# Class GBS_MCA_Comm

## Constructor:

```
GBS_MCA_Comm();
```

## Destructor:

```
~GBS_MCA_Comm();
```

## Public methods:

### Functions similar to the Windows communication dll:

```
bool COMM_INIT(long timeout = 1000,            // Read timeout
               int tryAgain = 8,               // number if attempts, if
                                               // communication fails
               unsigned long baudRate = 0);
```

Establishes a connection to an MCA. The first responding MCA that is found will be used.
If baudRate = 0: the highest possible baud rate is set
Return value: returns true if successful

```
void COMM_CLOSE();
```

The connection to the MCA is closed.

```
E_ERROR_FLAG MMCA_RESET();
```

Resets all MCA parameters to their initial state, all spectra are cleared and the measurement is aborted.

Return value:

| | | |
|---|---|---|
| 0: | ERROR_OK | successful data transfer |
| 1: | ERROR_INTERFACE | communication port is not initialized |
| 2: | ERROR_UNKNOWN_COMMAND | unknown command |
| 3: | ERROR_COMMUNICATION | faulty data transfer |
| 4: | ERROR_INVALID_PARAM | invalid parameter |
| 5: | ERROR_RUNNING_MEAS | measurement is running, but stopped measurement is required for this command |
| 6: | ERROR_VIOLATED_RIGHT | execution right violation |
| 7: | ERROR_STOPPED_MEAS | measurement is stopped, but running measurement is required for this command |
| 8: | ERROR_WRONG_MODE | wrong mode for using this command |
| 9: | ERROR_UNHANDLED_COMMAND | not handled by this firmware version |
| 10: | ERROR_FILE_WRITING_IN_PROCESS | file writing is in process, this command must not be called before the process is finished |

```
E_ERROR_FLAG MMCA_START_ACQUIRE
      (uint16_t Flags,        Flags (see firmware command CMD_START)
       uint32_t StartTime);   Start time (see firmware command CMD_START)
```

The acquisition is started or continued with the current parameters.
Return value: see MMCA_RESET()

```
E_ERROR_FLAG MMCA_STOP_ACQUIRE();
```

The acquisition is stopped.
Return value: see MMCA_RESET()

```
E_ERROR_FLAG MMCA_SET_PRESET_NONE();
```

Sets none automatic stop condition.
Return value: see MMCA_RESET()

```
E_ERROR_FLAG MMCA_SET_PRESET_LIVE_TIME
      (uint32_t LiveTime);    Live time in seconds
```

The function sets the time for the automatic stop condition (dead time corrected).
Return value: see MMCA_RESET()

```
E_ERROR_FLAG MMCA_SET_PRESET_REAL_TIME
      (uint32_t RealTime);    Real time in seconds
```

The function sets the time for the automatic stop condition.
Return value: see MMCA_RESET()

```
E_ERROR_FLAG MMCA_SET_ADC_RES_DISCR
      (uint16_t Channels,     number of Channels
                              MCA166: 128, 256, 512, 1024, 2048 or 4096
                              MCA527: 128, ... maximum channels count
       uint16_t LLD,          Upper level discriminator  (0 <= LLD < ULD)
       uint16_t ULD);         Low level discriminator
                              MCA166: LLD < ULD < Channels – Channels/32
                              MCA527: LLD < ULD < Channels
```

The function sets the ADC resolution and the software discriminator range.
Return value: see MMCA_RESET()

```
E_ERROR_FLAG MMCA_SET_GAIN
      (uint16_t CoarseGain,   Coarse gain: 2, 5, 10, 20, 50, 100, 200, 500
                                           or 1000
       uint16_t FineGain);    Fine gain
                              MCA166: 5000 ... 15000 * 0.0001
                              (if coarse gain=1000, max. fine gain=10000 )
                              MCA527: 5000 ... 65000 * 0.0001
```

The function sets the amplifier coarse and fine gain.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_MCA_INPUT_AMPLIFIER_POS**();

The function sets the amplifier input polarity to positive.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_MCA_INPUT_AMPLIFIER_NEG**();

The function sets the amplifier input polarity to negative.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_THRESHOLD**
(uint16_t Threshold); Threshold (0 ... 60 %)

The function sets the analog threshold.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_THRESHOLD_TENTHS**
(uint16_t Threshold); Threshold (0 ... 600 * 0.1)

The function sets the analog threshold. Only available for MCA527.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_SHAPING_TIME_LOW**();

The function sets the low shaping time.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_SHAPING_TIME_HIGH**();

The function sets the high shaping time.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_SHAPING_TIME_PAIR**
(uint16_t Low, Low shaping time [µs] *10 (1...254)
uint16_t High); High shaping time [µs] *10 (2...255)

The function sets the values for the low and high shaping time. It is available for MCA-527 only. Afterwards the shaping time can be selected with MMCA_SET_SHAPING_TIME_LOW or MMCA_SET_SHAPING_TIME_HIGH.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_TRIGGER_FILTER**
(uint16_t Low, Trigger filter for low shaping time (0..5)
uint16_t High); Trigger filter for high shaping time (0..5)

The function sets the trigger filter used for low and high shaping time. It is available for MCA-527 only.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_PZC_MANUAL**
(uint16_t PzcValue, PZC value (0 ... 2499)
int16_t* offset, The offset is returned
bool HighPrecision = true); true: offset is returned with higher

precision. This feature is available for the MCA-527 since firmware version 12.05.

The function sets the PZC value and returns the PZC offset (Pole Zero Cancellation). In case of an error the offset remains unchanged. This command needs about one second to return, because a measurement is run.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_HIGH_VOLTAGES**
(uint16_t HighVoltage, High voltage (0 ... 3600 V)
int32_t Inhibit); High voltage inhibit:
0 = Inhibit off
1 = ”Canberra HPGe mode”
2 = “DSG HPGe mode”
-1 = ”Ortec HPGe mode”

The function sets the detector high voltage and controls the HV-inhibit-signal.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_SET_FLAT_TOP_TIME**
(uint16_t FlatTopTime); flattop time (0 ... 255 * 0.1 µs)

The function sets the flattop time.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_QUERY_STATE**
(struct QUERY_STATE* rec_data); Pointer to a QUERRY_STATE structure to hold the read data

The function reads the MCA state. If rec_data is null, no data are copied (e.g. for test if the MCA answers).
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_QUERY_STATE527**
(struct QUERY_STATE527* rec_data); Pointer to a QUERRY_STATE527 structure to hold the read data

The function reads additional state information, and is available for MCA-527 only. If rec_data is null, no data are copied (e.g. for test if MCA answers).
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_QUERY_STATE527_EX**
(struct QUERY_STATE527_EX* rec_data); Pointer to a QUERRY_STATE527_EX structure to hold the read data

The function reads additional state information, and is available for MCA-527 only. If rec_data is null, no data are copied.
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_QUERY_POWER**
(struct QUERY_POWER* rec_data); Pointer to a QUERRY_POWER structure to hold the read data

The function reads the MMCA power state.
Return value: see MMCA_RESET()

```
E_ERROR_FLAG MMCA_QUERY_SYSTEM_DATA
      (struct QUERY_SYSTEM_DATA* rec_data); Pointer to a QUERY_SYSTEM_DATA
                                            structure to hold the read data

The function reads the MMCA system data.
Return value: see MMCA_RESET()


E_ERROR_FLAG MMCA_QUERY_SPECTRA
      (uint16_t FirstChannel,       Number of first channel (bit 11 ... 0):
                                    0 ... channelsCount – 32* compressFactor–1
                                    and buffer control (bit 15 ... 12):
                                    0 ... 15 (see  firmware command
                                    CMD_QUERY_SPECTRA_EX)
       uint32_t CompressFactor,     MCA166: 1 ... 32, MCA527: 1 ... 128
       QUERY_SPECTRA* rec_data);    Pointer to a QUERRY_SPECTRA structure to
                                    hold the data

The function reads the MMCA spectrum data (measurement result) (132 Bytes)
Return value: see MMCA_RESET()



E_ERROR_FLAG MMCA_QUERY_SPECTRA_EX
      (uint16_t FirstChannel,       Number of first channel:
                                    0 ... channelsCount – 32* compressFactor–1
       uint16_t CompressFactor,     MCA166: 1 ... 32, MCA527: 1 ... 128
       uint16_t BufferControl,      see  firmware command CMD_QUERY_SPECTRA_EX
       QUERY_SPECTRA* rec_data);    Pointer to a QUERRY_SPECTRA structure to
                                    hold the data

The function reads the MMCA spectrum data (measurement result) (132 Bytes)
Return value: see MMCA_RESET()
```

## Additional functions, which are not available in the traditional Windows communication dll:

```
E_ERROR_FLAG MMCA_QUERY_SPECTRA_TO_BUFFER
      (uint16_t FirstChannel,       Number of first channel (bit 11 ... 0):
                                    0 ... channelsCount – 32* compressFactor–1
                                    and buffer control (bit 15 ... 12):
                                    0 ... 15 (see  firmware command
                                    CMD_QUERY_SPECTRA_EX)
       uint32_t CompressFactor,     MCA166: 1 ... 32, MCA527: 1 ... 128
       uint32_t* buffer);           pointer to the buffer position for data
                                    output

This function reads the spectrum data (measurement result) (132 Bytes) directly
to the specified buffer position
Return value: see MMCA_RESET()



E_ERROR_FLAG MMCA_QUERY_SPECTRA_EX_TO_BUFFER
      (uint16_t FirstChannel,       Number of first channel:
                                    0 ... channelsCount – 32* compressFactor–1
       uint16_t CompressFactor,     MCA166: 1 ... 32, MCA527: 1 ... 128
       uint16_t BufferControl,      see  firmware command CMD_QUERY_SPECTRA_EX
       uint32_t* buffer);           pointer to the buffer position for data
```

output

This function reads the spectrum data (measurement result) (132 Bytes) directly to the specified buffer position
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_QUERY_COMPLETE_SPECTRUM**
(uint16_t FirstChannel, Number of first channel
uint16_t LastChannel, Number of last channel
uint16_t CompressFactor, MCA166: 1 ... 32, MCA527: 1 ... 128
uint32_t* buffer); pointer to the buffer position for data output

This function reads a complete spectrum or part of a spectrum by calling MMCA_QUERY_SPECTRA_EX_TO_BUFFER several times
Return value: see MMCA_RESET()

E_ERROR_FLAG **MMCA_QUERY_COMPLETE_SPECTRUM**
(uint16_t FirstChannel, Number of first channel
uint16_t LastChannel, Number of last channel
uint16_t CompressFactor, MCA166: 1 ... 32, MCA527: 1 ... 128
uint16_t BufferControl, see firmware command CMD_QUERY_SPECTRA_EX
uint32_t* buffer); pointer to the buffer position for data output

This function reads a complete spectrum or part of a spectrum by calling MMCA_QUERY_SPECTRA_EX_TO_BUFFER several times
Return value: see MMCA_RESET()

### Some information about the communication object:

bool **IsConnected**() const;

The function returns true, if a connection to an MCA is established.

Unsigned long **GetBaudrate**() const;

The function returns the data baud rate.

# Functions

uint32_t **ToMcaTime**(time_t time);

The function converts <time> into the time format used by the MCA (seconds since 1969-12-31, 16:00)

void **FromMcaTime**(uint32_t time, struct tm* outTime);

The function converts <time> in MCA format (seconds since 1969-12-31, 16:00) into struct tm.

std::string **GetTriggerFilterDesc**(uint16_t index);

The function returns the related trigger filter description.

# Data Structures

Structures of the query results
Please see CMD_QUERY... in the description of the MCA527 firmware commands for detailed information about the data.

## MCA state

```
struct QUERY_STATE
    { uint16_t  McaMode,
                Presets;
      uint32_t  PresetValue,
                ElPreset;
      uint16_t  Repeat,
                ElSweeps,
                TimePerChannel,
                ElTimePerChan;
      uint32_t  RealTime,
                CountsPerSec,
                DeadTime,
                BusyTime;
      uint16_t  Channels,
                Threshold,
                Lld,
                Uld,
                RoiBegin,
                RoiEnd,
                CoarseGain,
                FineGain,
                SlowDiscr,
                FastDiscr,
                DetectorBias,
                DetectorBiasPoly,
                PreampPower,
                PzcValue,
                PzcDtc1Offset,
                PzcDtc3Offset,
                StabState,
                StabResult,
                StabRoiBegin,
                StabRoiEnd,
                McaInputAdc,
                McaInputPol,
                Dtc,
                McaPur,
                McaInputMcs,
                McaNumber,
                HardwareVersion,
                FirmwareVersion,
                McsChannels,
                LastPowerState,
                BatteryCapacity,
                BatteryLifeTime;
```

```
      uint32_t  StartTime;
      uint16_t  Tdf,
                CommandAndParameter[4],
                BufferState;
      uint32_t  AmpliDacVal;
      uint16_t  DiffDeadTime;
      int16_t   HvInhibitMode;
      uint16_t  HvInhibitState,
                CheckSum,
                McaState,
                Reserve;
    };
```

## MCA 527 state

```
Struct QUERY_STATE527
    { uint16_t  HwVersion,
                FwVersion,
                HwModification,
                FwModification;
      uint32_t  Features;
      uint16_t  Year;
      uint8_t   Month,
                Day,
                Hour,
                Minute,
                Second,
                Reserve;
      uint32_t  TestingPhase;
      int16_t   McaTemperature;
      uint16_t  GeneralMode;
      uint32_t  DiscardedCycles;
      uint16_t  CoreClock;
      uint8_t   TriggerFilterLow,
                TriggerFilterHigh;
      uint16_t  Expander,
                OffsetDac;
      int16_t   DetectorTemperature;
      int16_t   PowerModuleTemperature;
      uint16_t  McaNumber;
      int16_t   AmIRightHolder;
      uint8_t   RightHoldersIp[4];
      uint16_t  RightHoldersPort;
      int16_t   ValidityOfRight;
      uint16_t  MaxChannels;
      uint8_t   PowerModuleVersions[2];
      uint16_t  PowerModuleNumber,
                PowerModuleId,
                PowerModuleMaxHv,
                ThresholdTenths;
      uint32_t  FastDeadTime;
      uint16_t  FilterType,
                FlatTopTime,
                FilterSize,
                TriggerLevel;
      int16_t   McaTemperatureAtStop,
                DetectorTemperatureAtStop;
      uint8_t   IpAddressSet[4];
      uint8_t   IpAddressActual[4];
      uint32_t  TimePerChannel,
                ElTimePerChan;
      int32_t   TriggerThreshold;
```

```
      int16_t   PowerModuleTemperatureAtStop;
      uint16_t  CommandAndParameter[4];
      uint8_t   JitterCorrection,
                BaselineRestoring;
      int32_t   SetTriggerThreshold;
      uint8_t   Input,
                MaxShapingTime,
                GatingMode,
                GatingPol;
      uint8_t   GatingShift;
      uint8_t   CoarseGainLevels;
      uint16_t  CheckSum,
                McaState,
                DiffFastDeadTime;
    };
```

## MCA 527 state ex

```
struct QUERY_STATE527_EX
    {
      uint32_t CommonMemorySize,
               CommonMemoryFillStop,
               CommonMemoryFillLevel;
      int16_t  OsciTimeResolution;
      uint16_t OsciTriggerSource,
               OsciTriggerPosition,
               OsciTriggerThreshold;
      uint32_t PurCounter;
      uint8_t  ExtPortPartA,
               ExtPortPartB,
               ExtPortPartC,
               ExtPortPartD,
               ExtPortPartE,
               ExtPortPartF,
               ExtPortParts,
               ExtPortStates,
               ExtPortPolarities,
               MaxFlattopTime;
      uint16_t BootPresetsDataSize;
      uint32_t ExtPortPulser1Period,
               ExtPortPulser2Period,
               ExtPortPulser1Width,
               ExtPortPulser2Width;
      uint16_t ExtPortRs232Baudrate,
               ExtPortRs232Flags;
      uint32_t ExtPortCounter1,
               ExtPortCounter1Cps,
               ExtPortCounter1Prev,
               ExtPortCounter2,
               ExtPortCounter2Cps,
               ExtPortCounter2Prev;
      uint16_t ExtPortRs232TxByteCount,
               FractionalRealTime;
      uint32_t PurCounterPrev,
               TriggerFilterAvailability;
      int16_t  TriggerFilterValue1,
               TriggerFilterValue2;
      uint8_t  TtlLowLevel,
               TtlHighLevel;
      uint16_t CoeffAutoThreshold;
      uint32_t AdcOverflowsPerSecond;
```

```
      uint16_t AdcSamplingRate,
               CommandAndParameter[4],
               NeededFileSize;
      uint32_t SdCardTotalMemorySize,
               SdCardFreeMemorySize;
      int8_t   FileWritingState,
               FileWritingResult;
      uint16_t CheckSum,
               McaState,
               Rs485BaudRate;
    };
```

## MCA Power state

```
struct QUERY_POWER
    {  uint32_t BatteryCurrent,
               HvPrimaryCurrent,
               P12PrimaryCurrent,
               M12PrimaryCurrent,
               P24PrimaryCurrent,
               M24PrimaryCurrent,
               BatteryVoltage,
               HighVoltage,
               HvState;
       uint8_t  P12Voltage,
               M12Voltage,
               P24Voltage,
               M24Voltage;
       uint32_t BiasCurrentValue;
       uint16_t Pin3Voltage,
               Pin5Voltage;
       uint32_t PowerSwitches,
               ChargerCurrent;
       uint16_t Pin5CurrentSourceValue,
               Pin5CurrentSourceState,
               Pin5Resistor;
       int8_t   Pin5Offset,
               Pin5Gain;
       uint32_t BatteryCurrentAtStop,
               HvPrimaryCurrentAtStop,
               P12PrimaryCurrentAtStop,
               M12PrimaryCurrentAtStop,
               P24PrimaryCurrentAtStop,
               M24PrimaryCurrentAtStop,
               BatteryVoltageAtStop,
               HighVoltageAtStop;
       int8_t   Pin3Offset,
               Pin3Gain;
       uint16_t HvControlVoltage;
       uint8_t  P12VoltageAtStop,
               M12VoltageAtStop,
               P24VoltageAtStop,
               M24VoltageAtStop;
       uint16_t Pin3VoltageAtStop,
               CommandAndParameter[4],
               Pin5VoltageAtStop;
       uint32_t ChargerCurrentAtStop;
       uint8_t  NotUsed1[4],
               PowerModulInfo,
               PowerModuleFeatures;
       uint16_t CheckSum,
               McaState;
```

```
       uint8_t  NotUsed2[2];
    };
```

## MCA System Data

```
Struct QUERY_SYSTEM_DATA
    {
       uint64_t PdCounter,// Peak detect counter, only for MCA-166
                Imp,      // Fast detect counter
                PdPre,    // MCA-166: Peak detect counter at time -1
                          // MCA-527: Counts outside the spectrum
                ImpPre;   // MCA-166: Fast detect counter at time -1
                          // MCA-527: Counts outside the spectrum of prev. sweep
       uint32_t BtPre,    // Busy time [ms] at time -1, only for MCA-166
                Time,              // MCA on time [s]
                RealTimePrevSweep, // Real time [s] of previous sweep
                DeadTimePrevSweep, // Dead time [ms] of previous sweep
                StartTimePrevSweep,// Start time of previous sweep
                FastDTimePrevSweep,// MCA-527: Fast dead time of prev.sweep
                ElSweeps,          // Elapsed sweeps
                BusyTimePrevSweep; // Busy time [ms] of previous sweep
       uint16_t FractRTPrevSweep;  // MCA-527:fract.digits of RT[ms] prev.sweep
       uint64_t PdPrevSweep;       // MCA-166:Peak detect counter of prev. sweep
       uint16_t ImpPrevSweepLow;   // Fast detect counter of prev. sweep
       uint32_t ImpPrevSweepHigh,
                StabCounter;       // Counter of stabilization steps
       int32_t  AmpliDacOffset,    // Current stabilization offset
                AmpliDacOffsetMin, // Maximal negative stabilization offset
                AmpliDacOffsetMax; // Maximal positive stabilization offset
       uint32_t RecCounter,        // Counter of received commands
                RecErrorCounter;   // Counter of unsuccessful commands
       uint16_t RecRate,
                CommandAndParameter[4],
                BufferState;
       uint32_t StabArea;          // Stabilization area preset
       uint16_t StabTime;          // Stabilization time preset [s]
       uint8_t  LowShapingTime,
                HighShapingTime,
                MinRecomClock,     // MCA-527: Minimum recommended core clock
                MaxAllowedClock;   // MCA-527: Maximum allowed core clock
       uint16_t CheckSum,
                McaState,
                ADCsampleRate;
    };
```

## Spectrum data

```
struct QUERY_SPECTRA
    { uint32_t  ChannelContents[32];
      uint16_t  BufferState,
                CheckSum;
    };
```