

**Development kit for DATAMAN 770  
Programmer's Guide  
C#.NET  
Version 2.01**



Copyright © 2004-2008 Dataman Programmers Ltd

Thank you for choosing the Dataman 770 Series Oscilloscope with Development Kit. We are confident, that it will meet your expectations.

For any further information or consultations, please contact us via phone or preferably e-mail on the following address:

Address:

Dataman Programmers Ltd  
Station Road  
Maiden Newton  
Dorset  
DT2 0AE  
United Kingdom

Phone:

Sales/General information: +44 (0) 1300 320719  
Technical support: +44 (0) 1300 322903

Fax:

All Enquiries: +44 (0) 1300 321012

Internet:

URL: <http://www.dataman.com/>  
e-mail: [support@dataman.com](mailto:support@dataman.com) - technical support  
[sales@dataman.com](mailto:sales@dataman.com) - sales  
[info@dataman.com](mailto:info@dataman.com) - other information

## Contents

1. Basic information.....	6
1.1. Development kit contents .....	6
1.2. DK usage.....	6
1.3. Application deployment.....	6
2. Controlling the device.....	7
2.1. Device initialization .....	7
2.2. Error handling .....	7
2.3. Data acquisition loop .....	7
2.4. Acquired data allocation .....	8
2.5. Measurement in the sampling mode .....	8
2.5.1. Random sampling method .....	8
2.5.2. Measurement in sampling mode .....	10
2.6. Using more 770 devices.....	10
3. Reference .....	11
3.1. Functions, that returns information about device .....	12
3.1.1. GetDKError.....	12
3.1.2. ResetDKError .....	12
3.1.3. GetDeviceID .....	12
3.1.4. GetDeviceSerialNumber .....	13
3.1.5. GetDKVersion .....	13
3.1.6. GetTimeBaseList .....	13
3.1.7. GetRangeList .....	14
3.2. Initialization functions .....	14
3.2.1. LoadDriver.....	14
3.2.2. InitHardware .....	15
3.2.3. GetDeviceList .....	15
3.2.4. InitHardwareFromList .....	15
3.2.5. UseDevice .....	16
3.3. Functions, that set the data acquisition parameters.....	16
3.3.1. SetTimeBase .....	16
3.3.2. SetRange .....	16
3.3.3. SetCoupling.....	17
3.3.4. SetVert .....	17
3.3.5. SetTriggerLevel .....	17
3.3.6. SetTriggerCount.....	18
3.3.7. SetTriggerLength .....	18
3.3.8. SetTriggerMode .....	18
3.3.9. SetAfterTriggerLength.....	19
3.3.10. SetHoldOff .....	19
3.3.11. SetTriggerSource .....	19
3.3.12. SetTriggerEdge .....	19
3.3.13. SetProbe .....	20
3.3.14. SetMemorySize.....	20
3.3.15. SetCompensationGenerator .....	20
3.3.16. SetGround .....	21
3.3.17. SetDigitalShielding .....	21
3.4. Data acquisition functions.....	21

3.4.1. StartMeasurement .....	21
3.4.2. IsDataReady .....	22
3.4.3. GetReconstructionPercentage .....	22
3.4.4. GetData .....	22
3.5. Other functions.....	23
3.5.1. GroundPositionToShift .....	23

## Figures and tables

Table 1.1. –Development kit (DK) contents .....	6
Fig. 2.4.1. – Data allocation in the array.....	8
Fig. 2.5.1.1. – Random sampling principle.....	9

## 1. Basic information

### 1.1. Development kit contents

All development kit (DK) parts are located in the installation directory.

Directory	Contents
Examples\C#.NET	C#.NET example
Examples\VB.NET	Visual Basic .NET example
Examples\VB	Visual Basic 6.0 example
Examples\Delphi	Delphi example
Examples\CBuilder	C++ Builder example
Examples\VC	Visual C++ example
Include\C#.NET	C#.NET header files
Include\VB.NET	Visual Basic .NET header files
Include\VB	Visual Basic 6.0 header files
Include\Delphi	Delphi header files
Include\CBuilder	C++ Builder header files
Include\VC	Visual C++ 6.0 header files
Bin	M770drvdk.dll and m770drv.dll libraries

*Table 1.1. –Development kit (DK) contents*

### 1.2. DK usage

In order to make the DK work properly, it is necessary to have the DATAMAN 770 oscilloscope drivers installed. The m770drv.cs header file contains the cm770drv class, which encapsulates all DK functions and constants. Add this file to project to gain access to the cm770drv class. The m770drvdk.dll and m770drv.dll must be present in the same directory as .exe file (bin\debug) during debugging.

### 1.3. Application deployment

The m770drvdk.dll and m770drv.dll libraries must be distributed together with your application. The drivers for the DATAMAN 770 oscilloscope must be installed in the system in order to communicate with the device. The application will work with every device with an activated DK.

## 2. Controlling the device

### 2.1. Device initialization

First of all, it is necessary to load the driver using the function `LoadDriver`.

```
cm770drv.LoadDriver();
```

After the driver is loaded it is possible to initialize the device using the function `InitHardware`. This function also returns the information for the calibration data.

```
byte calibok;  
cm770drv.InitHardware(out calibok);
```

### 2.2. Error handling

In case an error occurs all subsequent calls of functions will fail. Therefore it is necessary to check if the operations were successful (for example check if the initialization was successful). Use `GetDKError` to obtain the error code.

```
int res;  
res = cm770drv.GetDKError();
```

In case of an error it is necessary to reset the error flag (to indicate to the DK, that the error has been handled). Use `ResetDKError` function to do so (otherwise no other function will be successful).

```
cm770drv.ResetDKError();
```

### 2.3. Data acquisition loop

The data acquisition process can be started by calling `StartMeasurement` function.

```
cm770drv.StartMeasurement();
```

After the data acquisition starts the software must wait until the data is ready in the device. Use `IsDataReady` function to check the acquisition status.

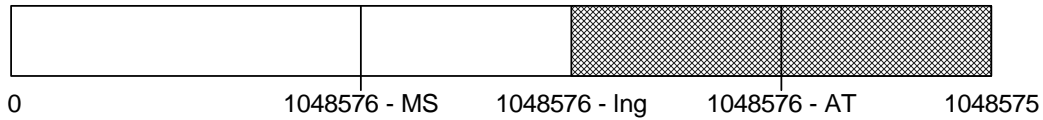
```
if (cm770drv.IsDataReady() == cm770drv.DATA_READY)  
{  
}
```

When the data is ready for transfer to the computer (return value `DATA_READY`), it is possible to transfer them to the computer using `GetData` function.

```
Sample[] data = new Sample[1024*1024];  
int lng;  
cm770drv.GetData(out data[0], out lng);
```

## 2.4. Acquired data allocation

The array that is passed as the first parameter of the GetData function must be at least 1048576 items long. Acquired data will be written in the array like this:



*Fig. 2.4.1. – Data allocation in the array*

MS is the memory size, AT is the amount of samples acquired after trigger, lng is the amount of received data (second parameter of the GetData function). If the  $AT + HoldOff \geq MS$ , MS equals to lng.

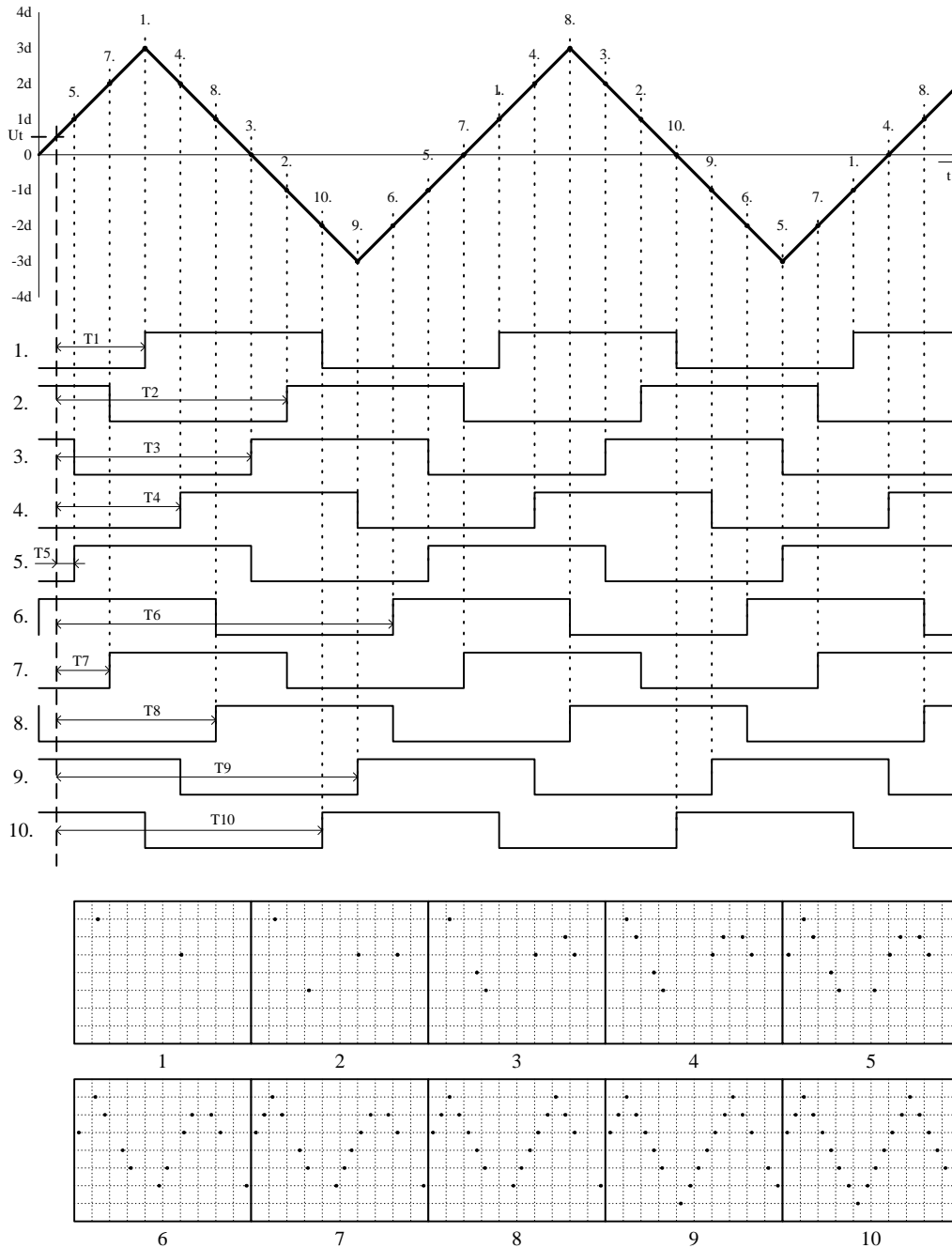
## 2.5. Measurement in the sampling mode

The DATAMAN 774 oscilloscope samples at 100MS/s in real time. To achieve a higher sampling speed a random sampling method is used.

### 2.5.1. Random sampling method

The sampling technology is applicable to periodical waveforms only. It is also known as “random sampling” or “equivalent sampling”. It uses the fact, that the sampled waveform is repeated periodically and the sampling clock is asynchronous in relation to the waveform. The principle is shown on the figure 2.5.1.1. The sampled waveform with triangular shape is sampled with the rising edge of the sampling clock.





**Fig. 2.5.1.1. – Random sampling principle**

The trigger's threshold is set to  $U_t$  with the rising edge of the trigger input selected. Below the sampled waveform there is shown ten sampling clocks after valid trigger events. Because the phase of the sampling clock and the trigger event are independent (asynchronous) in time, the time period between the trigger event and the rising edge of the first sampling clock after the trigger event is a random value ( $T_1$  thru  $T_{10}$ ). Using these time periods it is possible to reconstruct the measured waveform. Every step of the reconstruction is shown at the bottom of the picture. This is the way in which you can multiply the sampling rate of the oscilloscope.

For sampling rate multiplication by  $N$  it is necessary to perform a minimum  $N$  subsequent measurements. In the majority of cases it is necessary to perform more

measurements because the random nature of this process causes measurements with the same clock phase.

When the random sampling method is used it is necessary to trigger with a well defined and stable waveform.

### **2.5.2. Measurement in sampling mode**

Synthesis and interpolation of the samples is done automatically by the DK. Application received synthesized data. The only difference between real mode and sampling mode is that it is necessary to perform more acquisition loops to get a real image of the signal. Use `GetReconstructionPercentage` function to get the quality of the signal image (ratio of the measured and calculated samples).

## **2.6. Using more 770 devices**

In case your application needs to use more than one device from the 770 family, please use following initialization process:

First of all it is necessary to load the driver using `LoadDriver` function.

```
cm770drv.LoadDriver();
```

Before initialization of the device the device list must be obtained by calling the `GetDeviceList` function.

```
uint[] devs = new uint[16];  
cm770drv.GetDeviceList(out devs[0]);
```

The function fills the array with serial numbers of the present devices. Item index identifies the device in further calls of initialization functions.

The device can be initialized by calling the `InitHardwareFromList` function.

```
cm770drv.InitHardwareFromList(1, out calibok);
```

First parameter (device index in the array) indicates which device will be initialized.

After the device is initialized the `UseDevice` function can be used to select the device. All following calls of other functions (such as `SetTimeBase`, `SetRange` ...) will affect the selected device.

```
cm770drv.UseDevice(1);
```

### 3. Reference

Functions available in the DK can be divided into five groups:

#### **Functions, that returns information about device**

- GetDKError
- ResetDKError
- GetDeviceID
- GetDeviceSerialNumber
- GetDKVersion
- GetTimeBaseList
- GetRangeList

#### **Initialization functions**

- LoadDriver
- InitHardware
- GetDeviceList
- InitHardwareFromList
- UseDevice

#### **Functions, that set the data acquisition parameters**

- SetTimeBase
- SetRange
- SetCoupling
- SetVert
- SetTriggerLevel
- SetTriggerCount
- SetTriggerLength
- SetTriggerMode
- SetAfterTriggerLength
- SetHoldOff
- SetTriggerSource
- SetTriggerEdge
- SetProbe
- SetMemorySize
- SetCompensationGenerator
- SetGround
- SetDigitalShielding

#### **Data acquisition functions**

- StartMeasurement
- IsDataReady
- GetReconstructionPercentage
- GetData

## Other functions

GroundPositionToShift

### 3.1. Functions, that returns information about device

#### 3.1.1. GetDKError

In the case where the error occurs during the call of any DK function the error code is stored in the DK internal variable. All subsequent calls of DK functions will fail. GetDKError returns error code.

**Declaration:**

```
public static extern int GetDKError();
```

**Parameters: -**

**Return value:**

*ERROR\_OK* – no error occurred  
*ERROR\_DRIVER\_NOT\_LOADED* – unable to load driver/driver wasn't loaded before call  
*ERROR\_DRIVER\_INCOMPATIBLE* - driver is not compatible with DK  
*ERROR\_UNABLE\_TO\_LOAD\_EM77X* – unable to load em77x driver  
*ERROR\_INIT\_FAILED* – device initialization failed  
*ERROR\_FPGA\_CONFIG\_FAILED* – FPGA configuration failed  
*ERROR\_COMMUNICATION\_FAILED* – the communication with device is broken  
*ERROR\_OPERATION\_FAILED* – unable to finish last operation  
*ERROR\_DK\_NOT\_ENABLED* – the device doesn't have DK enabled  
*ERROR\_INCORRECT\_PARAMETER* – the function was called with incorrect parameter value

#### 3.1.2. ResetDKError

In the case where the error occurs during the call of any DK function the error code is stored in the DK internal variable. All subsequent calls of any DK functions will fail. ResetDKError function resets this variable thus allowing you to call the DK functions again.

**Declaration:**

```
public static extern void ResetDKError();
```

**Parameters: -**

**Return value: -**

#### 3.1.3. GetDeviceID

Returns ID of the connected device.

```
public static extern int GetDeviceID();
```

**Parameters:** -

**Return value:**

Device code is calculated as  $771 + \text{return value}$ .

### 3.1.4. GetDeviceSerialNumber

Returns device serial number.

```
public static extern uint GetDeviceSerialNumber();
```

**Parameters:** -

**Return value:**

Device serial number.

### 3.1.5. GetDKVersion

Returns the version of the selected DK part.

```
public static extern void GetDKVersion(int module, out DKVersion version);
```

**Parameters:**

*module* – identifies DK part, which version is to be obtained

    VERSION\_EM770 – em770 driver

    VERSION\_DK – DK (m770drvdk.dll)

*version* – structure, which will be filled with the version

**Return value:** -

### 3.1.6. GetTimeBaseList

Returns the list of available timebases.

```
public static extern void GetTimeBaseList(out TimeBaseItem timebaselist, out int timebaselistlength);
```

**Parameters:**

*timebaselist* – the first item of the array which will be filled with the available timebases. Array must be at least 31 items long.

*timebaselistlength* – the number of array items which was filled (number of available timebases)

**Return value:** -

**Remark:**

Each array item is of TTimeBaseItem type. This structure comprises of the following items:

*nsPerDiv* – amount of nanoseconds per screen division (50 samples); this value is passed to the SetTimeBase function

*RealSamplingRate* – real sampling rate (in case, that the timebase doesn't run in sampling mode, this value equals to *SamplingRate*)

*SamplingRate* – equivalent sampling rate; this frequency determines distance between two samples

*SamplingMulti* – indicates ratio of *SamplingRate* and *RealSamplingRate*; the value of 1 indicates sampling in real time, value greater than 1 indicates sampling mode

### 3.1.7. GetRangeList

Returns the list of available ranges.

```
public static extern void GetRangeList(out RangeItem rangelist, out int rangelistlength);
```

#### Parameters:

*rangelist* – the first item of the array which will be filled with the available ranges. Array must be at least 13 items long.

*rangelistlength* – the number of array items which was filled (number of available ranges)

**Return value:** -

#### Remark:

Each array item is of TRangeItem type. This structure comprise of following items:

*mVPerDiv* – amount of mV per one screen division (32 quantization levels); this value is passed to the SetRange function

## 3.2. Initialization functions

### 3.2.1. LoadDriver

Loads m770drv.dll driver.

```
public static extern int LoadDriver();
```

**Parameters:** -

**Return value:**

*ERROR\_OK* – driver loaded successfully

*ERROR\_DRIVER\_NOT\_LOADED* – unable to load m770drv.dll library

*ERROR\_DRIVER\_INCOMPATIBLE* – m770drv.dll isn't compatible with DK

#### Remark:

The return value is stored in the internal DK variable as well. Use GetDKError function to access this internal variable.

### 3.2.2. InitHardware

Initializes device. After successful call of this function, the device can be used.

```
public static extern int InitHardware(out byte CalibOK);
```

**Parameters:**

*CalibOK* – this variable will be filled with information whether the calibration data in the device is ok

1 – calibration data are ok

0 – calibration data in the device are broken

**Return value:**

*ERROR\_OK* – device was initialized successfully

*ERROR\_UNABLE\_TO\_LOAD\_EM77X* – unable to load em77x driver

*ERROR\_INIT\_FAILED* – device initialization failed (one of the reason can be, that the device isn't connected)

*ERROR\_FPGA\_CONFIG\_FAILED* – FPGA initialization failed

*ERROR\_DK\_NOT\_ENABLED* – the DK isn't enabled in the connected device

*ERROR\_DRIVER\_NOT\_LOADED* – the driver m770drv.dll wasn't loaded before this call (use LoadDriver function to load it)

### 3.2.3. GetDeviceList

Obtains the list of the connected devices. This function is used when several 770 oscilloscopes are connected.

```
public static extern void GetDeviceList(out uint DevList);
```

**Parameters:**

*DevList* – array with length of 16, which is filled with the serial numbers of the found devices

**Return value:** -

### 3.2.4. InitHardwareFromList

Initializes selected device. This function is used when several 770 oscilloscopes are connected. After successful call, the active device can be selected by *UseDevice* function.

```
public static extern int InitHardwareFromList(int index,out byte CalibOK);
```

**Parameters:**

*index* – indicates device index from the list (obtained by *GetDeviceList* function)

*CalibOK* – this variable will be filled with information whether the calibration data in the device are ok

1 – calibration data are ok

0 – calibration data in the device are broken

**Return value:**

*ERROR\_OK* – device was initialized successfully  
*ERROR\_UNABLE\_TO\_LOAD\_EM77X* – unable to load em77x driver  
*ERROR\_INIT\_FAILED* – device initialization failed (one of the reason can be, that the device isn't connected)  
*ERROR\_FPGA\_CONFIG\_FAILED* – FPGA initialization failed  
*ERROR\_DK\_NOT\_ENABLED* – the DK isn't enabled in the connected device  
*ERROR\_DRIVER\_NOT\_LOADED* – the driver m770drv.dll wasn't loaded before this call (use LoadDriver function to load it)

**3.2.5. UseDevice**

Selects the device, which will be affected by calls of other functions (such as *SetTimeBase*, *SetRange* ...). This function is used when several 770 oscilloscopes are connected.

```
public static extern void UseDevice(int index);
```

**Parameters:**

*index* – indicates device index from the list (obtained by *GetDeviceList* function)

**Return value:** -

**3.3. Functions, that set the data acquisition parameters****3.3.1. SetTimeBase**

Sets selected timebase.

```
public static extern void SetTimeBase(uint ns);
```

**Parameters:**

*ns* – amount of nanoseconds per one division (50 samples)

**Return value:** -

**Remark:**

The list of available timebases can be obtained by *GetTimeBaseList* function.

**3.3.2. SetRange**

Sets selected range to the selected channel.

```
public static extern int SetRange(int channel, int mV);
```

**Parameters:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*mV* – determines range, which is to be set (amount of mV per division)



**Return value:**

The vertical shift which must be set to keep the position of 0V at the same place.

**Remark:**

If it is necessary to keep the 0V position on the screen intact, it is necessary to set the vertical shift using function SetVert to the return value.

**3.3.3. SetCoupling**

Sets selected coupling on the channel.

```
public static extern void SetCoupling(int channel, int coupling);
```

**Parameters:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*coupling* – determines coupling

*COUPLING\_AC* – AC coupling

*COUPLING\_DC* – DC coupling

**Return value: -****3.3.4. SetVert**

Sets specified shift to the selected channel.

```
public static extern int SetVert(int channel, int shift);
```

**Parameters:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*shift* – shift value from 0 to 4095, where 0 moves the waveform maximally downwards and 4095 moves the waveform maximally upwards

**Return value:**

Position on the screen (quantization level), which corresponds with 0V.

**3.3.5. SetTriggerLevel**

Sets threshold on the selected source.

```
public static extern void SetTriggerLevel(int channel,int level);
```

**Parameters:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*level* – determines quantization level (0 to 255), which will be considered as threshold

**Return value:** -

### 3.3.6. SetTriggerCount

Sets amount of trigger events necessary to start acquisition on the selected trigger level.

```
public static extern void SetTriggerCount(int level, int amount);
```

**Parameters:**

*level* – determines level of the trigger system

*TRIGGER\_LEVEL\_PRIMARY* – primary level

*TRIGGER\_LEVEL\_SECONDARY* – secondary level

*amount* – amount of trigger events necessary to start acquisition

**Return value:** -

### 3.3.7. SetTriggerLength

Sets minimal length of valid trigger event.

```
public static extern void SetTriggerLength(int level, int samples);
```

**Parameters:**

*level* – determines level of the trigger system

*TRIGGER\_LEVEL\_PRIMARY* – primary level

*TRIGGER\_LEVEL\_SECONDARY* – secondary level

*samples* – minimal length of the valid trigger event from 8 to 131068 with step of 4. Value 0 turns the digital filter off.

**Return value:** -

### 3.3.8. SetTriggerMode

Sets trigger mode.

```
public static extern void SetTriggerMode(int mode);
```

**Parameters:**

*mode* – trigger mode

*TRIGGER\_MODE\_AUTO* – in case, that the trigger event doesn't occur for longer time, the acquisition starts anyway

*TRIGGER\_MODE\_NORMAL* – the acquisition starts only on valid trigger event

*TRIGGER\_MODE\_MANUAL* – the acquisition starts immediately after call of StartMeasurement function

**Return value:** -

### 3.3.9. SetAfterTriggerLength

Sets amount of samples acquired after trigger event.

```
public static extern void SetAfterTriggerLength(int samples);
```

**Parameters:**

*samples* – amount of samples acquired after trigger event (from 1 to 1048576 samples)

**Return value:** -

### 3.3.10. SetHoldOff

Sets length of hold-off.

```
public static extern void SetHoldOff(int samples);
```

**Parameters:**

*samples* – length of hold-off in samples from 4 to 1048576 with step of 4

**Return value:** -

### 3.3.11. SetTriggerSource

Sets selected trigger sources on the selected level of the trigger system.

```
public static extern void SetTriggerSource(int level, int sources);
```

**Parameters:**

*level* – determines level of the affected trigger system

*TRIGGER\_LEVEL\_PRIMARY* – primary level

*TRIGGER\_LEVEL\_SECONDARY* – secondary level

*sources* – sum of following constants determines selected sources:

*TRIGGER\_SOURCE\_A* – channel A

*TRIGGER\_SOURCE\_B* – channel B

*TRIGGER\_SOURCE\_E* – external trigger input

**Return value:** -

### 3.3.12. SetTriggerEdge

Sets triggering on the selected edge (leading or trailing) on the selected source.

```
public static extern void SetTriggerEdge(int level, int source, int edge);
```

**Parameters:**

*level* – determines level of the affected trigger system

*TRIGGER\_LEVEL\_PRIMARY* – primary level

*TRIGGER\_LEVEL\_SECONDARY* – secondary level

*source* – determines the source, whose sensitivity is affected

*TRIGGER\_SOURCE\_A* – channel A

*TRIGGER\_SOURCE\_B* – channel B

*TRIGGER\_SOURCE\_E* – external trigger input

*edge* – determines sensitivity

*TRIGGER\_EDGE\_LEADING* – sensitivity on the leading edge

*TRIGGER\_EDGE\_TRAILING* – sensitivity on the trailing edge

**Return value:** -

### 3.3.13. SetProbe

Sets device to measure with selected probe accurately.

```
public static extern void SetProbe(int channel, int probe);
```

**Parameters:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*probe* – determines connected probe

*PROBE\_1\_1* – probe 1:1

*PROBE\_1\_10* – probe with attenuation 1:10 or higher

**Return value:** -

**Remark:**

This function sets correction of the input impedance inaccuracy. This correction is turned on with probes with attenuation of 1:10 or higher.

### 3.3.14. SetMemorySize

Sets length of memory used for data acquisition.

```
public static extern void SetMemorySize(int size);
```

**Parameters:**

*size* – length of memory used for data acquisition (valid values are: 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288 and 1048576)

**Return value:** -

### 3.3.15. SetCompensationGenerator

Turns compensation generator (square wave with frequency of 1.465kHz on the E connector) on/off.

```
public static extern void SetCompensationGenerator(int onoff);
```

**Parameters:**

*onoff* – determines whether the compensation generator should be turned on/off

*ONOFF\_OFF* – turns generator off

*ONOFF\_ON* – turns generator on

**Return value:** -

### 3.3.16. SetGround

Grounds inputs on the selected channel.

```
public static extern void SetGround(int channel, int onoff);
```

**Parameters:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*onoff* – determines, whether the channel should be grounded

*ONOFF\_OFF* – grounds channel

*ONOFF\_ON* – disconnects channel from ground

**Return value:** -

### 3.3.17. SetDigitalShielding

Turns digital shielding on the selected channel on/off.

```
public static extern void SetDigitalShielding(int channel, int onoff, int level);
```

**Parametre:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*onoff* – determines whether the digital shielding will be turned on or off

*ONOFF\_OFF* – turns the digital shielding off

*ONOFF\_ON* – turns the digital shielding on

*level* – determines the digital shielding factor (valid values are: 2, 4, 8, 16, 32 and 64)

**Return value:** -

**Remark:**

In order to make the digital shielding work correctly, it is necessary to have a stable and well triggered signal on the input. Digital shielding prolongs the oscilloscope response to the signal change (the higher the factor the longer the response).

## 3.4. Data acquisition functions

### 3.4.1. StartMeasurement

Starts/restarts data acquisition.

```
public static extern void StartMeasurement();
```

**Parameters:** -

**Return value:** -

### 3.4.2. IsDataReady

Returns the data acquisition status.

```
public static extern int IsDataReady();
```

**Parameters:** -

**Return value:**

Indicates data acquisition status. It is one of following constants:

*DATA\_NOT\_TRIGGERED* – valid trigger event hasn't occurred

*DATA\_TRIGGERED* – the acquisition is triggered and is in progress

*DATA\_READY* – the acquisition is finished, data are prepared to be transferred to computer. This transfer can be done by GetData function.

*DATA\_ERROR* – the communication with device is broken, it is not possible to determine acquisition status

### 3.4.3. GetReconstructionPercentage

Returns ratio of the measured data to measured data + interpolated data.

```
public static extern void GetReconstructionPercentage(out int channelA, out int channelB);
```

**Parameters:**

*channelA* – amount of channel A, that was measured (percentage)

*channelB* – amount of channel B, that was measured (percentage)

**Return value:** -

**Remark:**

This function works only in case, that the selected timebase uses sampling mode (*SamplingMulti* > 1).

### 3.4.4. GetData

Transfers data from device to computer.

```
public static extern void GetData(out Sample data, out int length);
```

**Parameters:**

*data* – array, which contains 1048576 items of TSample type. This array is filled with the data transferred from the device. Last transferred sample from device is written to

the end of array (item index 1048575), first sample is written to the item index 1048576 - *length*

*length* – amount of samples, that were transferred from device

**Return value:** -

### 3.5. Other functions

#### 3.5.1. GroundPositionToShift

Returns vertical shift, which is necessary to set, to have the 0V at the selected position of the screen (quantization level).

```
public static extern int GroundPositionToShift(int channel, int position);
```

**Parameters:**

*channel* – determines channel

*CHANNEL\_A* – channel A

*CHANNEL\_B* – channel B

*position* – screen position (quantization level), where the 0V should be placed

**Return value:**

The vertical shift value, which places 0V to the selected screen position (quantization level). Set this value using SetVert to place 0V to the selected position.