Rev. 1 - February 7th, 2024

R7771

API Library

CAEN
Tools for Discovery

User Manual
UM9828

Purpose of this Manual

This document contains the description of the R7771 API library functions.

Change Document Record

Date	Revision	Changes
October 30th, 2023	00	Initial Release
February 7th, 2024	01	Modified Linux installation (Sec. **1.2**).

Symbols, Abbreviated Terms and Notation

PTR	Pulse Train Recorder
MCS	Multi Channel Scaler
HV	High Voltage
DDT	Differential Die-away Technique

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 The R7771 API

The CAEN R7771 device comes with a library of high level APIs that allow to configure the device, to manage acquisition, to save the collected data to file, to analyse them, and also to get the raw data from the boards. The library can be employed to develop custom readout programs for controlling devices working in attended mode. As described later in this manual, the library comes with a server program that must be running in background. This server is the R7771_ControlSoftware available on the CAEN website. It can be run normally as a standalone software (in this case it provides a user friendly GUI), or it can be run in 'demone' mode providing D as the input argument. If the server is running on a different computer with respect to the library, the computer IP address must be provided to the R7771 API library (see the Init function). Otherwise, a new instance of the server is created locally by the library Init function itself.

1.1 Windows

An installer is provided for installing the R7771 library. The default install path is: C:\Program Files\CAEN\R7771_API. The setup includes also a simple demo software written in C language that shows how to use the library (see section 1.10). The R7771 API setup copies also the server with all its dependencies in the r7771_service subfolder inside the library bin folder. For custom software development, it is necessary to copy the full library bin folder to the custom software directory, not only the library .dll file.

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 libr7771_api-X.Y.Z 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*.

After installing the library, the R7771 service must be installed: enter the r7771_service folder located inside the library folder and run the install script ("sudo sh install.sh").

The library package contains also a simple demo software (API_Demo) that shows how to use the library with simple examples. A *Makefile* is provided to build the demo software.

NOTE: do not delete the library folder once installed.

1.3 Return Codes

Error code	Value	Error explanation0
R7771_API_Success	0	No Error
R7771_API_ErrGeneric	-1	Generic error
R7771_API_SockInit	-2	Socket initialization error
R7771_API_SockConnect	-3	Socket connect error
R7771_API_OutOfMemory	-4	Out of memory
R7771_API_Handle	-5	Invalid device handle
R7771_API_Argument	-6	Invalid argument
R7771_API_SocketSend	-7	Send error
R7771_API_SocketReceive	-8	Receive error
R7771_API_Protocol	-9	Protocol error
R7771_API_Serialize	-10	Serialize error
R7771_API_Deserialize	-11	Deserialize error
R7771_API_Parameter	-12	Parameter error
R7771_API_ParameterValue	-13	Parameter value error
R7771_API_ParameterType	-14	Parameter type error
R7771_API_LibraryLoad	-15	Library dynamic load error (Linux only)
R7771_API_NotConnected	-16	Not connected
R7771_API_NotSupported	-17	Not supported
R7771_API_NotYetImplemented	-18	Not yet implemented
R7771_API_AlreadyConnected	-19	Device already connected
R7771_API_MaxConnections	-20	Max number of devices connected
R7771_API_OperatingMode	-21	Device operating mode not compliant
R7771_API_AcqIsOff	-22	Acquisition is off
R7771_API_AcqIsOn	-23	Acquisition is on
R7771_API_AnalysisInProgress	-24	Data analysis is still in progress
R7771_API_Connection_Refused	-25	Connection refused by device
R7771_API_ReadData	-26	Device reported a ReadData error
R7771_API_InvalidEvent	-27	Device found an invalid event
R7771_API_CommandFormat	-28	Wrong command format
R7771_API_InvalidArgument	-29	Invalid argument provided
R7771_API_WrongFPGAFirmware	-30	Device detected a wrong FPGA firmware
R7771_API_Timeout	-31	Timeout in communication with the server
R7771_API_NoData	-32	No data available to read
R7771_API_InternalDeviceError	-33	Connected device reported an internal error

1.4 Library Init

This function starts the communication with the R7771 server and must always be called before using any of the API functions. Return value is the error code (see error codes in Sec. **Return Codes**). If the server is already running on a different computer, the connection can be established by providing the IP address. If the localhost IP address (127.0.0.1) is provided as input value, a new instance of the R7771 server is launched locally on the computer. The server manages the raw data readout and analysis in background and the direct communication with the device.

```
R7771_API_ErrorCode_t R7771_DLLAPI Init(const char* sw_path);
```

Parameter	I/O	Description
*sw_path	Input	Path to the R7771 server: IP address of the server for ethernet connection.

1.5 Device Connection

A R7771 device can be connected to the PC via Ethernet or via USB. The following API functions allow to establish a new connection or to close the connection with a device. The connection is managed by the server running in background.

1.5.1 OpenDevice

These functions open the device. Return value is the error code (see error codes in Sec. **Return Codes**).

```
//in case of ethernet connection provide the IP address "xxx.xxx.xxx.xxx"
R7771_API_ErrorCode_t R7771_DLLAPI ConnectDeviceETH(const char* path);

//in case of usb connection provide the device PID
R7771_API_ErrorCode_t R7771_DLLAPI ConnectDeviceUSB(int PID);
```

Parameter	I/O	Description
*path	Input	Path to the device: IP address of the device for ethernet connection.
PID	Input	PID of the device for USB connection.

1.5.2 CloseDevice

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

```
R7771_API_ErrorCode_t R7771_DLLAPI DisConnectDevice();
```

1.6 Device Info

1.6.1 GetDevInfo

Several functions are available to get some general information from the device when it is connected to the server. Return value is the error code (see error codes in Sec. **Return Codes**).

```
R7771_API_ErrorCode_t R7771_DLLAPI GetDeviceNChannels(uint32_t *NCh);

R7771_API_ErrorCode_t R7771_DLLAPI GetDeviceNHVChannels(uint32_t*NHVCh);

R7771_API_ErrorCode_t R7771_DLLAPI GetDevicePID(uint32_t*PID);

R7771_API_ErrorCode_t R7771_DLLAPI GetDeviceFWRev(uint32_t *Rev);

R7771_API_ErrorCode_t R7771_DLLAPI GetDevicePath(char *path);

R7771_API_ErrorCode_t R7771_DLLAPI GetDeviceHVPolarity(char *pol);

R7771_API_ErrorCode_t R7771_DLLAPI GetDeviceMaxHV(uint32_t*MaxV);

R7771_API_ErrorCode_t R7771_DLLAPI GetDeviceMaxHVRamp(uint32_t* MaxVRamp);
```

Parameter	I/O	Description
*NCh	Output	Pointer to the variable to store the number of input channels
*NHVCh	Output	Pointer to the variable to store the number of HV channels
*PID	Output	Pointer to the variable to store the PID
*Rev	Output	Pointer to the variable to store the FPGA firmware revision
*path	Output	Pointer to the variable to store the connection path
*pol	Output	Pointer to the variable to store the HV channel polarity
*MaxV	Output	Pointer to the variable to store the maximum HV value
*MAxVRamp	Output	Pointer to the variable to store the maximum HV ramp value

1.7 Commands

Communication with the R7771 server provides a well defined set of commands. The server is configured to perform specific operations or to provide the requested data depending on the received command.

1.7.1 ExecCommand

The general ExecCommand function allows to execute the commands. Some commands can be used only when a device is connected in order to manage the hardware settings, to control data acquisition and to use the HV channel. Some other commands are relative to the data analysis, that is managed by the server, and to data save. Return value is the error code (see error codes in Sec. **Return Codes**).

```
R7771_API_ErrorCode_t R7771_DLLAPI ExecCommand(R7771_API_Command_t command, void *input_data, void *output_data);
```

Parameter	I/O	Description
command	Input	Command to send (see R7771_API_Command_t)
*input_data	Input	Pointer to the value to send (NULL in case no value is associated to the command)
*output_data	Output	Pointer to the value to receive (NULL in case no value is associated to the command)

The supported commands are listed under the R7771_API_Command_t enum, but some of them are only supported when the device is working in PTR or MCS mode. In the following, a brief description is provided for every command.

- **CMD_DEV_HVONOFF**: Switch on/off the HV channel of the connected device. The input_data should be a pointer to a uint8_t value (0 or 1 to switch off or on respectively).
- **CMD_DEV_ACQONOFF**: Start/Stop acquisition for the connected device. The input_data should be a pointer to a uint8_t value (0 or 1 to stop or start respectively).
- **CMD_DEV_CLEAR_MEMORY**: Clear the connected device memory.
- **CMD_DEV_CONFIG_RESET**: Reset the connected device configuration to default.
- **CMD_DEV_CONFIG_SAVE**: Save the current configuration to the connected device internal memory.
- **CMD_DEV_START_RAWACQ**: Start raw data acquisition for the connected device. The server will not save raw data to output files and it will be possible to read them directly (see section 1.9.4).
- **CMD_DEV_STOP_RAWACQ**: Stop raw data acquisition for the connected device (see section 1.9.4).
- **CMD_OUTFILE_FORMAT_GET**: Get the format of the output file, created by the server, that will store the raw acquisition data. The output_data should be a pointer to a uint8_t value (0 for .r77, 1 for .bin, 2 for .xml).
- **CMD_OUTFILE_FORMAT_SET**: Set the format of the output file that will be created by the server and will store the raw acquisition data. The input_data should be a pointer to a uint8_t value: 0 for .r77, 1 for .bin (allowed when the device works in PTR mode), 2 for .xml (allowed with the device working in MCS mode)).
- **CMD_OUTFILE_NAME_GET**: Get the name of the file (only file name with extension) that is currently storing the raw data from the device, or has stored the data of the last acquisition. The output_data should be a pointer to a char array of size *OUTFILE_STRLEN*.
- **CMD_OUTFILE_NAME_SET**: Set the name of the file (only file name with extension) that will store the raw data from the device. The input_data should be a pointer to a char array of size *OUTFILE_STRLEN*.
- **CMD_OUTFILE_DIR_GET**: Get the directory of the file (full path) that is currently storing the raw data from the device, or has stored the data of the last acquisition. The output_data should be a pointer to a char array of size *OUTFILE_STRLEN*.

- **CMD_OUTFILE_DIR_SET**: Set the directory (full path) where the raw data file will be saved. The input_data should be a pointer to a char array of size *OUTFILE_STRLEN*.
- **CMD_OUTFILE_ENABLE_GET**: Get the status of the acquisition data save. The output_data should be a pointer to a uint8_t value (0 if data save is disabled, 1 if it is enabled).
- **CMD_OUTFILE_ENABLE_SET**: Enable/Disable raw data save. The input_data should be a pointer to a uint8_t value (0 to disable data save, 1 to enable it). If data save is enabled the server will create a file named OUTFILE_NAME inside the OUTFILE_DIR.
- **CMD_INFILE_NAME_GET**: Get the name of the input file (full path including file name with extension) that will be analysed or has been analysed by the server. The output_data should be a pointer to a char array of size *INFILE_STRLEN*.
- **CMD_INFILE_NAME_SET**: Set the name of the file (full path including file name with extension) that will be analysed by the server. The input_data should be a pointer to a char array of size *INFILE_STRLEN*.
- **CMD_ANALYSIS_START**: Start raw data file analysis according to the input file type (pulse train or MCS histograms). The server will analyse the file provided as input and the analysis routine will be coincidence counting or differential die away depending on the file format.
- **CMD_ANALYSIS_STOP**: Stop the current raw data file analysis.
- **CMD_GENERAL_STATUS_GET**: Get the current global status: 0 when the system is ready, 1 when data acquisition is running without errors, 2 when data acquisition is running with errors, 3 when data analysis is running without errors, 4 when data analysis is running with errors.
- **CMD_ONLINE_ANALYSIS_ENABLE_GET**: Get the status of online data analysis. The output_data should be a pointer to a uint8_t value (0 if online data analysis is disabled, 1 if it is enabled). By default, it is supposed that data analysis follows data acquisition: when data acquisition is complete, the obtained raw data is analysed. The online analysis option allows to program the server in order to analyse the raw data while data acquisition is running. This option could slow down the system and it is not recommended for count rates higher than 1 MHz.
- **CMD_ONLINE_ANALYSIS_ENABLE_SET**: Enable/Disable online data analysis. The input_data should be a pointer to a uint8_t value (0 to disable online analysis, 1 to enable it).
- **CMD_MCS_CALIBFILE_NAME_GET**: Get the name of the MCS calibration file (full path including file name with .xml extension) that will be loaded during data analysis. The output_data should be a pointer to a char array of size *INFILE_STRLEN*.
- **CMD_MCS_CALIBFILE_NAME_SET**: Set the name of the MCS calibration file (full path including file name with .xml extension) that will be loaded during data analysis. The input_data should be a pointer to a char array of size. *INFILE_STRLEN*
- **CMD_PTR_CALIBFILE_NAME_GET**: Get the name of the PTR calibration file (full path including file name with .xml extension) that will be loaded during data analysis. The output_data should be a pointer to a char array of size *INFILE_STRLEN*.
- **CMD_PTR_CALIBFILE_NAME_SET**: Set the name of the MCS calibration file (full path including file name with .xml extension) that will be loaded during data analysis. The input_data should be a pointer to a char array of size *INFILE_STRLEN*.
- **CMD_MCS_CALIB_MATRIX_GET**: Get the name of the matrix whose calibration has been loaded for MCS data analysis.
- **CMD_MCS_CALIB_MATRIX_SET**: Set the name of the matrix whose calibration file will be loaded during MCS data analysis.
- **CMD_PTR_CALIB_MATRIX_GET**: Get the name of the matrix whose calibration has been loaded for PTR data analysis.
- **CMD_PTR_CALIB_MATRIX_SET**: Set the name of the matrix whose calibration has been loaded for PTR data analysis.

1.8 Device Parameters

1.8.1 Get and Set Device Parameter

The R7771 devices can be configured by means of a set of key parameters. 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**).

```
R7771_API_ErrorCode_t R7771_DLLAPI GetDeviceParameter(R7771_API_DevParam_t dparam, uint32_t *value);

R7771_API_ErrorCode_t R7771_DLLAPI SetDeviceParameter(R7771_API_DevParam_t dparam, uint32_t value);
```

Parameter	I/O	Description
dparam	Input	Parameter to be read/set (see allowed values for R7771_API_DevParam_t)
*value	Output	Pointer to the parameter value that is read from the device.
value	Input	Parameter value to set.

In the following a brief description of the parameters can be found and the corresponding allowed values are listed.

- **DEVPARAM_WMODE** (readonly parameter): the working mode. Value is 0 corresponding to ATTENDED working mode.
- **DEVPARAM_AMODE**: the acquisition mode (0 = PTR, 1 = MCS)
- **DEVPARAM_VSET**: the HV value (V unit).
- **DEVPARAM_VRAMP**: the HV ramp value (V/s unit).
- **DEVPARAM_VMON** (readonly parameter): the HV monitored value (V unit).
- **DEVPARAM_IMON** (readonly parameter): the HV monitored current (µA unit).
- **DEVPARAM_HVSTATUS** (readonly parameter): the HV status (see allowed values for R7771_API_HVStatus_t).
- **DEVPARAM_CHMASK**: the input channels enablemask (32 bit hex value).
- **DEVPARAM_MTIME**: the length of an acquisition cycle when in PTR mode (0.1 s unit).
- **DEVPARAM_NRUNS**: the total number of acquisition cycles in PTR mode.
- **DEVPARAM_DWELL**: the dwell time for the MCS mode (dwell time resolution units).
- **DEVPARAM_DWELLRES**: the dwell time resolution when in MCS mode (0 = 10 ns, 1 = 1 µs, 2 = 1 ms).
- **DEVPARAM_DELAY**: the delay time for the MCS mode (delay time resolution units).
- **DEVPARAM_DELAYRES**: the delay time resolution when in MCS mode (0 = 10 ns, 1 = 1 µs, 2 = 1 ms).
- **DEVPARAM_NMCSBINS**: the number of bins used for the MCS histograms.
- **DEVPARAM_NTRG**: the number of sweeps for the MCS data acquisition (number of triggers on channel 1).
- **DEVPARAM_MSUBF**: enable/disable monthly subfolders for internal data save.
- **DEVPARAM_ACQSTATUS**: device data acquisition status (0 = not running, mask of the active channels when acquisition is active)
- **DEVPARAM_CONNSTATUS**: device connection status (0 = not connected, 1 = connected)
- **DEVPARAM_MCSTRGCOUNT**: number of input triggers detected on channel 1 in MCS mode, value is updated while acquisition is running.
- **DEVPARAM_INPUTTHR**: channels input threshold (mV unit).

R7771_API_HVStatus_t	Value	Error explanation0
HVSTATUS_OFF	0	HV channel is off
HVSTATUS_OFF	1	HV channel is on
HVSTATUS_RAMPUP	2	HV ramp up active
HVSTATUS_RAMPDOWN	2	HV ramp down active
HVSTATUS_SETTINGUP	2	HV channel setting up
HVSTATUS_OVC	2	HV channel over current
HVSTATUS_OVT	2	HV channel over temperature
HVSTATUS_TWARN	2	HV channel temperature warning
HVSTATUS_INHIBIT	2	HV channel in inhibit
HVSTATUS_MAXV	2	HV channel at max V

1.9 Data Analysis

The R7771 API library provides advanced functions to be used to get the data analysis results or to directly access the raw data collected by the device. The R7771 server can be programmed to manage the device raw data (case 1), or to provide them directly (case 2):

- **case 1** In this case, the server can save the raw data to files (if data save is enabled), it can analyse them while data acquisition is running (if online data analysis is enabled), or it can analyse previously saved files. A set of analysis parameters can be used to configure data analysis and a set of functions allow to get the analysis results. A pdf report file is also saved when data analysis is complete.
- **case 2** In this case, the user is free to manage, decode and analyse the raw data for custom applications. A set of specific functions are available for data readout and decoding.

1.9.1 Get and Set Analysis Parameter

The R7771 data analysis can be configured by means of a set of parameters. 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**).

```
R7771_API_ErrorCode_t R7771_DLLAPI GetAnalysisParameter(R7771_API_AnalysisParam_t dparam, uint32_t* value);
R7771_API_ErrorCode_t R7771_DLLAPI SetAnalysisParameter(R7771_API_AnalysisParam_t dparam, uint32_t value);
```

Parameter	I/O	Description
dparam	Input	Parameter to be read/set (see allowed values for R7771_API_AnalysisParam_t)
*value	Output	Pointer to the parameter value that is read.
value	Input	Parameter value to set.

In the following a brief description of the parameters can be found and the corresponding allowed values are listed.

- **PARAM_CMODE**: counting mode for coincidence counting (0 = Counting, 1 = Coincidence, 2 = Multiplicity, 3 = RossiAlpha).
- **PARAM_INPUTSCC**: coincidence counting channels enablemask (32 bit hex value where bit n corresponds to channel n: bit n = 1 if channel n is enabled, 0 if disabled).
- **PARAM_INPUTAGGR**: channels aggregation (0 = ALL, 1 = INDIVIDUAL, 2 = GROUPED).
- **PARAM_GSIZE**: channels group size if aggregation is set to GROUPED (1,2,4,8,16,32).
- **PARAM_GCC**: coincidence counting groups enablemask (hex value where bit n corresponds to group n: bit i = 1 if the group is enabled, 0 if disabled).
- **PARAM_PDEL**: predelay value for coincidence counting (10 ns unit).
- **PARAM_GATE**: gate value for coincidence counting (10 ns unit).
- **PARAM_LDEL**: long delay value for coincidence counting (µs unit).
- **PARAM_NMULTBINS**: number of bins for the multiplicity distributions.
- **PARAM_RAWIN**: Rossi Alpha analysis window (10 ns unit).
- **PARAM_RASTEP**: Rossi Alpha analysis step (10 ns unit).
- **PARAM_DTSTEP**: deltaT distribution step (10 ns unit).
- **PARAM_STATUS**: data analysis status (1 = running, 0 = not running).
- **PARAM_MCS_CAL_STATUS**: MCS analysis calibration status (1 = active, 0 = disabled).

- **PARAM_PTR_CAL_STATUS**: PTR analysis calibration status (1 = active, 0 = disabled).
- **PARAM_MCS_ANALYSIS_MODE**: MCS data analysis mode (0 = DDT, 1 = basic).
- **PARAM_MCS_ROI_ENABLEMASK**: MCS ROIs enablemask (8 bit hex number where bit i is 1 if ROI i is enabled: 2 ROIs for every MCS histograms are defined).
- **PARAM_OBM_AIR**: OBM air coefficient for mass calculation according to the DDT technique (value multiplied by 1000).
- **PARAM_MASS_B_ERROR**: type B error (percent) to be added to mass error (value multiplied by 1000). This extra error is added to the statistical error when the mass is calculated.

Important Note: the R7771 device stores in its internal configuration the hardware settings and the most relevant analysis settings. When a device is connected, the server automatically loads all the stored settings so that data acquisition and analysis can be reconfigured according to the latest saved settings.

1.9.2 PTR mode analysis functions

1.9.2.1 GetCounts

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

```
R7771_API_ErrorCode_t R7771_DLLAPI GetCounts(int32_t Ch_or_Group, uint64_t* counts);
```

Parameter	I/O	Description
Ch_or_Group	Input	the input channel number (-1 for the total collected events)
*counts	Output	total number of detected events

1.9.2.2 GetDeltaTDistrib

This function allows to read the DeltaT distribution (distribution of the time differences between one event and the next) even if the data analysis is still in progress. Data are from one channel or from all the channels together, according to the channels aggregation mode.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetDeltaTDistrib(int32_t Ch_or_Group, uint32_t * nbins, uint64_t* data);
```

Parameter	I/O	Description
Ch_or_Group	Input	Input channel number (-1 for the total distribution built merging data from all channels)
*nbins	Output	Total number of bins used to build the distributions
*data	Input	Pointer to the pre-allocated array of size DISTRIB_MAX_NBINS that will store the DeltaT distribution

1.9.2.3 GetRossiAlphaDistrib

This function allows to read the Rossi Alpha distribution even if the data analysis is still in progress. Data are from one channel or from all the channels together, according to the channels aggregation mode. The distribution is available only if the counting mode is set to Rossi Alpha.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetRossiAlphaDistrib(int32_t Ch_or_Group, uint32_t* nbins, uint64_t* data);
```

Parameter	I/O	Description
Ch_or_Group	Input	Input channel number (-1 for the total distribution built merging data from all channels)
*nbins	Output	Total number of bins used to build the distributions
*data	Input	Pointer to the pre-allocated array of size ROSSI_ALPHA_MAX_NBINS that will store the distribution

1.9.2.4 GetRealsAndAccidentals

This function allows to get some coincidence counting results even if the analysis is still in progress. Results are available if the counitng mode is set to Coincidence, Multiplicity or Rossi Alpha.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetRealsAndAccidentals(int32_t Ch_or_Group, uint64_t* ReA, uint64_t* A);
```

Parameter	I/O	Description
Ch_or_Group	Input	the channel number (-1 for total results)
*ReA	Output	total number of accumulated coincidence counts in the ReA gate
*A	Output	total number of accumulated coincidence counts in the A gate

1.9.2.5 GetMultiplicityDistrib

This function allows to read the ReA and A multiplicity distributions even if the data analysis is still in progress. Data are from one channel or from all the channels together, according to the coincidence counting enablemask. Results are available if the counitng mode is set to Multiplicity or Rossi Alpha.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetMultiplicityDistrib(int32_t Ch_or_Group, uint32_t* nbins, uint64_t* ReAdistr, uint64_t* Adistr);
```

Parameter	I/O	Description
Ch_or_Group	Input	Input channel number (-1 for the total distribution built merging data from all channels)
*nbins	Output	Total number of bins used to build the distributions
*ADistrib	Input	Pointer to the pre-allocated array of size MULTIPLICITY_MAX_NBINS that will store the A distribution
*ReADistrib	Input	Pointer to the pre-allocated array of size MULTIPLICITY_MAX_NBINS that will store the ReA distribution

1.9.2.6 GetSDT

This function allows to read the Singles, Doubles and Triples rates calculated at the end of the data analysis. Data are from one channel or from all the channels together, according to the coincidence counting enablemask. Results are available if the counitng mode is set to Multiplicity or Rossi Alpha.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetSDT(int32_t Ch_or_Group, double* S, double* D, double* T);
```

Parameter	I/O	Description
Ch_or_Group	Input	Input channel number (-1 for the total results)
*S	Input	Pointer to the variable that will store the S result
*D	Input	Pointer to the variable that will store the D result
*T	Input	Pointer to the variable that will store the T result

1.9.3 MCS mode analysis functions

1.9.3.1 GetMCSHistograms

This function allows to read the 4 MCS histograms even if acquisition is running.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetMCSHistograms(uint32_t *bins, uint32_t *MCSdata);
```

Parameter	I/O	Description
*bins	Input	Pointer to the variable that will store the number of bins of one MCS histrogram
*MCSdata	Input	Pointer to the pre-allocated array (of size 4*MCS_MAX_NBINS) that will store the MCS histrograms

1.9.3.2 GetMCSHistoN

This function allows to read a single MCS histogram even if acquisition is running.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetMCSHistoN(uint32_t N, uint32_t* bins, uint32_t* MCSdata);
```

Parameter	I/O	Description
N	Input	Number of the MCS histrogram to read (0-3)
*bins	Input	Pointer to the variable that will store the number of bins of the MCS histrogram
*MCSdata	Input	Pointer to the pre-allocated array (of size MCS_MAX_NBINS) that will store the MCS histrogram

1.9.3.3 Get/SetMCSLabel

This function allows to read or set the label of the MCS histogram. This label is used in the results report.

```
R7771_API_ErrorCode_t R7771_DLLAPI SetMCSLabel(uint8_t MCS_histo, char *label);

R7771_API_ErrorCode_t R7771_DLLAPI GetMCSLabel(uint8_t MCS_histo, char* label);
```

Parameter	I/O	Description
MCS_histo	Input	Number of the MCS histrogram (0-3)
*label	Input or Output	Pointer to the variable that stores or that will store the label of the MCS histrogram

1.9.3.4 Get/SetROI

This function allows to read or set a ROI of one MCS histogram (the start and end channels). For every MCS histogram up to 2 ROIs can be defined.

```
R7771_API_ErrorCode_t R7771_DLLAPI SetMCSROI(uint8_t MCS_histo, uint8_t NROI, uint32_t roi_start, uint32_t roi_stop);
R7771_API_ErrorCode_t R7771_DLLAPI GetMCSROI(uint8_t MCS_histo, uint8_t NROI, uint32_t *roi_start, uint32_t *roi_stop);
```

Parameter	I/O	Description
MCS_histo	Input	Number of the MCS histrogram (0-3)
NROI	Input	Number of the ROI (0-1)
*roi_start	Input	Pointer to the variable that will store the start channel of the ROI
*roi_stop	Input	Pointer to the variable that will store the stop channel of the ROI
roi_start	Input	The start channel of the ROI
roi_stop	Input	The stop channel of the ROI

1.9.3.5 Get/SetROILabel

This function allows to read or set the label of a ROI of one MCS histogram. This label is used in the results report.

```
R7771_API_ErrorCode_t R7771_DLLAPI SetROILabel(uint8_t MCS_histo, uint8_t roi, char* label);
R7771_API_ErrorCode_t R7771_DLLAPI GetROILabel(uint8_t MCS_histo, uint8_t roi, char* label);
```

Parameter	I/O	Description
MCS_histo	Input	Number of the MCS histrogram (0-3)
roi	Input	Number of the ROI (0-1)
*label	Input or Output	Pointer to the variable that stores or that will store the label of the ROI

1.9.3.6 GetMass

This function allows to read the Mass value of $^{240}Pu_{eff}$ or ^{235}U calculated at the end of the PTR or MCS data analysis respectively. The Mass result is available only if a calibration file has been loaded and calibration has been enabled.

```
R7771_API_ErrorCode_t R7771_DLLAPI GetMass(double *m, double *m_error);
```

Parameter	I/O	Description
*m	Input	Pointer to the variable that will store the mass value (in grams)
*m_error	Input	Pointer to the variable that will store the mass error (in grams). This error includes the statistical error and the type B error.

1.9.3.7 AddToAnalysisReport

This function allows to enter a custom message that will be added to the pdf analysis report file. It can be used, for example, to add some info about the sample under measurement, or to add a comment about the current analysis. The maximum number of allowed characters is 1000 and is defined as INFO_MESSAGE_LEN.

```
R7771_API_ErrorCode_t R7771_DLLAPI AddToAnalysisReport(char* message);
```

Parameter	I/O	Description
*message	Input	Pointer to the array that stores the message

1.9.4 Raw Data Management

If the user has chosen to manage the device raw data, it is possible to decode them to get the relevant values depending on the device working mode (PTR or MCS). When the device works in PTR mode, the detected pulse train can be reconstructed by decoding the device raw data buffer. When it works in MCS mode, the raw buffer can be decoded to get the 4 MCS histograms. The library provides one generic function to read the binary raw data buffer and specific functions to decode the raw data and get events or MCS histograms.

1.9.4.1 GetRawDeviceData

This function can be used to get the binary raw data from the device. Some general info are also obtained from the read and a *R7771_Data_t* structure is filled. The readout buffer inside the struct 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 R7771_RAW_DATA_BUFF_SIZE. Return value is the error code (see error codes in Sec. **Return Codes**).

```
R7771_API_ErrorCode_t R7771_DLLAPI GetRawDeviceData(R7771_Data_t* data);
```

Parameter	I/O	Description
*data	Input	Pointer to the R7771_Data_t struct

1.9.4.2 DecodeR7771Word

This function can be used to decode a binary data word read from the device in PTR mode. In this case, the buffer filled by GetRawDeviceData 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 (see the R7771_MallocEventsBuffer function). 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**).

```
R7771_API_ErrorCode_t R7771_DLLAPI DecodeR7771Word(uint32_t word, R7771_Events_Buffer_t* EvBuffer, uint32_t* NEvents, uint64_t* LastEvTimetag);
```

The R7771_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:

- 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.

R7771_Data_t field	Description
read_finished	Data readout complete: 1 = ok, 0 = not all data have been read
VMon	Current monitored HV voltage value
IMon	Current monitored HV current value
HVStatus	HV channel status (hex value read from FPGA)
DevStatus	Device status (0 = data acquisition off, 1 = data acquisition running)
GenericError	Generic error FPGA register value
Datasize	Size of the data buffer read (32 bit words)
data_buffer	Raw buffer read from the device

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.
size	total size of the arrays in events number
occupancy	current filled size of the arrays in events number

- 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.

```
//malloc events buffer
R7771_API_ErrorCode_t R7771_DLLAPI R7771_MallocEventsBuffer(R7771_Events_Buffer_t* EvBuffer, uint32_t size);
//free events buffer
R7771_API_ErrorCode_t R7771_DLLAPI R7771_FreeEventsBuffer(R7771_Events_Buffer_t* EvBuffer);
//clear events buffer
R7771_API_ErrorCode_t R7771_DLLAPI R7771_ClearEventsBuffer(R7771_Events_Buffer_t* EvBuffer, uint32_t size);
```

1.9.4.3 DecodeR7771Histograms

This function can be used to decode a binary buffer read from the device in MCS mode. In this case, the buffer filled by GetRawDeviceData is made of 4 bytes unsigned words that contain the bin contents of the 4 MCS histograms. The histogram buffers inside the R7771_MCS_Histo_t structs should already be allocated when the function is called (see the R7771_Malloc_MCSHisto function). The maximum size of one MCS histogram is defined as MCS_MAX_NBINS. Return value is the error code (see error codes in Sec. **Return Codes**).

```
R7771_API_ErrorCode_t R7771_DLLAPI DecodeR7771Histograms(uint32_t* buffer, R7771_MCS_Histo_t* MCS);
```

Parameter	I/O	Description
*buffer	Input	The binary data buffer to be decoded
*MCS	Input	Pointer to the array of structs each containing the pre-allocated histogram that will be filled with the data

The R7771_MCS_Histo_t contains the MCS histogram as decoded from the binary buffer. The histogram size is also stored.

R7771_MCS_Histo_t field	Description
*h	Pointer to the array filled with the 4 histograms
size	Size of a single MCS histogram

The library also provides the needed functions to allocate, clear and free the MCS histograms buffers.

```
//malloc MCS histograms
R7771_API_ErrorCode_t R7771_DLLAPI R7771_Malloc_MCSHisto(R7771_MCS_Histo_t* h, uint32_t size);
//free MCS histograms
R7771_API_ErrorCode_t R7771_DLLAPI R7771_Free_MCSHisto(R7771_MCS_Histo_t* h);
//clear MCS histograms
R7771_API_ErrorCode_t R7771_DLLAPI R7771_Clear_MCSHisto(R7771_MCS_Histo_t* h, uint32_t size);
```

1.10 The R7771 API_Demo

A demo software is released together with the R7771 API library. Through very simple examples it shows how to employ the library functions to manage the device, data acquisition and data analysis. The user can manage the source files and modify them to connect a device, configure it and start acquisition.

The main screen of the console demo software main menu is shown in Fig. **1.1**. It is possible to analyse binary data files in different ways, or to connect a device and start a new data acquisition.

```
D : Connect device
d : Disconnect device
S : Start acquisition
s : Stop acquisition
X : Start raw data acquisition
x : Stop raw data acquisition
c : Clear device memory
f : Configure data save
p : Get/Set device parameters
H : HV Menu
P : Get/Set analysis parameters
A : Start analysis
a : Stop analysis

O : Enable Online Analysis
o : Disable Online Analysis
B : Set MCS analysis ROI
b : Get MCS analysis ROIs
L : Set MCS spectrum label
l : Get MCS spectrum label
N : Set MCS spectrum label
n : Get MCS spectrum label
C : Get counts
T : Get DeltaT distribution
M : Get Coincidence and Multiplicity results
R : Get Rossi Alpha distribution
m : Get MCS histograms
U : Get calculated Mass
z : Print Menu
q : Quit client
```

Fig. 1.1: Demo program main menu.

2 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:

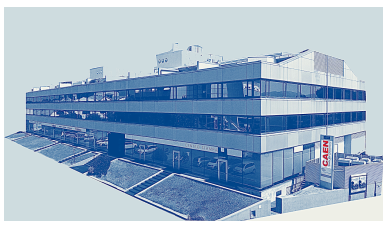
https://www.caen.it/support-services/getting-started-with-mycaen-portal/

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

UM9828 - R7771 - API Library rev. 1 - February 7th, 2024 00109/18:R777X.MUTX/00