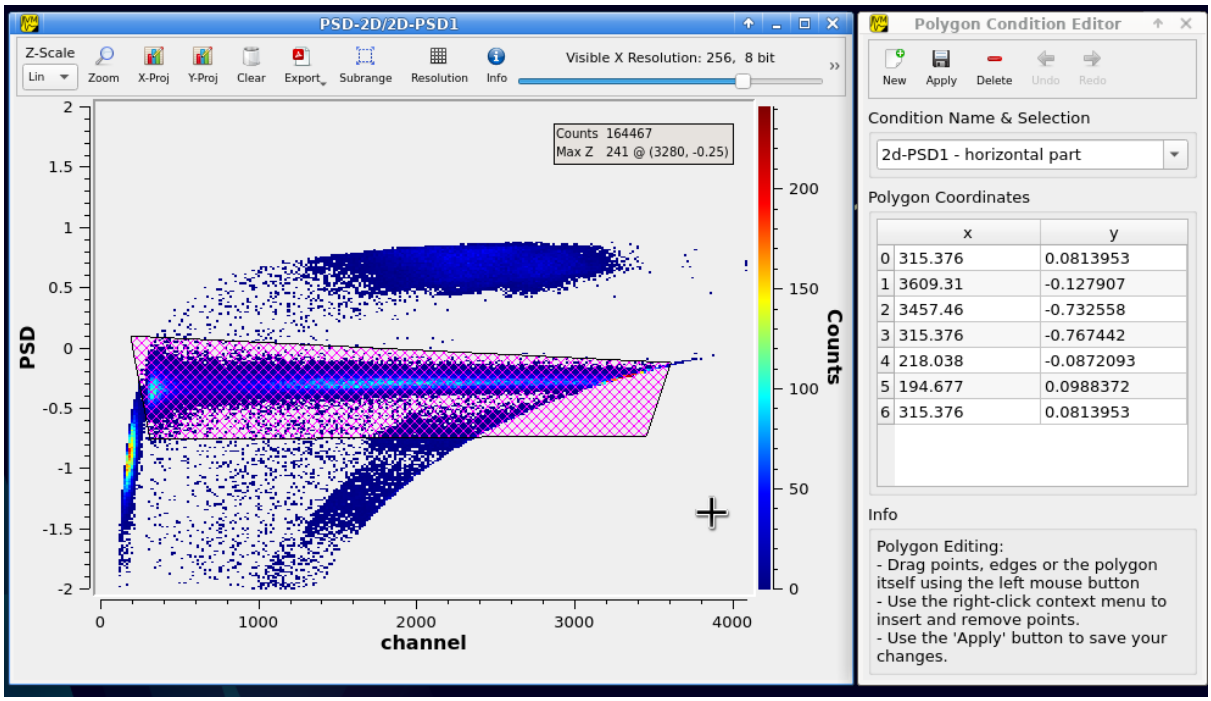


Fig. 4.19: Polygon Condition Editor

Expression conditions are created the same way as other operators: using the context menu and selecting `New -> Expression Condition`.

Using conditions

Once you have created one or more condtions you can apply them to operators and (histogram) sinks using the right-click context menu and selecting “conditions”. This works the same way as the input selection when creating operators. If multiple conditions are applied to the same operator they all have to evaluate to **true** in the current event for the operator to be executed.

Condition based listfile filtering

> **Note**
>
> Condition based listfile filtering is implemented for the MVLC only.

Since mvme-1.10 analysis conditions can be used to filter input listfiles and produce smaller output files which are faster to re-analyze than the original input file.

For each readout event defined in the VME config a condition can be selected. During a replay with filtering enabled only those events for which the selected condition evaluates to true are written to the output file.

To use the feature first create a condition used for filtering, e.g. a simple interval condition, then load a listfile and press the `Listfile Filtering` button in the analysis window.

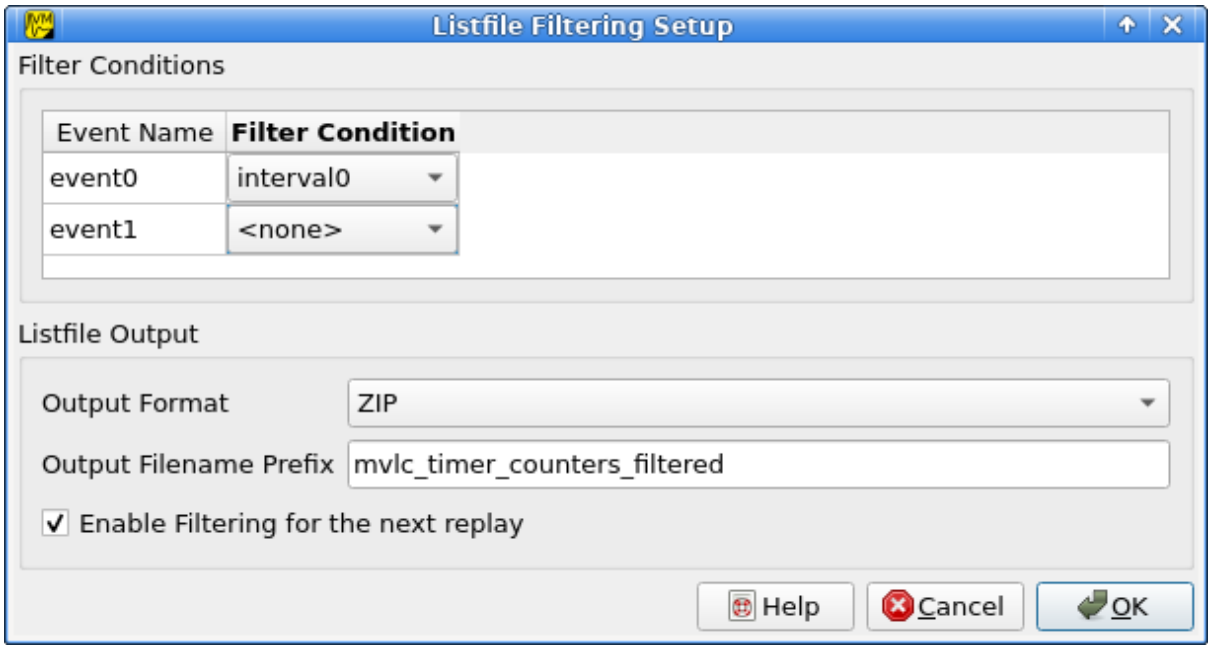


Fig. 4.20: Listfile filtering setup. `event0` is filtered by the condition `interval0`, `event1` is not filtered.

In the dialog that opens up a single condition can be selected for each defined event. If no condition is selected the respective event will always end up in the output listfile. For a condition to show up in the selection box it has to run in the same event context as the respective event.

Optionally set a prefix for the output listfile filename. To guarantee uniqueness a timestamp string is appended to the filename prefix.

Make sure the `Enable filtering` box is checked then accept the dialog and start the replay. The current output filename will be printed to the mvme log window.

4.5.1.5 Inspecting Dependencies

Since version 1.5 mvme includes a dependency graph viewer for analysis objects. It allows to visualize the data path and any active conditions affecting the processing. The graph can be opened from within histogram windows using the `Dependency Graph` button via the right-click context menu in the main analysis window.

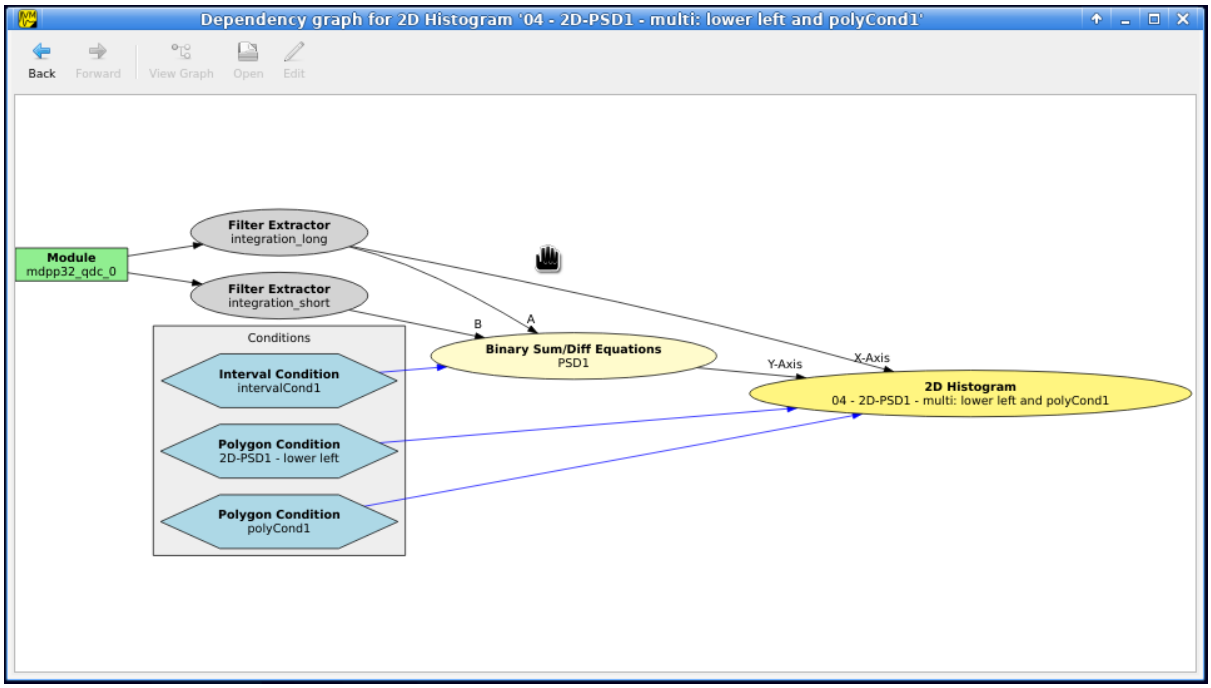


Fig. 4.21: Dependency graph for a 2D histogram. Active conditions in the data path are grouped together in the grey box.

Within the graph double-clicking on histograms or other sinks opens the respective view, while double clicks on other objects start the appropriate editor. Ctrl-clicking navigates to the graph for the clicked object.

4.5.2 System Details

As outlined in the *introduction* the analysis system is a set of interconnected objects with data flowing from *Sources* through *Operators* into *Sinks*.

The system is structured the same way as the VME Configuration: VME modules are grouped into events. An event contains the modules that are read out on activation of a certain trigger condition. The result of the readout is the modules event data (basically an array of 32-bit words). This module event data is the input to the analysis system.

When processing data from a live DAQ run or from a listfile replay the analysis system is "stepped" in terms of events: in each step all the *Data Sources* attached to a module get passed the modules event data. The task of each source is to extract relevant values from its input data and make these values available to subsequent operators and sinks.

After all sources have processed the module event data, the dependent operators and sinks are stepped in order. Each object consumes its input and generates new output or in the case of sinks accumulates incoming data into a histogram.

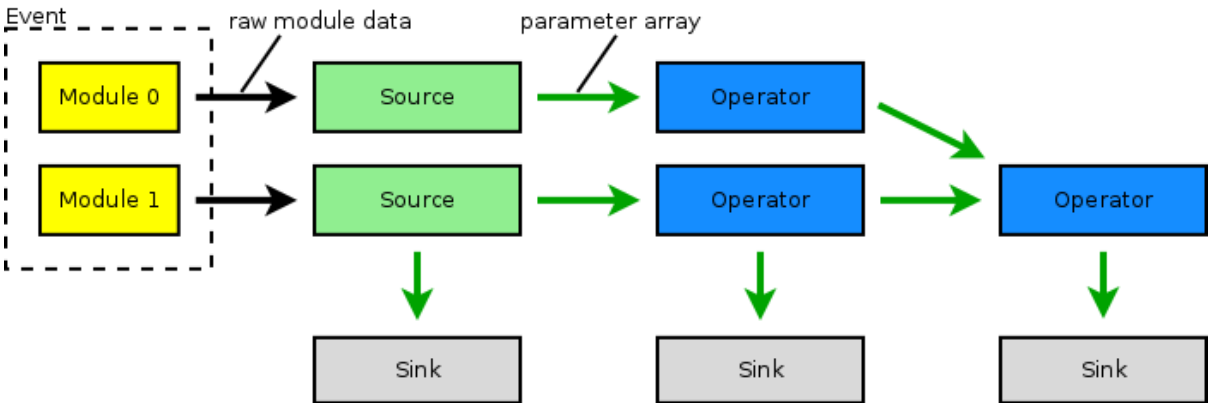


Fig. 4.22: Example analysis dataflow

4.5.2.1 Parameter Arrays

The transport container carrying data between objects is the Parameter Array:

Parameter Array			
size	unit label		
Parameters			
0	value	valid	limits
1	value	valid	limits
2	value	valid	limits
...			
size-1	value	valid	limits

The *size* of parameter arrays is determined at analysis startup time and is constant throughout the run. The *unit label* is a string which currently can be set through the use of the *Calibration Operator*. The index of a parameter in the array is usually the channel address that was extracted from the modules data.

Each parameter has the following attributes:

- *value* (double)

  The parameters data value.

- *valid* (bool)

  True if the parameter is considered valid, false otherwise.

  A parameter can become invalid if for example a data source did not extract a value for the corresponding channel address or an operator wants to explicitly filter out the address or could not calculate a valid result for the input value.

- *limits* (two doubles)

  Two double values forming the interval `[lowerLimit, upperLimit)` that the parameters value should fall into. This is used by histogram sinks and calibration operators to determine the parameters range and thus calculate the binning.

4.5.2.2 Connection types

Different operators have different requirements on their input types. The *Calibration Operator* for example can use whole parameter arrays as its input, transforms each data value and produces an output array of the same size as the input size.

Other operators can only act on individual values and thus connect directly to a specific *index* into the parameter array. An example is the *2D Histogram Sink*: it requires exactly two input values, X and Y, neither of which can be an array.

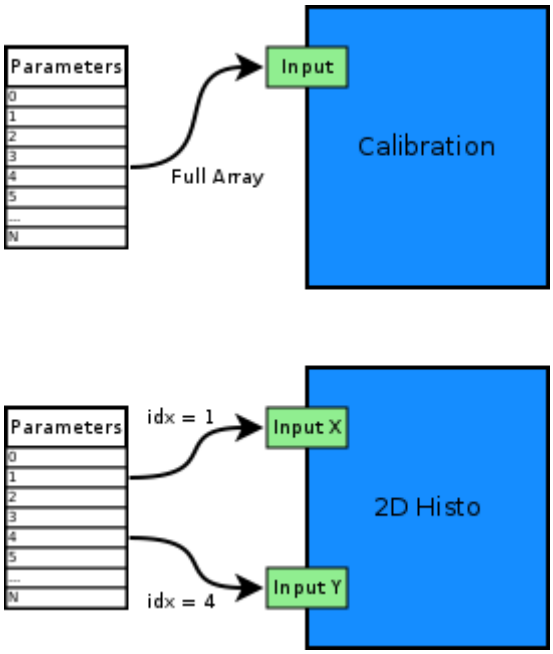


Fig. 4.23: Example of different input types

Each Operator implementation decides which types of input connections it accepts. Some operators even change the type of inputs they accept based on the first input type that is connected (they either accept full arrays for all their inputs or single values for all their inputs).

The *Analysis UI* will highlight valid input nodes in green when selecting an operators input.

4.5.3 Data Sources

Analysis Data Sources attach directly to a VME module. On every step of the analysis system they're handed all the data words produced by that module in the corresponding readout cycle. Their job is to extract data values from the raw module data and produce an output parameter array.

4.5.3.1 Filter Extractor

The Filter Extractor uses a list of bit-level filters to classify input words and extract address and data values.

Filter Basics

A single filter consists of 32 characters used to match a 32-bit data word. The filter describes the static parts of the data used for matching and the variable parts used for data extraction. The first (leftmost) character of a filter line matches bit 31, the last character bit 0.

The following characters are used in filter strings:

Character	Description
`0`	bit must be cleared
`1`	bit must be set
`A`	address bit
`D`	data bit
others	don't care

The following conventions are used in the default filters that come with mvme:

- `X` is used if any bit value is allowed.

- `O` (the letter) is used to denote the position of the *overflow* bit.
- `U` is used to denote the position of the *underflow* bit.
- `P` is used to denote the position of the *pileup* bit.

These characters are merely used to make it easier to identify certain bits when editing a filter. With regards to matching any character other than `0` or `1` means that any bit value is allowed.

Example: The default *Amplitude* filter for the MDPP-16_SCP:

```
0001 XXXX PO00 AAAA DDDD DDDD DDDD DDDD
```

The filter above contains a 4-bit address and a 16-bit data value. The positions of the pileup and overflow bits are marked using `P` and `O`. This helps when adjusting the filter to e.g. match only pileup data (replace the `P` with a `1`).

The number of address bits (`A`) determine the size of the Filter Extractors output array.

Data extraction from an input data word is done by keeping only the bits matching the address or data mask and then right shifting to align with the 0 bit.

Note

Address and data bit masks do not need to be consecutive. `A0AA` will produce 3-bit address values by gathering all extracted `A` bits on the right: `0AAA`.

Each filter has an optional *word index* attached to it. If the word index is set to a value >= 0, then the filter can only produce a match on the module data word with the same index.

Multiple filter words

The Filter Extractor implementation allows combining multiple 32-bit filters to match and extract data from multiple input words.

Filters are tried in order. If a previously unmatched filter produces a match no further filters will be tried for the same data word.

Once all individual filters have been matched the whole combined filter matches and address and data values can be extracted.

When extracting values the filters are again used in order: the first filter produces the lowest bits of the combined result, the result of the next filter is left-shifted by the amount of bits in the previous filter and so on.

Note

The maximum number of bits that can be extracted for address and data values is limited to 64.

See *Timestamp extraction* for an example of how a multiword filter can be used.

Matching and data extraction

During a DAQ run or a replay the Filter Extractor gets passed all the data that was produced by a single module readout (*Event Data*). Each data word is passed to the internal filter.

Once the filter has completed *Required Completion Count* times, address and data values will be extracted.

The data value is cast to a double and a uniform random value in the range `[0, 1)` is added. This resulting value is stored in the output parameter array at the index specified by the extracted address value.

User Interface

In the Analysis UI right-click a Module and select *New -> Filter Extractor* to add a new filter.

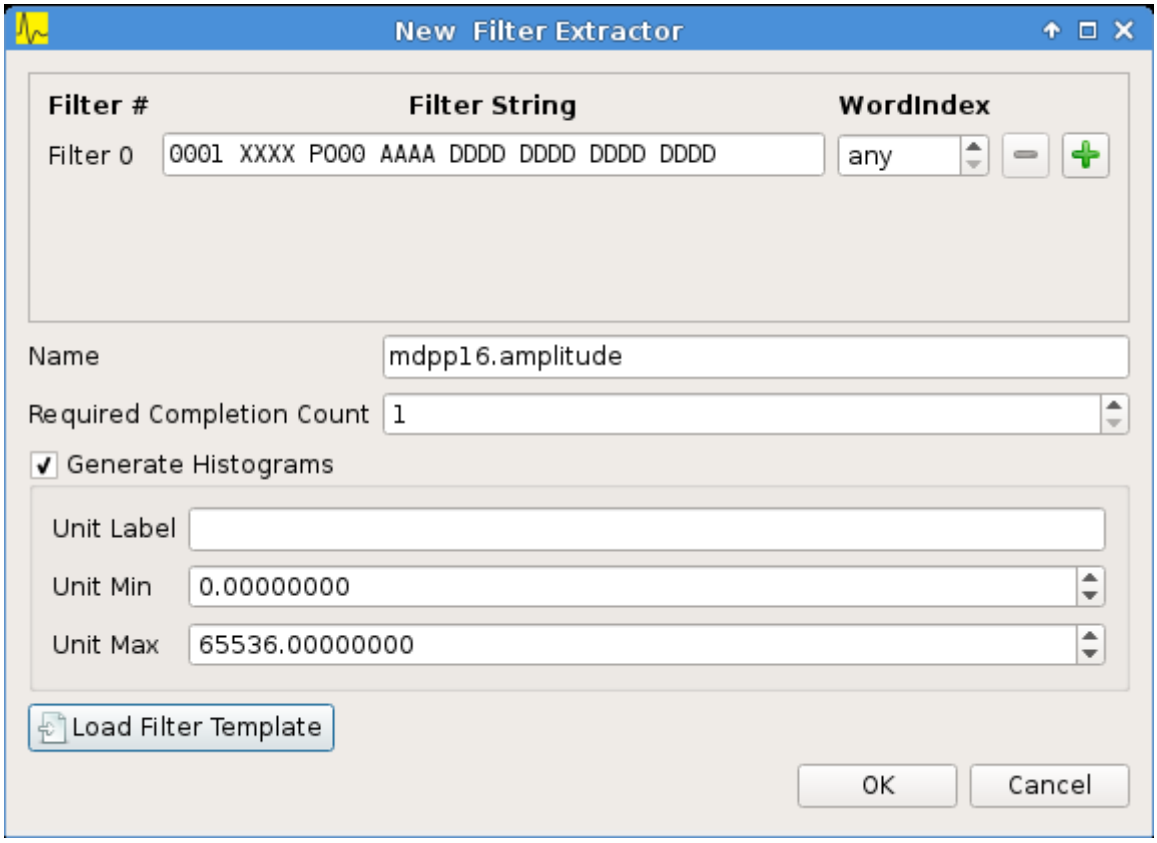


Fig. 4.24: Filter Extractor UI

Use the + and - symbols to add/remove filter words. The spinbox right of the filter string lets you specify a word index for the corresponding filter.

Required Completion Count allows you to specify how many times the filter has to match before it produces data. This completion count starts from 0 on every module event and is incremented by one each time the complete filter matches.

If *Generate Histograms* is checked raw and calibrated histograms will be created for the filter. *Unit Label*, *Unit Min* and *Unit Max* are parameters for the *Calibration Operator*.

Predefined filters can be loaded into the UI using the *Load Filter Template* button.

4.5.3.2 MultiHit Extractor

The *Filter Extractors* are limited to a single hit per extracted (channel) address. If multiple hits per address can occur in the same event the MultiHit Extractor can be used to gain access to these hits.

Properties

- filter

  A single *bit-level filter* specifying the bit-pattern to match and the position and number of address and data bits.

- maxHits

  Maximum number of hits to record per address. The total number of hits will be counted in a separate *hitCounts* array.

- shape
  - Array per Hit

    Hit0 for all addresses is stored in the first array, hit1 in the second, etc. A total of `maxHits` arrays is created each of length $2^{addrBits}$.
  - Array per Address

    Hits for the same address are recorded in an array of length `maxHits`. Creates a total of $2^{addrBits}$ arrays.

Diagram showing the number and size of the MultiHit Extractor output arrays depending on the selected shape type:

```
                Array per Hit
              =================

          +----------------------+ <---.
    hit0  |                      |     |
          +----------------------+     |
          +----------------------+     |
    hit1  |                      |  maxHits
          +----------------------+  arrays
                    ...                |
          +----------------------+     |
    hitN  |                      |     |
          +----------------------+ <---'

          ^                      ^
          '----len=2^addrBits----'

          +----------------------+
hitCounts |                      |
          +----------------------+

          ^                      ^
          '----len=2^addrBits----'


                Array per Address
              ====================

          +----------------------+ <------.
  hits[0] |                      |        |
          +----------------------+        |
          +----------------------+        |
  hits[1] |                      |   2^addrBits
          +----------------------+     arrays
                    ...                   |
          +----------------------+        |
  hits[N] |                      |        |
          +----------------------+ <------'

          ^                      ^
          '------len=maxHits-----'

          +---------------------------+
hitCounts |                           |
          +---------------------------+
```

(continues on next page)

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

(continued from previous page)

```
        ^                           ^
        '------len=2^addrBits-------'
```

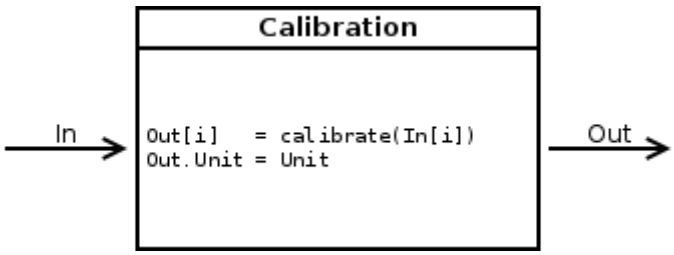
4.5.4 Operators

The following operators are currently implemented in mvme:

4.5.4.1 Calibration

The calibration operator allows to add a unit label to a parameter array and to calibrate input parameters using *unitMin* and *unitMax* values.

Each input parameters `[lowerLimit, upperLimit)` interval is mapped to the outputs `[unitMin, unitMax)` interval.



With *calibrate()*:

```
Out = (In - lowerLimit) * (unitMax - unitMin) / (upperLimit - lowerLimit) + unitMin
```

Limits can be specified individually for each address in the input array. Use the *Apply* button to set all addresses to the global min and max values.

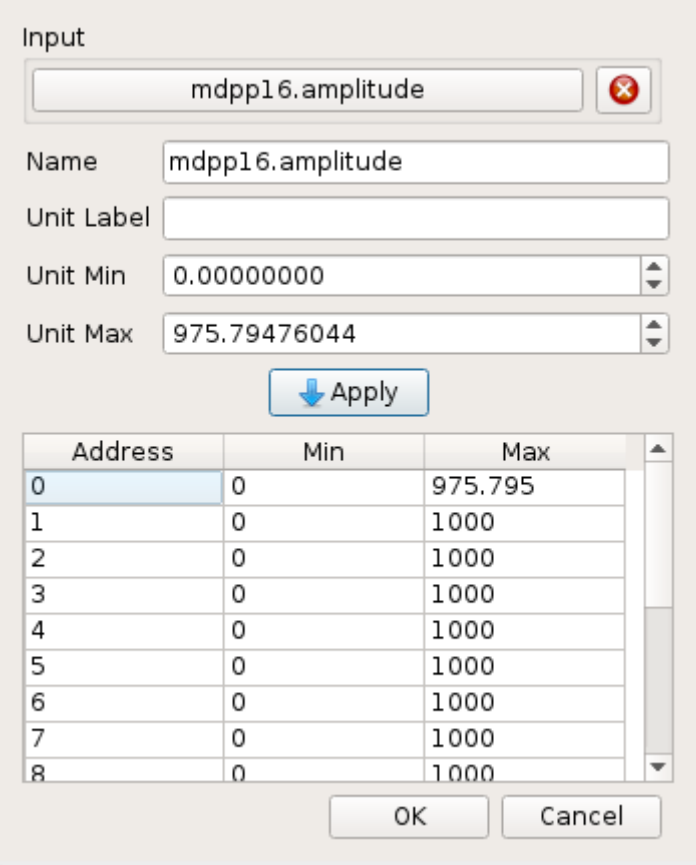


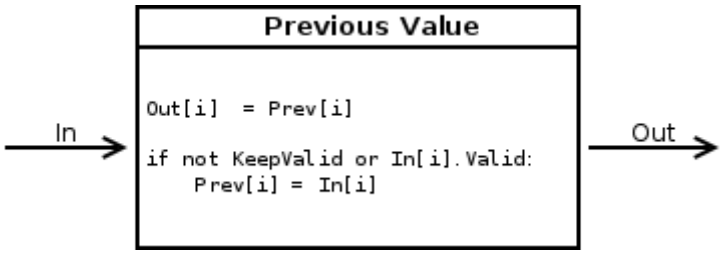
Fig. 4.25: Calibration UI

Note

Calibration information can also be accessed from adjacent 1D histograms. Refer to *Working with 1D Histograms* for details.

4.5.4.2 Previous Value

Outputs the input array from the previous event. Optionally outputs the last input that was valid.



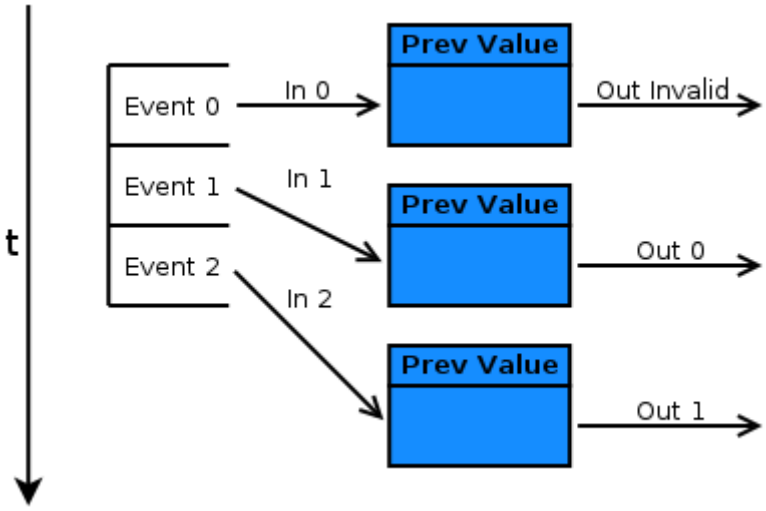


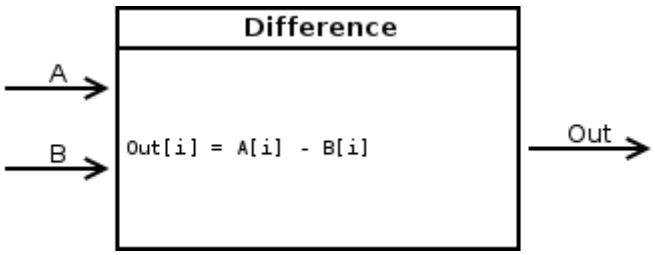
Fig. 4.26: Behaviour of Previous Value over time.

If *keepValid* is set the output will always contain the last valid input values.

This operator can be combined with the *Difference Operator* to accumulate the changes of a parameter across events. See *Rate Estimation Setup* for an example.

4.5.4.3 Difference

Produces the element-wise difference of its two inputs *A* and *B*:



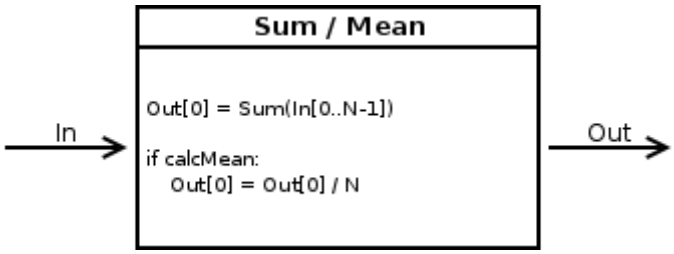
The output unit label is taken from input *A*. If `A[i]` or `B[i]` are invalid then `Out[i]` will be set to invalid:

```
Out.Unit = A.Unit
Out[i].lowerLimit = A[i].lowerLimit - B[i].upperLimit
Out[i].upperLimit = A[i].upperLimit - B[i].lowerLimit
Out[i].value      = A[i].value - B[i].value
```

4.5.4.4 Sum / Mean

Calculates the sum (optionally the mean) of the elements of its input array.

This operator produces an output array of size 1.



When calculating the mean the number of *valid* input values is used as the denominator.

4.5.4.5 Aggregate Operations

This operator combines various aggegrate functions in a single operator: `sum`, `mean`, `min`, `max`, `sigma`, `multiplicity`, `min x`, `max x`, `mean x`, `sigma x`. The output is always an array of size 1.

The input values used for the selected calculation can optionally be filtered by threshold limits: values that fall outside the threshold range are not used for the calculation of the result. The threshold can be either a lower limit, an upper limit or both. Lower and upper limits are inclusive.

Operations

- sum

  Sum of all array elements.

- mean

  Sum of valid array elements divided by the number of valid elements.

- min

  Minimum value in the input array.

- max

  Maximum value in the input array.

- sigma

  Standard deviation of the array elements.

- multiplicity

  The number of valid array elements. Example: determine the number of responding channels in an event.

- min x

  Zero based index of the minimum value in the array.

- max x

  Zero based index of the maximum value in the array.

- mean x

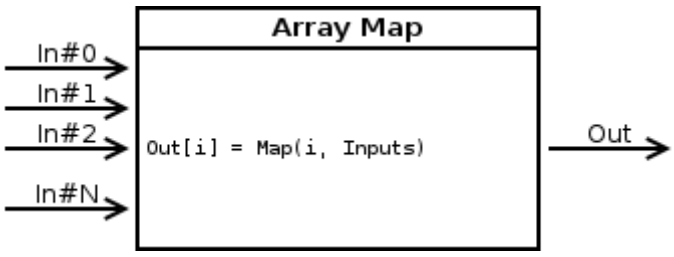
  Mean over the x values: `sum(input[i] * i) / sum(input[i])`.

- sigma x

  Standard deviation of the x values.

4.5.4.6 Array Map

Allows selecting and reordering arbitrary indices from a variable number of input arrays.



The mappings are created via the user interface:

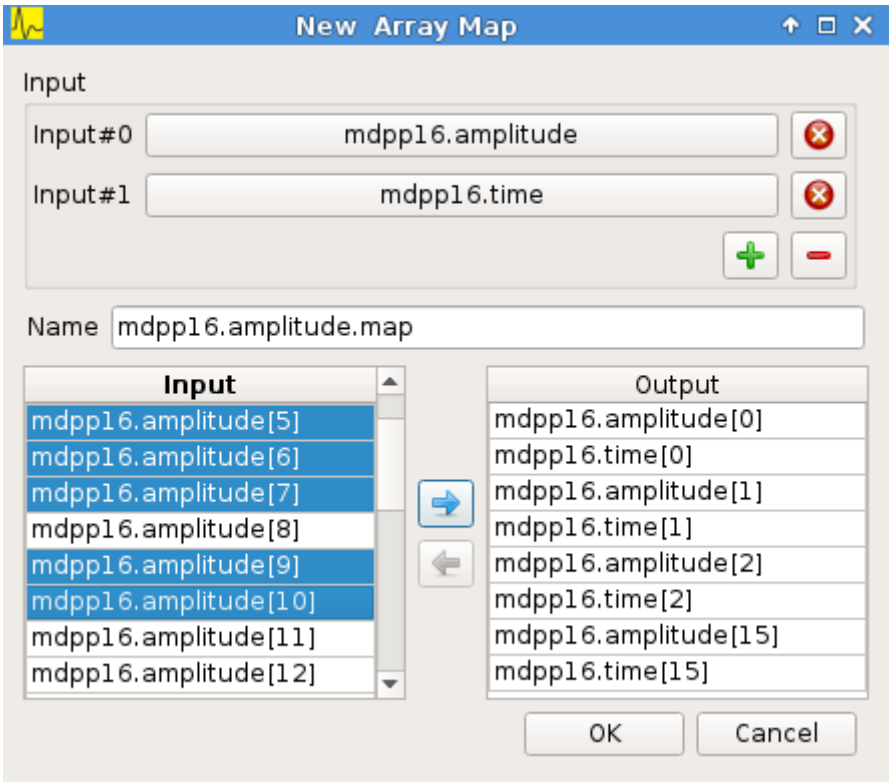


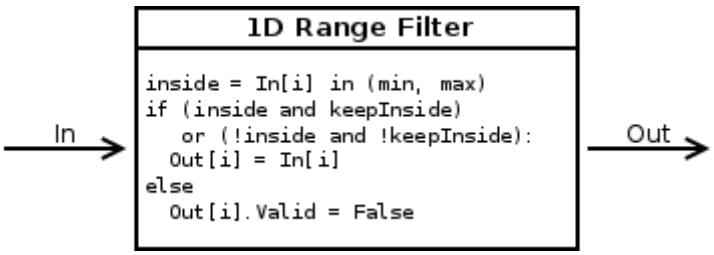
Fig. 4.27: Array Map UI

- Use the + and - buttons to add/remove inputs. The elements of newly added inputs will show up in the left *Input* list.
- Select elements in the *Input* and *Output* lists and use the arrow buttons to move them from one side to the other.

Multiple items can be selected by control-clicking, ranges of items by shift-clicking. Both methods can be combined to select ranges with holes in-between them. Focus a list and press `Ctrl-A` to select all items.

4.5.4.7 1D Range Filter

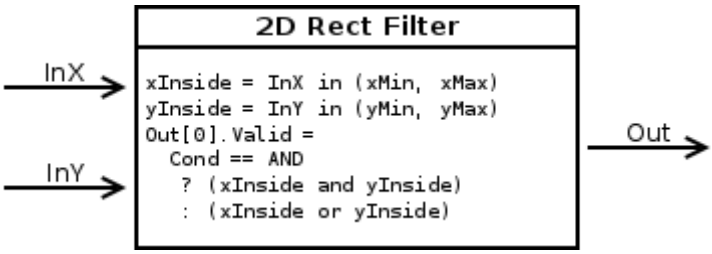
Deprecated since version 1.14: Use the new condition system and optionally the ConditionFilter instead.



Keeps values if they fall inside (optionally outside) a given interval. Input values that do not match the criteria are set to *invalid* in the output.

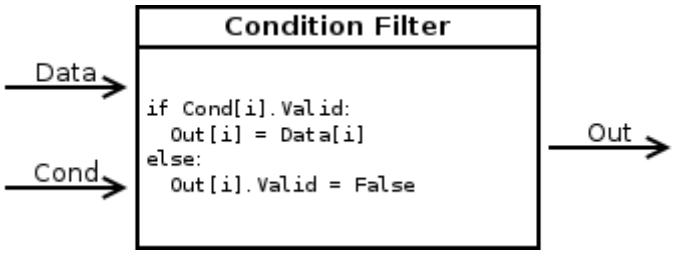
4.5.4.8 2D Rectangle Filter

Deprecated since version 1.14: Use the new condition system and optionally the ConditionFilter instead.



Produces a single *valid* output value if both inputs satisfy an interval based condition.

4.5.4.9 Condition Filter



Copies data input to output if the corresponding element of the condition input is valid.

4.5.4.10 Expression Operator

This operator uses the exprtk library to compile and evaluate C-like, user-defined expressions.

The operator supports multiple inputs and outputs. The definition of the outputs is done using an exprtk script, which means arbitrary calculations can be performed to calculate the number of outputs, their sizes and their parameter limits.

During analysis runtime a second script, the *step script*, is evaluted each time event data is available. The script calculates and assigns parameter values to the operators output arrays.

Details about the syntax and semantics are provided in the online help in the Expression Operator user interface.

4.5.4.11 ScalerOverflow

The ScalerOverflow operator outputs a contiguous value given an input value that overflows. This can be used to handle data like module timestamps which wrap after a certain time.

4.5.5 Data Sinks

mvme currently implements the following data sinks:

4.5.5.1 1D Histogram

Accumulates incoming data into 1D histograms. Accepts a full array or an individual value as input. If given a full array the number of histograms that will be created is equal to the array size.

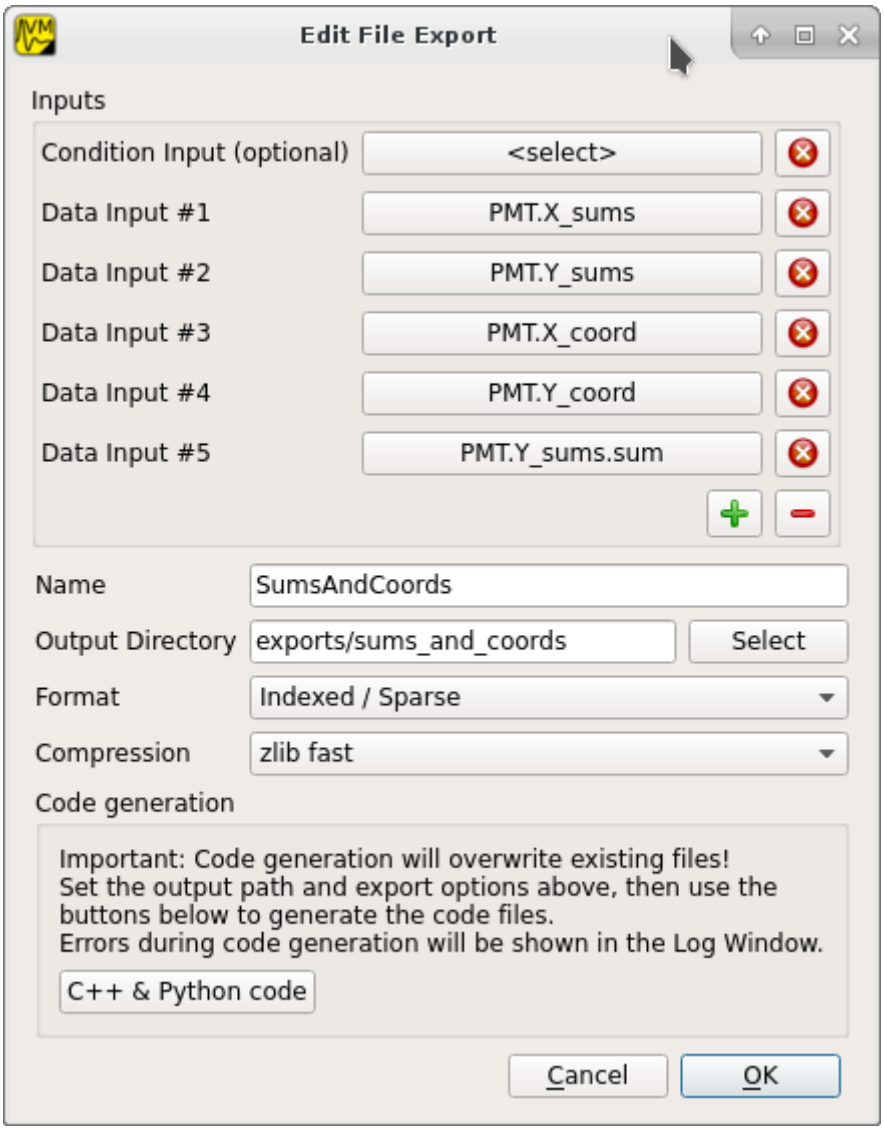
See *Working with 1D histograms* for usage details.

4.5.5.2 2D Histogram

Accumulates two incoming parameters into a 2D histogram. On each event input data will only be accumulated if both the X- and Y inputs are *valid*.

See *Working with 2D histograms* for details.

4.5.5.3 Export Sink



Implements data export to binary files and C++/Python example code generation.

This operator does not create an exported version of all the readout data but instead lets the user select a specific subset of analysis data arrays - all belonging to modules in the same VME event - to be exported to a binary file on disk. Additionally skeleton code can be generated and used as a base for reading in the generated file and working with the data.

For a complete, network-based readout data export including ROOT tree generation see *EventServer*.

The Export Sink has a variable number of data input arrays that will be written to disk. Additionally a single parameter condition input can be used to pre-filter data: output data will only be generated if the condition input is *valid*.

For each DAQ run or replay an export file named *data_<runid>.bin* is generated and the data from each event is appended to that file. If zlib compression is enabled the extension *.bin.gz* is used.

The inputs define the layout of the exported data (in the case of the "Plain/Full" format the export file contains plain, packed C-structs).

Use the "C++ & Python Code" button to generate code examples showing how to read and work with export data.

To compile the C++ code run `cmake . && make` inside the export directory. The CMake file will try to find a ROOT installation using the environment variable `${ROOTSYS}` and will search for **zlib** in the standard system

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

paths.

Most of the generated executables take an export binary file as their first command line argument, e.g:

```
./root_generate_histos my_run_001.bin.gz
```

4.5.5.4 Rate Monitor

The rate monitor uses its input values as precalculated rates or calculates the rate using the difference of successive input values. Rate values are kept in a circular buffer and a plot of the rate over time can be displayed.

Details can be found in the Rate Monitor user interface.

4.5.6 Conditions

See *Condition System* for a general description of how conditions work in mvme.

4.5.6.1 Interval Condition (1D Gates)

Accepts a parameter array as the single input value. Holds an interval for each member of the input array.

When being evaluated the Interval Condition checks if each input parameter value is inside its respective interval. The final condition result is the **logical OR** over the individual interval checks.

Note: intervals are interpreted as half-open with the lower border considered part of the interval.

4.5.6.2 Polygon Condition

A two-dimensional condition checking if the input coordinates are contained within a polygon.

4.5.6.3 Expression Condition

Higher level condition, accepting multiple other conditions as its input. Uses the exprtk library to evaluate a user-defined expression. The expression result becomes the conditions output value.

4.5.7 Readout Data Preprocessing

The standard data path into the analysis is straighforward: a readout parser component interprets the data stream and produces reassembled module data buffers which are then handed to the analysis system:

```
raw readout data -> readout_parser -> analysis
```

Using the **Event Settings** dialog in the Analysis UI, the following additional processing steps can be enabled.

4.5.7.1 Multi Event Processing

The purpose of Multi Event Processing module is to split multievent data obtained from VME block reads into separate, individual events.

Multievent data can be read out from mesytec modules if `multi event mode (0x6036=0xb)` is enabled and `max transfer data (0x601A)` is set to a value greater than 1. With this setup each VME block read from the module will yield up to the number of events specified in `max transfer data` provided the module has enough events buffered up. This way of reading out modules can dramatically improve performance as the modules can make full use of their internal buffers and less individual VME transfers are used to read out the same number of events. The drawbacks are that non-perfect trigger handling can lead to unsynchronized events across modules and that data processing becomes more complicated.

Splitting is performed on the module data as shown in the diagram below:

```
 multievent
+------------+                 split0           split1           split2
| m0_event0 |              +-------------+   +------------+   +------------+
```

(continues on next page)

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

(continued from previous page)

```
| m0_event1 |              |            |   |           |   |           |
| m0_event2 |              | m0_event0  |   | m0_event1 |   | m0_event2 |
|           | == split ==> | m1_event0  |   | m1_event1 |   | m1_event2 |
| m1_event0 |              |            |   |           |   |           |
| m1_event1 |              |            |   |           |   |           |
| m1_event2 |              +------------+   +-----------+   +-----------+
+-----------+
```

In the example the input multievent data obtained from a single event readout cycle is split into three separate events.

Data splitting is performed by using analysis DataFilters to look for module header words. If the filter contains the matching character S it is used to extract the size in words of the following event. Otherwise each of the following input data words is tried until the header filter matches again and the data in-between the two header words is assumed to be the single event data.

```
+-----------+
|m0_header  | <- Filter matches here. Extract event size if 'S' character in
|m0_e0_word0|    filter, otherwise try the following words until another match is
|m0_e0_word1|    found or there is no more input data left.
|m0_e0_word2|
|m0_header1 | <- Filter matches again
|m0_e1_word0|
|m0_e1_word1|
|m0_e1_word2|
+-----------+
```

When using mesytec modules the correct header filters for the modules are setup by default and just enabling `Multi Event Processing` in the `Event settings` dialog is enough to make the splitter system work. For non-mesytec modules the header filter bitmask can be specified in the `Module Settings` dialog found in the context menu of each module in the analysis UI.

Note

When using the MVLC VME Controller a info/debug system for the readout_parser and the multi_event_splitter is available via `Debug & Stats -> Debug next buffer`.

4.5.7.2 Event Builder

Since version 1.4.7 mvme contains a timestamp based EventBuilder module which can be enabled if using the MVLC VME Controller. The purpose of the EventBuilder is to create optimally matched events across modules while tolerating non-perfect trigger setups.

Note

In mvme-1.15.2 the EventBuilder has been reimplemented to yield better results while also having a simpler configuration. The description below refers to this new implementation.

The EventBuilder algorithm works as follows:

1. Recording phase
    1. Record incoming module data into memory buffers.
    2. Extract an event timestamp for each module.
    3. Apply the configured module timestamp offsets.

4. Record all extracted stamps in a list. This can be seen like a virtual module containing the timestamps of all incoming modules.

2. Yield data
    1. Pick the oldest stamp from the virtual stamp list. This is the reference stamp for this cycle.
    2. Ensure all modules have at least one timestamp that is too far in the future to be included in the output event. If this is not the case no data can be flushed yet and the yield phase terminates.
    3. Discard module data that is too old to fall into a modules timestamp match window.
    4. For each module pick the first recorded stamp that falls into the match window. Use the corresponding module data for the output event.

       Break if a match was found or the next recorded module stamp is too far in the future.
    5. If there is at least one module with non-empty output data, yield the output event.
    6. Remove the used reference stamp from the stamp list.
    7. Repeat until no timestamp match is possible anymore, meaning there is no more data or all data is too far in the future.

Note

Timestamp extraction is currently hard-coded to work with the non-extended 30 bit timestamps of mesytec modules.

Event building is only implemented for the mesytec MVLC controller.

Event Building can be enabled in the `Event Settings` dialog. The timestamp offset and match windows can be set in `Event Settings -> Event Builder Settings`.

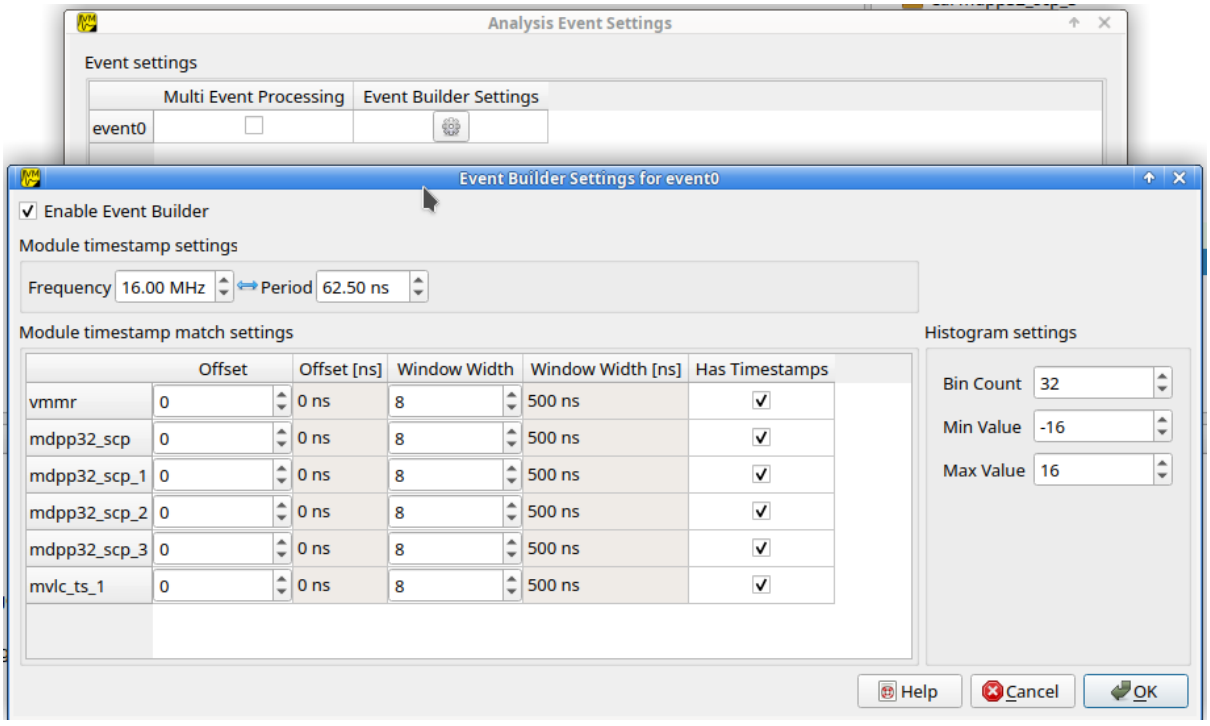


Fig. 4.28: Event Builder Settings

A tool to monitor the EventBuilder operation is available via the `Event Builder Monitor` button in the analysis window. This monitor prints internal counters and shows timestamp delta histograms for all module pairs.

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

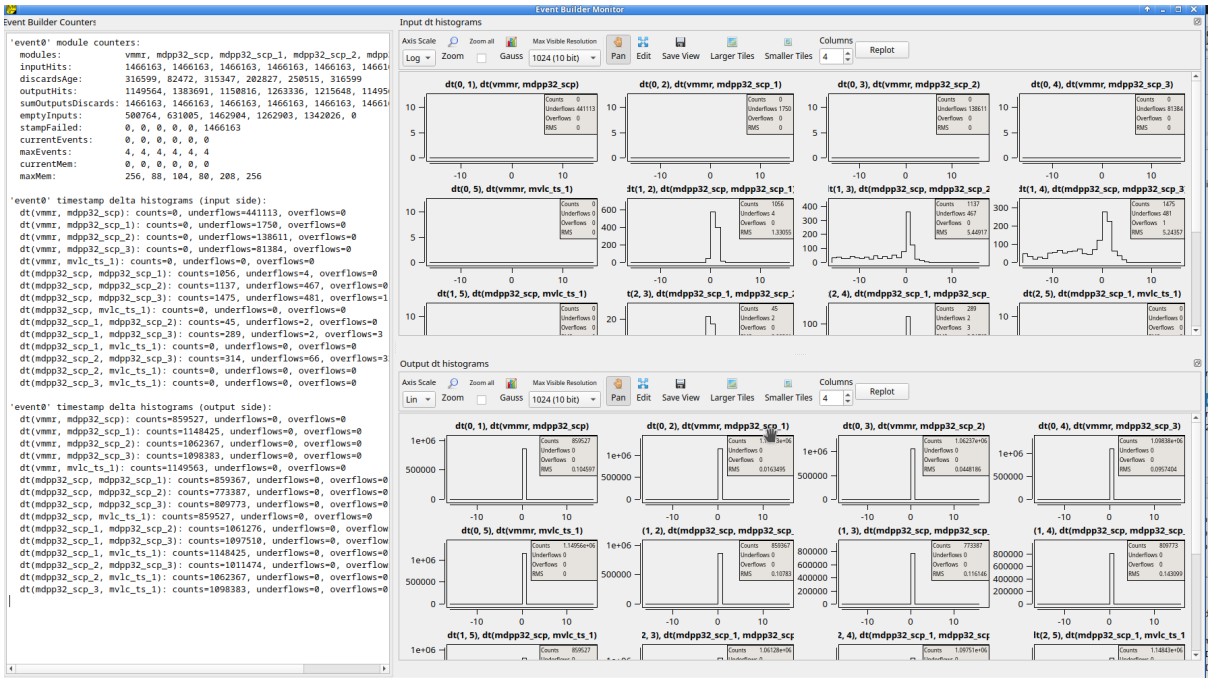


Fig. 4.29: Event Builder Monitor

4.5.8 More UI structuring and interactions

4.5.8.1 Directories

Analysis *Directory* objects can be created in all but the first userlevels (L > 0). Directories are placed in either the top or bottom trees and keep that assignment throughout their lifetime. Directories can contain any analysis objects from the corresponding tree and other subdirectories.

Creating a new directory is done via **New -> Directory** in the context menu.

A directory can serve as the destination of a Drag and Drop operation. All moved objects will be reparented to this destination directory.

4.5.8.2 Drag and Drop

Objects can be moved in-between trees by dragging and dropping. Selected objects from the source tree will be moved to the destination tree. If the destination is a directory all dropped objects will be reparented into that directory.

4.5.8.3 Plot Grid

A plot grid is a view for existing 1D and 2D histograms. A plot grid for an array of 1D histograms can be created on the fly via the context menu entry *Open in Plot Grid*. Alternatively a new, empty grid object can be created using *New -> Plot Grid* in the *Data Display* area.

H1D and H2D objects can be added to the grid by dragging them from the analysis trees onto the grid view.

Use *Save View* in the grid widgets toolbar to permanently save the view in the analysis setup.

The arrangement of plots in the grid can be changed by selecting *Edit* in the toolbar, then dragging plots into the desired order. Right-clicking a plot in *Edit* mode allows to set a custom plot title or remove the plot from the grid view.

4.5.8.4 Multiselections

By holding *Ctrl* and clicking analysis objects it is possible to create a (cross-tree) multiselection. Combine holding *Ctrl* and *Shift* at the same time to add ranges of objects to an existing selection.

> **Note**
>
> Cross-tree multiselections do not apply to drag and drop operations as these start and end on specific trees. Thus using a cross-tree selection as the source of a drag operation would be counterintuitive.

4.5.8.5 Copy/Paste

Object selections can be copied to clipboard by using *Ctrl-C* or choosing *Copy* in a context menu.

Pasting is done via *Ctrl-V* or *Paste* in a trees or directories context menu.

If a selection containing internally connected objects is pasted the connections will be restored on the newly created copies of the original objects. This way a network of operators and sinks can be duplicated quickly as only the "external" inputs will need to be reconnected on the copies.

If a directory has been copied the paste operation will create clones of the directory and all of its subobjects.

Copy/paste of data sources is possible but newly pasted objects will not be assigned to a specific module yet.

4.5.8.6 Import/Export

A way to share parts of an analysis is to **export** a cross-tree multiselection to file and later on **import** from file. Use the *Export* and *Import* toolbar entries in the analysis UI to perform these actions.

The behavior is similary to the copy/paste operations: all selected objects will be exported to file. On import clones of these objects are created, internal connections are restored and all objects are placed in the same userlevels as their originals.

4.5.8.7 Loading foreign analysis files

Internally VME modules are uniquely identified by a UUID and the module type name. This information is stored in both the VME and analysis configs.

When opening (or importing from) a "foreign" analysis file, module UUIDs and types may not match. In this case auto-assignment of analysis objects to VME modules is tried first. If auto-assignment is not possible the unassigned objects are collected under a special node in the top left tree of the analysis window. Data sources from these unassigned modules can be dragged onto modules existing in current DAQ setup to assign them.

4.6 JSON-RPC remote control support

MVME uses the jsonrpc specification to implement basic remote control functionality. To enable/disable the RPC-Server use the "Workspace Settings" button in the DAQ Controls Window and follow the instructions there.

If the RPC-Server is enabled mvme will open a TCP listening socket and accept incoming connections. By default mvme binds to all interfaces and listens on port 13800.

Requests are currently not length-prefixed, instead data is read until a full JSON object has been received. The received JSON is then interpreted according to the JSON-RPC spec.

Note: The JSON-RPC batch feature is not supported by the server implementation.

A command line client written in Python3 can be found in the mvme distribution under `extras/mvme_jsonrpc_client.py`. It allows to (repeatedly) call a single RPC method:

```
$ python3 extras/mvme_jsonrpc_client.py localhost 13800 getDAQState
---> {"id": "0", "jsonrpc": "2.0", "method": "getDAQState", "params": []}
<--- {"id": "0", "jsonrpc": "2.0", "result": "Running"}
```

Since mvme-1.8.0 there is a second script `extras/mvme_jsonrpc_replay.py` which can replay data from multiple listfile archives.

4.6.1 Methods

4.6.1.1 getVersion

Returns the version of MVME running the JSON RPC services.

- Returns

  String containing version information.

4.6.1.2 getLogMessages

Returns the messages buffered up in the MVME log.

- Returns

  StringList - List containing the log messages from oldest to newest.

4.6.1.3 getDAQStats

Returns information about the current DAQ run including counter values, the current run ID, listfile output filename and start time.

- Returns

  Object

4.6.1.4 getVMEControllerType

Returns the type of VME controller in use.

- Returns

  String

4.6.1.5 getVMEControllerStats

Returns detailed VME controller statistics if available for the current controller type.

- Returns:

  Object

4.6.1.6 getVMEControllerState

Returns the connection state of the VME controller.

- Returns:

  String - “Connected”, “Disconnected” or “Connecting”

4.6.1.7 reconnectVMEController

Starts a reconnection attempt of the VME controller. The operation is asynchronous, thus the result will not be directly available. Instead the controller state needs to be polled via `getVMEControllerState` to see the result of the reconnection attempt.

Note: this might in the future be changed to a synchronous version, which immediately returns the result or any errors that occured.

- Returns:

  String - “Reconnection attempt initiated”

4.6.1.8 getSystemState

Combined system state for both the readout/replay side and the analysis/data consumer side.

- Returns

  String - “Idle”, “Starting”, “Running”, “Stopping” or “Paused”

4.6.1.9 getDAQState

Returns the current state of the DAQ/replay side.

- Returns

  String - “Idle”, “Starting”, “Running”, “Stopping” or “Paused”

4.6.1.10 getAnalysisState

Returns the current state of the analysis/data consumer side.

- Returns

  String - “Idle”, “Starting”, “Running”, “Stopping” or “Paused”

4.6.1.11 startDAQ

Starts a new DAQ run. The system must be idle, meaning any previous DAQ runs must have been stopped.

- Returns

  true on success, error status and additional information otherwise.

4.6.1.12 stopDAQ

Stops the current DAQ run or replay.

- Returns

  true on success, error status and additional information otherwise.

4.6.1.13 getGlobalMode

Returns the global mode mvme is in. Either “daq” or “replay”.

- Returns

  string - “daq” or “replay”

4.6.1.14 loadAnalysis

```
loadAnalysis(string:  filepath)
```

Attempts to load a mvme analysis from the given file (*.analysis).

4.6.1.15 loadListfile

```
loadListfile(string:  filepath, bool:  loadAnalysis = false, bool replayAllParts =
true)
```

Attempts to open the specified file as a mvme listfile (*.zip).

- Parameters
  - string: filepath - Path to the ZIP listfile. Absolute or relative to the current workspace directory.
  - bool: loadAnalysis = false - If true load the analysis contained in the listfile archive.
  - bool: replayAllParts = true - If a split listfile is detected (filename ending in ‘partNNN’) all parts starting from the given filename will be replayed.

- Returns

  true on success, error info object otherwise.

4.6.1.16 closeListfile

Closes the currently open listfile. The system goes into 'DAQ" state. Has no effect if no listfile is open.

4.6.1.17 startReplay

`startReplay(bool: keepHistoContents = false)`

Attempts to replay from the currently open listfile.

- Parameters:
  - bool: keepHistoContents = false

    If keepHistoContents is true the analysis histograms will not be cleared before starting the replay.

4.6.1.18 stopReplay

Stops the active listfile replay.

4.6.2 Examples

4.6.2.1 Raw JSON data

- Requesting DAQ State:

```
---> {"id": "0", "jsonrpc": "2.0", "method": "getDAQState", "params": []}
<--- {"id": "0", "jsonrpc": "2.0", "result": "Running"}
```

- Starting data acquisition:

```
---> {"id": "0", "jsonrpc": "2.0", "method": "startDAQ", "params": []}
<--- {"id": "0", "jsonrpc": "2.0", "result": true}
```

- An error response:

```
---> {"id": "0", "jsonrpc": "2.0", "method": "startDAQ", "params": []}
<--- {"error": {"code": 102, "message": "DAQ readout worker busy"}, "id": "0",
↪"jsonrpc": "2.0"}
```

- Requesting DAQ stats:

```
---> {"id": "0", "jsonrpc": "2.0", "method": "getDAQStats", "params": []}
<--- {"id": "0", "jsonrpc": "2.0", "result": {"analysisEfficiency": 1,
↪"analyzedBuffers": 4644, "buffersWithErrors": 0, "currentTime": "2018-06-
↪14T11:45:21", "droppedBuffers": 0, "endTime": null, "listFileBytesWritten": 0,
↪"listFileFilename": "", "runId": "180614_114412", "startTime": "2018-06-
↪14T11:44:13", "state": "Running", "totalBuffersRead": 4644, "totalBytesRead":␣
↪6366924, "totalNetBytesRead": 5851300}}
```

4.6.2.2 mvme_jsonrpc_replay.py

- Load analysis from `listfiles/mvmelstrun004_part001.zip` and replay data from the same archive only:

  `mvme_jsonrpc_replay.py listfiles/mvmelstrun004_part001.zip --loadAnalysis`

- Keep the currently opened analysis and accumulate data from `listfiles/mvmelstrun004_part001.zip` without clearing the histograms first.

```
mvme_jsonrpc_replay.py listfiles/mvmelstrun004_part001.zip --keepHistoContents
```

- Replay all parts of a split run starting from `part001`:

  ```
  mvme_jsonrpc_replay.py listfiles/mvmelstrun004_part001.zip --replayAllParts
  ```

4.7 EventServer for data export

Currently there are two ways built into mvme which allow readout data to be accessed by external software:

- The *Analysis ExportSink* which is described in the Analysis reference.

  This method should be used when a limited number of analysis data arrays - all belonging to modules in the same VME event - are to be exported.

- The **EventServer** component described in this section. It provides export functionality of the complete event data produced during a DAQ run. The data is transmitted over a TCP network connection.

  The EventServer should be used when you want to export all of the data produced by an experiment.

  The source code for a minimal example client and for a ROOT client is shipped with mvme.

4.7.1 Overview

The EventServer component is a TCP based network server built into mvme. It uses a custom protocol to transmit extracted VME module data across the network.

The "extracted" data is produced by the data filters attached to each of the VME modules present in the experiment setup. E.g. with the default filters for a MTDC-32 module the server will send 32 time values, 2 trigger_times and a module timestamp.

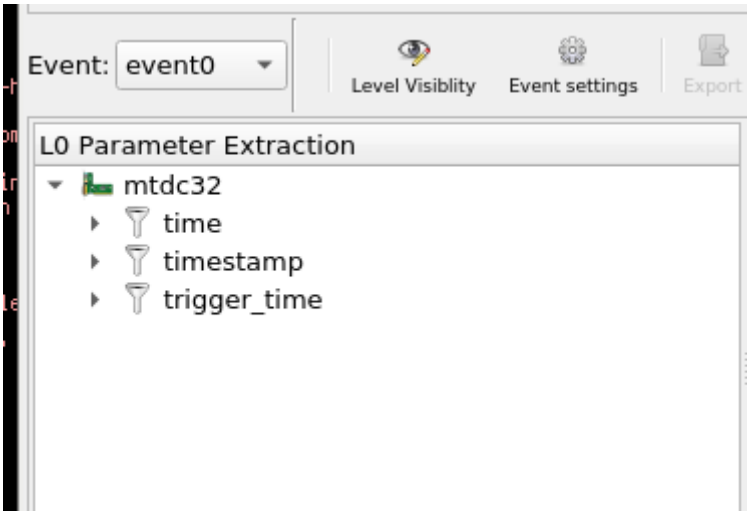


Fig. 4.30: MTDC-32 with default data extraction filters.

The server is attached to the analysis side of mvme which means during a live DAQ run it will not slow down the readout. Instead if the server, or the attached clients, cannot keep up with the readout data rate then the normal buffer loss mechanism at the core of the mvme DAQ will start to discard data. This means during a live run clients may only see parts of the data, whereas when replaying from a mvme listmode file all data will be transmitted.

The supplied ROOT client recreates the full *VME Event -> VME Module -> Data Array* structure present in an mvme setup using automatically generated ROOT classes. Additionally histograms of the raw parameter values are created and written to a separate ROOT file.

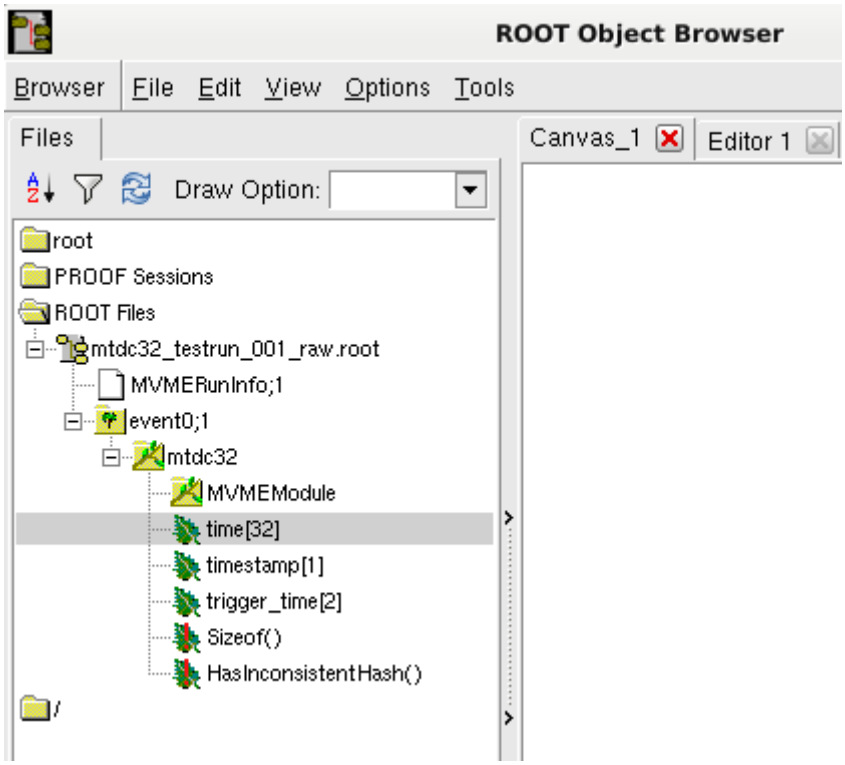


Fig. 4.31: ROOT tree structure created from an MTDC-32.

Note

The ROOT client requires ROOT6 and has so far only been tested on Linux.

Note

An example mvme workspace which contains pre-recorded listmode data can be found on our website: https://mesytec.com/kundendaten/mvme-examples/mvme-example-workspace-01.tar.bz2

This workspace can be used to test the server and client without needing to record your own data first.

4.7.2 Using the EventServer

- In the mvme main window go to `Workspace Settings` and check the `Enable Event Server` box. Optionally enter an IPv4 address or a hostname to bind the server to a specific network interface.

  Also enter an `Experiment Name` in the box at the top of the dialog. This will affect the filenames produced by the mvme_root_client.

- For a first test the *mvme_event_server_example_client* can be used. It can be found in the `bin/` directory of the mvme installation. Additionally the source code and a small Makefile are present in `share/mvme_event_server_example_client`.

  Start the client on the same machine: `./bin/mvme_event_server_example_client --print-data`.

  The client should be able to connect to the server running on localhost.

- Open an example listfile via `File -> Open Listfile` and also load the analysis from the file so that some data extraction filters are present.

- Start the replay. The client prints the incoming data to stdout.

If you want to write your own client you can find the protocol definition and implementation of the base `Client` class in `${MVME}/include/mvme/event_server/common`.

4.7.3 Using the ROOT client

The ROOT client is not shipped in binary form but has to be compiled manually so that it links against the locally installed version of ROOT.

To build the client run the following shell commands:

```
# cd into the mvme installation directory
cd mvme
# setup the enviroment ($PATH, $LD_LIBRARY_PATH)
source bin/initMVME
# compile and install the client
make -C ${MVME}/share/mvme_root_client install
```

You can now create a data directory to hold the generated code and output ROOT files and start the mvme_root_client from within this directory.

The mvme_root_client works in two modes:

1) In client mode it connects to an mvme instance and receives incoming readout data to produce ROOT output files.
2) The program replays data from a previously created ROOT output file.

In the first case the data description sent by mvme at the beginning of a run is used to generate ROOT classes. These classes are then compiled and loaded into the client. Compilation is done using a simple Makefile. The resulting ROOT library is then loaded via TSystem::Load().

During the run incoming event data is interpreted and used to fill instances of the generated ROOT classes. At the end of the run the class hierarchy is written to file in the form of TTrees. Additionally raw histograms are created, filled and written to a separate output file.

In the 2nd case the given input file is opened and mvme specific information is used to locate the previously built ROOT object library. The library is then loaded like in case 1) and the ROOT objects are filled from the data in the input file.

In both cases user-editable analysis code (`analysis.cxx`) is loaded (via dlopen()/dlsym()) and is invoked for each complete event received from mvme or read from the input file. For each defined VME event the client will attempt to call a function called `analyze_<eventname>()`, e.g. `analyze_event0()`.

For details on what the generated classes contain see the `<ExperimentName>_mvme.{h,cxx}` files. The base classes are defined in `${MVME}/share/mvme_root_client/mvme_root_event_objects.{h,cxx}`.

4.8 Metrics

Since version 1.10 mvme exports prometheus compatible metrics. By default an embedded webserver exposes metrics on port 13802. `curl` can be used to list the available metrics: `curl http://localhost:13802/metrics`.

Included in the mvme distribution is a docker-compose project for setting up local prometheus and grafana instances to monitor metrics exported by mvme. Also included is a simple dashboard which can be imported into existing grafana instances. See `extras/metrics/README.md` for details.

CHAPTER

FIVE

HOW-TO GUIDES

5.1 Rate Estimation Setup

This example shows how to use the rate estimation feature built into 1D histograms. Rate estimation works with statistical data that forms an exponential function.

In this example and MDPP-16_SCP is used but any mesytec VME module should work. The rate estimation is setup for channel 8 of the MDPP.

To make use of the rate estimation feature a 1D histogram accumulating the *difference* between timestamp values of incoming events needs to be created.

Note

For the module to produce timestamps instead of event counter values the register `0x6038` needs to be set to `1`. mvme does this by default for newly created modules in the *VME Interface Settings* script.

Steps for creating the histogram:

5.1.1 Timestamp extraction

Add a new Extraction Filter with two filter words to the mdpp16:

```
0001 XXXX XX00 1000 XXXX XXXX XXXX XXXX     any
11XX XXXX XXXX XDDD DDDD DDDD DDDD DDDD     any
```

The first filter word matches on the data word for channel 8: the prefix `0001` identifies a data word, the pattern `00 1000` is used to select channel 8 specifically.

The second filter word is used to extract the timestamp value. The prefix `11` marks an *End of Event* word which contains the 30-bit timestamp. Using the `D` character 19-bits of the timestamp value are extracted.

Set the name of the filter to *ts_c8* and click *Ok* to create the filter.

The complete filter will thus match if there was a data word for channel 8 and the event contained a timestamp data word.

5.1.2 Timestamp calibration

By default the timestamp is generated using the 16 MHz VME SYSCLK. This means the 19-bit timestamp value is in units of 16 MHz.

The goal is to convert the value to μs which is easier to use: $2^{19}/16 = 2^{19}/2^4 = 2^{15} = 32768$.

Thus instead of ranging from $[0, 2^{19}[$ the timestamp should use the range $[0, 2^{15}[$.

To transform the value create a *Calibration Operator* by right-clicking inside the *L1 Processing* tree (or any higher level processing tree) and selecting *New -> Calibration*. Use the green + button in the top-right corner to add a new user level if necessary.

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

As input select the *ts_c8* node created in the previous step. Keep the default name *ts_c8.cal*. Type `µs` in the *Unit Label* field, set *Unit Max* to `32768` and press the *Apply* button. Accept the dialog by pressing *Ok*.

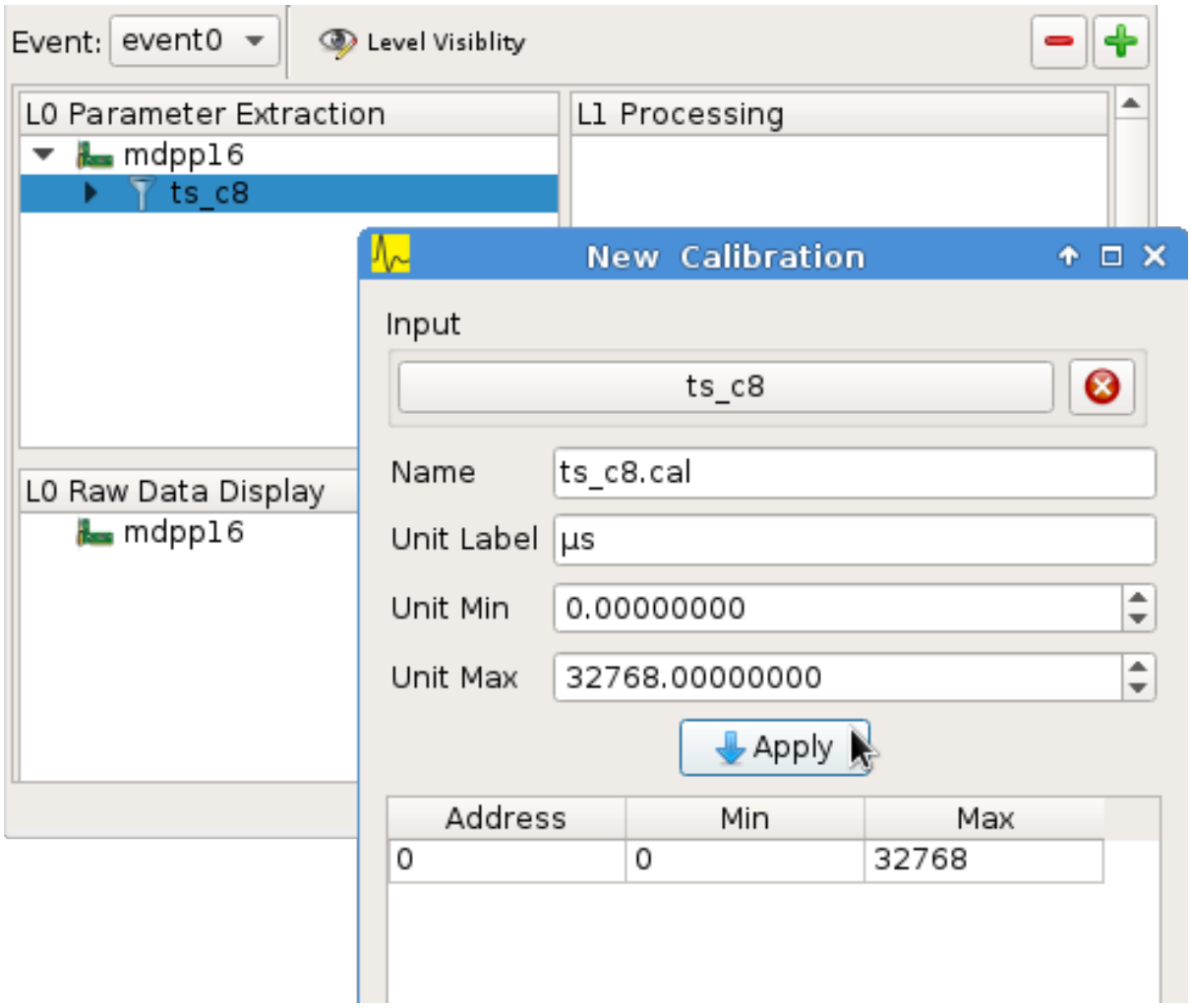


Fig. 5.1: Timestamp calibration

5.1.3 Making the previous timestamp available

Next right-click in the *L1 Processing* tree (or any higher level processing tree) and select *New -> Previous Value*.

As input select the *ts_c8.cal* node created in the previous step. Make sure the *Keep valid parameters* box is checked. Set the name to *ts_c8.cal.prev*.

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

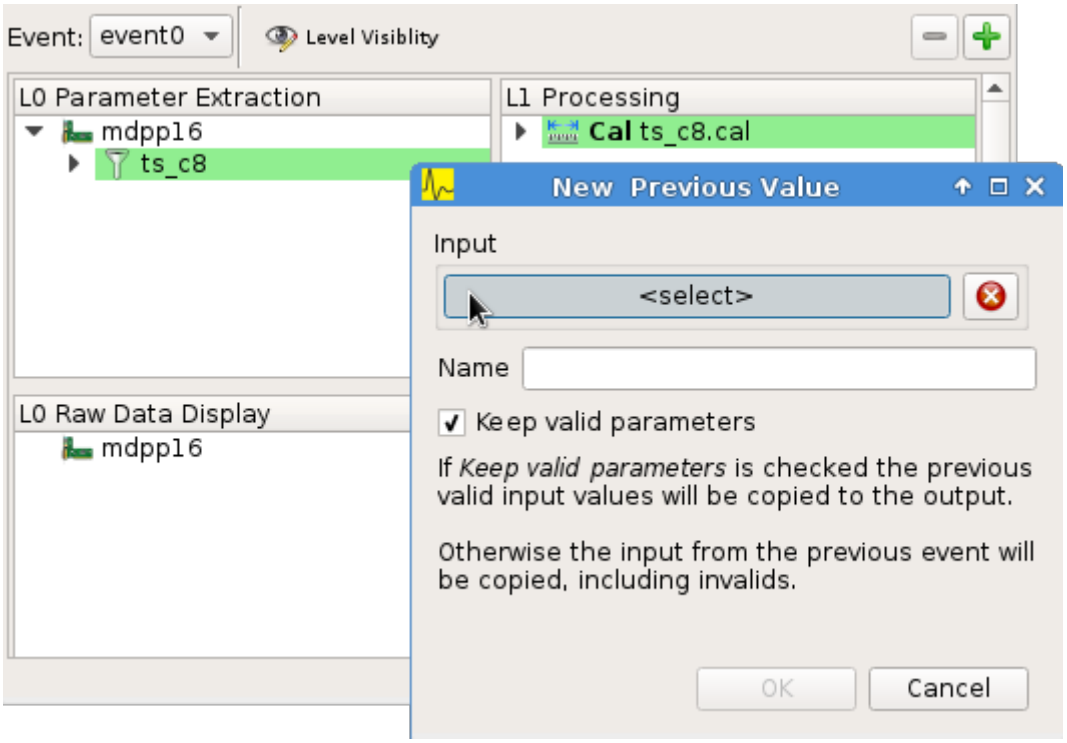


Fig. 5.2: Adding the PreviousValue operator

5.1.4 Histogramming the timestamp difference

Again right-click in one of the processing trees and choose *New -> Difference*. Set input A to *ts_c8.cal* and input B to *ts_c8.cal.prev*. Set the name to *ts_c8.diff*.

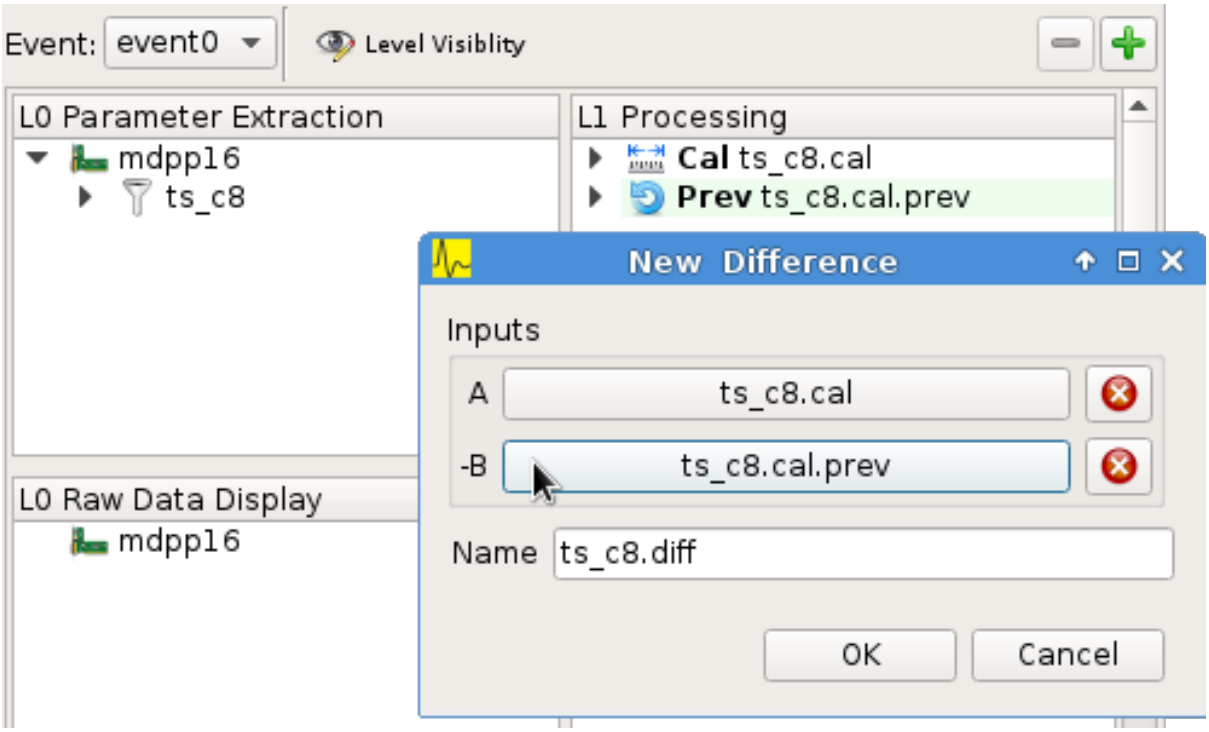


Fig. 5.3: Calculating the timestamp difference

Right-click in the display tree below and add a new 1D histogram using the difference *ts_c8.diff* as the input.

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

Open the newly created histogram click on *Subrange*, select *Limit X-Axis* and enter `(0.0, 200.0)` as the limits. This step limits the large default parameter range calculated by the *Difference operator*.

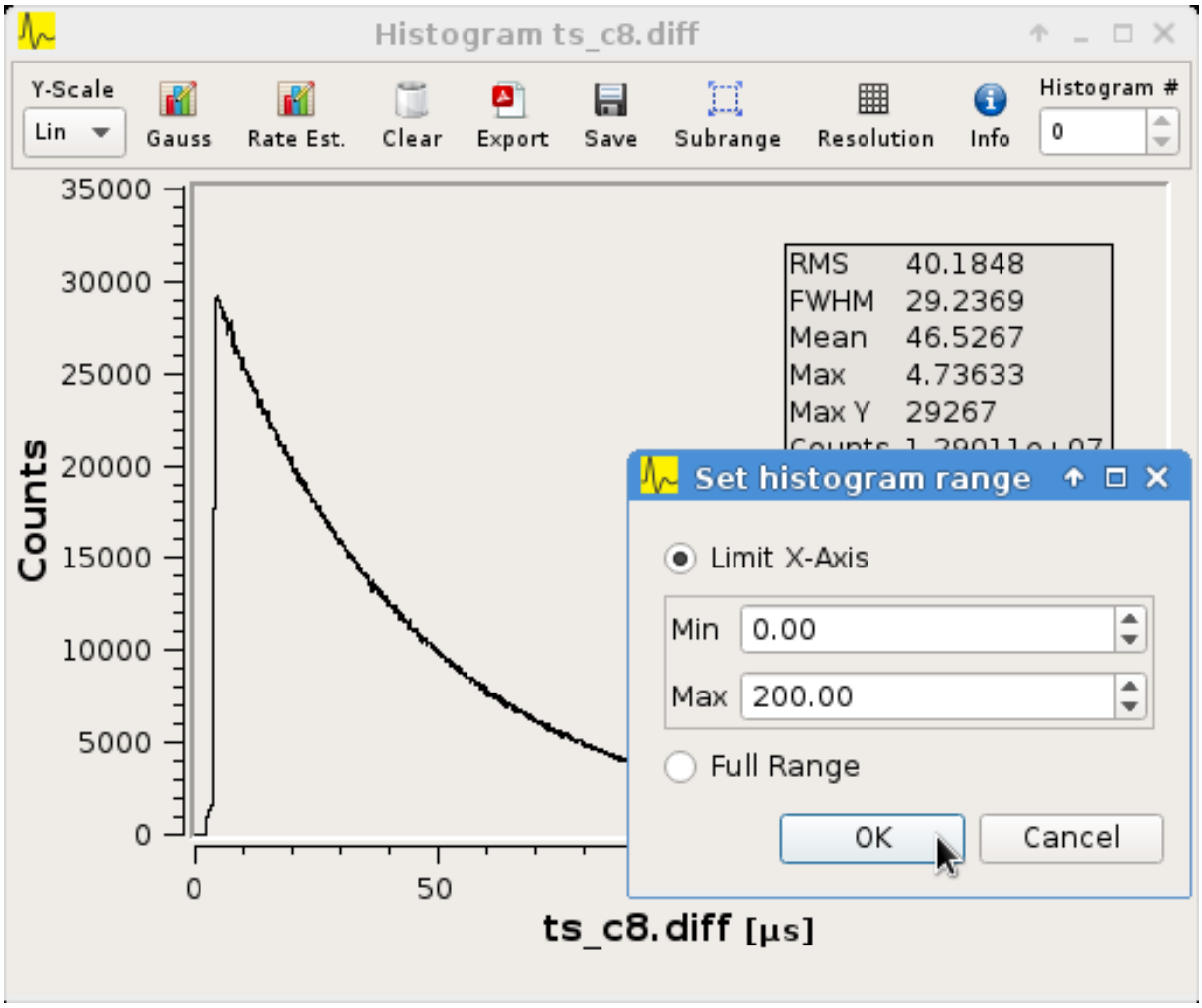


Fig. 5.4: Setting the histogram subrange

Next click the *Rate Estimation* button in the toolbar and then select two points on the x-axis to use for the rate estimation.

mesytec GmbH & Co. KG
Wernher-von-Braun-Str.1, D-85640 Putzbrunn, Germany
phone: +49 - 89 / 456007-30 fax: +49 - 89 / 456007-39
info@mesytec.com - www.mesytec.com

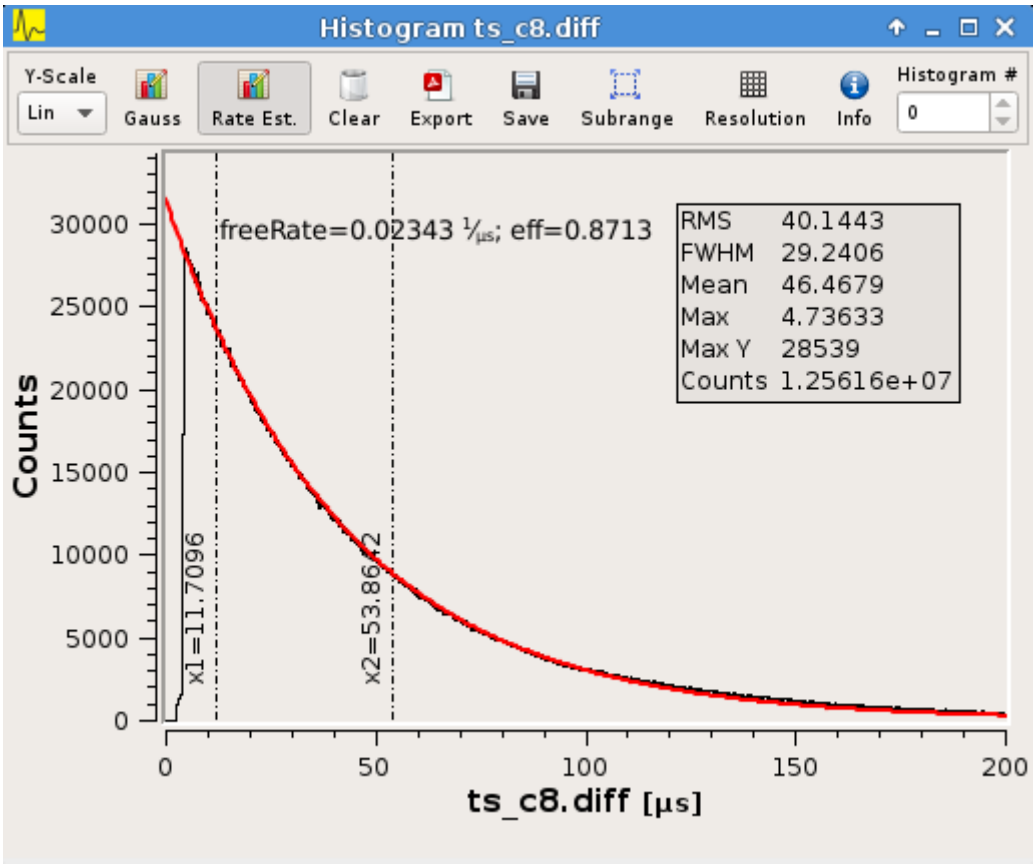


Fig. 5.5: Rate estimation data and curve visible

The calculation performed is:

$$\tau = (x_2 - x_1)/log(y_1/y_2)$$
$$y = y_1 * (e^{-x/\tau}/e^{-x_1/\tau})$$
$$freeRate = 1.0/\tau$$

5.2 VM-USB Firmware Update

The VM-USB firmware update functionality can be found in the mvme main window under *Tools -> VM-USB Firmware Update*. The latest firmware file is included in the mvme installation directory under *extras/vm-usb*.

Before starting the update set the *Prog* dial on the VM-USB to one of the programming positions P1-P4.

The controller will start the newly written firmware immediately after writing completes. Reset the *Prog* dial to C1-C4 to make the controller start the correct firmware on the next power cycle.

CHAPTER

SIX

CHANGELOG

6.1 Version 1.21.0

- analysis:
  - fix upper bound of the multiplicity operator
  - add 'sum' row to the histogram list stats
  - multiple fixes for both 2d and 1d list projections
  - limit projections to zoomed in area of the source histogram
  - remember zoom state and axis scaling when closing histograms. restore when opening
  - fix titles and axis labels when exporting plots to pdf or image
  - plotgrid: if there are pending changes ask if the view should be saved before closing
- analysis waveforms:
  - change the color table to have more contrast.
  - fix the phase correction setting not being applied to the ui

6.2 Version 1.20.0

- new feature: Ability to update the analysis config stored in listfiles.

  Use case: the baseline analysis config at DAQ time is stored in the listfile zip archive at the end of a run. During offline analysis changes are made to the analysis config. At some point these changes should be saved to the listfile, so that subsequent replays automatically use the updated analysis setup.

  Use 'Save to Listfile' from the toolbar in the analysis window to start the update process.

  The update is done by writing a new listfile zip archive containing the original listmode data and the updated analysis config. The original file is atomically replaced with the updated file to minimize the risk of data loss.
- event_builder: Suppress completely empty output events. Fixes weird looking counts in the analysis window.
- internal: The non-gui MVLC Trigger/IO abstractions now live in the mesytec-mvlc driver.

6.3 Version 1.19.0

- new feature: A rudimentary diff tool to compare an existing VME config against the template baseline.

  The Diff UI can be accessed by right-clicking on an Event or Module and selecting 'Compare with Template' in the context menu.
- templates:
  - Add new templates for MDPP32-v2_SCP/QDC

  - Update ‘Module Init’ of MTDC-32 (bank input config + bank input threshold)
  - Update the MDPP-16/32 (Streaming) SCP/QDC templates: new event header filters for streaming mode,
- doc: Cleanup the MVLC Trigger I/O section. No more reserved units.
- Fix the MVLC VME scanbus UI looping forever if the MVLC cannot be reached.

6.4 Version 1.18.0

Important: mvme-1.18 contains the new event builder implementation which has a slightly different configuration than the old version. An attempt is made to convert the old config to the new format on load, but settings have to be manually reviewed in ‘Processing Settings -> Event Builder Settings’.

- general:
  - Multiple ui related crash and bugfixes.
  - Disabled qt debug output in release builds. Might speed things up, especially under windows.
  - Compatibility fix for very old listfiles recorded with the VMUSB.
  - Fix a deadlock when pausing the DAQ while the event builder is active.
  - Can now load and replay listfiles that contain only a MVLC CrateConfig but not a mvme VMEConfig.
- analysis:
  - Fixes for the histogram projection code.
  - Fix for a crash when projecting from a ‘1d listview’ histogram.
  - h1d: add ‘relative width’ stat to the gauss info box.
  - Fix rare crash in ‘show dependency graph’.
  - Allow larger ‘rate monitor’ capacities (2^30 up from 2^20).
  - Immediately replot after selecting a new histogram from a histo array. Forces and update of the axis scales which previously was delayed.
  - ui: Allow larger range for event builder histograms axes.
  - Improve the event builder (aka timestamp sorter) to gracefully handle the case where none of the modules in an incoming event yields a valid timestamp. This case was not handled properly and lead to excess buffering and eventual halt of the analysis processing.
- daq:
  - MVLC export: add more mvme metadata to exported mvlc crateconfig yaml files.
  - Better config checks prior to daq start. It’s now an error to have data producing readout commands in the event multicast start/stop sequences.
  - Add a stream server for streaming raw readout data over TCP or UNIX. If enabled this currently forwars the raw unmodified MVLC readout stream. Uses async send calls to efficiently stream buffers to multiple clients in parallel.

    A simple settings GUI is under ‘Workspace Settings -> Stream Server’.
- templates: Update the description of ‘sample_source’ in the MDPP32_QDC template.

6.5 Version 1.17.5

- Fix a regression with MVLC FW0046 where the ‘daq_start’ trigger_io signal did not become active when starting the DAQ.

6.6 Version 1.17.4

- internal changes to stay compatible with newer graphviz versions(13, 14).
- event_server: inline the functions to avoid duplicate definitions under linux

6.7 Version 1.17.3

- event_server: fix broken socket reads in the example (and ROOT) client under windows.

6.8 Version 1.17.2

- Merge in VMUSB fixes from Chang, Jin-Hee <ChangJ@frib.msu.edu> & Ron Fox. This fixes IRQ mask register handling in certain cases.
- Compatibility with old (2017) VMUSB listfiles restored. These were recorded before the SIS and MVLC controller existed.
- Crash fixes in some of the listfile handling code.

6.9 Version 1.17.1

- analysis:
  - Fixes and updates to histogram projections. Zooming and on-the-fly resolution reduction have been improved.
  - Fix various UI related crashes
- Add/update Dockerfiles for debian:testing, ubuntu:latest and ubuntu:rolling. Remove the Dockerfiles running only tests but not doing packaging (less files, less maintenance).
- Compile fixes for gcc-15 and clang-19 (removed pre-ansi-c style MBS code, will revive if needed)

6.10 Version 1.17.0

- config:
  - Remove special handling for MVLC periodic triggers realized via the trigger io subsystem from the UI.

    These triggers have been superseded by the new StackTimers added in MVLC FW0037. If higher frequency timers are needed they can still be implemented via the trigger io system. This change only removes them as a first class trigger condition from the UI.

    Existing event configs will now show 'MVLC Trigger I/O' as their condition instead of 'Periodic (via MVLC Trigger I/O)'.
- analysis:
  - fix crashes in 'Debug & Stats' and the plot grid widget
  - increase initial size of the 'new operator' dialogs
  - speed up and improve the 'MDPP Sample Decoder Monitor' widget (it was able to lag the whole application)
  - fix crashes in the multiplot widget (plot grid)
  - allow editing polygon vertex coordinates directly in the table
  - zero init memory segments in the internal arena under windows
  - fix convert_to_signed() yielding wrong results under windows.

- templates:
  - various fixes when saving/loading custom modules (.mvmemodule)
  - fixed updating of module meta info when loading a VME config
  - mdpp32_scp/mdpp32_qdc
    * lower readout_num_events from +2 to +1 (buffer limitation if sampling is enabled)
    * corrected number of max samples (488 for SCP, 1000 for QDC)

    This only affects newly created modules, existing modules need to be updated manually.
- trigger_io:
  - remove special hightlight for the previously reserved timer and stackstart units
  - improve the gate generator editors
  - better LUT strobe input explanation
- ui:
  - disable 'Recent Workspaces' action when DAQ is not idle
  - improve log output for errors and the 'accu test' command
  - fix broken listfile recovery under windows
  - fix silly listfile recovery progress values
  - when importing a .mvmemodule file, disable the module type dropdown
  - fix incorrect module meta info when loading a module from file
  - fix multiple issues with exporting & importing .mvmemodule files
  - update the copyright notices
- vme_script:
  - add new commands for the software accu: 'accu_add' and 'accu_write'. These were added for the PTP support in FW0046: read the current ptp seconds value, add a constant to it and write that value back to a special PTP trigger register.
  - fix evaluation of numerical expressions exceeding 32-bit unsigned
- Fix crash when starting a second mvme instance under windows (logfile related).
- add a Dockerfile for debian trixie and fix some issues with trixie

6.11 Version 1.16.3

- Fix non-multievent readouts for the VMUSB and SIS controllers. This has been broken since 1.16.1 Note: multievent splitting is currently not supported for these old controller.

6.12 Version 1.16.2

- vme_templates: lower max sample count for mdpp32_scp from 1k to 500 (buffer limitation)
- add doc/mdpp_sampling_mode.md with a short description of how to use MDPP sampling mode and decode the data

6.13 Version 1.16.1

- multi_event_splitter:
  - The splitter can now make use of multiple event header filters per module. This is needed to support mdpp16/32 sampling formats which use a new header word allowing to encode larger frame sizes.
  - Revive error counters (moduleHeaderMismatches and moduleEventSizeExceedsBuffer are counted again).
  - Display more info when inspecting buffers.
- vme_templates:
  - Add strict firmware version checks for MTDC-32 IRQ masks.
  - Add new module header filters to support sampling mode for mdpp16_qdc, mdpp16_scp, mdpp32_scp and mdpp32_qdc.
  - Improve the load/save module functionality and ui.
- analysis:
  - Add spline interpolation option to the waveform viewer
  - Rework of the waveform viewer logic. Most update and mode related bugs should be fixed now. Note: the plot legend may start to jitter at times. Still have to find a fix for this.
  - Add a deconvolution operator to be used with mdpp sampling mode.
  - Improve the multiplot widget
    - Can now drop single 1D histograms onto the widget.
    - Autosave the view on dropping an object.
- Add a tiny lmd2mvme converter tool. Works with GSI MBS TRIVA structured readout configs only.
- Add 'Tools -> Scan VME Bus' for the MVLC. Does a full scan of the VME address space and tries to identify mesytec modules.

6.14 Version 1.16.0

- Updated MVLC DAQ start sequence order to fix initial timestamp sync under very high trigger load: MVLC DAQ mode is now enabled _after_ the MCST DAQ start sequence, not before. Note: this breaks multicrate. Will be fixed in an upcoming firmware+software release.
- Event Builder:
  - Add module timestamp delta histograms and more detailed counters.
  - Improve the configuration ui.
- vme_templates:
  - Add new MVLC trigger signal mappings implemented since MVLC FW0041
  - Add the new trigger signal mappings to the TRIVA7 master init script
  - Add/update the 'Streaming and Sampling' init scripts for MDPP modules.
- analysis:
  - Improve the MDPP Sample Trace viewer.
  - Updates for the upcoming MDPP-32 firmware with Sampling support.

6.15 Version 1.15.2

- Improved Event Builder module for timestamp based event building.

6.16 Version 1.15.1

- analysis: implement auto phase correction in the EventHistory data sink.

Note: breaking change, users of 1.15.0 need to recreate the MDPP sample decoder or reconnect the EventHistory sink.

6.17 Version 1.15.0

The major user facing improvement is support for MDPP-16/32-SCP sampling mode. A new analysis data source for decoding sample traces from MDPP modules has been added. Additionally a new EventHistory data sink with supporting widgets has been added to visualize acquired sample traces.

- vme_templates:
  - Add eventHeaderFilter to the mvlc_event_stamper module. This makes the stamper work with multi-event readouts.
  - Add 'Streaming and Sampling' init script to the MDPP-16_SCP module.
- mvlc:
  - Support the new MVLC smart transaction retry mechanism in MVLC FW0039. This ensures that VME transactions are never executed more than once, even in the face of ETH packet loss.
  - Various fixes and improvements from the mesytec-mvlc driver library.
- analysis:
  - Fix a crash when opening a histogram sink without any allocated histograms.
  - Fix output limit calculations for the `BinarySumDiff` operator. Swap inverted limits on analysis load.
  - Fix some crashes in the histogram accumulation code where calculated bin numbers could become negative when extreme input limits were used.
  - MDPP_SAM: Add new data source to support MDPP sampling mode: DataSourceMdppSampleDecoder.
  - MDPP_SAM: Add new EventHistory data sink and a corresponding widget for rendering input data as waveforms.

6.18 Version 1.14.4

- analysis: fix crash when a specific parameter array index is selected in the ui

6.19 Version 1.14.3

- analysis: fix Calibration when input is a single parameter instead of an array.

6.20 Version 1.14.2

- analysis ui: Improve input/output node highlighting in the tree views.

6.21 Version 1.14.1

- analysis ui: Nest operators by category in the operator creation menu.

6.22 Version 1.14.0

- analysis
  - Change Condition Filter to be compatible with the output of the newer condition operators: Invalids and 0.0 are considered false, everything else is true. Before this change only invalids were considered false.
  - Deprecate 1d range and 2d rect filters. Use the newer condition system and optionally the Condition-Filter instead.
  - Store and restore geometries of operator editor dialogs.
  - Fix drag & drop for directories in the data display area.

6.23 Version 1.13.0

- vme_templates:
  - New templates for MDPP-16/32-SCP streaming mode
  - Update templates for MDPP-32 SCP, QDC and PADC
- analysis:
  - Add new `extended timestamp` filter template to relevant VME modules.
  - Improve runtime performance.

6.24 Version 1.12.2 - works with MVLC FW0037 or earlier

- mvlc:
  - Update FTDI driver for windows from 1.3.0.2 to 1.3.0.8.
  - Fix broken trigger setup with MVLC < FW0037.
  - Fixes and improvements to the mesytec-mvlc YAML export/import.
- vme_script
  - Fix wrong FIFO mode setting when parsing block transfer commands.
  - Do not accept invalid VME amods for block transfers (e.g. MBLT with a24).
- vme_templates: Add template for MHV-4 readout via RC-Bus connection from a parent VME module.
- analysis
  - Improve the Calibration editor.
  - Fix potential corruption in the Difference operator when input sizes are not equal (old analysis files).
- vme ui: Bugfixes in the Add/Edit Module GUI.
- Fix hang on shutdown under windows (promteheus-cpp related issue).

6.25 Version 1.11.0 - MVLC FW0037 or later only!

- Support for new MVLC firmware FW0037 features:

  - The 4 new StackTimer units can now be used to create periodic readout events without having to use `TriggerResource` and `StackStart` units.
  - New event trigger condition to activate command stacks on one of the master trigger signals.
  - The total number of MVLC command stacks has been raised from 8 to 16, so now 15 command stacks are available for readout events.
- mvlc:
  - The default Trigger I/O script is now empty as the previous default script was confusing.
  - Remember last connected ETH hostname and other settings when switching controller types.
  - Add more utility VME scripts for the MVLC related to multicrate master/slave handling.
  - Trigger I/O DSO: Fix simulation of strobed LUT outputs.
- analysis:
  - histo1d: log scale plotting and stat calculation fixes
  - ExportSink: updates to the generated code: port to python3, require cmake 3.0, suppress compiler warnings, do not force c++14 but pick the standard used by ROOT instead. Tested with python-3.11.
- Only attempt to connect to VME controllers once instead of multiple times to reduce log spam and wait time in case of errors.
- Crash fix with old mvmelst formatted data and multi event splitting.

6.26 Version 1.10.4

- analysis: Fix bug leading to ExportSink output files being truncated on listfile load.

6.27 Version 1.10.3

- mvlc: Fix `mblts` and `mbltsfifo` commands producing broken readout stacks.
- analysis:
  - correctly set the output limits of the `Previous Value` operator
  - allow toggling between scientific and full number display in the “Show Parameters” window.

6.28 Version 1.10.2

- Fix large values for listfile split size and split duration being truncated.

6.29 Version 1.10.1

- Add the prometheus+grafana docker-compose project to the binary packages (extras/metrics).

6.30 Version 1.10.0

- new: filter listfiles based on analysis conditions. Produces MVLC_USB formatted output listfiles.
- new: prometheus metrics for the MVLC readout and the analysis. Metrics are exposed on port 13802 by default.
- vme_templates: Updated VMMR template. Merge vmmr and vmmr_1ns variants.
- MVLC Trigger IO: fixes for the LUT simulation code (FW0036 related).

- analysis: conditions now output 0.0 if false instead of invalid_param()/NaN. Allows to track condition true/false and total counts in a single 1d histogram.
- bugfixes: empty listfile filename display, uninitialized data in multi_event_splitter

6.31 Version 1.9.2

- mvlc: abort daq start sequence on error in trigger io init.
- histo2d: enable slicing only for 2d histos, not for 1d array views

6.32 Version 1.9.1

- vme_script: Add new `mvme_require_version` command: software side check of minimum mvme version required to run the script.
- MVLC Trigger IO
  - Add `mvme_require_version 1.9.1` at the top of the script. Older versions cannot parse the updated format (`mvlc_stack_begin/end` blocks) correctly. They will now error out due to the unknown `mvme_require_version` command.
  - Trigger IO script exec fixes: the standard `run_script` function is now used to run the script.
- mvlc: Fix bogus value of `moduleData.hasDynamic` in mvlc_readout_parser.

6.33 Version 1.9.0

- Major mesytec-mvlc update for MVLC FW0036 and later:
  - MVLC now supports the FIFO flag for block reads. The `mvlc_set_address_inc_mode` command has been removed. New VME Script commands have been added instead (see below).
  - Larger command stack uploads are now possible. The stack is uploaded in parts. The max size of each part depends on the transport being used: ETH is limited by the max UDP packet size, USB by MVLC internal buffer sizes.
- vme_script: Implement new commands for 2eSST fifo and memory block reads: `2esstfifo`, `2esstsfifo`, `2esstmem`, `2esstsmem`.
- vme_script: Better error handling and log output for MVLC inline stacks.
- MVLC Trigger IO: unit initialization is now wrapped in `mvlc_stack_begin/end` blocks to get atomic init behavior. This means executing the Trigger IO script won't interfere with the DSO or active readouts that are using the Trigger IO system. This change also speeds up execution of the Trigger IO init script. Note: the Trigger IO script has to be regenerated via the GUI for this change to take effect!
- vme_script: Better error handling and log output for MVLC inline stacks (`mvlc_stack_begin/end`).
- MVLC Trigger IO: unit initialization is now wrapped in `mvlc_stack_begin/end` blocks to get atomic init behavior. This means executing the Trigger IO script won't interfere with the DSO or active readouts that are using the Trigger IO system. This change also speeds up execution of the Trigger IO init script. Note: the Trigger IO script has to be regenerated via the GUI for this change to take effect!
- MVLC DSO
  - Fix DSO readout returning early before having received a trigger.
  - DSO readout does not use an internal timeout anymore. This means pulses with very long interval times can now be reliably sampled.
  - Rework the UI: can now enter measurement duration instead of post-trigger time. Max measurement duration is limited to 65500 ns by the MVLC.
  - Plot: Fix issue where the trigger edge was not aligned with the 0 coordinate.

- Implement 2D Histogram slicing. Works for X and Y and uses the currently visible area. The slices are opened in a new 1D histogram window.
- vme_templates: Add hardware id checks for mesytec modules similar to MDPP-16 firmware type checks.
- Merge PR from wvonseeg to make the sparse ExportSink python code work with python-3.10.
- Use FIFO block reads in VME Debug Widget.

6.34 Version 1.8.2

- Better fix for the EventServer reconnect race: clients are not disconnected anymore when loading listfiles or switching VME controllers. Also remove the sleep from mvme_jsonrpc_replay.py
- Readd the `mvme.sh` script to directly start mvme with the correct env variables set. Note: mvme.sh sources the `initMVME` script to setup the environment.

6.35 Version 1.8.1

- mvme_root_client: Abort if the DAQ run/replay is already in progress whenn connecting. Can be disabled by passing "run-in-progress-is-ok" on the command line.
- mvme_jsonrpc_replay.py: Sleep between loading a listfile and starting the replay. This works around a race where the mvme_root_client was not connected yet but the replay was already running.

6.36 Version 1.8.0

- [mesytec-mvlc]
  - mvlc_eth: Do not send a frame to the data pipe when connecting. This way ongoing readouts won't be redirected when a second process connects to the MVLC.
  - eth and usb: Do not reset the stack trigger registers when connecting. It made reading back the last trigger configuration impossible. Now only the DAQ mode register is written when requested via disableTriggersOnConnect().
  - New SplitZipReader to replay from split listfiles stored across multiple zip archives. To consumers it looks like the data came from a single file.

    Replaying all parts from a split listfile is done in the 'listfile browser (ctrl+4)' by checking 'replay all parts' before opening the first part that should be replayed.
- [vme_templates] Update integration parameter ranges for MDPP-16/32-QDC
- When a listfile is opened do not try to auto connect to the VME controller.
- Updates to the JSON-RPC listfile handling methods: 'loadAnalyis', 'keepHistoContents' and 'replayAllParts' are now explicit parameters to the respective methods.
- The Qt Assistant binary is now again contained with the linux package.

6.37 Version 1.7.2-1

- Use current workspace directory as the starting point for MVLC CrateConfig exports.

6.38 Version 1.7.2

- Fix mvme_root_client compilation issue against root 6.22.06

- New JSON-RPC remote control methods for loading analysis configs and opening listfiles.

  extras/mvme_jsonrpc_replay.py shows how to replay from a list of input listmode files while accumulating into the same analysis.
- Close projection plots when the parent h2d plot is closed.
- Better error logging in multi_event_splitter.
- Fix 'read_to_accu' missing the 'late' flag when exporting a VMEConfig to mesytec-mvlc CrateConfig.

6.39 Version 1.7.1

- [analysis]
  - Show module/group names in readout parser debug.
  - Improve histo stats widget table formatting and show the RMS value of each column.
  - Fix 1D histo statistics not following the zoom under Windows.
  - Clear 2D histograms when their subrange was modified.
  - Replace the histo resolution slider with a combo box.
  - Interval condition can now exclude/ignore specific intervals from affecting the conditions result.
  - Many improvements to graphical interval/polygon condition editing.
  - Dependency Graph View now starts editing data sources on double-click.
  - Fix a source of frequent crashes when modifying the analysis (periodic histo counter updates).
- [vme_templates]
  - Slightly improve the bus_time filters for VMMR modules
  - Calibrate mesytec vme module timestamps to µs.
- [mvlc]
  - DSO plot and logic updates (recommended to use MVLC Firmware FW0031 or later).
  - Start/stop the DSO using a single stack transaction instead of multiple individual commands. Fixes issues when the DSO is running while the Triger IO script is being written to the MVLC.
- [doc]
  - Update to the "Manual ARP setup" section for the MVLC.
- [packaging]
  - Add missing graphviz dependencies to the linux packages.

6.40 Version 1.7.0

- [vme_script]
  - Breaking change: spaces are not allowed in variables names anymore. The UI now also rejects attempts to uses spaces in variable names.
  - Can now place complete vme_script command lines in variables, e.g.:

    ```
    set readout_cmd "mbltfifo a32 0x0100 65535"
    ${readout_cmd}
    ```

    The second line above is now correctly parsed as a **mbltfifo** command.

    Variable references can also be used on the right-hand side:

```
set my_addr 0x1234
set readout_cmd "mbltfifo a32 ${my_addr} 65535"
${readout_cmd} # Will be expanded to "mbltfifo a32 0x1234 65535"
```

This process is not recursive.

- [ui]
  - Save/restore node expansion state of the VME Config tree.
  - VME script editor: add new “Run (ignore errors)” action. Useful for temporarily ignoring errors from VME scripts and running the script to the end.
  - Remove **BerrMarker** and **EoMMarker** text from buffer debut output. These values were only added for the VM-USB and are misleading when looking at MVLC buffers.
  - Show RMS value in 1d plot grid tiles.
- [mvme_root_client]

  Breaking change: improve handling of TTrees split across multiple files.

  The *TTree::SetMaxTreeSize()* can now be specified on the command line when recording: *–root-max-tree-size=<maxBytesPerFile>*. The default value is set to the ROOT default of 100000000000LL.

  Replay mode is now enabled via *–replay*. In this mode mvme_root_client now accepts a list of filenames instead of a single file. The filenames are used to create a TChain object which becomes the source for the replay data.

6.41 Version 1.6.3

- Another mvme_root_client compilation fix.

6.42 Version 1.6.2

- vme_templates: Add support for the MVHV-4 VME High Voltage Bias Supply
- Fix mvme_root_client compilation issue: do not set c++ standard in the Makefile.
- Packaging: do not package libz.so anymore.

6.43 Version 1.6.1

- [gui]
  - New feature: recover corrupted listfiles.

    If a listfile ZIP archive is corrupted due to a crash/power outage the UI now offers a way to attempt to recover the data when opening the corrupted archive.

    Recovery works by searching for the first local file header in the zip archive and attempting to unpack the following data. The recovery process also works for listfile archives containing LZ4 compressed readout data.
  - New feature: can now save/load VME event configs to/from file

    Saving is done via the events context menu entry “Save Event to file”.

    To load an event and add it as a new event use the top-level “Events” node context menu and select “Add Event from file”.

    Saved events can also be merged into existing events: Use “Merge with Event from file” from the destination events context menu. This will add all modules from the source event to the target event. Non-system and non-mesytec VME Script variables defined in the source event will be added to the destination event. Existing variables are overwritten.

  - add “Save Script” to the VME tree context menu
  - Do not allow deleting the MVLC Trigger/IO script
  - Fix file saving logic across the GUI. The logic was flawed and could lead to files being overwritten.
- [vme/readout]
  - Return earlier if errors occur during the DAQ start sequence. Return points are: after global start scripts, after VME module init scripts and after event start scripts.
  - Update module template for the MDLL: init script udpates and analaysis filter and naming fixes.
- [analysis]
  - Implement on-the-fly histogram creation when attempting to graphically edit a condition that does not have a matching histogram.
  - Increase initial size of plot windows so that all toolbar buttons are visible (hopefully).
  - Crash fix in the ExportSink operator UI.
- [doc] Changelog was missing from PDF file in windows builds.
- [mesytec-mvlc]
  - Add a command line vme-scan-bus tool. This is in its early stages and needs more polish.
- Updated build system for linux binaries: Debian Stretch with glibc-2.24 is used with custom built gcc-10.4 and Qt-5.15.8 libraries. Deployment is done using ‘linuxdeployqt’.

  The binaries should now run on a wider range of systems (all using glibc>=2.24) while still containing a modern version of Qt. A detailed list of glibc versions used in common distributions can be found here: https://repology.org/project/glibc/versions

6.44 Version 1.6.0

- [analysis]
  - Add plot grid views: configurable window for showing multiple plots in a grid layout.
  - Reworked the 1d histogram statistics window: it now uses a table to display the data and the statistics are synchronized to the zoomed area of the histogram widget.
  - Add multi_event_splitter counter output to the analysis info widget.
- [vme_script]
  - VME amod parsing is not case-sensitive anymore. By default the user/non-privileged VME amods are used but numeric amod arguments are now also accepted to allow full control of the amod.
  - The effective vme amod value is now logged in the output of script commands.
  - read and readabs now accept “late” in addition to “slow”
  - Improve the script level accumulator commands to make them similar to the MVLC accu stack commands.
- Fix VME Debug Widget block reads not working anymore (wrong VME amod was used)
- mvme now requires c++17!

6.45 Version 1.5.0

- [analysis]
  - Implemented a *condition system* and 1d-interval, 2d-polygon and expression (exprtk) conditions.
  - Added a new *dependency viewer* to visualize data processing and active conditions.

- [vme/readout]
  - Revert a change from 1.4.9 where lowercase amod specifiers used the *privileged* value, while uppercase specifiers where converted to the *user* value. Now by default the user amods are used but numeric amod arguments can be given to single and block read commands for full control over the amod.
  - Add the raw VME amod value to the log output of vme script commands.
  - Add new commands for the fast 2eSST VME transfer modes: *2esst* and the word swapped version *2essts*.
  - Add new module templates for mesytec MDLL, mesytec MCPD-8_MPSD and the CAEN v1742
  - Add a new software accumulator and related functions: *accu_set*, *accu_mask_rotate*, *accu_test*
  - Update MDPP-16/32 scripts to check if the correct firmware revision is loaded.
  - Listfile filenames can now be specified using format strings (fmt library).

6.46 Version 1.4.9.5

- Bugfix release: listfile archives where missing the analysis config and log file.

6.47 Version 1.4.9.4

- Fix data rate monitoring and display when using MVLC_USB (read timeout issue)

6.48 Version 1.4.9.3

- Improved listfile filename generation: an fmt format string can now be used to specify the output filename. Currently the run number and the timestamp are passed as arguments when generating the output filename.
- Add untested templates for the CAEN v775 TDC module.

6.49 Version 1.4.9.2

- [analysis] Suppress completely empty events when using the SIS3153 controller.

6.50 Version 1.4.9

- [analysis]
  - Add a new MultiHitExtractor data source allowing to extract multiple hits per address.
  - Add 'Generate Histograms' context menu action to data sources and operators to quickly generate histograms for selected objects.
  - Raise maximum number of data sources and operators per VME event context from 256 to 65536.
  - Improve histo1d stats output.
- New feature: listfile splitting (MVLC only)

  When recording readout data the output listfile can now be split either based on file size or elapsed time. Each partial listfile ZIP archive is in itself a complete, valid mvme listfile and includes the VME config, analysis config and logged messages.

  Replaying from split listfiles currently has to be done manually for each part. Using the 'keep histo contents' in mvme allows to accumulate data from multiple (partial) listfiles into the same analysis.
- Listfile output directory can now be selected in the Workspace Settings GUI.
- Add new optional suffix part to listfile filename generation.

- New feature: VME modules can now be saved to and loaded from JSON files. This can be used to create custom VME modules without having to use the mvme VME template system.
- DAQ run number is now incremented on MVLC readout stop to represent the *next* run number.
- Show the original incoming data rate in the analysis window when replaying from listfile.
- VME Config: allow moving modules between VME Events via drag&drop.
- [mvlc]
  - Revert the MVLC readout parser simplification done in 1.4.8

    The parser now allows prefix, dynamic and suffix parts again. The parser data callback remains unchanged, passing the parsed data as a single pointer + size.
  - Fix command timeout errors with older USB2 chipsets.
  - Fix USB2 connection issues by retrying opening the device.
  - Periodically add stack error information received on the command pipe to recorded listfile data. Uses a new system_event::StackErrors section to store stack error locations, flags and counts.
  - Fix 'VME Script -> Run' in the MVLC Debug GUI
- [vme_templates]
  - Add 'stop_acquisition', 'reset_fifo' and 'readout_reset' commands to mesytec module reset scripts. Fixes an issue where the modules could signal a VME IRQ during the init sequence but before the DAQ was properly started with the multicast start sequence.
  - Improve Triva7 VME module templates.
- Improved VME Script Execution: log messages from commands are now immediately visible. Progress dialog shows progress based on number of commands.
- Fix wrong VME -> analysis module assignments when disabled VME modules are present in the config.
- New ZMQ publisher listfile output (MVLC only).

  Sends readout buffers through a ZMQ PUB socket. Based on code from GANIL.

6.51 Version 1.4.8.2

Raise MVLC command timeout (request/response) from 500ms to 1000ms.

6.52 Version 1.4.8.1

Make mvme build against qwt versions older than 6.2.0 again.

6.53 Version 1.4.8

- [mvlc]
  - Simplify the readout parser: modules readout data may now consist of either a dynamic or a fixed part instead of prefix, dynamic and suffix parts. This allows for a simpler callback interface for the parser.

    The previous, more complex structure can be recrated by adding multiple modules to the VME config, each performing either fixed size reads or a block transfer.
  - Add support for new features in firmware FW0021:
    - New vme_script commands to work with the MVLC stack accumulator.

      See *mvlc_signal_accu* and the commands following it.

    * Add ability to define custom and inline MVLC stacks in VME scripts.

      See *mvlc_stack_begin/end* and *mvlc_custom_begin/end*.

    * The readout parser now knows about the accumulator and emulated accumulator block reads.

    * Support CR/CSR addressing modes.

- [analysis]
  - Improvements to the EventBuilder module. This version does work with non-mesytec modules being present in an event and allows to exclude modules from the timestamp matching algorithm.
  - Improve Histo1D 'Print Stats' output
  - Crash fix when loading a session file with unconnected histograms.
- [vme_templates]

  Add module templates for the GSI Triva 7 trigger module.
- [build]
  - Upgrade to Qt 5.15.2 and Qwt 6.2.0

6.54 Version 1.4.7

- Reopen to the last used VME config when closing a listfile.
- When saving VME/analysis config files suggest a filename based on the workspace directory.
- Add a `--offline` option to mvme which disables any connection attempts to the VME controller. Useful for replay-only sessions.
- Improve MVLC stack error reporting.
- Decrease number of readout buffers in-flight to reduce latency when stopping a run/replay.
- Various bug and crash fixes.
- [analysis]
  - Add an EventBuilder module to the analysis processing chain.
  - Fix analysis stats display when using more than 12 modules in an event.
  - Prepend the module name to analysis objects generated when adding the default filters.
- [vme_script]
  - Add support for MVLC stacks containing custom data (mvlc_custom_begin).
  - Add support for new MVLC commands in Firmware 0x0020.
- [packaging]
  - make installed files and directories group and world readable.
  - re-add the mvme.sh startup shell script to the bin/ directory.

6.55 Version 1.4.6

- [mvlc]
  - Improve immediate MVLC/VME command latency when using the DSO.
  - Trigger/IO updates
- [analysis]
  - Fix crash in the ExportSink ("File Export") operator.

  - Add CSV output option to the ExportSink.
- [vme] Change default vme amods from the privileged to the user variants.

6.56 Version 1.4.5

- Create an empty analysis when opening a workspace and no existing analysis could be loaded from the workspace. This fixes an issue where analysis objects from the previously opened workspace still existed after changing the workspace.

6.57 Version 1.4.4

- [vme_script] Behavior changes:
  - Do not accept octal values anymore. '010' was parsed as 8 decimal while '080' - which is an invalid octal literal - was parsed as a floating point value and interepreted as 8 decimal.
  - Floating point parsing is now only applied if the literal contains a '.'.
- [analysis] Module hit counts in the top left tree now display the count and rate of non-empty readout data from the module. Previously they showed all hits and where thus equal to parent event rate unless multi-event splitting was in effect.
- [vmusb] Fix readout being broken.
- Do not auto create non-existing workspace directories on startup. Instead ask the user to open an existing workspace or create a new one.
- Do not set default vme and analysis config file names when creating a workspace or no previously loaded files exist in the current workspace. This makes the user have to pick a name when saving each of the files and should make it less likely to accidentially overwrite existing configs.

6.58 Version 1.4.3

- [mvlc] Add support for the oscilloscope built into the MVLC since firmware FW0018.
- [analysis]
  - Remove the vme module assignment dialog. Instead show data sources belonging to unassigned modules in a hierarchy in the top left tree of the analysis window. Data sources can be dragged from there onto known modules to assign them.
  - Add static variables to the Expression Operator. These variables exist per operator instance and persist their values throughout a DAQ or replay run.
  - Add a ScalerOverflow operator which outputs a contiguous increasing value given an input value that overflows. This can be used to handle data like module timestamps which wrap after a certain time.
  - The RateMonitor can now display a plain value on the x axis instead of time values. Useful when plotting timestamp or counter values.
  - Added division to the binary equation operator.
- Better handling of vme/analysis config files when opening listfiles to reduce the number of instances where the vme and analysis configs diverge.
- Add print statements to the module reset vme template scripts.

6.59 Version 1.4.2

- [vme_templates]
  - Wait 500ms instead of 50ms in the reset scripts of MDPP-32_PADC/QDC

  - Update MDPP-32_QDC calibration to 16 bits
  - Do not set vme mcst address in the mvlc_timestamper `VME Interface Settings` script.
- [analysis]
  - Improve Rate Monitor draw performance
  - Make Rate Estimation work in projections of 2D histograms
  - Analysis session data parsing fixes

6.60 Version 1.4.1

- [vme_templates] Fix gain calculation in MDPP16-SCP `Frontend Settings` script.

6.61 Version 1.4.0

- [mvlc] Trigger/IO updates for firmware FW0017
  - Replace IRQ, SoftTrigger and SlaveTrigger units with the new TriggerResource units
  - Support the IRQ input, L1.LUT5/6 and L2.LUT2 units
  - Support Frequency Counter Mode for Counter units
  - Basic support for the Digital Storage Oscilloscpe built into the Trigger/IO system.
  - Crash fixes when parsing Trigger/IO scripts
- [mvlc] Updates to the DAQ Start and Stop sequence
- [vme_config] The order of Modules within an Event can now be changed via drag and drop.
- [analysis]
  - Performance and visual updates for the RateMonitors
  - Display directory hierarchy in Histogram and RateMonitor window titles
- [vme_templates]
  - Add the new MDPP-16/32 channel based IRQ signalling.
  - Add the 'stop acq' sequence to all module 'VME Interface Settings' scripts. This makes modules not produce data/triggers directly after being intialized but only after the 'Event DAQ Start' script has been executed.

6.62 Version 1.3.0

- [mvlc] Support MVLC ethernet readout throttling
  - Throttling is done by sending 'delay' commands to the MVLC which then adds small gaps between outgoing ethernet packets thus effectively limiting the data rate.
  - The MVLC will block the VME readout side if it cannot send out enough ethernet packets either due to reaching the maximum bandwidth or due to throttling. This behaves in the same way as USB readouts when the software side cannot keep up with the USB data rate.
  - The delay value is currently calculated based on the usage level of the readout socket receive buffer. Throttling starts at 50% buffer usage level and increases exponentially from there.

  This method of ethernet throttling is effective when the receiving PC cannot handle the incoming data rate, e.g. because it cannot compress the listfile fast enough. Instead of bursts of packet loss which can lead to losing big chunks of readout data the readout itself is slowed down, effectively limiting the trigger rate. The implementation does not compensate for packet loss caused by network switches or other network equipment.

Throttling and socket buffer statistics are shown at the bottom of the main window, below the VME config tree.

- [mvlc] readout_parser fixes: - disabled VME modules where confusing the readout parser - stale data from the previous DAQ run was remaining in the buffers
- [mvlc] Updates and fixes for the trigger IO editor.
- [mvlc] When creating a new VME config a new default trigger IO setup is loaded. The setup provides 5 trigger inputs, 5 gated trigger outputs, a free trigger output and daq_start, stack_busy and readout_busy signals on the NIMs. The setup is intended to be used with two events: one for the readout and one periodic event for counter readout.
- [analysis] Allow directories, copy/paste and drag/drop for raw histograms (bottom-left tree view). When generating default filters and histograms for a module the histograms are also placed in a directory instead of being attached to special module nodes. When loading analysis files from previous versions the missing directories are automatically created.
- [analysis] Updated the multievent_splitter to work with modules which do not contain the length of the following event data in their header word. Instead the event length is determined by repeatedly trying the module header filter until it matches the next header or the end of the readout data is reached.
- [analysis] Updates and fixes for the RateMonitors
- [vme_templates]
  - Updates to the mesytec VMMR template.
  - Updates to the CAEN v785 template.
  - Add templates for the CAEN V1190A Multihit TDC.
- [vme_script] add 'readabs' command
- [core] Improve the high level stopDAQ logic and resulting state updates. This in turn makes stopping the DAQ via JSON-RPC work reliably.

6.63 Version 1.2.1

- [analyis] Fix two crashes when using the ExportSink

6.64 Version 1.2.0

- [mvlc] Update mesytec-mvlc lib to work around an issue were MVLC_ETH was not able to connect under Windows 10 Build 2004.

  This issue has also been fixed in MVLC Firmware FW0008.
- [vme_templates] Add VME and analysis templates for the mesytec MDPP-16_CSI, MDPP-16_PADC and MDPP-32_PADC module variants.
- [vme_templates] Add templates for the MDI-2 starting from firmware FW0300.
- [vme_templates] Add files for the CAEN V830 latching scaler.
- [vme_script] Add a new 'mblts' (swapped block read) command for the MVLC which swaps the two 32-bit words received from MBLT64 block reads.

  This was added to the MVLC to support the CAEN V830 and possibly other modules which have the data words swapped compared to the mesytec modules.
- [analysis] Generate histograms and calibrations for ListfilterExtractors found in module analysis template files. This was added for the V830 which is the first template file to use ListfilterExtractors.
- [core] Add facilities for storing the log messages generated by mvme to disk:

  - All messages generated during DAQ runs (from 'DAQ start' to 'DAQ stop') are written to a file in the workspace 'run_logs/' directory.

    The maximum number of files kept is limited to 50. On exceeding the limit the oldest file is removed. Filenames are based on the current date and time.

    This feature was added because previously only the logs from *successful* DAQ starts where kept on disk (inside the listfile ZIP archive generated by mvme). Log contents from aborted starts had to be manually copied from the log window.

  - All messages generated by mvme are written to 'logs/mvme.log'. On opening a workspace an existing logfile is moved to 'logs/last_mvme.log' and a new logfile is created.

    These files contain all messages generated by mvme, even those produced while no DAQ run was active.

- [event_server] Use relative path for dlopen() in mvme_root_client. Attempts to fix an issue where the analysis.so could not be loaded on some machines.

6.65 Version 1.1.0

- MVLC support is now implemented using the mesytec-mvlc library. Listfiles created by this version of mvme can be replayed using the library (e.g. the mini-daq-replay program).

6.66 Version 1.0.1

- [vme_templates] Add new VMMR_Monitor module intended for reading out MMR monitor data (power, temperature, errors).
- [vme_templates] Module templates can now specify a set of default variables to create when the module is instantiated.
- [vme_templates] Allow using ListFilterExtractors in module analysis templates in addition to MultiWordDataFilters.
- [mvlc] Update trigger io editor connection bars to reflect changes to the firmware.
- [mvlc] Fix potential data loss under very high data rates.
- [doc] Updates to the Installation section.

6.67 Version 1.0.0

- Add ability to run the data acquisition for a limited amount of time before automatically stopping the run.
- Add VME templates for the MDPP-32 (SCP and QDC variants).
- [vme_script] Drop support for the 'counted block read` commands. They are complex, rarely used and the MVLC does not currently support them. As long as a VME module supports either reading until BERR or can be read out using a fixed amount of (M)BLT cycles there is no need for these special commands.
- [vme_script] VME scripts now support floating point values, variables and embedded mathematical expressions.
- [vme_config] Updates to the mesytec module templates and the internal config logic to make use of the new VME script variables.

  These changes make IRQ and MCST handling with multiple modules and events much simpler. When using only mesytec modules no manual editing of scripts is required anymore.

  When loading a config file from a previous mvme version all module and event scripts will be updated to make use of the standard set of variables added to each VME event.
- Improve UI responsiveness with the MVLC at low data rates.
- Multiple MVLC fixes and improvements.

- Various bugfixes and UI improvements
  - VME Script error messages are now highlighted in red in the log view.
  - Speed up creating and updating the analysis tree views. This is especially noticeable when using many modules or many VME events.
- Upgrade Qt to version 5.14.1 on the build servers.
- Do not ship libstdc++ with the linux binary package anymore. It caused issues in combination with setting LD_LIBRARY_PATH as is done in the initMVME shell script.

6.68 Version 0.9.6

- Improved support for the MVLC. Among others VME Scripts can now be directly executed during a DAQ run without having to pause and resume the DAQ.
- New UI for setting up the MVLC Trigger and I/O logic system.
- Updates to the auto-matching of vme and analysis objects on config load.
- Improved the mvlc_root_client
- Documentation updates
- Improved VME module templates
- Various stability and bugfixes

6.69 Version 0.9.5.5

- This is the first version with support for the upcoming mesytec MVLC VME controller.
- Added the EventServer component which allows to transmit extracted readout data over a TCP connection.
- Added a client for the EventServer protocol which generates and loads ROOT classes, fills instances of the generated classes with incoming readout data and writes these objects out to a ROOT file. Additionally user defined callbacks are invoked to perform further analysis on the data.

6.70 Version 0.9.5.4

- Log values written to the VMUSB ActionRegister when starting / stopping the DAQ

6.71 Version 0.9.5.3

- Allow access to all VMUSB registers via vme_script commands `vmusb_write_reg` and `vmusb_read_reg`
- Fix a crash in Histo1DWidget when resolution reduction factor was set to 0

6.72 Version 0.9.5.2

- Fix a race condition at DAQ/replay startup time
- Remove old config autosave files after successfully loading a different config. This fixes an issue where apparently wrong autosave contents where restored.
- Rewrite the analysis session system to not depend on HDF5 anymore. This was done to avoid potential issues related to HDF5 and multithreading.

Note

Session files created by previous versions cannot be loaded anymore. They have to be recreated by replaying from the original readout data.

6.73 Version 0.9.5.1

This release fixes issues with the code generated by the analysis export operator.

Specifically the generated CMakeLists.txt file was not able to find the ROOT package under Ubuntu-14.04 using the recommended way (probably other versions and other debian-based distributions where affected aswell). A workaround has been implemented.

Also c++11 support is now properly enabled when using CMake versions older than 3.0.0.

6.74 Version 0.9.5

Note

Analysis files created by this version can not be opened by prior versions because the file format has changed.

This version contains major enhancements to the analysis user interface and handling of analysis objects.

- It is now possible to export an object selection to a library file and import objects from library files.
- Directory objects have been added which, in addition to the previously existing userlevels, allow to further structure an analysis.

  Directories can contain operators, data sinks (histograms, rate monitors, etc.) and other directories.
- Objects can now be moved between userlevels and directories using drag and drop.
- A copy/paste mechanism has been implemented which allows creating a copy of a selection of objects.

  If internally connected objects are copied and then pasted the connections will be restored on the copies.

Other fixes and changes:

- New feature: dynamic resolution reduction for 1D and 2D histograms.

  Axis display resolutions can now be adjusted via sliders in the user interface without having to change the physical resolution of the underlying histogram.
- Improved hostname lookups for the SIS3153 VME controller under Windows. The result is now up-to-date without requiring a restart of mvme.
- Add libpng to the linux binary package. This fixes a shared library version conflict under Ubuntu 18.04.
- SIS3153: OUT2 is now active during execution of the main readout stack. Unchanged: OUT1 is active while in autonomous DAQ mode.
- The Rate Monitor can now take multiple inputs, each of which can be an array or a single parameter.

  Also implemented a combined view showing all rates of a Rate Monitor in a single plot.
- Add new VM-USB specific vme script commands: `vmusb_write_reg` and `vmusb_read_reg` which allow setting up the VM-USB NIM outputs, the internal scalers and delay and gate generators.

  Refer to the VM-USB manual for details about these registers.

6.75 Version 0.9.4.1

- Fix expression operator GUI not properly loading indexed parameter connections
- Split Histo1D info box into global and gauss specific statistics. Fixes to gauss related calculations.

6.76 Version 0.9.4

- New: *Analysis Expression Operator*

  This is an operator that allows user-defined scripts to be executed for each readout event. Internally exprtk is used to compile and evaluate expressions.

- New: *Analysis Export Sink*

  Allows exporting of analysis parameter arrays to binary files. Full and sparse data export formats and optional zlib compression are available.

  Source code showing how to read and process the exported data and generate ROOT histograms can be generated.

- New: *Analysis Rate Monitor*

  Allows to monitor and plot analysis data flow rates and rates calculated from successive counter values (e.g. timestamp differences).

- Moved the MultiEvent Processing option and the MultiEvent Module Header Filters from the VME side to the analysis side. This is more logical and allows changing the option when doing a replay.
- General fixes and improvements to the SIS3153 readout code.
- New: JSON-RPC interface using TCP as the transport mechanism.

  Allows to start/stop DAQ runs and to request status information.

6.77 Version 0.9.3

- Support for the Struck SIS3153 VME Controller using an ethernet connection
- Analysis:
  - Performance improvments
  - Better statistics
  - Can now single step through events to ease debugging
  - Add additional analysis aggregate operations: min, max, mean, sigma in x and y
  - Save/load of complete analysis sessions: Histogram contents are saved to disk and can be loaded at a later time. No new replay of the data is neccessary.
  - New: rate monitoring using rates generated from readout data or flow rates through the analysis.
- Improved mesytec vme module templates. Also added templates for the new VMMR module.
- More options on how the output listfile names are generated.
- Various bugfixes and improvements

6.78 Version 0.9.2

- New experimental feature: multi event readout support to achieve higher data rates.
- DataFilter (Extractor) behaviour change: Extraction masks do not need to be consecutive anymore. Instead a “bit gather” step is performed to group the extracted bits together and the end of the filter step.

- UI: Keep/Clear histo data on new run is now settable via radio buttons.
- VMUSB: Activate output NIM O2 while DAQ mode is active. Use the top yellow LED to signal “USB InFIFO Full”.
- Analysis performance improvements.
- Major updates to the VME templates for mesytec modules.

6.79 Version 0.9.1

- Record a timetick every second. Timeticks are stored as sections in the listfile and are passed to the analyis during DAQ and replay.
- Add option to keep histo data across runs/replays
- Fixes to histograms with axis unit values >= 2^31
- Always use ZIP format for listfiles

INDEX

A

Analysis, 43
Analysis Conditions, 66
Analysis Dependency Graph, 52
Analysis Processing Chain, 66

C

Changelog, 81
Changes, 81
Condition System, 50
Counter, 38

D

DAQ, 17
DAQ Controls, 17
Data Export, 64, 74
Dependency Viewer, 52

E

Event Builder, 67
EventBuilder, 67
EventServer, 74
ExportSink, 64
Expression Condition, 66

G

Grafana, 76

I

Installation, 5
Interval Condition, 66
Introduction, 1
IRQ, 33, 34
IRQ Input Units, 33
IRQ Utility Units, 34

J

JSON-RPC, 70

L

Listfile Filtering, 51
Lookup Table, 34
LUT, 34
LVDS, 33

M

MasterTrigger, 38
Metrics, 76
Multi Event, 66
Multi Event Splitter, 66
MultiEventSplitter, 66
MVLC Trigger/IO, 30
MVLC VME Script Commands, 26
mvlc_trigger_io, 30
mvlc_trigger_io_Counter, 38
mvlc_trigger_io_IRQ, 33
mvlc_trigger_io_IRQ_util, 34
mvlc_trigger_io_LUT, 34
mvlc_trigger_io_LVDS, 33
mvlc_trigger_io_MasterTrigger, 38
mvlc_trigger_io_NIM, 33
mvlc_trigger_io_SlaveTrigger, 34
mvlc_trigger_io_SoftTrigger, 34
mvlc_trigger_io_StackBusy, 34
mvlc_trigger_io_StackStart, 38
mvlc_trigger_io_Sysclk, 34
mvlc_trigger_io_Timer, 33
mvlc_trigger_io_TriggerResource, 34

N

NIM, 33

P

Polygon Condition, 66
Prometheus, 76

Q

Quickstart, 9

R

Readout Data Processing, 66
Reference, 16
Replay, 17
ROOT, 64, 74
ROOT export, 64, 74

S

Slave Trigger, 34
SlaveTrigger, 34
Soft Trigger, 34
SoftTrigger, 34

StackBusy, 34
StackStart, 38
Sysclk, 34

T

Timer, 33
Trigger Resource, 34
TTL, 33

V

VME Config, 18
VME IRQ, 33, 34
VME Script, 22
VME Script Accumulator Commands, 25
VME Tree, 18
VMEConfig, 18
VMEScript, 22