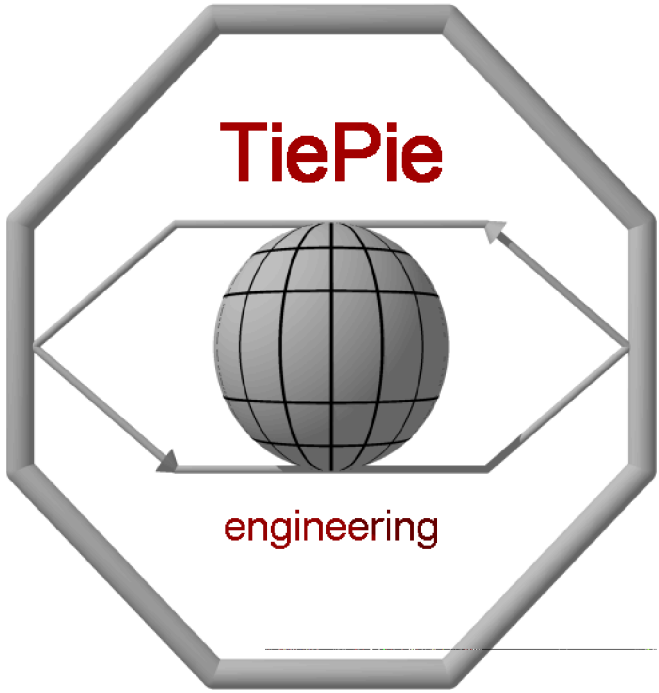
Programmer's Manual

TiePie Dll's



for: TP112
TP208
TP508
TP801 AWG ISA
TP801 AWG PCI
TE6100
TiePieSCOPE HS508
TiePieSCOPE HS801 AWG
Handyprobe HP2
Handyscope 2
Handyscope 3 AWG

Revision 1.10

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 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                                      {and allocate resources}
  while MeasureMore do
  begin
    if ChangeSettings then
    begin
      OldFreq := GetFrequency;                        {query a setting}
      SetFrequency((OldFreq* 10)-1000 );             {change a setting}
    end; { if }
    if StartMeasurement = E_NO_ERRORS   then                {measure}
    begin
      for Sample := 0 to GetRecordLengh  do
      begin                                              {retrieve data}
        GetOneMeasurement(Data1[Sample], Data2[Sample]);
      end; { for }
    end; { if }
  end; { while }
  ExitInstrument;                                    { free resources}
end
else
begin
  writeln( 'Error: No hardware found...' );
end; { else }
```

Legend:

bold = reserved words
123 = number
italic = comment

Understand the error codes

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*/

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.

Input: -

Output: Returnvalue E_NO_ERRORS;
E_NOT_INITIALIZED

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

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

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	Return address for the array of input sensitivities
Returnvalue	E_NO_ERRORS

Determine the available input resolutions

word GetAvailableResolutions(double *dResolutions);

description: The Handyscope 3 supports 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	Return address for the array of input sensitivities
Returnvalue	E_NO_ERRORS

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

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

Check for a function generator

word GetFunctionGenStatus(void);

Description: The TiePieSCOPE HS801, TP801 and Handyscope 3 can have a built-in arbitrary waveform generator. When this function returns E_NO_GENERATOR, the HS801, TP801 or Handyscope 3 is equipped with a simple square wave generator.
Input: -
Output: Returnvalue E_NO_ERRORS
E_NO_GENERATOR

Get the maximum amplitude of the function generator

word GetFuncGenMaxAmplitude(double *dAmplitude);

Description: The maximum output voltage for the TiePieSCOPE HS801 and Handyscope 3 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

Start the 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.

Input: -

Output: Returnvalue E_NO_ERRORS
E_NOT_INITIALIZED

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)
```

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

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.

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

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, 32767 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

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, 32767 corresponds to 0 and 65535 to +Sensitivity in Volts.

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

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

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

The previously mentioned routine StartMeasurement takes care of a complete measurement. It sets up the hardware to perform a measurement and then starts the hardware measurement. Then it will wait for the measurement to be ready. In the mean while, it checks the triggered flag of the hardware and checks if a trigger time out has occured. If that has happened, it will force a trigger. Then it will wait for the measurement to be ready. When the measurement is ready, the measured data will be transferred from the hardware to memory inside the DLL. While doing that, it will check if auto range is required. When all data is transferred and checked, the routine will end.
The application can then transfer the data from the DLL memory to it's own memory and process it.

For certain applications it might be usefull to split up this process into individual steps. The following routines enable this. When these routines are used, no Trigger timout is available.

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

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

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

Retrieve the measured data

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, 32767 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

Example of use of the routines

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

```
.
.
ADC_Start;
while bContinue  do
begin
  if ADC_Ready =  1 then
  begin
  ADC_Getdata( @Ch1WordArray, @Ch2WordArray );
  ADC_Start;
  ApplicationProcessData;
end; { if }
.
.
```

Controlling the input resolution

The Handyscope 3 supports 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
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

Get the current measure mode

word GetMeasureMode(byte *byMode);

Description: This routine returns the current Measure Mode:

1 :	Ch1	the signal at channel 1 is measured
2 :	Ch2	the signal at channel 2 is measured
3 :	Ch1 & Ch2	the signals at channel 1 and 2 are measured simultaneously

Input: -

Output:

byMode	The return address for the Measure Mode.
Returnvalue	E_NO_ERRORS
	E_INVALID_VALUE

Set the measure mode

word SetMeasureMode(byte byMode);

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

Input:

byMode	The new measure mode (1, 2 or 3).

Output:

Returnvalue	E_NO_ERRORS
	E_INVALID_VALUE

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 the-routine 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

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

Get the current sampling frequency

dword GetSampleFrequency(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 SetSampleFrequency(dword *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

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

Set the sample clock status

word SetExternalClock(word wMode);

Description:	This routine sets the sampling clock mode: is te 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

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)
Output:	dSens	The return address for the sensitivity.
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL

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)
	dSens	The new Sensitivity in Volts
Output:	dSens	Contains the actual set Sensitivity, on return
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL

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).
Output:	byMode	Return address for the Autoranging mode.
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL

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).
	byMode	The new value for the Autoranging mode.
Output:	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE

Get the current input coupling

word GetCoupling(byte byCh, byte *byMode);

Description: This routine returns the current signal coupling for the selected channel:
0 : coupling AC
1 : coupling DC.
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)

Output: byMode Return address for the current coupling.
Returnvalue E_NO_ERRORS
E_INVALID_CHANNEL
E_INVALID_VALUE

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).
byMode The new coupling for the selected channel (0 or 1).

Output: Returnvalue E_NO_ERRORS
E_INVALID_CHANNEL
E_INVALID_VALUE

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)
Output:	dLevel	Return address for the current DC Level.
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_NOT_SUPPORTED

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)
	dLevel	The new DC Level in Volts
Output:	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE E_NOT_SUPPORTED

Get the current trigger source

word GetTriggerSource(byte *bySource);

Description: This routine is used to retrieve the current Trigger Source. The hardware waits for a trigger condition, or a Timeout, before it starts to measure.

0	:	Ch1Trig	Channel 1
1	:	Ch2Trig	Channel 2
2	:	ExtTrig	a digital external signal
3	:	AndTrig	Channel 1 AND Channel 2
4	:	OrTrig	Channel 1 OR Channel 2
5	:	NoTrig	no source, measure immediately
6	:	PxiExtTrig	PXI bus digital trigger signals
7	:	XorTrig	Channel 1 XOR Channel 2
8	:	AnaExtTrig	an analog external signal

Input: -

Output: bySource The current trigger source.
Returnvalue E_NO_ERRORS
E_INVALID_VALUE

Set the trigger source

word SetTriggerSource(byte bySource);

Description: This routine sets the trigger source, see GetTriggerSource for the codes of the different sources.

Input: bySource The new trigger source.

Output: Returnvalue E_NO_ERRORS
E_INVALID_VALUE
E_NOT_SUPPORTED

Note Not all instruments support all Trigger Sources. Look at the manual of your instrument for details, which Trigger Sources your instrument supports. 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.

0	Rising	trigger on rising slope
1	Falling	trigger on falling slope
2	InWindow	trigger when signal gets inside window
3	OutWindow	trigger when signal gets outside window
4	TVLine	trigger on TV line sync pulse
5	TVFieldOdd	trigger on TV odd frame sync pulse
6	TVFieldEven	trigger on TV even frame sync pulse

Input: -

Output:

byMode	The current trigger mode.
Returnvalue	E_NO_ERRORS
	E_INVALID_VALUE

Set the trigger mode

word SetTriggerMode(byte byMode);

Description: This routine is used to set the Trigger Mode. See also GetTriggerSource. 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

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 or 2).
Output:	dLevel	Return address for the Trigger Level.
	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL

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 or 2).
	dLevel	The new Trigger Level in Volts.
Output:	Returnvalue	E_NO_ERRORS E_INVALID_CHANNEL E_INVALID_VALUE

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

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 or 2).
Output:	dHysteresis	Return address for the Trigger Hysteresis.
	Returnvalue	E_NO_ERROR E_INVALID_CHANNEL

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 or 2).
	dHysteresis	The new trigger hysteresis.
Output:	Returnvalue	E_NO_ERRORS E_INVALID_VALUE E_INVALID_CHANNEL

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.

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

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
	Returnvalue:	E_NO_ERRORS E_NOT_SUPPORTED

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

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

Get the current trigger timeout value

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

Set the trigger timeout value

dword SetTriggerTimeOut(dword lTimeout);

Description: This routine sets the Timeout value, see also GetTimeOut.
Input: lTimeout The new timeout value in msec.
Output: Returnvalue E_NO_ERRORS

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

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

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

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
- 0 Sine wave
- 1 Triangular wave
- 2 Square wave
- 3 DC
- 4 Noise
- 5 Arbitrary signal

Output: Returnvalue: E_NO_ERRORS
E_NO_GENERATOR
E_INVALID_VALUE

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

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

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

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

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

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 poitive 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

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

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 65536 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

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

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 32767 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:	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 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

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

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

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