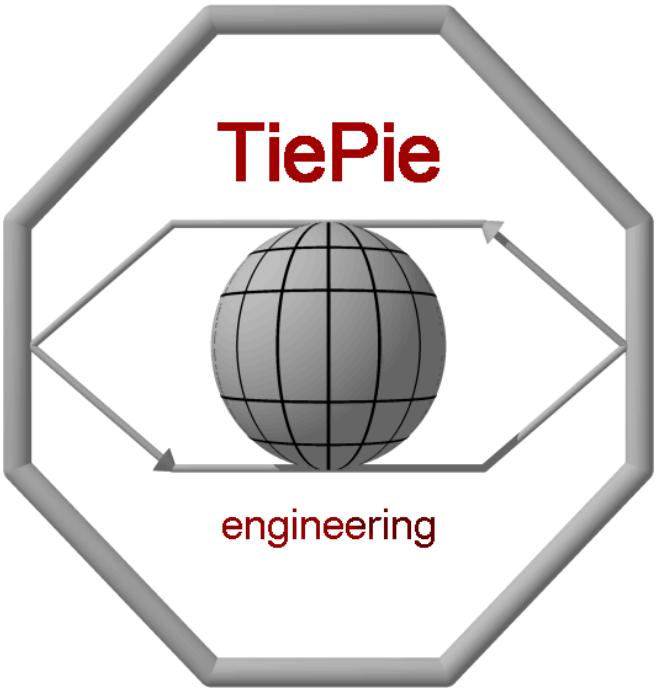
Programmer's Manual

TiePie DLLs



for:

TP112	TiePieSCOPE HS508
TP208	TiePieSCOPE HS801 AWG
TP508	Handyprobe HP2
TP801 AWG ISA	Handyscope HS2
TP801 AWG PCI	Handyscope HS3
TE6100	Handyscope HS4 (DIFF)

Revision 1.29

Table of contents

This manual describes the available functions in the DLLs for the various TiePie engineering measuring instruments.

For each instrument, a specific DLL is available. All DLLs have the same routines and the same programming interface.

Since all instruments have different specifications, a number of functions are available to determine the specifications of the instrument, like e.g. maximum sampling frequency, maximum record length, number of channels etc.

Not all instruments have the same functionality as other instruments, like e.g. the availability of a function generator or digital inputs and outputs. When a certain function is called and the instrument does not support that functionality, the routine will return an error code indicating that the functionality is not supported.

Since the initial development of the DLLs, many routines have been added to the DLL, to improve the performance of performing measurements using the DLL. Several of those routines are replacing older routines, but are not entirely compatible. To avoid that existing software would no longer function, the old routines are still available in the DLL, but are marked in the manual as being obsolete. It is advised to stop using these routines and use the new routines instead.

Understand the codes

Error codes

Most routines in the DLL return a status value, that indicates whether the routine was executed successfully or not. In case of a non successfull execution, the return value will indicate the possible cause of the error. The following codes are used:

Code Names	Code Values	
	Hexadecimal	Binary

```
E_NO_ERRORS              = 0x0000;  /*0000000000000000*/
E_NO_HARDWARE            = 0x0001;  /*0000000000000001*/
E_NOT_INITIALIZED        = 0x0002;  /*0000000000000010*/
E_NOT_SUPPORTED          = 0x0004;  /*0000000000000100*/
E_NO_GENERATOR           = 0x0008;  /*0000000000001000*/
E_INVALID_CHANNEL        = 0x0010;  /*0000000000010000*/
E_INVALID_VALUE          = 0x0020;  /*0000000000100000*/
E_I2C_ERROR              = 0x0040;  /*0000000001000000*/
E_I2C_INVALID_ADDRESS    = 0x0080;  /*0000000010000000*/
E_I2C_INVALID_SIZE       = 0x0100;  /*0000000100000000*/
E_I2C_NO_ACKNOWLEDGE     = 0x0200;  /*0000001000000000*/
```

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:

lCh1 = 1
lCh2 = 2
lCh3 = 3
lCh4 = 4

The routines that deal with the MeasureMode use different values:

mmCh1 = 1
mmCh2 = 2
mmCh3 = 4
mmCh4 = 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: **Return value** 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 automatically 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: **Return value** 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. Therefore, 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
	Return value	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 hard coded in the hardware. TP112, TP208 and TP508 do not have a serial number in the instrument.

Input:	-	
Output:	dwSerialNumber	the serial number
	Return value	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 a 20 elements large array. The caller must ensure that there is enough space in the array to contain the data. Therefore the size of the array in bytes 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
Return value	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 a 20 elements large array. The caller must ensure that there is enough space in the array to contain the data. Therefore the size of the array in bytes 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
Return value	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
	Return value	E_NO_ERRORS E_NO_HARDWARE

Get the maximum sampling frequency

double GetMaxSampleFrequencyF(void)

Description:	The different instruments have different maximum sampling frequencies. This routine queries the maximum sampling frequency.	
Input:	-	
Output:	**Return value**	The maximum sampling frequency the instrument supports, in Hz.

Note The above function replaces the existing, old and deprecated function GetMaxSampleFrequency.

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:	**Return value**	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: Return value 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: Return value 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: Return value 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.
Return value	E_NO_ERRORS E_NO_GENERATOR E_NO_HARDWARE

Start a measurement

word ADC_Start(void)

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: Return value E_NOT_INITIALIZED
E_NO_ERRORS
E_NO_HARDWARE

Check if the hardware is measuring

word ADC_Running(void)

Description: This routine checks if the hardware is currently measuring

Input: -

Output: Return value 0 = not measuring
1 = measuring

Abort a running measurement

word ADC_Abort(void)

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, ADC_Start does this already.

Input: -

Output: Return value E_NOT_INITIALIZED
E_NO_ERRORS
E_NO_HARDWARE

Read the trigger status

word ADC_Triggered(void)

Description: This routine reads the trigger status from the hardware. The returned value indicates which trigger source caused the trigger, this value is different for various instruments.

Input: -

Output:

Return value	HS4 / HS4 DIFF	other instruments
0	not triggered	not triggered
1	Ch1	Ch1
2	Ch2	Ch2
4	Ch3	External
8	Ch4	-
16	External	-

Remark: Return value can be a combination of indicated values.

Read the measurement status

word ADC_Ready(void)

Description: This routine checks if the measurement is ready or not.

Input: -

Output: Return value 0 = not ready
1 = ready

Force a trigger

word ADC_ForceTrig(void)

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: Return value E_NOT_INITIALIZED
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. Therefore the size of the array in bytes 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.
	Return value	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. Therefore the size of the array in bytes 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.
	Return value	E_NO_ERRORS E_NO_HARDWARE

Get all digital input values

word GetDigitalInputValues(word *wValues)

Description: 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. Therefore the size of the array in bytes must be at least

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

Input: -

Output: Return value 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.
Return value E_NO_ERRORS
E_NOT_SUPPORTED
E_NO_HARDWARE

Example of use of the routines

To use the measurement routines, your application could contain a loop like the following (for a two channel instrument):

```
type TDoubleArray = array[0 .. 128 * 1024 - 1] of double;

var wCh           : word;
    wChCount      : word;
    dSampleFreq   : double;
    ChSensArray   : array[lCh1 .. lCh2] of double;
    ChDoubleArray : array[lCh1 .. lCh2] of TDoubleArray;


if InitInstrument( 0 ) = E_NO_ERRORS then
begin
  GetNrChannels( wChCount );
 {*
  * Setup Ch1, 8 Volt full scale range, AC coupling
  *}
  ChSensArray[lCh1] := 8.0;
  SetSensitivity( lCh1, ChSensArray[lCh1] );
  SetCoupling( lCh1, lctAC );
 {*
  * Setup Ch2, 20 Volt full scale range, DC coupling
  *}
  ChSensArray[lCh1] := 20.0;
  SetSensitivity( lCh2, ChSensArray[lCh2] );
  SetCoupling( lCh2, lctDC );
 {*
  * Setup the trigger, source Ch1, rising slope, level 0 Volt
  *}
  SetTriggerSource( ltsCh1 );
  SetTriggerMode( ltmRising );
  SetTriggerLevel( lCh1, 0 );
 {*
  * Setup the time base:
  *   5000 samples record length,
  *   50% pre trigger (=2500 post samples, 2500 pre samples )
  *   10 MHz sampling frequency
  *}
  dSampleFreq := 10e6;
  SetRecordLength( 5000 );
  SetPostSamples( 2500 );
  SetSampleFrequencyF( dSampleFreq );
 {*
  * select the channel(s) to measure
  *}
  SetMeasureMode( mmCh1 + mmCh2 );
 {*
  * start performing measurements
  *
  * see next page
  *}
```

```
  ADC_Start;
  StartTime := GetCurrentTime;
  while bContinue do
  begin
    if GetCurrentTime > ( StartTime + TimeOut ) then
    begin
      ADC_ForceTrig;
    end; { if }
    if ADC_Ready = 1 then
    begin
      for wCh := lCh1 to wChCount do
      begin
        ADC_GetDataVoltCh( wCh, ChDoubleArray[wCh] );
      end; { for }
      ADC_Start;
      StartTime := GetCurrentTime;
      ApplicationProcessData;
    end; { if }
    Application.ProcessMessages;
  end; { while }
end; { if }
```

Legend:

bold	= reserved words
123	= number
italic	= comment
green	= pseudo code

Setup for streaming measurements

It is possible to do streaming measurements with the Handyscope HS3 and Handyscope HS4 (DIFF). 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, following 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:
		ItmBlock (0) default value. During the measurement, all data is stored in the instrument. When the measurement is ready, all data is transferred in one block to the computer. This is normal oscilloscope mode
		ItmStream (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:	Return value	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.
	Return value	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.

Control the input resolution

The Handyscope HS3 and Handyscope HS4 (DIFF) 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:	Return value	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 current 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
	Return value	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 SetInstrumentConfig(word wMode)

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

Get the current instrument configuration

word GetInstrumentConfig(word *wMode)

Description:	This routine returns the current Instrument configuration.	
Input:	-	
Output:	**wMode**	The current configuration
	Return value	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, e.g.:

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.
Return value	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:

Return value	E_NO_ERRORS E_NOT_SUPPORTED 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 occurred) is calculated like this:
`PreSamples = RecordLength - PostSamples.`

Input: -

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

Remark: Setting a record length smaller than the number of post samples gives an E_INVALID_VALUE error. See also the routines Get/SetPostSamples.

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 the routine GetMaxRecordLength(). The minimum value equals the current number of post samples. When an invalid value is passed on to the routine, this value is ignored and no changes in the instrument setting are made.

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

Output: Return value 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. See also the routines Get/SetPostSamples.

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 occurred).

Input: -

Output: **Return value** The current selected number of post samples to measure.

Remark: Setting a number of post samples larger than the record length gives an E_INVALID_VALUE error. See also the routines Get/SetRecordLength.

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. When an invalid value is passed on to the routine, this value is ignored and no changes in the instrument setting are made.

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

Output: **Return value** 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. See also the routines Get/SetRecordLength.

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: **Return value** 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

Return value E_NO_ERRORS
E_NO_HARDWARE

Note The above two functions are replacing the existing, old and deprecated 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 and faster devices support external clock input	
Input:	-	
Output:	wMode	The status of the internal clock, 0 = clock internal 1 = clock external
	Return value	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 and faster devices support external clock input	
Input:	wMode	0 = internal clock 1 = external clock
Output:	Return value	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:

lCh1 = 1
lCh2 = 2
lCh3 = 3
lCh4 = 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.
	Return value	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
	Return value	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 auto ranging mode:
0 : Auto ranging is off
1 : Auto ranging is on.
If Auto ranging is switched on 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. Auto ranging has no effect on a current measurement.

Input:	byCh	The channel whose current Auto ranging mode is requested (1, 2, 3 ,4).
Output:	byMode	The Auto ranging mode.
	Return value	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 Auto ranging mode:
0 : turn Auto ranging off
1 : turn Auto ranging on.
See also GetAutoRanging.

Input:	byCh	The channel whose Auto ranging mode has to be set (1, 2, 3, 4).
	byMode	The new value for the Auto ranging mode.
Output:	Return value	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:
lctAC : coupling AC (0)
lctDC : 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.
- Return value — 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:
- Return value — 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.
	Return value	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 necessary. 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:	Return value	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE

Note Not all devices support DC Level. If DC Level is not supported, the error value E_NOT_SUPPORTED is returned.

Get the current trigger source

word GetTriggerSource(byte *bySource)

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

ltsCh1	(0) Channel 1
ltsCh2	(1) Channel 2
ltsCh3	(2) Channel 3
ltsCh4	(3) Channel 4
ltsExternal	(4) a digital external signal
ltsAnalogExt	(5) an analog external signal
ltsAnd	(6) Channel 1 **AND** Channel 2
ltsOr	(7) Channel 1 **OR** Channel 2
ltsXor	(8) Channel 1 **XOR** Channel 2
ltsNoTrig	(9) no source, measure immediately
–	(10) not used
ltsPxiExt	(11) PXI bus digital trigger signals
lts GenStart	(12) start of the Handyscope HS3 generator
ltsGenStop	(13) stop of the Handyscope HS3 generator
ltsGenNew	(14) each new period of the HS3 generator

Input: -

Output:

bySource	The current trigger source.
Return value	E_NO_ERRORS, E_INVALID_VALUE E_NO_HARDWARE

Set the trigger source

word SetTriggerSource(byte bySource)

Description: This routine sets the trigger source of the acquisition system.

Input:

bySource	The new trigger source.

Output:

Return value	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.

ltmRising	(0)	trigger on rising slope
ltmFalling	(1)	trigger on falling slope
ltmInWindow	(2)	trigger when signal gets inside window
ltmOutWindow	(3)	trigger when signal gets outside window
ltmTVLine	(4)	trigger on TV line sync pulse
ltmTVFieldOdd	(5)	trigger on TV odd frame sync pulse
ltmTVFieldEven	(6)	trigger on TV even frame sync pulse

Input: -

Output:

byMode	The current trigger mode.
Return value	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:

Return value	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:	Return value	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:	Return value	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.
	Return value	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:	**Return value**	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.
	Return value	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:	Return value	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:	Return value	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 SetPXITriggerEnables
	Return value:	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:	Return value	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.
	Return value	E_NO_ERRORS E_NOT_SUPPORTED E_NO_HARDWARE

Set the digital outputs

word SetDigitalOutputs(byte byValue)

Description:	The TP112 is equipped 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:	Return value	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.
	Return value	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 square wave generator, the HS508 for example. This routine returns the generator frequency in Hz.

Input: -

Output: **Return value** 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.
Return value 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

Control the Arbitrary Waveform Generator

The Arbitrary Waveform Generator can operate in two different modes, DDS mode and Linear mode.

In DDS mode, the generator frequency refers to the frequency of the signal that is generated. In linear mode, the generator frequency refers to the internal sampling clock of the generator.

Set the generator mode

word SetFuncGenMode(dword dwMode)

Description:	The Handyscope HS3 function generator can be set to either linear mode or to DDS mode:	
	lfmDDS	(1) DDS mode
	lfmLinear	(2) Linear mode
Input:	dwMode	the requested function generator mode
Output:	Return value	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
	Return value	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE

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

lstSine	(0)	Sine wave
lstTriangle	(1)	Triangular wave
lstSquare	(2)	Square wave
lstDC	(3)	DC
lstNoise	(4)	Noise
lstArbitrary	(5)	Arbitrary signal

Output: **Return value:**
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. This does not apply to the Handyscope HS3 generator, which has two independent waveform buffers.

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

Return value
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.	
Input:	dAmplitude	the function generator amplitude in Volts: 0 <= value <= MaxAmplitude
Output:	Return value	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
	Return value	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.	
Input:	dDCOffset	the requested offset in Volts: -MaxAmpl <= value <= +MaxAmpl
Output:	Return value	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
	Return value	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:	Return value	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
	Return value	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Set the generator frequency

word SetFuncGenFrequency(double *dFrequency)

Description: In DDS mode, this routine sets the output signal frequency of the generator. In linear mode it sets the sample frequency of the generator.

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. The AWG of the TiePieSCOPE HS801, the TP801 ISA and the TP801 PCI support setting the sampling frequency in 15 steps: 38.1, 610, 2441, 9765, 39062, 78125, 156250, 312500, 625000, 1250000, 2500000, 5000000, 10000000, 25000000, 50000000 The Handyscope HS3 AWG supports setting the sampling frequency at the same frequencies as the sampling frequency of the acquisition system of the instrument.

Output:

dFrequency	the hardware can not support any arbitrary frequency within the available range. The value that was actually selected is returned.
Return value	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
Return value	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 be set to be started by an external TTL trigger signal on pin 21 of the extension connector, see also the instrument manual, chapter 4.
This routine sets the function generator trigger source:

ltsExtTrig	(4) a digital external signal
ltsNoTrig	(9) no source, generate immediately

The default value is ltsNoTrig

Input:	bySource	the requested trigger source
Output:	Return value	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
	Return value	E_NO_ERRORS E_INVALID_VALUE E_NOT_SUPPORTED E_NO_HARDWARE

Fill the function generator waveform memory

word FillFuncGenMemory(dword dwNrPoints, word *wFuncGenData)

description: This routine fills the function generator waveform memory with user defined data. The data must be in unsigned 16 bits values. A value of 0 corresponds to the negative full output scale, 32768 to 0 Volt and 65535 to the positive 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:

dwNrPoints	the number of waveform points that must be loaded, see remarks.
wFuncGenData	an array of unsigned 16 bits values, containing the signal that must be loaded. Must contain at least dwNrPoints samples.

Output:

Return value	E_NO_ERRORS E_NO_GENERATOR E_INVALID_VALUE E_NO_HARDWARE

Remarks: The number of samples (dwNrPoints) that can be uploaded to the generator is different per instrument. The Handyscope HS3 accepts any power of 2 up to 2^17 = 262144. Older generators' buffer sizes are 1024 samples in DDS mode and 65536 or 131072 samples in linear mode. These instruments automatically change the generator mode depending on dwNrPoints. See SetFuncGenMode for information about DDS and linear mode. When a number of samples is uploaded to the instrument that is smaller than the preferred value for that instrument, the buffer will be enlarged to the appropriate value and the additional samples will be filled with "zero Volt".

When generating a predefined signal, like e.g. a sine wave, 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. This does not apply to the Handyscope HS3 generator, which has two independent waveform buffers.

Set the generator output state

word SetFuncGenOutputOn(word wValue)

Description: For the TiePieSCOPE HS801 and the TP801 PCI/ISA, this routine switches the output of the function generator on or off.
For the Handyscope HS3, this routine switches on the internal logic of the function generator, but does not start the generation of the signal. Refer to SetFuncGenEnable() of FuncGenBurst() for starting/stopping the generator.

Input: wValue — The new output state
- 0 output is off.
  The output of a Handyscope HS3 is floating at an undefined voltage
- 1 output is on
  The output of a Handyscope HS3 is equal to the DC offset that is set

Output: Return value
- 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

Return value
- E_NO_ERRORS
- E_NO_GENERATOR
- E_INVALID_VALUE
- E_NO_HARDWARE

Set the generator enabled state

word SetFuncGenEnable(word wValue)

Description: This routine enables the Handyscope HS3 function generator. Prior to calling this function, the generator must have been switched on using SetFuncGenOutputOn().

Input: **wValue** The new enabled state
- **0** Stop signal generation, bringing the generator in idle mode
- **1** Start continuous signal generation

Output: **Return value** E_NO_ERRORS
E_NO_GENERATOR
E_INVALID_VALUE
E_NO_HARDWARE

Get the current generator enabled state

word GetFuncGenEnable(word *wValue)

Description: This routine determines the current setting of the function generator enabled setting

Input: -

Output: **wValue** The current setting of the enabled state
- **0** generator is in idle state, no signal generation
- **1** generator is in continuous signal generation state

Return value 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 will switch off burst mode and bring the generator back in idle mode.
Output:	Return value	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, using SetFuncGenOutpuOn().

Note The generator has to be placed in idle mode before burst mode is selected, using SetFuncGenEnabled().

Use the I²C bus

Some instruments have an I²C connection on the extension connector. Refer to the hardware manual for the exact pin numbers on the extension connector of the instrument.

Support of I²C requires instrument drivers of version 6.0.5.0 or higher. If your driver version is lower, please refer to www.tiepie.nl for the latest version of the drivers.

To control devices on this bus, the following routines are available.

Get the I²C bus speed

```
word I2CGetSpeed( dword *dwSpeed )
```

Description:	The I²C bus can operate on two frequencies, 100 kHz and 400 kHz. This routine will read the current bus speed.	
Input:	-	
Output:	**dwSpeed**	The bus frequency in Hz
	return value	E_NO_ERRORS E_NO_HARDWARE E_NOT_SUPPORTED

Set the I²C bus speed

```
word I2CSetSpeed( dword *dwSpeed )
```

Description:	The I²C bus can operate on two frequencies, 100 kHz and 400 kHz. This routine will set the bus speed to the closest valid bus speed.	
Input:	**dwSpeed**	The requested bus frequency in Hz
Output:	**dwSpeed**	The bus frequency that was actually set, in Hz
	return value	E_NO_ERRORS E_NO_HARDWARE E_NOT_SUPPORTED

Write data to the I²C bus

Two routines are available to write data to the I²C bus.

word I2CWrite(dword dwAddress, void * pBuf, dword dwSize)

Description:	This routine writes the data that is placed in the memory where pBuf points to, to a specified address on the I²C bus. When the data is sent, a stop command is sent to the I²C bus.	
Input:	dwAddress	the address of the device the data is written to
	*pBuf	pointer to the begin of memory location that contains the data to be written.
	dwSize	the size of the buffer in bytes
Output:	return value	E_NO_ERRORS E_NO_HARDWARE E_NOT_SUPPORTED E_I2C_ERROR E_I2C_INVALID_ADDRESS E_I2C_INVALID_SIZE E_I2C_NO_ACKNOWLEDGE

word I2CWriteNoStop(dword dwAddress, void * pBuf, dword dwSize)

Description:	This routine writes the data that is placed in the memory where pBuf points to, to a specified address on the I²C bus. When the data is sent, no stop command is sent to the I²C bus.	
Input:	dwAddress	the address of the device the data is written to
	*pBuf	pointer to the begin of memory location that contains the data to be written.
	dwSize	the size of the buffer in bytes
Output:	return value	E_NO_ERRORS E_NO_HARDWARE E_NOT_SUPPORTED E_I2C_ERROR E_I2C_INVALID_ADDRESS E_I2C_INVALID_SIZE E_I2C_NO_ACKNOWLEDGE

Read data from the I²C bus

Two routines are available to read data from the I²C bus.

word I2CRead(dword dwAddress, void * pBuf, dword dwSize)

Description:	This routine reads the data from a specified address on the I²C bus and places it in the memory where pBuf points to. When the data is read, a stop command is sent to the I²C bus.	
Input:	dwAddress	the address of the device the data is read from
	*pBuf	pointer to the begin of memory location where the read data will be placed.
	dwSize	the size of the buffer in bytes
Output:	return value	E_NO_ERRORS E_NO_HARDWARE E_NOT_SUPPORTED E_I2C_ERROR E_I2C_INVALID_ADDRESS E_I2C_INVALID_SIZE E_I2C_NO_ACKNOWLEDGE

word I2CReadNoStop(dword dwAddress, void * pBuf, dword dwSize)

Description:	This routine reads the data from a specified address on the I²C bus and places it in the memory where pBuf points to. When the data is sent, no stop command is sent to the I²C bus.	
Input:	dwAddress	the address of the device the data is read from
	*pBuf	pointer to the begin of memory location where the read data will be placed.
	dwSize	the size of the buffer in bytes
Output:	return value	E_NO_ERRORS E_NO_HARDWARE E_NOT_SUPPORTED E_I2C_ERROR E_I2C_INVALID_ADDRESS E_I2C_INVALID_SIZE E_I2C_NO_ACKNOWLEDGE

Perform 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, auto ranging, 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:	Return value	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
	Return value	E_NO_ERRORS
		E_NOT_INITIALIZED
		E_NOT_SUPPORTED
		E_NO_HARDWARE

Deprecated 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.

Get the maximum sampling frequency

```
dword GetMaxSampleFrequency( void )
```

Continuing using this routine is deprecated, use the routine

```
GetMaxSampleFrequencyF( )
```

instead.

Start a measurement

```
word StartMeasurement( void )
```

Continuing using this routine is deprecated, use the routines

```
ADC_Start( )
ADC_Ready( )
```

instead.

Get all measurement data in Volts

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

Continuing using this routine is deprecated, use the routine

```
ADC_GetDataVoltCh( )
```

instead.

Get one sample of the measurement data, in Volts

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

Continuing using this routine is deprecated.

Get all measurement data, binary

word GetMeasurementRaw(word *wCh1, word *wCh2)

Continuing using this routine is deprecated, use the routine

ADC_GetDataCh( )

instead.

Get one sample of the measurement data, binary

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

Continuing using this routine is deprecated.

Retrieve the measured data in binary format

word ADC_GetData(word *wCh1, word *wCh2)

Continuing using this routine is deprecated, use the routine

ADC_GetDataCh( )

instead.

Retrieve the measured data in Volts

word ADC_GetDataVolt(double *dCh1, double *Ch2)

Continuing using this routine is deprecated, use the routine

ADC_GetDataVoltCh( )

instead.

Get the current sampling frequency

dword GetSampleFrequency(void)

Continuing using this routine is deprecated, use the routine

GetSampleFrequencyF( )

instead.

Set the sampling frequency

word SetSampleFrequency(dword *dwFreq)

Continuing using this routine is deprecated, use the routine

SetSampleFrequencyF( )

instead.

Get the current trigger timeout value

dword GetTriggerTimeOut(void)

Continuing using this routine is deprecated.

Set the trigger timeout value

word SetTriggerTimeOut(dword lTimeout)

Continuing using this routine is deprecated.

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 timeout, the user will have to implement that self, by using ADC_ForceTrig

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

TiePie engineering
Koperslagersstraat 37
8601 WL SNEEK
The Netherlands

Tel.: +31 (0)515 415 416
Fax: +31 (0)515 418 819
E_mail: support@tiepie.nl
Website: www.tiepie.nl