muonic Documentation

Release 1.0.1

robert.franke,achim.stoessl

May 24, 2013

CONTENTS

CHAPTER

ONE

DOCUMENTATION FOR RELEASE 1.0.1, DOCUMENTATION BUILT ON MAY 22, 2013

Contents:

1.1 muonic - a python gui for QNET experiments

The muonic project provides an interface to communicate with QNet DAQ cards and to perform simple analysis of QNet data. Its goal is to ensure easy and stable access to the QNet cards and visualize some of the features of the cards. It is meant to be used in school projects, so it should be easy to use even by people who do not have lots of programming or LINUX backround. Automated data taking can be used to ensure no valuable data is lost.

1.1.1 Licence and terms of agreement

Muonic is ditributet under the terms of GPL (Gnu Public License). With the use of the software you accept the condidions of the GPL. This means also that the authors can not be made responsible for any damage of any kind to hard- or software.

1.2 muonic setup and installation

Muonic consists of two main parts: 1. the python package *muonic* 2. a python executable

1.2.1 prerequesitories

muonic needs the following packages to be installed (list may not be complete!)

- python-scipy
- python-matplotlib
- python-numpy
- python-qt4
- python-serial

1.2.2 installation with the setup.py script

Run the following command in the muonic main directory

python setup.py install

This will put the muonic package into your python site-packages directory and alsot the exectuables *muonic* and *which_tty_daq* to your user/bin directory.

The use of python-virtualenv is recommended.

1.2.3 installing muonic without the setup script

You just need the script *./bin/muonic* to the upper directory and rename it to *muonic.py*. You can do this by typing

mv bin/muonic muonic.py

while being in the muonic main directory.

Afterwards you have to create the folder *muonic_data* in your home directory.

mkdir ~/muonic_data

1.3 How to use muonic

1.3.1 start muonic

If you have setup muonic via the provided setup.py script or if you hav put the package somewhere in your PYTHONPATH, simple call from the terminal

```
muonic [OPTIONS] xy
```

where `xy` are two characters which you can choose freely. You will find this two letters occuring in automatically generated files, so that you can identify them.

For help you can call

```
muonic --help
```

[OPTIONS]

-s
use the simulation mode of muonic. This should only used for testing and developing the so

-d
debug mode. Use it to generate more log messages on the console.

-t sec
change the timewindow for the calculation of the rates. If you expect very low rates, you
default is 5 seconds.

-p
automatically write a file with pulsetimes in a non hexadecimal representation

-n
supress any status messages in the output raw data file, might be useful if you want use mu

1.3.2 Saving files with muonic

All files which are saved by muonic are ASCII files. The filenames are as follows:

Warning: currently all files are saved under $HOME/muonic_data. This directory must exist. If you use the provided setup script, it is created automatically

YYYY-MM-DD_HH-MM-SS_TYPE_MEASUREMENTTME_xy

- *YYYY-MM-DD* is the date of the measurement start
- *HH-MM-SS* is the GMT time of the measurement start
- *MEASUREMENTTIME* if muonic is closed, each file gets is corresponding measurement time (in hours) assigned.
- *xy* the two letters which were specified at the start of muonic
- *TYPE* might be one of the following:
- *RAW* the raw ASCII output of the DAQ card, this is only saved if the 'Save to file' button in clicked in the 'Daq output' window of muonic
- *R* is an automatically saved ASCII file which contains the rate measurement data, this can then be used to plot with e.g. gnuplot later on
- *L* specifies a file with times of registered muon decays. This file is automatically saved if a muon decay measurement is started.
- *P* stands for a file which contains a non-hex representation of the registered pulses. This file is only save if the *-p* option is given at the start of muonic

Representation of the pulses:

(69.15291364, [(0.0, 12.5)], [(2.5, 20.0)], [], [])

This is a python-tuple which contains the triggertime of the event and four lists with more tuples. The lists represent the channels (0-3 from left to right) and each tuple stands for a leading and a falling edge of a registered pulse. To get the exact time of the pulse start, one has to add the pulse LE and FE times to the triggertime

---

Note: For calculation of the LE and FE pulse times a TMC is used. It seems that for some DAQs cards a TMC bin is 1.25 ns wide, allthough the documentation says something else. The triggertime is calculated using a CPLD which runs in some cards at 25MHz, which gives a binwidth of the CPLD time of 40 ns. Please keep this limited precision in mind when adding CPLD and TMC times.

1.3.3 Performing measurements with muonic

Setting up the DAQ

For DAQ setup it is recommended to use the 'settings' menu, allthough everything can also be setup via the command line in the DAQ output window (see below.) Muonic translates the chosen settings to the corresponding DAQ commands and sends them to the DAQ. So if you want to change things like the coincidence time window, you have to issue the corresponding DAQ command in the DAQ output window.

Two menu items are of interest here: * Channel Configuration: Enable the channels here and set coincidence settings. A veto channel can also be specified.

Note: You have to ensure that the checkboxes for the channels you want to use are checked before you leave this dialogue, otherwise the channel gets deactivated.

Note: The concidence is realized by the DAQ in a way that no specific channels can be given. Instead this is meant as an 'any' condition. So 'twofold' means that 'any two of the enabled channels' must claim signal instead of two specific ones (like 1 and 2).

Warning: Measurements ad DESY indicated that the veto feature of the DAq might not work properly in all cases.

- Thresholds: For each channel a threshold (in milliVolts) can be specified. Pulse which are below this threshold are rejected. Use this for electronic noise supression.

Note: A proper calibration of the individual channels is the key to a succesfull measurement!

Looking at raw DAQ data

The first tab of muonic displays the raw ASCII DAQ data. This can be saved to a file. If the DAQ status messages should be supressed in that file, the option *-n* should be given at the start of muonic. The edit field can be used to send messages to the DAQ. For an overview over the messages, look here (link not available yet!). To issue such an command periodically, you can use the button 'Periodic Call'

Note: The two most importand DAQ commands are 'CD' ('counter disable') and 'CE' ('counter enable'). Pulse information is only given out by the DAQ if the counter is set to enabled. All pulse related features may not work properly if the counter is set to disabled.

Muon Rates

In this tab a plot of the measured muonrates is displayed. A triggerrate is only shown if a coincidence condition is set. In the block on the left side of the tab the average rates are displayed since the measurement start. Below you can find the number of counts for the individual channels. The measurement can be reset by clicking on 'Restart'. The 'Stop' button can be used to temporarily hold the plot to have a better look at it.

Note: You can use the displayed 'max rate' at the left bottom to check if anything with the measurement went wrong.

Note: Currently the plot shows only the last 200 seconds. If you want to have a longer timerange, you can use the information which is automatically stored in the 'R' file.(see above)

Muon Lifetime

A lifetime measurement of muons can be performed here. A histogram of time differences between succeding pulses in the same channel is shown. It can be fit with an exponential by clicking on 'Fit!'. The fit lifetime is then shown in the above right of the plot, for an estimate on the errors you have to look at the console.

Warning: This feature might not work properly, especially when used with the standard scintilators! Use it with extreme care.

Pulse Analyzer

You can have a look at the pulsewidhts in this plot. The height of the pulses is lost during the digitization prozess, so all pulses have the same height here.

1.4 Fermilab DAQ - hardware documentation

1.4.1 ASCII DAQ output format

sample line of DAQ output - example for the daq data format

triggers	r0	f0	r1	f1	r2	f2	r3	f3	onepps	gpstime	gpsdte	gps-valid	gps-satelites
92328FE2	00	3D	00	3E	00	00	00	00	915E10CF	034016.021	060180	V	00

1.4.2 DAQ onboard documentation

Online help on the DAQ cards is available by sending the following commands to the DAQ

- V1, V2, V3
- H1,H2

V1

Setting	example value	description
Run Mode	Off	CE (cnt enable), CD (cnt disable)
Ch(s) Enabled	3,2,1,0	Cmd DC Reg C0 using (bits 3-0)
Veto Enable	Off	VE 0 (Off), VE 1 (On)
Veto Select	Ch0	Cmd DC Reg C0 using (bits 7,6)
Coincidence 1-4	1-Fold	Cmd DC Reg C0 using (bits 5,4)
Pipe Line Delay	40 nS	Cmd DT Reg T1=rDelay Reg T2=wDelay 10nS/cnt
Gate Width	100 nS	Cmd DC Reg C2=LowByte Reg C3=HighByte 10nS/cnt
Veto Width	0 nS	Cmd VG (10nS/cnt)
Ch0 Threshold Ch1 Threshold Ch2 Threshold Ch3 Threshold	0.200 vlts 0.200 vlts 0.200 vlts 0.200 vlts	
Test Pulser Vlt Test Pulse Ena	3.000 vlts Off	

Example line for 1 of 4 channels. (Line Drawing, Not to Scale) Input Pulse edges (begin/end) set rising/falling tags bits. ____~~~~~~_____ Input Pulse, Gate cycle begins _____~_____ Delayed Rise Edge 'RE' Tag Bit _____~_____ Delayed Fall Edge 'FE' Tag Bit _____ Tag Bits delayed by PipeLnDly ___| |_____ **PipeLineDelay : 40nS** _____ _____| |___ **Capture Window: 60nS** _____ ___| |___ Gate Width : 100nS

If 'RE','FE' are outside Capture Window, data tag bit(s) will be missing. CaptureWindow = GateWidth - PipeLineDelay The default Pipe Line Delay is 40nS, default Gate Width is 100nS. Setup CMD sequence for Pipeline Delay. CD, WT 1 0, WT 2 nn (10nS/cnt) Setup CMD sequence for Gate Width. CD, WC 2 nn(10nS/cnt), WC 3 nn (2.56uS/cnt)

H2

Barometer Qnet Help Page 2 BA - Display Barometer trim setting in mVolts and pressure as mBar. BA d - Calibrate Barometer by adj. trim DAC ch in mVlts (0-4095mV). Flash FL p - Load Flash with Altera binary file(*.rbf), p=password. FR - Read FPGA setup flash, display sumcheck. FMR p - Read page 0-3FF(h), (264 bytes/page) Page 100h= start fpga *.rbf file, page 0=saved setup. GPS NA 0 - Append NMEA GPS data Off,(include 1pps data). NA 1 - Append NMEA GPS data On, (Adds GPS to output). NA 2 - Append NMEA GPS data Off,(no 1pps data). NM 0 - NMEA GPS display, Off, (default), GPS port speed 38400, locked. NM 1 - NMEA GPS display (RMC + GGA + GSV) data. NM 2 - NMEA GPS display (ALL) data, use with GPS display applications. Test Pulser TE m - Enable run mode, 0=Off, 1=One cycle, 2=Continuous. TD m - Load sample trigger data list, 0=Reset, 1=Singles, 2=Majority. TV m - Voltage level at pulse DAC, 0-4095mV, TV=read. Serial # SN p n - Store serial # to flash, p=password, n=(0-65535 BCD). SN - Display serial number (BCD). Status ST - Send status line now. This resets the minute timer. ST 0 - Status line, disabled. ST 1 m - Send status line every (m) minutes.(m=1-30, def=5). ST 2 m - Include scalar data line, chs S0-S4 after each status line. ST 3 m - Include scalar data line, plus reset counters on each timeout. TI n - Timer (day hr:min:sec.msec), TI=display time, (TI n=0 clear). U1 n - Display Uart error counter, (U1 n=0 to zero counters). VM 1 - View mode, 0x80=Event_Demarcation_Bit outputs a blank line. - View mode returns to normal after 'CD','CE','ST' or 'RE'.

H1 Quarknet Scintillator Card, Qnet2.5 Vers 1.11 Compiled Jul 15 2009 HE=Help Serial#=6531 uC_Volts=3.33 GPS_TempC=0.0 mBar=1023.8

CE - TMC Counter Enable. CD - TMC Counter Disable. DC - Display Control Registers, (C0-C3). WC a d - Write Control Registers, addr(0-6) data byte(H). DT - Display TMC Reg, 0-3, (1=PipeLineDelayRd, 2=PipeLineDelayWr). WT a d - Write TMC Reg, addr(1,2) data byte(H), if a=4 write delay word. DG - Display GPS Info, Date, Time, Position and Status. DS - Display Scalar, channel(S0-S3), trigger(S4), time(S5). RE - Reset complete board to power up defaults. RB - Reset only the TMC and Counters. SB p d - Set Baud,password, 1=19K, 2=38K, 3=57K ,4=115K, 5=230K, 6=460K, 7=920K SA n - Save setup, 0=(TMC disable), 1=(TMC enable), 2=(Restore Defaults). TH - Thermometer data display (@ GPS), -40 to 99 degrees C. TL c d - Threshold Level, signal ch(0-3)(4=setAll), data(0-4095mV), TL=read. Veto - Veto select, Off='VE 0', On='VE 1', Gate='VG c', 0-255(D) 10ns/cnt. View - View setup registers. Setup=V1, Voltages(V2), GPS LOCK(V3). HELP - HE,H1=Page1, H2=Page2, HB=Barometer, HS=Status, HT=Trigger.

VE2 V2 Barometer Pressure Sensor Calibration Voltage = 1495 mVolts Use Cmd 'BA' to calibrate. Sensor Output Voltage= 1655 mVolts (2.93mV * 565 Cnts) Pressure mBar = 1023.6 (1655.5 - 1500)/15 + 1013.25 Pressure inch = 30.63 (mBar / 33.42)

Timer Capture/Compare Channel TempC = 0.0 Error? Check sensor cable connection at GPS unit. TempF = 32.0 (TempC * 1.8) + 32

Analog to Digital Converter Channels(ADC) Vcc 1.80V = 1.82 vlts (2.93mV * 621 Cnts) Vcc 1.20V = 1.19 vlts (2.93mV * 407 Cnts) Pos 2.50V = 2.45 vlts (2.93mV * 837 Cnts) Neg 5.00V = 5.03 vlts (7.38mV * 682 Cnts) Vcc 3.30V = 3.33 vlts (4.84mV * 689 Cnts) Pos 5.00V = 4.84 vlts (7.38mV * 656 Cnts) 5V Test Max=4.86v Min=4.84v Noise=0.015v

V3 10 Second Accumulation of 1PPS Latched 25MHz Counter. (20 line buffer) Buffer Now (hex) Prev-Now (dec) (25e6*10) 1 0 0 2 0 0 3 0 0 4 0 0 5 0 0 6 0 0 7 0 0 8 0 0 9 0 0 10 0 0 11 0 0 12 0 0 13 0 0 14 0 0 15 0 0 16 0 0 17 0 0 18 0 0 19 0 0 20 0 0

1.5 muonic package software reference

1.5.1 main package: muonic

muonic.daq muonic.gui muonic.analysis

1.5.2 daq i/o with muonic.daq

Provide a connection to the QNet DAQ cards via python-serial. For software testing and development, (very) dumb DAQ card simulator is available

muonic.daq.DAQProvider

Control the two I/O threads which communicate with the DAQ. If the simulated DAQ is used, there is only one thread.

class muonic.daq.DAQProvider.**DAQProvider**(*opts*, *logger*, *root*)

Launch the main part of the GUI and the worker threads. periodicCall and endApplication could reside in the GUI part, but putting them here means that you have all the thread controls in a single place.

muonic.daq.DAQConnection

The module provides a class which uses python-serial to open a connection over the usb ports to the daq card. Since on LINUX systems the used usb device (which is usually /dev/tty0) might change during runtime, this is catched automatically by DaqConnection. Therefore a shell script is invoked.

class muonic.daq.DaqConnection.**DaqConnection**(*inqueue*, *outqueue*, *logger*)

get_port()

check out which device (/dev/tty) is used for DAQ communication

read()

Get data from the DAQ. Read it from the provided Queue.

write()

Put messages from the inqueue which is filled by the DAQ

muonic.daq.SimDaqConnection

This module provides a dummy class which simulates DAQ I/O which is read from the file "simdaq.txt". The simulation is only useful if the software-gui should be tested, but no DAQ card is available Provides a simple DAQ card simulation, so that software can be tested

class muonic.daq.SimDaqConnection.**SimDaq**(*logger*, *usefile='simdaq.txt'*, *createfakerates=True*)

_physics()

This routine will increase the scalars variables using predefined rates Rates are drawn from Poisson distributions

inWaiting()

simulate a busy DAQ

readline()
read dummy pulses from the simdaq file till the configured value is reached

write(*command*)
Trigger a simulated daq response with command

class muonic.daq.SimDaqConnection.**SimDaqConnection**(*inqueue*, *outqueue*, *logger*)

read()
Simulate DAQ I/O

1.5.3 pyqt4 gui with muonic.gui

This package contains all gui relevant classes like dialogboxes and tabwidgets. Every item in the global menu is utilizes a "Dialog" class. The "Canvas" classes contain plot routines for displaying measurements in the TabWidget. The gui of the programm, written with PyQt4

muonic.gui.MainWindow

Contains the "main" gui application. It Provides the MainWindow, which initializes the different tabs and draws a menu. Provides the main window for the gui part of muonic

class muonic.gui.MainWindow.**MainWindow**(*inqueue*, *outqueue*, *logger*, *opts*, *root*, *win_parent=None*)
The main application

about_menu()
Show a link to the online documentation

closeEvent(*ev*)
Is triggered when the window is closed, we have to reimplement it to provide our special needs for the case the program is ended.

config_menu()
Show the config dialog

exit_program(**args*)
This function is used either with the 'x' button (then an event has to be passed) Or it is used with the File->Exit button, than no event will be passed.

get_scalars_from_queue(*msg*)
Explicitely scan a message for scalar informatioin Returns True if found, else False

get_thresholds_from_queue(*msg*)
Explicitely scan message for threshold information Return True if found, else False

help_menu()
Show a simple help menu

processIncoming()
Handle all the messages currently in the inqueue and pass the result to the corresponding widgets

query_daq_for_scalars()
Send a "DS" message to DAQ and record the time when this is don

sphinxdoc_menu()
Show the sphinx documentation that comes with muonic an a browser

threshold_menu()
Shows the threshold dialogue

widgetUpdate()
Update the widgets

muonic.gui.MainWindow.**tr**()
QCoreApplication.translate(str, str, str disambiguation=None, QCoreApplication.Encoding encoding=QCoreApplication.CodecForTr) -> QString QCoreApplication.translate(str, str, str, QCoreApplication.Encoding, int) -> QString

muonic.gui.MuonicWidgets

The functionality of the software Provide the different physics widgets

class muonic.gui.MuonicWidgets.**DAQWidget**(*logger*, *parent=None*)

on_file_clicked()
save the raw daq data to a automatically named file

on_hello_clicked()
send a message to the daq

on_periodic_clicked()
issue a command periodically

class muonic.gui.MuonicWidgets.**DecayWidget**(*logger*, *parent=None*)

activateMuondecayClicked()
What should be done if we are looking for mu-decays?

calculate(*pulses*)

is_active()

mufitClicked()
fit the muon decay histogram

update()

class muonic.gui.MuonicWidgets.**PulseanalyzerWidget**(*logger*)
Provide a widget which is able to show a plot of triggered pulses

calculate(*pulses*)

is_active()

update()

class muonic.gui.MuonicWidgets.**RateWidget**(*logger*, *parent=None*)
Display rate plot

calculate(*rates*)

is_active()

startClicked()
restart the rate measurement

stopClicked()
hold the rate measurement plot till buttion is pushed again

update()

class muonic.gui.MuonicWidgets.**VelocityWidget**(*logger*)

activateVelocityClicked()
Perform extra actions when the checkbox is clicked

calculate(*pulses*)

is_active()

update()

velocityFitClicked()
fit the muon velocity histogram

muonic.gui.MuonicWidgets.**tr**()
QCoreApplication.translate(str, str, str disambiguation=None, QCoreApplication.Encoding encoding=QCoreApplication.CodecForTr) -> QString QCoreApplication.translate(str, str, str, QCoreApplication.Encoding, int) -> QString

muonic.gui.MuonicDialogs

Provide the dialog fields for user interaction

class muonic.gui.MuonicDialogs.**ConfigDialog**(**args*)
Set Channel configuration

class muonic.gui.MuonicDialogs.**DecayConfigDialog**(**args*)
Settings for the muondecay

class muonic.gui.MuonicDialogs.**HelpDialog**(**args*)

helptext()
Show this text in the help window

class muonic.gui.MuonicDialogs.**MuonicDialog**
Base class of all muonic dialogs

createButtonBox(*objectname='buttonBox'*, *leftoffset=80*, *topoffset=900*)
Create a custom button for cancel/apply

createCheckGroupBox(*label='Single Pulse'*, *objectname='singlecheckbox'*, *radio=False*, *leftoffset=20*, *setchecked=None*, *checkable=False*, *itemlabels=['Chan0', 'Chan1', 'Chan2', 'Chan3']*)
Create a group of choices

class muonic.gui.MuonicDialogs.**PeriodicCallDialog**(**args*)
Issue a command periodically

class muonic.gui.MuonicDialogs.**ThresholdDialog**(*thr0*, *thr1*, *thr2*, *thr3*, **args*)
Set the Thresholds

class muonic.gui.MuonicDialogs.**VelocityConfigDialog**(**args*)

muonic.gui.MuonicPlotCanvases

Provide the canvases for plots in muonic

class muonic.gui.MuonicPlotCanvases.**LifetimeCanvas**(*parent*, *logger*)
A simple histogram for the use with mu lifetime measurement

class muonic.gui.MuonicPlotCanvases.**MuonicHistCanvas**(*parent*, *logger*, *binning*, *histcolor='b'*, ***kwargs*)
A base class for all canvases with a histogram

show_fit(*bin_centers*, *bincontent*, *fitx*, *decay*, *p*, *covar*, *chisquare*, *nbins*)

update_plot(*data*)

class muonic.gui.MuonicPlotCanvases.**MuonicPlotCanvas**(*parent*, *logger*, *ymin=0*, *ymax=10*, *xmin=0*, *xmax=10*, *xlabel='xlabel'*, *ylabel='ylabel'*, *grid=True*)
The base class of all muonic plot canvases

color(*string*, *color='none'*)
output colored strings on the terminal

update_plot()
Instructions to updated this plot implement this individually

class muonic.gui.MuonicPlotCanvases.**PulseCanvas**(*parent*, *logger*)
Matplotlib Figure widget to display Pulses

update_plot(*pulses*)

class muonic.gui.MuonicPlotCanvases.**PulseWidthCanvas**(*parent*, *logger*, *histcolor='r'*)

update_plot(*data*)

class muonic.gui.MuonicPlotCanvases.**ScalarsCanvas**(*parent*, *logger*)

reset()
reseting all data

update_plot(*result*)

class muonic.gui.MuonicPlotCanvases.**VelocityCanvas**(*parent*, *logger*)

1.5.4 analyis package muonic.analysis

muonic.analysis.PulseAnalyzer

Transformation of ASCII DAQ data. Combination of Pulses to events, and looking for decaying muons with different trigger condi Get the absolute timing of the pulses by use of the gps time Calculate also a non hex representation of leading and falling edges of the pulses

class muonic.analysis.PulseAnalyzer.**DecayTriggerThorough**(*logger*)
We demand a second pulse in the same channel where the muon got stuck Should operate for a 10mu sec triggerwindow

trigger(*triggerpulses*, *single_channel=2*, *double_channel=3*, *veto_channel=4*, *selfveto=False*, *mindecaytime=0*, *minsinglepulsewidth=0*, *maxsinglepulsewidth=12000*, *mindoublepulsewidth=0*, *maxdoublepulsewidth=12000*)
Trigger on a certain combination of single and doublepulses

class muonic.analysis.PulseAnalyzer.**PulseExtractor**(*pulsefile=''*)
get the pulses out of a daq line speed is important here

_calculate_edges(*line*, *counter_diff=0*)
get the leading and falling edges of the pulses Use counter diff for getting pulse times in subsequent lines of the triggerflag

_get_evt_time(*time*, *correction*, *trigger_count*, *onepps*)
Get the absolute event time in seconds since day start If gps is not available, only relative eventtime based on counts is returned

_order_and_cleanpulses()
Remove pulses which have a leading edge later in time than a falling edge and do a bit of sorting Remove also single leading or falling edges NEW: We add virtual falling edges!

close_file()

extract(*line*)
Analyze subsequent lines (one per call) and check if pulses are related to triggers For each new trigger, return the set of pulses which belong to that trigger, otherwise return None

class muonic.analysis.PulseAnalyzer.**VelocityTrigger**(*logger*)

trigger(*pulses*, *upperchannel=1*, *lowerchannel=2*, *omit_early_pulses=True*)
Timedifference will be calculated t(upperchannel) - t(lowerchannel)

muonic.analysis.fit

Provide a fitting routine Script for performing a fit to a histogramm of recorded time differences for the use with QNet

muonic.analysis.fit.**gaussian_fit**(*bincontent*)

muonic.analysis.fit.**main**(*bincontent=None*)

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

m

INDEX

Symbols

A

C

D

E

G

T

U

V

W