Programmer's Manual

TiePie Dll's

for:

TP112
TP208
TP508
TP801 AWG ISA
TP801 AWG PCI
TE6100

Handyprobe HP2
Handyscope HS2
Handyscope HS3 AWG
Handyscope HS4 (Diff)
TiePieSCOPE HS508
TiePieSCOPE HS801 AWG

Revision 1.20

Table of contents

Perform my first measurement

Before performing a measurement, the instrument must first be initialized, using the routine InitInstrument. If this routine returns a non-zero value the initialization has failed and it is not possible to perform any measurements.

After initializing the hardware you can:
- modify the measurement settings
- start a measurement

These two actions can be executed in any order and as often as required.

Finally, when the instrument is no longer required, the routine ExitInstrument has to be called to deactivate the instrument and free any used resources.

Example in Pascal code:

```
const E_NO_ERRORS = 0;

if InitInstrument = E_NO_ERRORS then      {initialize instrument}
begin
  while bMeasureMore do
  begin
    if bChangeSettings then
    begin
      dwOldFreq := GetFrequency;                  {query setting}
      SetFrequency( ( dwOldFreq * 10 ) - 1000 ); {change setting}
    end; { if }
    if ADC_Start = E_NO_ERRORS then               {start measure}
    begin
      while not ADC_Ready do                   {wait until ready}
      begin
        { do nothing }
      end; { while }
      ADC_GetData( @wCh1Data, @wCh2Data );        {retrieve data}
    end; { if }
  end; { while }
end { if }
else
begin
  writeln( 'Error: No hardware found...' );
end; { else }
```

Legend:		
	bold	= reserved words
	123	= number
	italic	= comment

Error codes

Code Names   Code Values

```
                              Hexadecimal         Binairy
E_INVALID_VALUE               = 0x0020;  /*0000000000100000*/
E_INVALID_CHANNEL             = 0x0010;  /*0000000000010000*/
E_NO_GENERATOR                = 0x0008;  /*0000000000001000*/
E_NOT_SUPPORTED               = 0x0004;  /*0000000000000100*/
E_NOT_INITIALIZED             = 0x0002;  /*0000000000000010*/
E_NO_HARDWARE                 = 0x0001;  /*0000000000000001*/
E_NO_ERRORS                   = 0x0000;  /*0000000000000000*/
```

Defined constants

For several programming environments declaration files (header files) are available. These files contain declarations for all the available functions in the DLL, but also declarations of many used constants, like for trigger sources.

It is recommended that the constants from these declaration files are used in the application that uses the DLL. When in a future release of the DLL some values have changed, they will be adapted in the declaration file as well, so the application only needs to be recompiled, it will not affect the rest of the program.

All channel related routines use a channel parameter to indicate for which channel the value is meant:

Ch1 = 1
Ch2 = 2
Ch3 = 3
Ch4 = 4

The routines that deal with the MeasureMode use different values:

mCh1 = 1
mCh2 = 2
mCh3 = 4
mCh4 = 8

Search and Initialize the Instrument

word InitInstrument (word wAddress);

Descriptions: Initialize the hardware of the instrument. Set default measurement settings, allocate memory and obtain the calibration constants etc.
Parallel port connected instruments, USB instruments and PCI bus instruments detect the hardware by themselves and ignore the address parameters.

Input: wAddress The hardware address of the instrument should be passed to this routine.

Output: Returnvalue E_NO_ERRORS;
E_NO_HARDWARE

Note All instruments have their calibration constants in internal, non-volatile memory, except for the TP208 and TP508. These have to be calibrated using internal routines. This is done automaticallly at first startup everyday. Some relays will begin to click.

Close the Instrument

word ExitInstrument (void);

Description: Close the instrument. Free any allocated resources and memory, place the relays in their passive state, etc.
Only call this routine when the instrument is no longer required

Input: -

Output: Returnvalue E_NO_ERRORS;
E_NOT_INITIALIZED

Note Calling ExitInstrument in LabView causes LabView no longer to be able to connect to the instrument. LabView has to be closed and opened again to restore the contact. Therefor, only use ExitInstrument when the instrument is no longer required, right before closing LabView.

Get the calibration date

word GetCalibrationDate (dword *dwDate);

Description: This routine returns the calibration date of the instrument. The date is encoded in a packed 32 bit variable:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
|<--------------------->|<--------------------->|<------------------------------------>|
|    day (8 bits)       |   (month (8 bits)     |         year (16 bits)               |
```

Example decoding routine in C/C++:

```
        day   =  number >> 24;                    /* highest 8 bits */
        month = (number >> 16) & 0xFF;            /* middle  8 bits */
        year  =  number & 0xFFFF;                 /* lowest 16 bits */
```

Input: -

Output: **dwDate** The calibration date

Returnvalue E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Get the instrument serial number

word GetSerialNumber (dword *dwSerialNumber);

Description: This routine returns the Serial Number of the instrument. This number is hardcoded in the hardware. TP112, TP208 and TP508 do not have a serial number in the instrument.

Input: -

Output: **dwSerialNumber** the serial number

Returnvalue E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Determine the available input sensitivities

word GetAvailableSensitivities(double *dSensitivities);

description: This routine retrieves the available input sensitivities from the hardware and stores them in an array.
dSensitivities is an 20 elements large array. The caller must ensure that there is enough space in the arrays to contain the data. Therefor both the arrays must be at least

```
     20 * sizeof(double)
```

At return, all elements containing a non-zero value, contain an input sensitivity. This is a full scale value. So if an element contains the value 4.0, the input sensitivity is 4 Volt full scale, enabling to measure input signals from -4 Volt - +4 Volt.

input: -

output:

dSensitivities	the array of input sensitivities
Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Determine the available input resolutions

word GetAvailableResolutions(double *dResolutions);

description: The Handyscope HS3 and Handyscope HS4 support different, user selectable input resolutions. This routine retrieves the available input resolutions from the hardware and stores them in an array.
dResolutions is an 20 elements large array. The caller must ensure that there is enough space in the arrays to contain the data. Therefor both the arrays must be at least

```
     20 * sizeof(double)
```

At return, all elements containing a non-zero value, contain an input resolution in number of bits.

input: -

output:

dResolutions	the array of input sensitivities
Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Get the number of input channels

word GetNrChannels(word *wNrChannels);

Description: This routine returns the number of input channels of the instrument.

Input: -

Output: wNrChannels the number of channels
Returnvalue E_NO_ERRORS
E_NO_HARDWARE

Get the maximum sampling frequency

dword GetMaxSampleFrequency(void);

Description: The different instruments have different maximum sampling frequencies. This routine queries the maximum sampling frequency.

Input: -

Output: Returnvalue The maximum sampling frequency the instrument supports, in Hz.

Get the maximum record length

dword GetMaxRecordLength(void);

Description: The different instruments have different record lengths. This routine queries the maximum available record length per channel, in samples.

Input: -

Output: Returnvalue The maximum record length the instrument supports, in number of samples.

Check for availability of DC hardware offset adjustment

word GetDCLevelStatus(void);

Description: Some instruments support DC Hardware offset adjustment. This routine checks if the DC Level is supported.

Input: -

Output: Returnvalue E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Check for a square wave generator

word GetSquareWaveGenStatus(void);

Description: Some instruments have a built-in square wave generator, the HS508 for example. This routine checks the presence of the generator.

Input: -

Output: Returnvalue E_NO_ERRORS
E_NO_GENERATOR
E_NO_HARDWARE

Check for a function generator

word GetFunctionGenStatus(void);

Description: The TiePieSCOPE HS801, TP801 and Handyscope HS3 can have a built-in arbitrary waveform generator. When this function returns E_NO_GENERATOR, the HS801, TP801 or Handyscope HS3 is equipped with a simple square wave generator.

Input: -

Output: Returnvalue E_NO_ERRORS
E_NO_GENERATOR
E_NO_HARDWARE

Get the maximum amplitude of the function generator

word GetFuncGenMaxAmplitude(double *dAmplitude);

Description: The maximum output voltage for the TiePieSCOPE HS801 and Handyscope HS3 generator is 12 Volt, the maximum output voltage for the TP801 generator is 10 Volt. This routine determines the maximum voltage.

Input: -

Output: dAmplitude The maximum amplitude the generator supports.

Returnvalue E_NO_ERRORS
E_NO_GENERATOR
E_NO_HARDWARE

Start a measurement

word ADC_Start;

Description:	This routine writes any new instrument setting information to the hardware and then starts the measurement. If the hardware is already measuring, this measurement is aborted. Previous measured data is lost	
Input:	-	
Output:	Returnvalue	E_NOT_INITIALIZED E_NO_ERRORS E_NO_HARDWARE

Check if the hardware is measuring

word ADC_Running;

Description:	This routine checks if the hardware is currently measuring	
Input:	-	
Output:	Returnvalue	0 = not measuring 1 = measuring

Abort a running measurement

word ADC_Abort;

Description:	This routine aborts a running measurement. Any measured data is lost. It is not required to abort a running measurement before starting a new one, StartMeasurement does this already.	
Input:	-	
Output:	Returnvalue	E_NOT_INITIALIZED E_NO_ERRORS E_NO_HARDWARE

Read the trigger status

word ADC_Triggered;

Description:	This routine reads the trigger status from the hardware.	
Input:	-	
Output:	Returnvalue	0 = not triggered 1 = Ch1 caused trigger 2 = Ch2 caused trigger 4 = External input caused trigger
Remark:	Returnvalue can be a combination of indicated values.	

Read the measurement status

word ADC_Ready;

Description:	This routine checks if the measurement is ready or not.	
Input:	-	
Output:	Returnvalue	0 = not ready 1 = ready

Force a trigger

word ADC_ForceTrig;

Description:	This routine forces a trigger when the input signal will not meet the trigger specifications. This allows to do a measurement and see the signal.	
Input:	-	
Output:	Returnvalue	E_NOT_INITIALIZED E_NO_ERRORS E_NO_HARDWARE

Retrieve the measured data in binary format

word ADC_GetData(word *wCh1, word *wCh2);

Description: This routine transfers the measured data from the acquisition memory in the hardware via the dll into the memory in the application. The measured data is returned in binary values. A value of 0 corresponds to -Sensitivity, 32768 corresponds to 0 and 65535 to +Sensitivity in Volts. wCh1 and wCh2 are arrays. The caller must ensure that there is enough space in the arrays to contain the data. Therefor the arrays must be at least

```
RecordLength * sizeof( word )
```

Input: -

Output:

wCh1 The array to which the measured data of channel 1 should be passed.

wCh2 The array to which the measured data of channel 2 should be passed.

Returnvalue E_NO_ERRORS
E_NO_HARDWARE

Retrieve the measured data in Volts

word ADC_GetDataVolt(double *dCh1, double *Ch2);

Description: This routine transfers the measured data from the acquisition memory in the hardware via the dll into the memory in the application. The measured data is returned in volt. dCh1 and dCh2 are arrays. The caller must ensure that there is enough space in the arrays to contain the data. Therefor the arrays must be at least `RecordLength * sizeof( double )`

Input: -

Output:

dCh1 The array to which the measured data of channel 1 should be passed.

dCh2 The array to which the measured data of channel 2 should be passed.

Returnvalue E_NO_ERRORS
E_NO_HARDWARE

Get the data from a specific channel in binary format

word ADC_GetDataCh(word wCh, word *wData);

Description: This routine transfers the measured data of one channel from the acquisition memory in the hardware via the dll into the memory in the application. The measured data is returned in binary values. A value of 0 corresponds to -Sensitivity, 32768 corresponds to 0 and 65535 to +Sensitivity in Volts. wData is an array. The caller must ensure that there is enough space in the array to contain the data. Therefor the array must be at least

```
RecordLength * sizeof( word )
```

Input:	wCh	Indicates from which channel the data has to be retrieved
Output:	wData	The array to which the measured data of the requested channel should be passed.
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Get the date from a specific channel in Volts

word ADC_GetDataVoltCh(word wCh, double *Data);

Description: This routine transfers the measured data of one channel from the acquisition memory in the hardware via the dll into the memory in the application. The measured data is returned in volt. dData is an array. The caller must ensure that there is enough space in the array to contain the data. Therefor the array must be at least

```
RecordLength * sizeof( double )
```

Input:	wCh	Indicates from which channel the data has to be retrieved
Output:	dData	The array to which the measured data of the requested channel should be passed.
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Get all digital input values

word GetDigitalInputValues(word *wValues);

Desription: The TP112 has eight digital inputs, which are sampled simultaneously with the analog input channels.
This routine transfers the measured digital values from the memory in the DLL into the memory in the application. The measured data is returned in binary values. Each bit in the digital data words represents a digital input. wValues is an array. The caller must ensure that there is enough space in the array to contain the data. Therefor the array must be at least

```
     RecordLength * sizeof(word)
```

Input:	-	
Output:	Returnvalue	E_NO_ERRORS E_NOT_SUPPORTED E_NO_HARDWARE

Get one sample of the digital input values

word GetOneDigitalValue(word wIndex; word *wValue);

Description: This routine transfers a single digital input value from the memory in the DLL to the memory of the application.

Input:	wIndex	The index of the measured data point, relative to the trigger point (negative for pre samples, positive for post samples)
Output:	wValue	Return address for the digital input value.
	Returnvalue	E_NO_ERRORS E_NOT_SUPPORTED E_NO_HARDWARE

Example of use of the routines

To use the measurement routines, your application could look like the following:

```
.
.
ADC_Start;
StartTime := GetCurrentTime;
while bContinue do
begin
  if GetCurrentTime > ( StartTime + TimeOut ) then
  begin
    ADC_ForceTrig;
  end; { if }
  if ADC_Ready = 1 then
  begin
    ADC_Getdata( Ch1WordArray, Ch2WordArray );
    ADC_Start;
    StartTime := GetCurrentTime;
    ApplicationProcessData;
  end; { if }
end; { while }
.
.
```

Streaming measurements

It is possible to do streaming measurements with the Handyscope HS3. Each time a specified number of samples is measured (the record length), they can be transferred to the computer and processed while the hardware continues measuring uninterrupted.

This way of measuring uses a callback function or an event to let the application know new samples are available.

Using DataReady callback function

When new data is available, a function in the application can be called. The DLL has a function pointer which has to be set to this function, using

word SetDataReadyCallback(TDataReady pAddress)

description	This routines sets the pointer for the Ready function, which will be called when new data is available	
input:	pAddress	a pointer to a function with the following prototype: void DataReady(void)
output	return value	E_NO_HARDWARE E_INVALID_VALUE E_NO_ERRORS

In the callback function, the data can be read from the instrument, using the ADC_GetData routines.

Using DataReady event

When new data is available, an event can be set by the DLL. The user must reset the event when the data is read.

word SetDataReadyEvent(HANDLE hEvent)

description	This routine sets the event handle for the DataReady event	
input	hEvent	the event handle
output	return value	E_NO_HARDWARE E_NO_ERRORS

Setting up streaming measurements

To tell the instrument a streaming measurement has to be performed, folowing routine has to be used.

word SetTransferMode(dword dwMode);

Description: This routine tells the instrument what kind of measurement has to be performed.

Input: dwMode determines the requested data transfer mode. Possible values are:

tmBlock (0) default value. During the measurement, all data is stored in the instrument. When the measurement is ready, all dat a is transferred in one block to the computer. This is normal oscilloscope mode

tmStream (1) Each time during the measurement that new data is available, it will be transferred to the computer. So a measurement gives a constant stream of data.

Output: Returnvalue E_NO_ERRORS
E_NO_HARDWARE
E_INVALID_VALUE

Getting the current transfer mode

word GetTransferMode(dword *dwMode);

Description: This routine reads the current set transfer mode from the instrument.

Input: -

Output: dwMode holds the current data transfer mode.
Returnvalue E_NO_ERRORS
E_NO_HARDWARE

Performing streaming measurements

When the callback function has been created and the transfer mode is set to streaming mode, streaming measurements can be performed.

The sampling speed has to be set to the required values and the input channels have to be set to appropriate values (auto ranging does not work in streaming mode). The record length has to be set to the number of samples that has to be measured each measurement. There is no trigger and no pre- or post trigger available in streaming mode.

A streaming measurement is started with the before mentioned routine ADC_Start( ). During the measurement the callback function will be called each time new data is available. These can be used to update the screen of the application and show the measured data.

To stop a running measurement, call ADC_Abort( ). This will stop the running measurement.

Controlling the input resolution

The Handyscope HS3 and Handyscope HS4 support a number of different input resolutions.

Set the input resolution

word SetResolution(byte byResolution);

Description:	This routine sets the input resolution of the hardware. Use GetAvailableResolutions() to determine which resolutions are available.	
Input:	byResolution	the new resolution, in bits
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE
Remark:	When setting a new input resolution, the maximum sampling frequency of the hardware changes as well. Use GetMaxSampleFrequency() to determine the new maximum sampling frequency.	

Get the currrent input resolution

word GetResolution (byte *byResolution);

Description:	This routine retrieves the currently set input resolution in bits.	
Input:	-	
Output:	byResolution	the return address for the resolution
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Control the instrument configuration

The Handyscope HS3 allows to change it's instrument configuration. It supports the following configurations:

licHS3Norm	(0) operate as a 2 channel 12 bit instrument with 128K samples per channel and an Arbitrary Waveform Generator.
licHS3256K	(1) operate as a 2 channel 12 bit instrument with 256K samples per channel, without generator.
licHS3512K	(2) operate as a 1 channel 12 bit instrument, with 512K samples for the channel, without generator.

Set the instrument configuration

word SetIinstrumentConfig(word wMode);

Description:	This routine changes the Instrument configuration.	
Input:	wMode	The new configuration
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NO_HARDWARE E_NOT_SUPPORTED

Get the current instrument configuration

word GetIinstrumentConfig(word *wMode);

Description:	This routine returns the current Instrument configuration.	
Input:	-	
Output:	wMode	The current configuration
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE E_NOT_SUPPORTED

Control which channels are measured

The routines to get or set the measure mode use channel numbers. The following numbers are used:

mmCh1 = 1
mmCh2 = 2
mmCh3 = 4
mmCh4 = 8

Get the current measure mode

word GetMeasureMode(byte *byMode);

Description: This routine returns the current Measure Mode:

mmCh1	the signal at channel 1 is measured
mmCh2	the signal at channel 2 is measured
mmCh1 + mmCh2	the signals at channel 1 and 2 are measured simultaneously
mmCh3	the signal at channel 3 is measured
mmCh1 + mmCh3	the signals at channel 1 and 3 are measured simultaneously

Input: -

Output: **byMode** The current Measure Mode.
Returnvalue E_NO_ERRORS
E_INVALID_VALUE
E_NO_HARDWARE

Set the measure mode

word SetMeasureMode(byte byMode);

Description: This routine changes the Measure Mode, see also GetMeasureMode.

Input: **byMode** The new measure mode.

Output: **Returnvalue** E_NO_ERRORS
E_INVALID_VALUE
E_NO_HARDWARE

Get the current record length

dword GetRecordLength(void);

Description: This routine returns the total number of points to be digitized. The number of pre samples (number of samples to measure before the trigger occured) is calculated like this:

```
PreSamples = RecordLength - PostSamples.
```

Input: -

Output: Returnvalue The total number of points to be digitized per channel.

Set the record length

word SetRecordLength(dword wTotal);

Description: This routine sets the total number of points to be digitized. The maximum record length can be determined with theroutine GetMaxRecordLength(). The minimum is 0.

Input: wTotal The total number of points to be digitized per channel.

Output: Returnvalue E_NO_ERRORS
E_INVALID_VALUE
E_NO_HARDWARE

Remark: Setting a record length smaller than the number of post samples gives an E_INVALID_VALUE error

Get the current number of post samples

dword GetPostSamples(void);

Description: This routine returns the number of post samples to measure (the number of samples **after** the trigger has occured).

Input: -

Output: Returnvalue The current selected number of post samples to measure.

Set the number of post samples

word SetPostSamples(dword wPost);

Description: This routine sets the number of post samples. This number must be between 0 and the record length.

Input: wPost The requested number of post samples to measure.

Output: Returnvalue E_NO_ERRORS
E_INVALID_VALUE
E_NO_HARDWARE

Remark: Setting a number of post samples larger than the record length gives an E_INVALID_VALUE error

Get the current sampling frequency

double GetSampleFrequencyF(void);

Description:	This routine returns the current set sampling frequency in Hz. The minimum/maximum frequency supported is instrument dependent.	
Input:	-	
Output:	Returnvalue	The current sampling frequency in Hz.

Set the sampling frequency

word SetSampleFrequencyF(double *dFreq);

Remarks:	The routine sets the sampling frequency. The hardware is not capable of creating every selected frequency so the hardware chooses the nearest allowed frequency to use, This is the frequency that is returned in dFreq.	
Input:	dFreq	The requested sampling frequency in Hz
Output:	dFreq	The actual selected sampling frequency in Hz
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Note The above two functions are replacing the existing functions GetSampleFrequency() and SetSampleFrequency().

Get the sample clock status

word GetExternalClock(word *wMode);

Description: This routine determines whether the sampling clock uses the internal Crystal oscillator or the external clock input
Only 50 MHz devices support external clock input

Input: -

Output: wMode The status of the internal clock,
0 = clock internal
1 = clock external

Returnvalue E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Set the sample clock status

word SetExternalClock(word wMode);

Description: This routine sets the sampling clock mode: is the internal crystal oscillator used or the external clock input?
Only 50 MHz devices support external clock input

Input: wMode 0 = internal clock
1 = external clock

Output: Returnvalue E_NO_ERRORS
E_INVALID_VALUE
E_NOT_SUPPORTED
E_NO_HARDWARE

Control the analog input channels

The routines to adjust channel settings use channel numbers. The following numbers are used:

Ch1 = 1
Ch2 = 2
Ch3 = 3
Ch4 = 4
etc.

Get the current input sensitivity

word GetSensitivity (byte byCh, double *dSens);

Description: This routine returns the current selected full scale input sensitivity in Volts for the selected channel.

Input: byCh The channel whose current Sensitivity is requested (1, 2, 3, 4)

Output: dSens The current sensitivity.

Returnvalue E_NO_ERRORS
E_INVALID_CHANNEL
E_NO_HARDWARE

Set the input sensitivity

word SetSensitivity(byte byCh, double *dSens);

Description: This routine sets the Sensitivity for the selected channel. The hardware can only deal with a limited number of ranges. The sensitivity that matches the entered sensitivity best is used.This is the value that will be returned in dSens.

Input: byCh The channel whose Sensitivity is to be changed (1, 2, 3, 4)

dSens The new Sensitivity in Volts

Output: dSens Contains the actual set Sensitivity, on return

Returnvalue E_NO_ERRORS
E_INVALID_CHANNEL
E_NO_HARDWARE

Get the current auto ranging status

word GetAutoRanging(byte byCh, byte *byMode);

Description: This routine returns the current autoranging mode:

0 : autoranging is off

1 : autoranging is on.

If autoranging is on then for a channel the sensitivity will be automatically adjusted if the input signal becomes too large or too small.

When a measurement is performed, the data is examined. If that data indicates another range will provide better results, the hardware is set to a new sensitivity. The next measurement that is performed, will be using that new sensitivity. Autoranging has no effect on a current measurement.

Input: byCh The channel whose current Autoranging mode is requested (1, 2, 3 ,4).

Output: byMode The Autoranging mode.

Returnvalue E_NO_ERRORS
E_INVALID_CHANNEL
E_NO_HARDWARE

Set the auto ranging status

word SetAutoRanging(byte byCh, byte byMode);

Description: This routine selects the autoranging mode:

0 : turn Autoranging off

1 : turn Autoranging on.

See also GetAutoRanging.

Input: byCh The channel whose Autoranging mode has to be set (1, 2, 3, 4).

byMode The new value for the Autoranging mode.

Output: Returnvalue E_NO_ERRORS
E_INVALID_CHANNEL
E_INVALID_VALUE
E_NO_HARDWARE

Get the current input coupling

word GetCoupling(byte byCh, byte *byMode);

Description:	This routine returns the current signal coupling for the selected channel: ctAC : coupling AC (0) ctDC : coupling DC (1) In DC mode both the DC and the AC components of the signal are measured. In AC mode only the AC component is measured.	
Input:	byCh	The channel whose current coupling is requested (1, 2, 3, 4)
Output:	byMode	The current coupling.
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE E_NO_HARDWARE

Set the input coupling

word SetCoupling(byte byCh, byte byMode);

Description:	This routine changes the signal coupling for the selected channel. See also GetCoupling.	
Input:	byCh	The channel whose Coupling is to be changed (1, 2, 3, 4).
	byMode	The new coupling for the selected channel (0 or 1).
Output:	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE E_NO_HARDWARE

Get the current DC level value

word GetDcLevel(byte byCh, double *dLevel);

Description:	This routine returns the current DC Level value for the selected channel. This voltage is added to the input signal before digitizing. This is used to shift a signal that is outside the current input range into the input range.	
Input:	byCh	The channel whose DC Level is requested (1, 2, 3, 4)
Output:	dLevel	The current DC Level.
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_NOT_SUPPORTED E_NO_HARDWARE

Set the DC level value

word SetDcLevel(byte byCh, double dLevel);

Description:	This routine is used to change the DC Level for the selected channel. The DC Level has a minimum of -2*sensitivity and a maximum of +2*sensitivity. If the sensitivity changes, the DC level is automatically checked and clipped if neccessary. See also GetDcLevel.	
Input:	byCh	The channel whose DC Level is to be set (1, 2, 3, 4)
	dLevel	The new DC Level in Volts
Output:	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE

Get the current trigger source

word GetTriggerSource(byte *bySource);

Description: This routine is used to retrieve the current Trigger Source.

tsCh1	(0)	Channel 1
tsCh2	(1)	Channel 2
tsCh3	(2)	Channel 3
tsCh4	(3)	Channel 4
tsExternal	(4)	a digital external signal
tsAnalogExt	(5)	an analog external signal
tsAnd	(6)	Channel 1 **AND** Channel 2
tsOr	(7)	Channel 1 **OR** Channel 2
tsXor	(8)	Channel 1 **XOR** Channel 2
tsNoTrig	(9)	no source, measure immediately
–	(10)	not used
tsPxiExt	(11)	PXI bus digital trigger signals
ts GenStart	(12)	start of the Handyscope HS3 generator
tsGenStop	(13)	stop of the Handyscope HS3 generator
tsGenNew	(14)	each new period of the HS3 generator

Input: -

Output: **bySource** The current trigger source.
Returnvalue E_NO_ERRORS,
E_INVALID_VALUE
E_NO_HARDWARE

Set the trigger source

word SetTriggerSource(byte bySource);

Description: This routine sets the trigger source.

Input: **bySource** The new trigger source.

Output: **Returnvalue** E_NO_ERRORS,
E_INVALID_VALUE
E_NOT_SUPPORTED
E_NO_HARDWARE

Note Not all devices support all Trigger Sources. If the Trigger Source is not supported, the error value **E_NOT_SUPPORTED** is returned.

Get the current trigger mode

word GetTriggerMode(byte *byMode);

Description: This routine is used to query the current Trigger Mode.

tmRising	(0)	trigger on rising slope
tmFalling	(1)	trigger on falling slope
tmInWindow	(2)	trigger when signal gets inside window
tmOutWindow	(3)	trigger when signal gets outside window
tmTVLine	(4)	trigger on TV line sync pulse
tmTVFieldOdd	(5)	trigger on TV odd frame sync pulse
tmTVFieldEven	(6)	trigger on TV even frame sync pulse

Input: -

Output: byMode The current trigger mode.

Returnvalue E_NO_ERRORS
E_INVALID_VALUE
E_NO_HARDWARE

Set the trigger mode

word SetTriggerMode(byte byMode);

Description: This routine is used to set the Trigger Mode for all channels. See also GetTriggerMode. Some trigger modes are not available on all instruments, in that case, the value E_NOT_SUPPORTED will be returned.

Input: byMode The new trigger mode.

Output: Returnvalue E_NO_ERRORS
E_INVALID_VALUE
E_NOT_SUPPORTED
E_NO_HARDWARE

Note When edge triggering (Rising or Falling) is selected, the instrument will not trigger on a constant level DC signal

Get the current trigger mode for a specific channel

word GetTriggerModeCh(byte byCh; byte *byMode);

Description:	This routine is used to get the current Trigger Mode for a specific channel. Some trigger modes are not available on all instruments, in that case, the value E_NOT_SUPPORTED will be returned.	
Input:	byCh	The channel to set the trigger mode for
	byMode	The new trigger mode.
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE E_INVALID_CHANNEL

Set the trigger mode for a specific channel

word SetTriggerModeCh(byte byCh; byte byMode);

Description:	This routine is used to set the Trigger Mode for a specific channel. See also GetTriggerMode. Some trigger modes are not available on all instruments, in that case, the value E_NOT_SUPPORTED will be returned.	
Input:	byCh	The channel to set the trigger mode for
	byMode	The new trigger mode.
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE E_INVALID_CHANNEL

Note When edge triggering (Rising or Falling) is selected, the instrument will not trigger on a constant level DC signal

Get the current trigger level

word GetTriggerLevel(byte byCh, double *dLevel);

Description:	This routine is used to retrieve the Trigger Level of the selected channel. The hardware starts to measure when the signal passes this level. The routine **SetTriggerMode** can be used to select the trigger slope.	
Input:	**byCh**	The channel whose Trigger Level is to be retrieved (1, 2, 3, 4).
Output:	**dLevel**	The current Trigger Level.
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_NO_HARDWARE

Set the trigger level

word SetTriggerLevel(byte byCh, double dLevel);

Description:	This routine is used to set the Trigger Level. The Trigger Level is valid if it is between **-sensitivity** and **+sensitivity**.	
Input:	**byCh**	The channel whose Trigger Level is to be set (1, 2, 3, 4).
	dLevel	The new Trigger Level in Volts.
Output:	**Returnvalue**	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE E_NO_HARDWARE

Note The Trigger Level applies only to analog trigger sources, not to digital trigger sources.

When window trigger is selected, the Trigger Level controls the upper level of the trigger window.

Get the current trigger hysteresis

word GetTriggerHys(byte byCh; double *dHysteresis);

Description: This routine is used to retrieve the current Trigger Hysteresis. The hysteresis is the minimum voltage change that is required to comply with the trigger conditions. This is used to minimize the influence of the noise on a signal on the trigger system.

Input:	byCh	The channel whose Trigger Hysteresis is to be retrieved (1, 2, 3, 4).
Output:	dHysteresis	The current Trigger Hysteresis.
	Returnvalue	E_NO_ERROR E_INVALID_CHANNEL E_NO_HARDWARE

Set the trigger hysteresis

word SetTriggerHys(byte byCh; double dHysteresis);

Description: This routine changes the hysteresis, see also GetTriggerHys.

Input:	byCh	The channel whose Trigger Hysteresis is to be set (1, 2, 3, 4).
	dHysteresis	The new trigger hysteresis.
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_INVALID_CHANNEL E_NO_HARDWARE

Upper and lower limits of the hysteresis:

Slope	Lower limit	Upper limit
rising	0	level + sens
falling	0	sens - level

Note The Trigger Hysteresis applies only to analog trigger sources, not to digital trigger sources.

When window trigger is selected, the Trigger Hysteresis controls the lower level of the trigger window.

The TE6100 has 8 digital external trigger inputs, at the PXI bus, which can be used to trigger the measurement. It is possible to select which inputs have to be used and if the inputs have to respond to a rising or a falling slope.

Select the PXI external trigger signals

word SetPXITriggerEnables(byte byEnables);

Description: This routine determines which of the eight PXI external trigger inputs have to be used. When more than one input is selected, trigger occurs when one or more inputs become active (logic OR). Which input state is active, is determined by the Slopes setting, see next page.

Input: byEnables a bit pattern that defines which inputs have to be used. Bit 0 represents input 0, bit 1 represents input 1 etc.
When a bit is high, the corresponding input is used.
When a bit is low, the corresponding input is not used.

Output: Returnvalue E_NO_ERRORS,
E_NOT_SUPPORTED
E_NO_HARDWARE

Get the current used PXI external trigger signals

word GetPXITriggerEnables(byte *byEnables);

Description: This routine retrieves the currently selected PXI external trigger inputs.

Input: -

Output: byEnables a bit pattern that defines which inputs are currently used. See also the routine SetPXITrigger-Enables

Returnvalue: E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Set the PXI external trigger slopes

word SetPXITriggerSlopes(byte bySlopes);

Description: This routine determines for each PXI external trigger input individually whether it should respond to a falling or a rising slope.

Input: bySlopes a bit pattern that defines how the slope settings for each input is set.
Each bit represents an input, bit 0 represents input 0, bit 1 represents input 1 etc.
When a bit is high, the corresponding input responds to a rising slope.
When a bit is low, the corresponding input responds to a falling slope.

Output: ReturnValue E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Get the current PXI external trigger slopes

word GetPXITriggerSlopes(byte *bySlopes);

Description: This routines determines how the slope sensitivities for the PXI external trigger inputs are set.

Input: -

Output: bySlopes a bit pattern that defines how the slope settings for each input is set.
Each bit represents an input, bit 0 represents input 0, bit 1 represents input 1 etc.
When a bit is high, the corresponding input responds to a rising slope.
When a bit is low, the corresponding input responds to a falling slope.

Returnvalue E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Set the digital outputs

word SetDigitalOutputs(byte byValue);

Description:	The TP112 is equiped with 8 digital outputs, which can be set individually. This routine sets the status of the digital outputs.	
Input:	byValue	the new status of the outputs. Each bit represents an output.
Output:	Returnvalue	E_NO_ERRORS E_NOT_SUPPORTED E_NO_HARDWARE

Get the current status of the digital outputs

word GetDigitalOutputs(byte *byValue);

Description:	This routine gets the current status of the digital outputs.	
Input:	-	
Output:	byValue	the status of the outputs. Each bit represents an output.
	Returnvalue	E_NO_ERRORS E_NOT_SUPPORTED E_NO_HARDWARE

Get the current square wave generator frequency

double GetSquareWaveGenFrequency(void);

Description: Some instruments have a built-in generator, the HS508 for example. This routine returns the generator frequency in Hz.

Input: -

Output: **Returnvalue** The generator frequency in Hz.

Remarks: Not all instruments have a square wave generator, use the routine GetSquareWaveGenStatus() to check if a square wave generator is available

Set the square wave generator frequency

word SetSquareWaveGenFrequency(double *dFreq);

Remarks: The routine sets the frequency. The hardware is not capable of using every frequency so the hardware chooses the nearest legal frequency to use, this is the frequency that is returned in dFreq. See also GetGeneratorFrequency.

Input:	dFreq	the requested frequency in Hz. A value "zero" switches the output off
Output:	dFreq	the frequency that is actually made.
	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_NO_HARDWARE

Remarks: Not all instruments have a square wave generator, use GetSquareWaveGenStatus() to check if a square wave generator is available

Set the generator signal type

word SetFuncGenSignalType(word wSignalType);

Description: This routine sets the signal type of the function generator.

Input: wSignalType The requested signal type

stSine	(0)	Sine wave
stTriangle	(1)	Triangular wave
stSquare	(2)	Square wave
stDC	(3)	DC
stNoise	(4)	Noise
stArbitrary	(5)	Arbitrary signal

Output: Returnvalue: E_NO_ERRORS
E_NO_GENERATOR
E_INVALID_VALUE
E_NO_HARDWARE

Remark: When Arbitrary is selected, the contents of the function generator memory will be "played" continuously. This memory is used for every signal type, so each time when selecting Arbitrary, use the function FillFuncGenMemory() to fill the memory with the requested signal.

Get the current generator signal type

word GetFuncGenSignalType(word *wSignalType);

Description: This routine returns the currently selected signal type.

Input: -

Output: wSignalType The currently selected signal type
See SetFuncGenSignalType for possible values for wSignalType

Returnvalue E_NO_ERRORS
E_NO_GENERATOR
E_INVALID_VALUE
E_NO_HARDWARE

Set the generator amplitude

word SetFuncGenAmplitude(double dAmplitude);

Description:	This routine sets the output amplitude of the function generator in volts. When the requested amplitude is smaller than zero or larger than the maximum supported amplitude, E_INVALID_VALUE is returned and the requested value is ignored. When signal type DC is selected, the absolute amplitude of the signal is determined by the amplitude and the polarity is determined through the DC offset.	
Input:	dAmplitude	the function generator amplitude in Volts: 0 <= value <= MaxAmplitude
Output:	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Get the current generator amplitude

word GetFuncGenAmplitude(double *dAmplitude);

Description:	This routine determines the currently selected amplitude of the function generator	
Input:	-	
Output:	dAmplitude	the function generator amplitude in Volts: 0 <= value <= MaxAmplitude
	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Set the generator DC Offset

word SetFuncGenDCOffset(double dDCOffset);

Description:	This routine applies a DC offset to the output signal. The value is entered in Volts. When signal type **DC** is selected, the DC offset value is used to determine the polarity of the output signal. A value >= 0 Volt results in a positive output signal, a value < 0 Volt results in a negative output signal. The amplitude of the DC signal is determined through the Amplitude value.	
Input:	dDCOffset	the requested offset in Volts: -MaxAmpl <= value <= +MaxAmpl
Output:	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Get the current generator DC Offset

word GetFuncGenDCOffset(double *dDCOffset);

Description:	This routine determines the currently selected DC offset value of the function generator	
Input:	-	
Output:	dDCOffset	the currently selected DC Offset value
	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Set the generator signal symmetry

word SetFuncGenSymmetry(double dSymmetry);

Description: This routine sets the symmetry of the output signal. The symmetry can be set between 0 and 100. With a symmetry of 50, the positive part of the output signal and negative part of the output signal are equally long. With a symmetry of 25, the positive part of the output signal takes 25% of the total period and the negative part takes 75% of the total period.
With signal types DC, Noise and Arbitrary, the symmetry value is ignored.

Input:	dSymmetry	The requested symmetry value: 0 <= value <= 100
Output:	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Get the current generator signal symmetry

word GetFuncGenSymmetry(double *dSymmetry);

Description: This routine retrieves the currently selected symmetry of the output signal.

Input:	-	
Output:	dSymmetry	the current symmetry value
	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Set the generator frequency

word SetFuncGenFrequency(double *dFrequency);

Description: When signal type Sine, Triangular, Square or Noise is selected (DDS mode), this routine sets the frequency of the output signal of the function generator.

When signal type Arbitrary is selected, the frequency settings behaves slightly different. When 1024 samples are loaded into the waveform memory (DDS mode), this routine sets the frequency of the output signal. When more samples are loaded into the waveform memory (linear mode), this routine sets the frequency of the sampling clock of the function generator. Only a limited number of frequencies are available.

Input: dFrequency DDS mode: the requested frequency of the output signal:
0.001 <= dFrequency <= 2,000,000
Linear mode: the requested frequency of the sampling clock in 15 steps:
38.1, 610, 2441,
9765, 39062, 78125,
156250, 321500, 625000,
1250000, 2500000, 5000000,
10000000, 25000000, 50000000

Output: dFrequency the hardware can not support any arbitrary frequency within the available range. The value that was actually selected is returned.

Returnvalue E_NO_ERRORS
E_NO_GENERATOR
E_INVALID_VALUE
E_NO_HARDWARE

Get the current generator frequency

word GetFuncGenFrequency(double *dFrequency);

Description: This routine determines the currently set frequency.

Input: -

Output: dFrequency The currently set frequency in Hz

Returnvalue E_NO_ERRORS
E_NO_GENERATOR
E_INVALID_VALUE
E_NO_HARDWARE

Set the generator trigger source

word SetFuncGenTrigSource(byte bySource);

Description:	The Handyscope HS3 function generator can set to be started by an external trigger signal. This routine sets the function generator trigger source:		
	tsExtTrig	(4)	a digital external signal
	tsNoTrig	(9)	no source, generate immediately
	The default value is tsNoTrig		
Input:	bySource	the requested trigger source	
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE	

Get the current generator trigger source

word GetFuncGenTrigSource(byte *bySource);

Description:	This routine determines the currently selected function generator trigger source	
Input:	-	
Output:	bySource	the currently selected trigger source
	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE

Set the generator mode

word SetFuncGenMode(dword dwMode);

Description:	The Handyscope HS3 function generator can set to either linear mode or to DDS mode:	
	fmDDS	(1) DDS mode
	fmLinear	(2) Linear mode
Input:	dwMode	the requested function generator mode
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE

Get the current generator mode

word GetFuncGenMode(dword *dwMode);

Description:	This routine determines the currently selected function generator mode.	
Input:	-	
Output:	dwMode	the currently selected function generator mode
	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE

Fill the function generator waveform memory

word FillFuncGenMemory(dword wNrPoints; word *wFuncGenData);

description: This routine fills the function generator waveform memory with user defined data.

The generator can operate in two different modes: DDS and Linear. When operating in DDS mode, 1024 samples must be loaded. These 1024 samples will form one period of the output signal. When operating in Linear mode, the maximum record length samples (depends on the instrument, e.g. 65536 or 131072) must be loaded. These samples will form one period of the output signal.

The data must be in unsigned 16 bits values. A value of 32768 produces a 0 Volt signal, 65535 results in positive full output scale and a value of 0 results in negative full output scale.

The amplitude parameter of the function generator determines the exact value of full scale. If an amplitude of 8 Volt is selected, full scale will be 8 Volt.

Input:

Input:	dNrPoints	the number of waveform points that must be loaded: 1024 or 65536 or 131072. Also determines whether the function generator operates in DDS or Linear mode.
	wFuncGenData	an array of 1024, 65536 or 131072 unsigned 16 bits values, containing the signal that must be loaded. The caller must ensure that enough data is allocated.
Output:	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Remark: When generating a predefined signal, like e.g. a sinewave, the memory is filled with a sine wave pattern and the generator operates in DDS mode. So each time one selects signal type Arbitrary, the memory has to be filled again with the user defined pattern.

Set the generator output state

word SetFuncGenOutputOn(word wValue);

Description:	This routine switches the output of the function generator on or off.	
Input:	wValue	The new output state 0 output is off 1 output is on
Output:	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Get the current generator output state

word GetFuncGenOutputOn(word *wValue);

Description:	This routine determines the current setting of the function generator output	
Input:	-	
Output:	*wValue	The current setting of the output 0 output is off 1 output is on
	Returnvalue	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Generate bursts

```
word FuncGenBurst( word wNrPeriods );
```

Description: This routine will make the Handyscope HS3 generator generate a burst with a requested number of periods of the selected signal. When the burst is finished, the output will remain at the last generated amplitude value.

Input: wNrPeriods the requested number of periods to generate. Any value > 0 will switch on burst mode. The value 0 wil switch off burst mode and start continuous generation again.

Output: Returnvalue E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Note The output of the generator has to be switched on before burst mode is selected.

Resistance measurements

Some instruments have special hardware to perform resistance measurements.

Setup resistance measurements

```
word SetupOhmMeasurements( word wMode );
```

Description: This routine sets the instrument up to perform resistance measurements. Several properties of the instrument are adapted: input sensitivity, signal coupling, record length, sampling frequency, autoranging, trigger source, trigger timeout, acquisition mode. These are all brought to the required state and should not to be set to other values afterwards.

Input: **wMode**
0 switch resistance measurements off
1 switch resistance measurements on

Output: **returnvalue**
E_NO_ERRORS
E_INVALID_VALUE
E_NOT_SUPPORTED
E_NO_HARDWARE

Retrieve the resistance values

After resistance measurements are switched on, and a measurement is performed in the normal way, the resistance values can be retrieved by using the function

```
word GetOhmValues( double *dValue1; double *dValue2 );
```

Description: This routine retrieved the determined resistance values from the instrument. This routine also performs averaging on the values, only after 5 measurements the value is valid.
The calling software is responsible for performing enough measurements

Input: -

Output:
***dValue1** resistance value for Channel 1
***dValue2** resistance value for Channel 2
returnvalue
E_NO_ERRORS
E_NOT_INITIALIZED
E_NOT_SUPPORTED
E_NO_HARDWARE

Obsolete measurement routines

The following described routines are considered obsolete. They were initially put in the DLL to perform measurements and collect the measured data. With the current instruments and computers, these routines will not give the required performance.

Continuing using these functions is deprecated.

Start a measurement

word StartMeasurement (void);

Description: This routine tells the hardware to perform a single measurement. The measurement is initiated, and then the routine will wait until the hardware is ready. When the hardware is ready, the measured data is transferred from the hardware acquisition memory into the computer memory, inside the DLL.

Continuing using this function is deprecated.

Input: -

Output: Returnvalue E_NO_ERRORS
E_NOT_INITIALIZED
E_NO_HARDWARE

Remark: Perform a measurement. One (software) measurement equals a record_length number of hardware-measurements. So the hardware will fill it's internal buffer. This routine will wait until the hardware is done.

Get all measurement data in Volts

word GetMeasurement (double *dCh1, double *dCh2);

Description: This routine transfers the measured data from the acquisition memory in the DLL into the memory in the application. For each sample, the value in Volts is calculated.
dCh1 and dCh2 are both array. The caller must ensure that there is enough space in the arrays to contain the data. Therefor both the arrays must be at least

```
     RecordLength * sizeof(double)
```

Continuing using this function is deprecated.

Input:	-	
Output:	dCh1	The array to which the data of channel 1 should be passed.
	dCh2	The array to which the data of channel 2 should be passed.
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Get one sample of the measurement data, in Volts

word GetOneMeasurement (dword wIndex, double *dCh1, double *dCh2);

Description: This routine transfers a single sample per channel from the acquisition memory in the DLL into the memory of the application. The value in Volts is calculated for each sample.

Continuing using this function is deprecated.

Input:	wIndex	The index of the measured data point.
Output:	dCh1	Return address for the measured data from channel 1.
	dCh2	Return address for the measured data from channel 2.
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Get all measurement data, binary

word GetMeasurementRaw (word *wCh1, word *wCh2);

Description: This routine transfers the measured data from the acquisition memory in the DLL into the memory in the application. The measured data is returned in binary values. A value of 0 corresponds to -Sensitivity, 32768 corresponds to 0 and 65535 to +Sensitivity in Volts. wCh1 and wCh2 are arrays. The caller must ensure that there is enough space in the arrays to contain the data. Therefor the arrays must be at least

```
RecordLength * sizeof(word)
```

Continuing using this function is deprecated.

Input: -

Output:

wCh1	The array to which the measured data of channel 1 should be passed.
wCh2	The array to which the measured data of channel 2 should be passed.
Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Get one sample of the measurement data, binary

word GetOneMeasurementRaw(dword wIndex, word *wCh1, word *wCh2);

Description: This routine transfers a single sample per channel from the acquisition memory in the DLL to the memory of the application. The measured data is returned in binary values. A value of 0 corresponds to -Sensitivity, 32768 corresponds to 0 and 65535 to +Sensitivity in Volts.

Continuing using this function is deprecated.

Input:	wIndex	The index of the measured data point
Output:	wCh1	Return address for the measured data from channel 1
	wCh2	Return address for the measured data from channel 2
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Get the current sampling frequency (deprecated)

dword GetSampleFrequency(void);

Description: This routine returns the current set sampling frequency in Hz. The minimum/maximum frequency supported is instrument dependent.

Continuing using this function is deprecated.

Input:	-	
Output:	Returnvalue	The current sampling frequency in Hz.

Set the sampling frequency (deprecated)

word SetSampleFrequencyF(dword *dwFreq);

Remarks: The routine sets the sampling frequency. The hardware is not capable of creating every selected frequency so the hardware chooses the nearest allowed frequency to use, This is the frequency that is returned in dFreq.

Continuing using this function is deprecated.

Input:	dwFreq	The requested sampling frequency in Hz
Output:	dwFreq	The actual selected sampling frequency in Hz
	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Note The above two functions are replaced by the new functions GetSampleFrequencyF() and SetSampleFrequencyF() which give better support of the hardware.
These functions are now deprecated and should no longer be used.

Get the current trigger timeout value (deprecated)

dword GetTriggerTimeOut(void);

Description: This routine is used to query the current Timeout value. When this Timeout period has elapsed and the hardware has not seen a trigger, then a trigger is forced so that the hardware can start to measure. This way it is possible to measure a signal that has not met the trigger conditions.

Continuing using this function is deprecated.

Input:	-	
Output:	Returnvalue	The current Timeout value in msec.

Set the trigger timeout value (deprecated)

word SetTriggerTimeOut(dword lTimeout);

Description: This routine sets the Timeout value, see also GetTimeOut.

Continuing using this function is deprecated.

Input:	lTimeout	The new timeout value in msec.
Output:	Returnvalue	E_NO_ERRORS E_NO_HARDWARE

Note The Trigger Timeout applies **only** to measurements that are started with the **obsolete** routine StartMeasurement().
Measurements that are started using ADC_Start do **not** react to the trigger timout, the user will have to implement that self, by using ADC_ForceTrig

If you have any suggestions and/or remarks concerning the DLL's or the manual, please contact:

TiePie engineering
PO Box 290
8600 AG SNEEK

Visitors address:

TiePie engineering
Koperslagersstraat 37
8601 WL SNEEK
Tel.: +31 515 415 416
Fax: +31 515 418 819
E_mail: support@tiepie.nl

TiePie engineering DLL programmer's manual rev 1.20