Rev. 1 - January 24th, 2025

CAEN SRR7780

Software Library for Shift Register R7780

CAEN
Tools for Discovery

User Manual
UM9815

Purpose of this Manual

This document contains the full description of CAEN SRR7780, the software library for the acquisition and control of the R7780 CAEN Shift Register Multiplicity and Time Recorder.

Change Document Record

Date	Revision	Changes
October 23rd, 2023	00	First official release
January 24th, 2025	01	Modified Sec. **2.3**.

Symbols, Abbreviated Terms and Notation

HPGe	High Purity Germanium detector
HVPS	High Voltage Power Supply
MCA	Multi-Channel Analyzer
MDA	Minimum Detectable Activity
OS	Operating System
PC	Personal Computer
PMT	Photo Multiplier Tube
USB	Universal Serial Bus

Manufacturer Contacts

CAEN S.p.A.
Via Vetraia, 11 55049 Viareggio (LU) - ITALY
Tel. +39.0584.388.398 Fax +39.0584.388.959
www.caen.it | info@caen.it

Limitation of Responsibility

If the warnings contained in this manual are not followed, CAEN will not be responsible for damage caused by improper use of the device. The manufacturer declines all responsibility for damage resulting from failure to comply with the instructions for use of the product. The equipment must be used as described in the user manual, with particular regard to the intended use, using only accessories as specified by the manufacturer. No modification or repair can be performed.

Disclaimer

The information contained herein has been carefully checked and is believed to be accurate; however, no responsibility is assumed for inaccuracies. CAEN SpA reserves the right to modify its products specifications without giving any notice; for up to date information please visit www.caen.it.

Made in Italy

We remark that all our boards have been designed and assembled in Italy. In a challenging environment where a competitive edge is often obtained at the cost of lower wages and declining working conditions, we proudly acknowledge that all those who participated in the production and distribution process of our devices were reasonably paid and worked in a safe environment (while this is true for the boards marked ”MADE IN ITALY”, we cannot guarantee for third-party manufactures).

Index

List of Figures

1 Introduction

The R7780 module is a Neutron Coincidence Analyzer and Multiplicity module combining the functions of a Shift Register and a Pulse Train Recorder. It manages the acquisition and analysis over up to 8 neutron detectors that can work in unattended mode. The eight single-ended TTL inputs (LEMO) feature independent counting capability. Moreover, adjustable input thresholds give the possibility to compensate TTL signal voltage drops in case of long-distance use.

The internal 100MHz sampling clock fits for high count-rate applications and the on-board intelligence synergy of a FPGA and a Single Board Computer (an ARM CPU running Linux) makes it possible to provide time-stamped lists and the overall neutron counting information (coincidence timing, multiplicity distributions of coincident events, etc.) required for the analysis in Nuclear Safeguards and nuclear material process monitoring.

After the start-up sequence based on a programmable configuration file, the device can collect data without external control on a local non-volatile memory. Two SD cards, externally accessible for insertion/extraction, store all measurement results and log information in two identical copies for redundancy reasons. The presence of a OTG USB port allows the automatic data retrieval by a USB stick.

The device can also operate in attended mode controlled by an external host computer using the USB port as virtual point-to-point serial connection (reserved for INCC software protocol[1]) and a remote network connection through the ethernet port.

High power outputs are available as well: one high-voltage channel for the detector biasing and two different low-voltage channels (+5V and +12V) to power the front-end electronics such as preamplifiers and discriminators.

[1] *INCC Software Users Manual*

2 The CAEN SRR7780 Library

The CAEN SRR7780 library is specific for R7780 boards and allows the user to configure the devices, to manage the acquisition and to read data from the boards. The library can be employed to develop custom readout programs for controlling devices working in both attended and unattended mode. A sample program with all the source files is provided in the form of a demo application. The CAEN SRR7780 library relies on the CAENUtility library for some communication protocol functions.

2.1 Library Installation

2.1.1 Windows

An installer is provided for installing the CAENSRR7780 library. The default install path is: C:\Program Files\CAEN\CAENSRR7780.

2.1.2 Linux

Before installing the library some dependencies should be installed: run the command *"sudo apt install gcc make pkg-config libxml2-dev"* or *"sudo yum install gcc make pkgconf-pkg-config libxml2-devel"*.

To install the library extract the source files from the archive, enter the folder and run the following standard commands in sequence:

"./configure [–prefix=<installdir>]", *"make"*, *"sudo make install"*. If not set, the default installation directory is /usr/lib.

2.2 Return Codes

Error code	Value	Error explanation
CAENSR_RetCode_Success	0	No Error
CAENSR_RetCode_Generic	-1	Generic error
CAENSR_RetCode_SockInit	-2	Socket initialization error (in case of ethernet connection)
CAENSR_RetCode_SockConnect	-3	Socket connect error (in case of ethernet connection)
CAENSR_RetCode_OutOfMemory	-4	Out of memory
CAENSR_RetCode_Handle	-5	Invalid handle
CAENSR_RetCode_Argument	-6	Invalid argument
CAENSR_RetCode_SocketSend	-7	Send error
CAENSR_RetCode_SocketReceive	-8	Receive error
CAENSR_RetCode_Protocol	-9	Protocol error
CAENSR_RetCode_Serialize	-10	Serialize error
CAENSR_RetCode_Deserialize	-11	Deserialize error
CAENSR_RetCode_Parameter	-12	Parameter error
CAENSR_RetCode_ParameterValue	-13	Parameter value error
CAENSR_RetCode_ParameterType	-14	Parameter type error
CAENSR_RetCode_LibraryLoad	-15	Library dynamic load error (Linux only)
CAENSR_RetCode_NotConnected	-16	Not connected
CAENSR_RetCode_NotSupported	-17	Not supported
CAENSR_RetCode_NotYetImplemented	-18	Not yet implemented
CAENSR_RetCode_AlreadyConnected	-19	Device already connected
CAENSR_RetCode_MaxDevices	-20	Max number of devices connected
CAENSR_RetCode_SerialInit	-21	Serial port open error (in case of serial connection)
CAENSR_RetCode_SerialWrite	-22	Serial port write error (in case of serial connection)
CAENSR_RetCode_SerialRead	-23	Serial port read error (in case of serial connection)
CAENSR_RetCode_OperatingMode	-24	Device operating mode not compliant
CAENSR_RetCode_AcqIsOff	-25	Acquisition is off
CAENSR_RetCode_AcqIsOn	-26	Acquisition is on
CAENSR_RetCode_AnalysisInProgress	-27	Data analysis is still in progress
CAENSR_RetCode_Connection_Refused	-28	Connection refused by device, wrong autentication
CAENSR_RetCode_ReadData	-29	Device reported a ReadData error
CAENSR_RetCode_InvalidEvent	-30	Device found an invalid event
CAENSR_RetCode_CommandFormat	-31	Wrong command format
CAENSR_RetCode_InvalidArgument	-32	Invalid argument provided
CAENSR_RetCode_SDAlreadyEnabled	-33	SD card is already enabled
CAENSR_RetCode_SDAlreadyDisabled	-34	SD card is already disabled
CAENSR_RetCode_SDIsLocked	-35	SD card is locked
CAENSR_RetCode_WrongFPGAFirmware	-36	Device detected a wrong FPGA firmware
CAENSR_RetCode_DoubleCopy	-37	Only one SD card found

2.3 Device Connection

A R7780 device can be connected to the PC via ethernet. For retro compatibility with old softwares, serial communication is also possible when usb is connected because the internal CPU is equipped with a virtual serial port. The following CAEN SRR7780 library functions allow to establish a new connection or to close the connection with a device.

2.3.1 OpenDevice

This function is now available for back compatibility only. Please refer to the new function OpenDeviceSecure.

This function opens the device and provides the device handler that can be used to interact with the device. Return value is the error code (see error codes in Sec. **Return Codes**).

```
//Connect to Device
//in case of ethernet cnnection path contains the IP address "eth:xxx.xxx.xxx.xxx"
//in case of serial connection path contains the serial port name "ser:COMX"
//in case of usb connection path contains the device serial number "usb:SN"
CAENSR_RetCode_t CAENSR_DLLAPI CAENSR_OpenDevice(const char *path, CAENSR_Access_Level access_level, char * password, SR_HANDLE *handle);
```

Parameter	I/O	Description
*path	Input	Path to the device: *"eth:hostname"* (where *hostname* is the IP address of the device) for ethernet connection, *"ser:port"* (where *port* is the name of the serial port) for serial connection.
access_level	Input	Access level (ADMIN, USER or GUEST)
*handle	Output	Handle of the device, NULL in case of errors.

2.3.2 OpenDeviceSecure

This function opens the device and provides the device handler that can be used to interact with the device. Return value is the error code (see error codes in Sec. **Return Codes**).

Two possible connection modes are available: TRUSTED connection mode allows the user to establish a secure connection; the exchanged TCP packets received from the unit contain an encrypted header that can be decryped only using the public key *R7780Public_key.pem* provided in the installation package. TRUSTED connection prevents possible network security issues. UNTRUSTED connection mode does not use any secure key.

```
CAENSR_RetCode_t CAENSR_DLLAPI CAENSR_OpenDeviceSecure(const char* path, CAENSR_Access_Level access_level,
                                                       char* password, CAENSR_ConnectionMode mode, char* public_key_pem, SR_HANDLE* handle);
```

Parameter	I/O	Description
*path	Input	Path to the device: *"eth:hostname"* (where *hostname* is the IP address of the device) for ethernet connection, *"ser:port"* (where *port* is the name of the serial port) for serial connection.
access_level	Input	Access level (ADMIN, USER or GUEST)
connection_mode	Input	Connection mode (CAENSR_TRUSTED, CAENSR_UNTRUSTED)
*public_key_pem	Input	Public key
*handle	Output	Handle of the device, NULL in case of errors.

2.3.3 CloseDevice

This function closes the connection with the device. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_CloseDevice(SR_HANDLE handle);
```

Parameter	I/O	Description
handle	Input	The handle of the device to be closed.

2.4 Device Info

2.4.1 GetDevInfo

This function allows to get some general information from the device and fills a dedicated structure with them. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetDevInfo(SR_HANDLE handle,
                  CAENSR_DeviceInfo_t *Dev_Info);

typedef struct {
    int Connected;
    CAENSR_ConnectionType ConnType;
    char ConnPath[CONN_PATH_MAX_LEN];
    uint32_t Model;
    uint32_t PCB_Revision;
    uint32_t nChannels;
    uint32_t nHVChannels;
    char HVPolarity;
    uint32_t SerialNum;
    uint32_t FWRevision;
    Operating_Mode ShReg_Mode;
    Counting_Mode ShReg_Count_Mode;
    CAENSR_HVInfo_t HVInfo[MAX_HVCHANNELS];
} CAENSR_DeviceInfo_t;

typedef enum CAENSR_ConnectionType {
    CAENSR_Ethernet = 0,
    CAENSR_Serial = 1,
} CAENSR_ConnectionType;

typedef enum Operating_Mode {
    ATTENDED = 0,
    UNATTENDED = 1,
    LIST_MODE = 2,
    UNKNOWN = 3,
} Operating_Mode;
```

Parameter	I/O	Description
handle	Input	The handle of the device to read info from
*Dev_Info	Input	Pointer to the structure to store the device info

CAENSR_DeviceInfo_t field	Description
Connected	connection status: 1 = ok, 0 = lost
ConnType	Active connection type (ethernet or serial)
ConnPath	Active connection path
Model	Device model (7780)
PCB_Revision	Revision number of the device PCB
nChannels	Number of input channels
nHVChannels	Number of HV channels
HVPolarity	Polarity of the HV channel: 'P'(positive), 'N'(negative)
SerialNum	Serial number of the device
FWRevision	FPGA firmware revision
ShReg_Mode	Device operating mode (see Operating_Mode type)
HVInfo	Structure filled with the HV channel info

2.4.2 GetHVInfo

This function allows to get information about the HV channel and fills a dedicated structure with them. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetHVInfo(SR_HANDLE handle,
                 CAENSR_HVInfo_t *HV_Info
                 );

typedef struct {
    uint32_t Polarity;
    uint32_t NHVRanges;
    uint32_t ActiveRange;
    CAENSR_HVRange_Info_t HVRangeInfo[MAX_HVRANGES];
} CAENSR_HVInfo_t;

typedef struct {
    uint16_t MaxHV;
    uint16_t MaxHVRamp;
}CAENSR_HVRange_Info_t;
```

Parameter	I/O	Description
handle	Input	The handle of the device
*HV_Info	Input	Pointer to the structure to store the HV info

CAENSR_HVInfo_t field	Description
Polarity	HV polarity code: 1=positive, 0=negative
NHVRanges	Number of ranges allowed for the HV channel
ActiveRange	Active HV range of the HV channel
HVRangeInfo	Info about the HV range (max output voltage and ramp-up/down)

2.5 Commands

Communication with the R7780 devices takes place according to a well defined set of commands. The devices are configured to perform specific operations or to provide the requested data depending on the received command. If a ethernet communication is active, commands and device replies follow a custom protocol described in Sec. **Communication Protocols**, while for serial communication a standard protocol is used (also described in Sec. **Communication Protocols**).

2.5.1 SendCommand

The CAENSRR7780 library handles both the different communication protocols depending on the active connection and provides a general SendCommand function to communicate with the device. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_SendCommand(SR_HANDLE handle,
                    CAENSR_Command_t command,
                    uint32_t *value);

typedef enum {
    CMD_ACQ_START = 0,
    CMD_ACQ_STOP = 1,
    CMD_DEVICE_CLEAR = 2,
    CMD_DEV_CONFIG_RESET = 3,
    CMD_DEV_SW_TRG = 4,
    CMD_MEAS_DATA_RESET = 5,
    CMD_MEAS_RESET_ALL = 6,
    CMD_SD_ACCESS_DISABLE = 7,
    CMD_SD_ACCESS_ENABLE = 8,
    CMD_GET_SD_ACCESS_STATUS = 9,
    CMD_HVCH_RESET = 10,
    CMD_LV5V_RESET = 11,
    CMD_LV12V_RESET = 12,
    CMD_OVERWRITE_CONFIG = 13,
} CAENSR_Command_t;
```

Parameter	I/O	Description
handle	Input	Handle of the device
command	Input	Command to send (see CAENSR_Command_t)
*value	Input/Output	Pointer to the value to send or to receive (NULL in case no value is associated to the command)

The supported commands are listed under the CAENSR_Command_t enum, but some of them are only supported by the devices when they are working in UNATTENDED mode. In the following, a brief description is provided for every command. For the currently supported commands *value is NULL.

- **CMD_ACQ_START**: Start acquisition for the device.
- **CMD_ACQ_STOP**: Stop acquisition for the device.
- **CMD_DEVICE_CLEAR**: Clear the device memory.
- **CMD_DEV_CONFIG_RESET**: Reset the device configuration to default.
- **CMD_DEV_SW_TRG**: Send a software trigger to the device. All channels are forced to generate an output event at the same time.
- **CMD_MEAS_DATA_RESET** (*only for unattended devices*): Reset current measurement. All analysis data are cleared and acquisition is restarted (if active).
- **CMD_MEAS_RESET_ALL** (*only for unattended devices*): Reset analysis data (if analysis is in progress), and reset device configuration to default. Acquisition is restarted if it was active.
- **CMD_SD_ACCESS_DISABLE**: Disable access to SD cards in order to allow their removal

- **CMD_SD_ACCESS_ENABLE**: Enable access to SD cards to restart data save and logging
- **CMD_GET_SD_ACCESS_STATUS**: Get the current status of the SD cards
- **CMD_HVCH_RESET**: Reset the HV channel
- **CMD_LV5V_RESET**: Reset the LV 5V channel
- **CMD_LV12V_RESET**: Reset the LV 12V channel
- **CMD_OVERWRITE_CONFIG**: Save current configuration and overwrite the previous one

2.5.2 HVOnOff

This function allows to switch on or off the HV channel of the device. At present all R7780 devices provide only one HV channel, but the function is ready to support a possible new model equipped with more than one HV channel. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_HVOnOff(SR_HANDLE handle,
               uint32_t hvch,
               uint8_t on_or_off
               );
```

Parameter	I/O	Description
handle	Input	Handle of the device
hvch	Input	HV channel number to switch on/off
on_or_off	Input	0 for switching off, 1 for switching on

2.5.3 LV5VOnOff and LV12VOnOff

These functions allow to switch on or off the LV channels of the device. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_LV5VOnOff(SR_HANDLE handle, uint8_t on_or_off);

CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_LV12VOnOff(SR_HANDLE handle, uint8_t on_or_off);
```

Parameter	I/O	Description
handle	Input	Handle of the device
on_or_off	Input	0 for switching off, 1 for switching on

2.5.4 Device ClockTime

The library allows to read the internal system clock time from the device and to adjust it. This is particularly important for devices working in unattended mode, because the analysis results are saved to files that are named according to the system date-time when acquisition was started. This is the same also for system log files. By checking and adjusting the clock time, all the storage files will be dated correctly. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetDevClockTime(SR_HANDLE handle,
                       char *time_str
                       );

CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_SetDevClockTime(SR_HANDLE handle,
                       char *time_str
                       );
```

The date-time string is formatted according to the following predefined format: *"MMDDYYhhmmss"* where MM is the month, DD is the day, YY is the year after 2000, hh are hours, mm are minutes and ss are seconds.

Parameter	I/O	Description
handle	Input	Handle of the device
*time_str	Input	Pointer to a pre-allocated string of size CLK_STR_LEN containing, or that will contain, the date-time formatted string

2.6 Parameters

2.6.1 Get and Set Parameter

The R7780 devices can be configured by means of a set of parameters that are related both to acquisition and analysis settings. The following functions allow to read the current value of a parameter and to set a new value for it. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_GetParameter(SR_HANDLE handle,
                     CAENSR_Parameter_t par, void *value);

CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_SetParameter(SR_HANDLE handle,
                     CAENSR_Parameter_t par, void *value);

typedef enum {
    PARAM_STATUS = 0,
    PARAM_OP_MODE = 1,
    PARAM_GATE = 2,
    PARAM_PDELAY = 3,
    PARAM_MTIME = 4,
    PARAM_NRUNS = 5,
    PARAM_LDELAY = 6,
    PARAM_MULT_BINS = 7,
    PARAM_ENABLEMASK = 8,
    PARAM_CH_AGGR = 9,
    PARAM_CHG_SIZE = 10,
    PARAM_G_ENABLEMASK = 11,
    PARAM_GATE_HRES = 12,
    PARAM_PDELAY_HRES = 13,
    PARAM_COUNT_MODE = 14,
    PARAM_TTL_LEVEL_14 = 15,
    PARAM_TTL_LEVEL_58 = 16,
    PARAM_ADC5VMON = 17,
    PARAM_ADC12VMON = 18,
    PARAM_LVSTATUS = 19,
    PARAM_COINC_ENABLE = 20,
    PARAM_AUTORESTART_TIME = 21,
    PARAM_NTP_ENABLE = 22,
    PARAM_NTP_IP = 23,
} CAENSR_Parameter_t;
```

Parameter	I/O	Description
handle	Input	Handle of the device
par	Input	Parameter to be read/set (see allowed values for CAENSR_Parameter_t)
*value	Input/Output	Pointer to the parameter value to set or that is read from the device.

In order to read or set a parameter, the pointer to its value is cast to a generic void *, but the value of every parameter has a well defined type. In the following a brief description of the parameters can be found and the corresponding type is specified.

- **PARAM_STATUS** (readonly parameter): the status of the acquisition. The value type should be unsigned char; the status value is 1 when acquisition is on and 0 when acquisition is off.
- **PARAM_OP_MODE**: the operating mode of the device (0=ATTENDED, 1=UNATTENDED, 2=LIST_MODE, 3=UNKNOWN). The value type should be unsigned char.
- **PARAM_GATE**: the length in µs of the A and ReA gate window to be used for data analysis. The value type should be unsigned short.

- **PARAM_PDELAY**: the length in µs of the predelay to be used for data analysis. The value type should be unsigned int.
- **PARAM_MTIME**: the length, in 0.1 s unit, of an acquisition cycle. The value type should be unsigned int.
- **PARAM_NRUNS**: the total number of acquisition cycles. The value type should be unsigned int.
- **PARAM_LDELAY**: the length in µs of the long delay to be used for data analysis. The value type should be unsigned int.
- **PARAM_MULT_BINS**: the number of bins used for multiplicity distributions. The value type should be unsigned short.
- **PARAM_ENABLEMASK**: the input channels enablemask. The value type should be unsigned int.
- **PARAM_GATE_HRES**: the length in 10ns unit of the A and ReA gate window to be used for data analysis. The value type should be unsigned short.
- **PARAM_PDELAY_HRES**: the length in 10ns unit of the predelay to be used for data analysis. The value type should be unsigned int.
- **PARAM_COUNT_MODE**: the counting mode of the device (0=COUNTING, 1=COINCIDENCE, 2=MULTIPLICITY, 3=HV_PLATEAU). The value type should be unsigned char.
- **PARAM_TTL_LEVEL_14**: the threshold in mV of input channels 1-4. The value type should be unsigned int.
- **PARAM_TTL_LEVEL_58**: the threshold in mV of input channels 5-8. The value type should be unsigned int.
- **PARAM_ADC5VMON**: the monitored output voltage in mV of the 5V LV channel. The value type should be unsigned int.
- **PARAM_ADC12VMON**: the monitored output voltage in mV of the 12V LV channel. The value type should be unsigned int.
- **PARAM_LVSTATUS**: the status of the LV channels. The value type should be unsigned int.
- **PARAM_COINC_ENABLE**: the channels enablemask for coincidence counting. The value type should be unsigned int.
- **PARAM_AUTORESTART_TIME**: autorestart time for unattended operation. The value type should be unsigned int.
- **PARAM_NTP_ENABLE**: enable or disable NTP (1=enable, 0=disable). The value type should be unsigned char.
- **PARAM_NTP_IP**: IP address of the NTP. The value type should be char.

2.6.2 Get and Set HV Parameter

The following functions allow to monitor the status of a HV channel and to read or set HV parameters. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetHVParameter(SR_HANDLE handle,
                      CAENSR_HVParameter_t par, uint32_t hvch, void *value);

CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_SetHVParameter(SR_HANDLE handle,
                       CAENSR_HVParameter_t par, uint32_t hvch, void *value);

typedef enum {
    PARAM_VSET = 0,
    PARAM_VMON = 1,
    PARAM_VRAMP = 2,
    PARAM_IMON = 3,
    PARAM_HVSTATUS = 4,
    PARAM_HVP_VMAX = 5,
    PARAM_HVP_VMIN = 6,
    PARAM_HVP_VSTEP = 7,
    PARAM_HVP_DELAY = 8,
    PARAM_HVP_MTIME = 9,
    PARAM_HVP_ENABLEMASK = 10,
} CAENSR_HVParameter_t;
```

Parameter	I/O	Description
handle	Input	Handle of the device
par	Input	HV Parameter to be read/set (see allowed values for CAENSR_HVParameter_t)
hvch	Input	HV channel number
*value	Input/Output	Pointer to the HV parameter value to set or read from the device.

Also for HV parameters, the pointer to the parameter value is cast to a generic void *, even though it has a well defined type. In the following, a brief description of the HV parameters can be found and the corresponding type is specified.

- **PARAM_VSET**: the high voltage value in V. The type value should be unsigned short.
- **PARAM_VMON**: the monitored value of the high voltage in mV. The type value should be unsigned int.
- **PARAM_VRAMP**: the value of the high voltage ramp-up and ramp-down in mV/s. The type value should be unsigned int.
- **PARAM_IMON**: the monitored value of the output current in nA. The type value should be unsigned int.
- **PARAM_HVSTATUS** (readonly parameter): the status of the HV channel. The type value should be unsigned int. The status is returned as a mask where some key bits store a precise information about the HV channel status:
  - bit 0: HV channel power status (this bit is 1 if high voltage is on).
  - bit 1: HV channel ramp up (this bit is 1 if ramp up is active).
  - bit 2: HV channel ramp down (this bit is 1 if ramp down is active).
  - bit 3: HV channel overcurrent (this bit is 1 in case of overcurrent).
  - bit 6: HV channel overvoltage (this bit is 1 in case of overvoltage).
  - bit 8: HV channel temperature warning (this bit is 1 in case of a temperature warning).
  - bit 9: HV channel overtemperature (this bit is 1 in case of overtemperature).
  - bit 10: HV channel inhibited (this bit is 1 in case an inhibit is active).

  - bit 11: HV channel warmup (this bit is 1 if warmup is active).
  - bit 13: HV channel switching off (this bit is 1 in case HV is going off).
- **PARAM_HVP_VMAX**: the value of the maximum high voltage for the HV Plateau in V. The type value should be unsigned int.
- **PARAM_HVP_VMIN**: the value of the minimum high voltage for the HV Plateau in V. The type value should be unsigned int.
- **PARAM_HVP_VSTEP**: the value of the high voltage step for the HV Plateau in V. The type value should be unsigned int.
- **PARAM_HVP_DELAY**: the value of the waiting time between the measurements for the HV Plateau in ms. The type value should be unsigned int.
- **PARAM_HVP_MTIME**: the measurement time for the HV Plateau in s. The type value should be unsigned int.
- **PARAM_HVP_ENABLEMASK**: the enablemask of channels that are active for the HV Plateau measurement. The type value should be unsigned int.

2.7 Attended Operation

The CAENSRR7780 library provides advanced functions to handle R7780 devices working in attended mode. The detected pulse train can be reconstructed by decoding the device raw data buffer. The binary raw data can be directly retrieved from the device and the user is free to decide how to use them. For example, the user can decide to get the raw data and store them to a binary file on its computer for later analysis, or to decode them online to analyse the pulse train. The library provides one function to read the binary raw data and one function to decode the raw data word by word and get events data.

2.7.1 ReadData

This function can be used to get the binary raw data from the device. Some general info are also obtained from the read. The readout buffer should already be allocated when the function is called. The maximum amount of data that can be received in a single read is defined in the library as RAW_DATA_BUFF_SIZE. Return value is the error code (see error codes in Sec. **Return Codes**).

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_ReadData(SR_HANDLE handle,
                CAENSR_Data_t *data, uint32_t * datasize)

typedef struct {
    CAENSR_InfoData_t Info;
    char *data_buffer;
} CAENSR_Data_t;

typedef struct {
    uint16_t read_finished;
    double VMon;
    double IMon;
    uint32_t HVStatus;
    uint32_t DevStatus;
    uint32_t  GenericError;
    uint32_t FIFOFullError;
    uint32_t LVStatus;
    uint8_t LV_5V_Error;
    uint8_t LV_12V_Error;
} CAENSR_InfoData_t;
```

Parameter	I/O	Description
handle	Input	The handle of the device
*data	Input	Pointer to the CAENSR_Data_t struct
*datasize	Output	Pointer to the total size of data read from the device

2.7.2 DecodeWord

This function can be used to decode a binary data word read from the device. The buffer filled by CAENSR_ReadData is made of 4 bytes unsigned words that contain the events to be decoded. The events buffer should already be allocated when the function is called. The decode function returns also a timetag value, this information is very useful because it represents the total time elapsed since acquisition has started. It must be taken into account that some kinds of words do not contain events but simply provide a time information or info about errors. Return value is the error code (see error codes in Sec. **Return Codes**).

The CAENSR_Events_Buffer_t contains events data as decoded from the binary words. The channel number that detected the event is stored, together with the corresponding event timestamp and flags.

Some specific bits of the event flags should be checked because they contain some information about the event itself:

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_DecodeWord(uint32_t word,
                  CAENSR_Events_Buffer_t *EvBuffer,
                  uint32_t *NEvents,
                  uint64_t *LastEvTimetag
                  );


typedef struct {
    uint64_t *Timetag;
    uint16_t *ch;
    uint16_t *flags;
    uint16_t *error_info;
    uint32_t size;  //size in events number
    uint32_t occupancy;
} CAENSR_Events_Buffer_t;
```

Parameter	I/O	Description
word	Input	The binary word to be decoded
*EvBuffer	Input	Pointer to the pre-allocated events buffer that will be filled with the events decoded from the word
*NEvents	Output	The total number of events decoded from the word
*LastEvTimetag	Output	The last time measured by the device (sometimes it can coincide with the timetag of the last decoded event)

CAENSR_Events_Buffer_t field	Description
*Timetag	Pointer to the array filled with event Timestamps
*ch	Pointer to the array filled with the channels from which events are collected
*flags	Pointer to the array filled with event flags
*error_info	Pointer to the array filled with errors info. To be used only if the flag indicates a generic error event.

- FLAG_GENERIC_ERROR (bit 0): the event is an error event. In this case the error_info field specifies the error type:
  - bit 0: if this bit is 1 the error is a HV error.
  - bit 1: if this bit is 1 the error is due to a channel FPGA FIFO that is full.
  - bit 2: if this bit is 1 the error is due to the final FPGA FIFO that is full.
  - bit 3: if this bit is 1 the error is due to a problem with the FPGA time counter rollover.
- FLAG_STOP_ACQ (bit 1): the event is a STOP event. Data read should continue until the STOP event has been decoded.

The library also provides the needed functions to allocate, clear and free the events buffer.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_MallocEventsBuffer(CAENSR_Events_Buffer_t *EvBuffer, uint32_t size);

CAENSR_RetCode_t
CAENSR_DLLAPI CAENSR_FreeEventsBuffer(CAENSR_Events_Buffer_t *EvBuffer);

CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_ClearEventsBuffer(CAENSR_Events_Buffer_t* EvBuffer, uint32_t size);
```

2.7.3 GetDeviceError

This function can be employed to check whether the device has errors and to know the error type. In particular, it should be called in case an error event is decoded from the binary raw buffer.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetDeviceError(SR_HANDLE handle,
                      uint32_t *generic_error, uint32_t *fifo_status)
```

Parameter	I/O	Description
handle	Input	The handle of the device
*generic_error	Output	The error word
*fifo_status	Output	The status of the channels FIFOs; every bit is associated to the corresponding input channel number and if it is 1 the corresponding channel FIFO is full

The bits of the generic error word provide info about the kind of error that may be reported by the device; the word can be decoded according to the following error masks:

```
//Device Generic Error masks
#define HV_ERROR                0x1
#define CH_FFULL_ERROR          0x2
#define FINAL_FFULL_ERROR       0x4
#define ROLLOVER_ERROR          0x8
#define TRGFIFO_ERROR           0x10
#define SCALER_OVF_ERROR        0x20
#define DISTRIB_OVF_ERROR       0x40
#define DISTRIB_OUT_ERROR       0x80
#define TSTBIT_ERROR            0x100
```

2.7.4 GetDeviceGlobalStatus

This function allows to get all the general status and monitor info from the device and fills a CAENSR_InfoData_t struct.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetDeviceGlobalStatus(SR_HANDLE handle, CAENSR_InfoData_t* data);
```

Parameter	I/O	Description
handle	Input	The handle of the device
*CAENSR_InfoData_t	Output	The monitor and status info filling the struct

2.8 Unattended Operation

The CAENSRR7780 library provides also advanced functions specific for R7780 devices working in unattended mode, that allow to get info about the data analysis status and to retrieve the analysis results.

2.8.1 GetMeasurementData

This function allows to monitor the status of a measurement even if still in progress and to get some results data.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetMeasurementData(SR_HANDLE handle,
                          uint64_t *Time,
                          uint64_t *Totals,
                          uint64_t *ReA,
                          uint64_t *A,
                          uint8_t *status
                          );
```

Parameter	I/O	Description
handle	Input	the handle of the device
*Time	Output	the time elapsed since the measurement was started (in 10 ns unit with usb or ethernet connection, in 0.1 s unit with serial connection)
*Totals	Output	total number of processed events
*ReA	Output	total number of accumulated coincidence counts in the ReA gate
*A	Output	total number of accumulated coincidence counts in the A gate
status	Output	measurement status (1=running, 0=stopped)

2.8.2 GetChTotals

This function allows to read the number of events detected by one of the input channels (number of triggers) even if acquisition is running.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetChTotals(SR_HANDLE handle,
                   uint8_t ch,
                   uint64_t *Totals
                   );
```

Parameter	I/O	Description
handle	Input	the handle of the device
ch	Input	the input channel number
*Totals	Output	total number of detected events

2.8.3 GetCounts

This function allows to read the number of events detected by all the input channels (number of triggers) even if acquisition is running. The Chcounts buffer should be allocated to MAX_CHANNELS before calling the function.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetCounts(SR_HANDLE handle,
                 uint64_t* Totals, uint64_t *ChCounts);
```

Parameter	I/O	Description
handle	Input	the handle of the device
*Totals	Output	total number of detected events
*ChCounts	Output	the number of events detected by the input channels

2.8.4 GetFinalCounts

This function allows to read the final number of events detected by all the input channels (number of triggers) when a data acquisition is complete. The Chcounts buffer should be allocated to MAX_CHANNELS before calling the function.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetFinalCounts(SR_HANDLE handle,
                      uint64_t* ChCounts);
```

Parameter	I/O	Description
handle	Input	the handle of the device
*ChCounts	Output	the final number of events detected by the input channels

2.8.5 GetNRunsCounter

This function allows to get the current acquisition run number when data acquisition is running.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetNRunsCounter(SR_HANDLE handle,
                       uint64_t* counter);
```

Parameter	I/O	Description
handle	Input	the handle of the device
*counter	Output	the current measurement run

2.8.6 GetDistributions

This function allows to read the ReA and A multiplicity distributions measured by the device even if the measurement is still in progress. Data are from all the channels together, according to the coincidence counting eneblemask.

```
CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_GetDistributions(SR_HANDLE handle,
                         uint16_t *nbins, uint32_t *ADistrib, uint32_t *ReADistrib);
```

Parameter	I/O	Description
handle	Input	Handle of the device
ch	Input	Input channel number (for single channel distributions)
*nbins	Output	Total number of bins used to build the distributions
*ADistrib	Input	Pointer to the pre-allocated array of size MAX_NBINS that will store the A distribution
*ReADistrib	Input	Pointer to the pre-allocated array of size MAX_NBINS that will store the ReA distribution

2.9 Files management

The following functions allow to set the data save path, to change the base name of the output files and to manage the SD cards.

2.9.1 Get/SetFilenamePrefix

These functions allow to get/set the filename prefix for both data and log files. The char array should already be allocated to FILENAME_MAX_SIZE before calling the Get function.

```
CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_GetFilenamePrefix(SR_HANDLE handle, char* filename);

CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_SetFilenamePrefix(SR_HANDLE handle, char *filename);
```

Parameter	I/O	Description
handle	Input	the handle of the device
*filename	Input/Output	the base filename of the files

2.9.2 Get/SetFilesPath

These functions allow to get/set the save path for data, log and config files. The path must be relative to the SD cards root folder and must start and end with / (for example /savepath/). The default save path is the root folder itself (/). The char array should already be allocated to FILENAME_MAX_SIZE before calling the Get function.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetFilesPath(SR_HANDLE handle, char* path);

CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_SetFilesPath(SR_HANDLE handle, char* path);
```

Parameter	I/O	Description
handle	Input	the handle of the device
*path	Input/Output	the save path of files

2.9.3 Get/SetAutomaticFileDeleteD

These functions allow to get/set the maximum age of the files for automatic deletion, in days. At every restart, the device automatically checks if some files older than this age are present on the SD cards and if yes deletes them.

```
CAENSR_RetCode_t CAENSR_DLLAPI
SetAutomaticFileDeleteD(SR_HANDLE handle, uint16_t days);

CAENSR_RetCode_t CAENSR_DLLAPI
GetAutomaticFileDeleteD(SR_HANDLE handle, uint16_t *days);
```

Parameter	I/O	Description
handle	Input	the handle of the device
(*)days	Input/Output	the limit age of the files in days

2.10 NTP management

The following functions allow to configure the NTP IP address and to force a manual update. However, the NTP is fully managed by the web interface, that allows to manage it more easily.

2.10.1 Get/SetNTPIP

These functions allow to get/set the IP address of the NTP server for time synchronization and to update it.

```
CAENSR_RetCode_t CAENSR_DLLAPI
CAENSR_GetNTPIP(SR_HANDLE handle, char* IP);

CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_SetNTPIP(SR_HANDLE handle, char* IP);

CAENSR_RetCode_t CAENSR_DLLAPI
 CAENSR_UpdateNTP(SR_HANDLE handle);
```

Parameter	I/O	Description
handle	Input	the handle of the device
(*)IP	Input/Output	the NTP IP address

2.11 Communication Protocols

Communication with a R7780 device can take place according to a ethernet or serial protocol. A server runs on the device internal CPU and waits for input commands from the connecting client.

2.11.1 Ethernet Procotol

In the Ethernet protocol, both the client and the server packets start with a header that is made of 4 sections:

- *protocol version*: 0xCAE7 (2 bytes word)
- *size*: the total size of the packet (4 bytes word)
- *command code*: the command to send (2 bytes word)
- *number of parameters*: the total size of the packet (4 bytes word)

In the client package, the header is followed by the parameters or data sent to the server. The server reply package contains also an error code after the header (4 bytes word) that specifies whether the command has been executed correctly or not.

2.11.2 Serial Procotol

The Serial protocol is the same used by INCC software for compatibility with the existing devices [1]. All the standard interactive mode commands for controlling the instrument in real time are supported, but some extra commands have been implemented for the R7780 devices. The new commands allow to set some specific parameters that are not supported by other devices or to get specific info from the device. In the following table *x* represents one byte, *c* represents one character.

Command	Reply	Explaination
tr		Send a software trigger
c		Clear the device memory
BR	*xx*	Read the number of bins for multiplicity distributions
BW*xx*		Set the number of bins for multiplicity distributions
QR	*xx*	Get long delay (value in ns = *xx* *250)
QW*xx*		Set long delay (value in ns = *xx* *250)
iF	*xxxx*	Get FW release
iN	*xxxx*	Get serial number
iC	*xxxx*	Get number of input channels
iH	*xxxx*	Get number of HV channels
iP	*xxxx*	Get PCB version
iV	*c*	Get HV polarity: P=positive, N=negative, M=mixed
iM	*xxxx*	Get Model
iE	*xxxx*	Get device error
iW	*xxxx*, *x*, *x*, *x*, *xx*, *xx*	Get HV info: number of HV channels, polarity code, number of HV ranges, active range, Vmax, Max HVRamp
MR*xxxx*	*xxxxxxxx*, *xxxx* ...	Get Rates of run: (8 bytes base, 4 bytes power) for singles, doubles and triples
Mr*xxxxxxxx*	*xxxxxxxx*, *xxxx* ...	Get Rates of run for a channel: (8 bytes base, 4 bytes power) for singles, doubles and triples
pr+*param*	*xxxx*	Get the value of parameter *param*(see param codes definition)
pw+*paramxxxx*		Set the value of parameter *param*(see param codes definition)
pru*x*	*xxxx*	Get the value of HV rampup/down in V/s for a given HV channel
pwu*xxx*		Set the value of HV rampup/down in V for a given HV channel
pri*x*	*xxxx*	Get the monitored current in µA for a given HV channel
prh*x*	*xxxx*	Get the status a given HV channel

The following parameter codes are supported for commands pr and pw:

- T: acquisition cycle time
- r: number of acquisition cycles
- a: channels aggregation mode
- e: channel enablemask
- g: channel groups mask
- s: channel groups size
- o: device operating mode

[1] See section Interactive Mode Commands of the INCC software manual.

2.12 The CAENSR_Demo

A demo software is released together with the CAENSRR7780 library. Through simple examples it shows how to employ the main library functions to communicate with the device. The user can manage the source files and modify them to connect a device, configure it and start acquisition. The demo source file contains comments that guide the user in the configuration steps. For example, the following figure shows the connection section: the device path should be specified according to the connection type.

```
//open the device according to the connection type
char path[] = "eth:10.0.0.10";  //ETHERNET connection -> provide IP address    (in this example the IP address is 10.0.0.10)
//char path[] = "ser:COM4";          //SERIAL connection   -> provide serial port   (in this example the serial port in COM4)

const char* pkey_filename = "R7780Public_key.pem";
char* public_key = read_public_key(pkey_filename);

if (public_key) {
    printf("Public key loaded:\n%s\n", public_key);
}
else {
    fprintf(stderr, "Error loading public key\n");
}

ret = CAENSR_OpenDeviceSecure(path, ADMIN, "admin", CAENSR_TRUSTED, public_key, &device); //new TRUSTED connection mode using signed packages
if (ret != CAENSR_RetCode_Success)
    goto QuitProgram;
```

The main screen of the console demo software is shown in Fig. **2.1**; if the connection is successfully established, the key information about the connected device is shown. If the device is already operating in unattended mode a warning message is shown. In this case the special commands for UNATTENDED working mode become active and it is possible to monitor the analysis status and to get results.

If the device is operated in ATTENDED mode, the demo shows how to collect raw data from the device and uses them to build the time difference (DeltaT) distributions and to plot them using gnuplot. The user is free to customize the data analysis section and to manage the collected data in a different way.

```
Connected to device model 7771, with serial number 24.
FPGA FW revision 0x37300005, PCB revision 1
Number of channels 32, number of HV channels 1
HV polarity: P.
Device operating mode is 1

Acquisition is already running!


S : Start acquisition
s : Stop acquisition
W : Send a software trigger
c : Clear Device memory
C : Reset Device configuration

O : Change device Operating mode

k : Get device clock time
K : Set device clock time

P : Get parameters

G : Set new Gate value
p : Set new PreDelay value
L : Set new LongDelay value
T : Set new Measurement time
B : Set new Multiplicity Bins

A : Set new Channel Aggregation mode

***** For ATTENDED working mode only *****
F : Print counting statistics
E : Plot DeltaT distribution

***** For UNATTENDED_A working mode only *****
t : Read Totals of a single channel
M : Read Measurement data
d : Read Single channel Multiplicity Distributions
D : Read Multiplicity Distributions

Z : Get Rates results (available when the measurement is complete)
z : Get Single channel Rates results (available when the measurement is complete)

r : Reset Measurement
R : Reset Measurement and configuration

V : Switch to HV Menu

x : Print Menu
q : Quit client
```

Fig. 2.1: Demo program main menu.

3 Technical Support

To contact CAEN specialists for requests on the software, hardware, and board return and repair, it is necessary a MyCAEN+ account on www.caen.it:

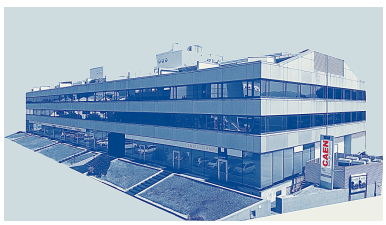
https://www.caen.it/support-services/getting-started-with-mycaen-portal/

All the instructions for use the Support platform are in the document:

A paper copy of the document is delivered with CAEN boards.
The document is downloadable for free in PDF digital format at:

https:/www.caen.it/safety-information-product-support

CAEN S.p.A.
Via Vetraia 11
55049 - Viareggio
Italy
Phone +39 0584 388 398
Fax +39 0584 388 959
info@caen.it
www.caen.it



CAEN GmbH
Brunnenweg 9
64331 Weiterstadt
Germany
Phone +49 212 254 40 77
Fax +49 212 254 40 79
info@caen-de.com
www.caen-de.com

CAEN Technologies, Inc.
1 Edgewater Street - Suite 101
Staten Island, NY 10305
USA
Phone: +1 (718) 981-0401
Fax: +1 (718) 556-9185
info@caentechnologies.com
www.caentechnologies.com

CAENspa INDIA Private Limited
B205, BLDG42, B Wing,
Azad Nagar Sangam CHS,
Mhada Layout, Azad Nagar, Andheri (W)
Mumbai, Mumbai City,
Maharashtra, India, 400053
info@caen-india.in
www.caen-india.in

UM9815 - CAEN SRR7780 - Software Library for Shift Register R7780 rev. 1 - January 24th, 2025 00100/20:R7780.MUTX/00