



***MotionPro Timing Hub***  
**SDK Reference Manual**  
**(Software Development Kit)**

**Software Release**

2.03

**Document Revision**

July 2011

**Products Information**

<http://www.idtvision.com>

**North America**

1202 E Park Ave  
TALLAHASSE FL 32301  
United States of America  
P: (+1) (850) 222-5939  
F: (+1) (850) 222-4591  
[llourenco@idtvision.com](mailto:llourenco@idtvision.com)

**Europe**

via Pennella, 94  
I-38057 - Pergine Valsugana (TN)  
Italy  
P: (+39) 0461- 532112  
F: (+39) 0461- 532104  
[pgallorosso@idtvision.com](mailto:pgallorosso@idtvision.com)  
Eekhoornstraat, 22  
B-3920 - Lommel  
Belgium  
P: (+32) 11- 551065  
F: (+32) 11- 554766  
[amarinelli@idtvision.com](mailto:amarinelli@idtvision.com)

**Copyright © Integrated Design Tools, Inc.**

The information in this manual is for information purposes only and is subject to change without notice. Integrated Design Tools, Inc. makes no warranty of any kind with regards to the information contained in this manual, including but not limited to implied warranties of merchantability and fitness for a particular purpose. Integrated Design Tools, Inc. shall not be liable for errors contained herein nor for incidental or consequential damages from the furnishing of this information. No part of this manual may be copied, reproduced, recorded, transmitted or translated without the express written permission of Integrated Design Tools, Inc.

<b>1. OVERVIEW</b>	<b>7</b>
1.1. DIRECTORIES STRUCTURE	8
<b>2. USING MOTIONPRO TIMING HUB SDK</b>	<b>9</b>
2.1. OVERVIEW OF SQUARE WAVES AND PULSES	9
2.2. PROGRAMMING LANGUAGE	11
2.3. 64 BIT PROGRAMMING	12
2.4. LOAD/UNLOAD THE DRIVER	13
2.5. ENUMERATE/OPEN A DEVICE	13
2.6. CONFIGURING A DEVICE	14
2.7. OPENING AND CLOSING OUTPUT CHANNELS	15
2.8. MEASUREMENT	16
2.9. SIGNALS TIME-BASE	16
2.10. INTERNAL MODE	17
2.11. EXTERNAL MODE	17
2.12. START/STOP MODE	17
2.13. RATE SWITCH MODE	18
2.14. BURST MODE	18
<b>3. MOTIONPRO TIMING HUB SDK REFERENCE</b>	<b>19</b>
3.1. INITIALIZATION FUNCTIONS	19
3.1.1. Overview: Initialization functions	19
3.1.2. ThGetVersion	20
3.1.3. ThLoadDriver	21
3.1.4. ThUnloadDriver	22
3.1.5. ThEnumDevices	23
3.1.6. ThOpenDevice	24
3.1.7. ThCloseDevice	25
3.2. CONFIGURATION FUNCTIONS	26
3.2.1. Overview: Configuration functions	26
3.2.2. ThGetDeviceInfo	27
3.2.3. ThReadDefaultSettings	28
3.2.4. ThReadDeviceSettings	29
3.2.5. ThRefreshDeviceSettings	30
3.2.6. ThValidateDeviceSettings	31
3.2.7. ThSetParameter	32
3.2.8. ThGetParameter	33
3.2.9. ThGetParameterAttribute	34
3.3. OUTPUTS ENABLE/DISABLE FUNCTIONS	35
3.3.1. Overview: Outputs Enable/Disable Functions	35
3.3.2. ThOpenOutputs	36
3.3.3. ThCloseOutputs	37
3.4. MEASURE FUNCTIONS	38
3.4.1. Overview: Measure functions	38
3.4.2. ThSynchMeasure	39
3.4.3. ThStartMeasure	40
3.4.4. ThAbortMeasure	41
3.5. MISCELLANEOUS FUNCTIONS	42
3.5.1. Overview: Miscellaneous functions	42

3.5.2.	ThGetHardwareError .....	43
<b>4.</b>	<b>MOTIONPRO TIMING HUB ACTIVEX CONTROL REFERENCE .....</b>	<b>44</b>
4.1.	OVERVIEW .....	44
4.2.	TIMING CONTROL FUNCTIONS .....	45
4.2.1.	Overview: Timing Control functions .....	45
4.2.2.	Open .....	46
4.2.3.	Close .....	47
4.2.4.	GetInfo .....	48
4.2.5.	SetParameter .....	49
4.2.6.	RefreshDeviceSettings .....	50
4.2.7.	ReadDefaultSettings .....	51
4.2.8.	GetParameter .....	52
4.2.9.	GetParameterAttribute .....	53
4.2.10.	OpenOutputs .....	54
4.2.11.	CloseOutputs .....	55
4.2.12.	Measure .....	56
<b>5.</b>	<b>MOTIONPRO TIMING HUB LABVIEW™ INTERFACE REFERENCE .....</b>	<b>57</b>
5.1.	OVERVIEW .....	57
5.2.	INITIALIZATION VIS .....	58
5.2.1.	Overview: Initialization VIs .....	58
5.2.2.	IDT TH Enum Devices .....	59
5.2.3.	IDT TH Open Device .....	60
5.2.4.	IDT TH Close Device .....	61
5.3.	CONFIGURATION VIS .....	62
5.3.1.	Overview: Configuration VIs .....	62
5.3.2.	IDT TH Get Info .....	63
5.3.3.	IDT TH Get Parameter .....	64
5.3.4.	IDT TH Set Parameter .....	65
5.3.5.	IDT TH Send Config .....	66
5.4.	OUTPUTS ENABLE/DISABLE VIS .....	67
5.4.1.	Overview: Outputs enable/disable VIs .....	67
5.4.2.	IDT TH Open Outputs .....	68
5.4.3.	IDT TH Close Outputs .....	69
5.5.	MEASUREMENT VIS .....	70
5.5.1.	Overview: Measurement VIs .....	70
5.5.2.	IDT TH Synch Measure .....	71
5.5.3.	IDT TH Measure Start .....	72
5.5.4.	IDT TH Measure Abort .....	73
5.5.5.	IDT TH Measure Read .....	74
5.5.6.	IDT TH Measure is Ready .....	75
5.6.	MISCELLANEOUS VIS .....	76
5.6.1.	Overview: Miscellaneous VIs .....	76
5.6.2.	IDT TH Get Error .....	77
5.7.	HOW TO USE THE VIS .....	78
5.7.1.	Opening and closing a device .....	78
5.7.2.	Configuring a device .....	78
5.7.3.	Measurement .....	78
5.7.4.	Error handling .....	78
5.8.	EXAMPLES VIS .....	79
5.8.1.	1_enum_devices .....	79
5.8.2.	2_getinfo .....	79

5.8.3.	3_ch0_generator.....	79
5.8.4.	4_internal_mode.....	79
5.8.5.	5_external_mode.....	79
5.8.6.	6_startstop_mode.....	80
5.8.7.	7_rateswitch_mode.....	80
5.8.8.	8_burst_mode.....	80
5.8.9.	9_synch_measure.....	80
5.8.10.	10_asynch_measure.....	80
<b>6.</b>	<b>MOTIONPRO TIMING HUB MATLAB™ INTERFACE REFERENCE .....</b>	<b>81</b>
6.1.	OVERVIEW .....	81
6.2.	INITIALIZATION FUNCTIONS.....	82
6.2.1.	Overview: Initialization functions.....	82
6.2.2.	IdtThGetVersion.....	83
6.2.3.	IdtThEnumDevices.....	84
6.2.4.	IdtThOpenDevice.....	85
6.2.5.	IdtThCloseDevice.....	86
6.3.	CONFIGURATION FUNCTIONS.....	87
6.3.1.	Overview: Configuration functions.....	87
6.3.2.	IdtThGetDeviceInfo.....	88
6.3.3.	IdtThGetParameter.....	89
6.3.4.	IdtThSendCfg.....	90
6.4.	OUTPUTS ENABLE/DISABLE FUNCTIONS.....	91
6.4.1.	Overview: Outputs enable/disable Functions.....	91
6.4.2.	IdtThOpenOutputs.....	92
6.4.3.	IdtThCloseOutputs.....	93
6.5.	MEASUREMENT FUNCTIONS.....	94
6.5.1.	Overview: Measurement Functions.....	94
6.5.2.	IdtThSynchMeasure.....	95
6.5.3.	IdtThStartMeasure.....	96
6.5.4.	IdtThAbortMeasure.....	97
6.5.5.	IdtThReadMeasure.....	98
6.5.6.	IdtThMeasureIsReady.....	99
6.6.	HOW TO USE THE INTERFACE FUNCTIONS.....	100
6.6.1.	Opening and closing a device.....	100
6.6.2.	Configuring a device.....	100
6.6.3.	Measurement.....	100
6.6.4.	Error handling.....	100
6.7.	EXAMPLES.....	101
6.7.1.	IdtThEnumEx.....	101
6.7.2.	IdtThInfoEx.....	101
6.7.3.	IdtThReadParmEx.....	101
6.7.4.	IdtThSynchMeasEx.....	101
6.7.5.	IdtThAsynchMeasEx.....	101
6.7.6.	IdtThWaveGenEx.....	101
6.7.7.	IdtThWaveDelayEx.....	101
6.7.8.	IdtThInternalEx.....	101
6.7.9.	IdtThExternalEx.....	102
6.7.10.	IdtThStartStopEx.....	102
6.7.11.	IdtThRateSwitchEx.....	102
6.7.12.	IdtThBurstEx.....	102
6.7.13.	IdtThMeasureEx.....	102
<b>7.</b>	<b>APPENDIX.....</b>	<b>103</b>

7.1.	APPENDIX A - RETURN CODES .....	103
7.2.	APPENDIX B – INFORMATION PARAMETERS .....	104
7.3.	APPENDIX C – DEVICE SETTINGS .....	105
7.4.	APPENDIX D – LABVIEW / MATLAB ERROR CODES.....	106
7.5.	APPENDIX E – DATA TYPES .....	107
7.5.1.	TH_DEV_MODEL .....	107
7.5.2.	TH_REVISION .....	107
7.5.3.	TH_OP_MODE .....	107
7.5.4.	TH_OUTPUT_CHANNEL .....	107
7.5.5.	TH_INPUT_CHANNEL .....	108
7.5.6.	TH_OUTPUT_MASK .....	108
7.5.7.	TH_INPUT_INVERT .....	108
7.5.8.	TH_INPUT_INVERT .....	109
7.5.9.	TH_STATE.....	109
7.5.10.	TH_MEASURE_TYPE .....	109
7.5.11.	TH_CALLBACK_FLAGS.....	109
7.5.12.	TH_ATTRIBUTE.....	109
7.5.13.	TH_ERROR.....	109
7.5.14.	TH_INFO .....	109
7.5.15.	TH_PARAM.....	110
7.6.	APPENDIX F – STRUCTURES .....	111
7.6.1.	TH_ENUMITEM .....	111
7.6.2.	TH_SETTINGS .....	112
7.6.3.	TH_AsyncCallback.....	113

# 1. Overview

The on-line documentation of the MotionPro Timing Hub Software Development Kit and its components is divided into the following parts:

## **Using the MotionPro Timing Hub SDK**

This section describes how to start using the Timing Hub SDK.

## **MotionPro Timing Hub SDK Reference**

This section contains a detailed description of the Timing Hub SDK functions.

## **XStreamTHX™ ActiveX Control Reference**

This section contains a detailed description of the XStreamTHX™ ActiveX functions.

## **MotionPro Timing Hub LabVIEW™ Interface Reference**

This section contains a detailed description of the Timing Hub LabVIEW™ VIs.

## **MotionPro Timing Hub MATLAB™ Interface Reference**

This section contains a detailed description of the Timing Hub MATLAB™ functions.

## **Appendix**

This section provides additional information about data structures, parameters and functions return codes.

---

**Important note:** ActiveX, MATLAB and LabVIEW plug-ins are not supported in MAC OS X SDK version, but only in Windows version.

---

## 1.1. Directories structure

The default installation directory of the SDK is “C:\Program Files\IDT\XsTH”. Under this directory a set of sub-directories is created:

**BIN:** it contains the files (drivers, INF, DLLs) that may be re-distributed with the hub and your application.

**DOCS:** it contains the SDK documentation and the manuals.

**INCLUDE:** it contains the SDK header files (H and BAS).

**LABVIEW:** it contains the LabVIEW™ example Virtual Instruments.

**LIB:** it contains the SDK lib files.

**MATLAB:** it contains the MATLAB™ drivers and examples.

**SOURCE:** it contains the Visual C++ SDK examples.



## 2. Using MotionPro Timing Hub SDK

### 2.1. Overview of square waves and pulses

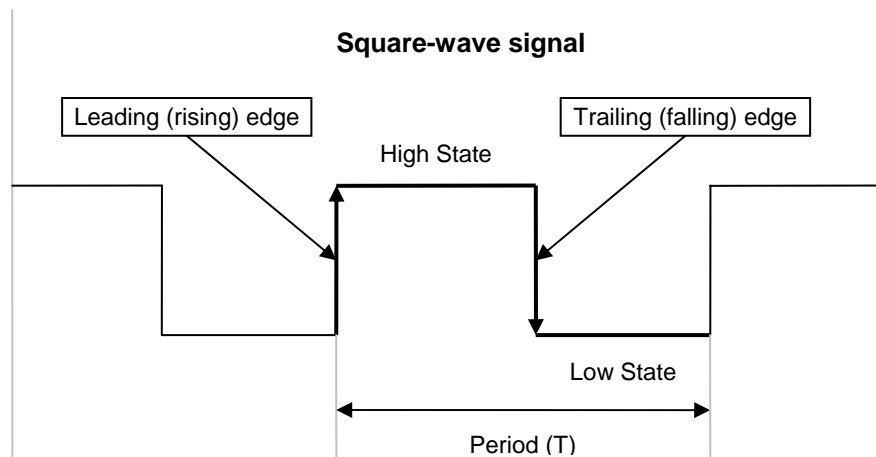
The Timing Hub generates square-waves and pulses.

A **SQUARE WAVE** is a periodic signal which changes instantaneously between two fixed levels. The values are usually 0 and 5 V (TTL) or 0 to 3.3 V (CMOS). The Timing Hub generates CMOS levels square waves, but it's also TTL compatible. The two fixed levels are called "**Low Level**" (or "Low State") and "**High level**" (or "High State"). The duration of the high state may be called "**High Time**", while the duration of the low state may be called "**Low Time**".

The "**Leading Edge**" (or "Rising Edge") of a signal is the part of the signal that changes from the Low State to the High State, while the "**Trailing Edge**" (or "Falling Edge") is the part of the signal that changes from the High State to the Low State.

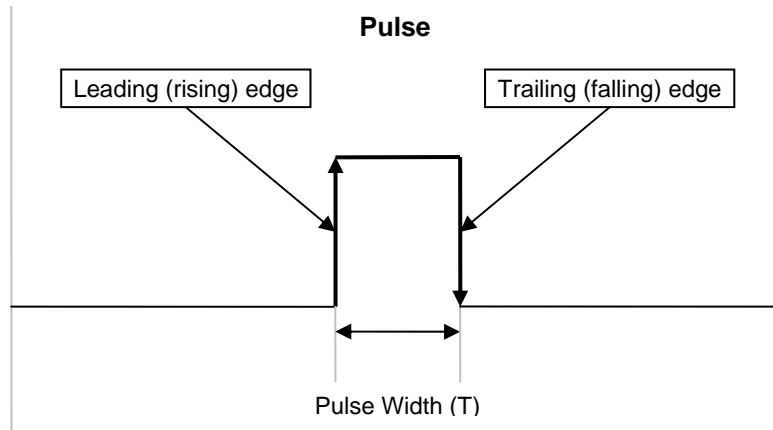
Other important parameters are the wave **Period** (T), e.g. the time between two consecutive rising edges, and the **Duty Cycle**, e.g. the ratio of the duration (time) that a signal is on high state to the total period of the signal.

The diagram below shows the main components of a square-wave signal.



A **PULSE** is a signal whose amplitude deviates from zero for a short period of time. A pulse may be periodic (pulse train) and not periodic (single pulse). The **Pulse Width** is the time between the rising edge and the falling edge of a pulse. If the pulse is periodic it has a period and a duty cycle.

The diagram below shows the main parts of a pulse.



## 2.2. Programming Language

A C/C++ header file is included in the SDK (TimHubAPI.h file in the Include sub-directory).

Most compiled languages can call functions; you will need to write your own header/import/unit equivalent based on the C header file.

In Windows version a Visual Basic module is included in the SDK (**TimHubAPI.bas** file in the Include sub-directory). VB cannot use **ThStartMeasure**, because these functions have callbacks which occur on a different thread. If you want to use VB, you might need to write some C code depending on your application's requirements. The same issue with asynchronous callbacks, above, also applies to Java.

The Windows driver is a DLL (XStreamDrv.dll) that resides in the system32 directory. It may be found also in the Bin sub-directory.

**MS Visual C++™**: A Visual C++ 6.0 stub COFF library is provided (**TimHubDrv.lib** in the Lib sub-directory); if you are using Visual C++, link to TimHubDrv.lib. The DLL uses Windows standard calling conventions (\_stdcall).

**Borland C++ Builder™**: the TimHubDrv.lib file is in COFF format. Borland C++ Builder requires the OMF format. To convert the library into to OMF format, use the IMPLIB Borland tool with the following syntax: "IMPLIB TimHubDrv.lib TimHubDrv.dll".

**Other compilers**: the Most other compilers can create a stub library for DLLs. The DLL uses Windows standard calling conventions (\_stdcall).

**MAC OS Project Builder™**: the driver is a Framework that resides in the **/Library/Frameworks** folder. If you use Apple Project Builder 2.1, add the XStream.framework file to your project.

**MS Visual C#**: the **XsTH.cs** file has been added to the include folder. It wraps the APIs into a C# class. Just include the file into your C# project and call the "**XsTH**" class members.

## 2.3. 64 bit programming

The Visual C++ stub COFF library for 64-bit programming is the **TimHubDrv64.lib** file, stored in the LIB subdirectory of the SDK.

One of the main issues in migrating software from 32 bit to 64 bit platforms is the size of types.

An “int” and a “long” are 32-bit values in on 64-bit Windows operating systems. For programs that you plan to compile for 64-bit platforms, you should be careful not to assign pointers to 32-bit variables. Pointers are 64-bit on 64-bit platforms, and you will truncate the pointer value if you assign it to a 32-bit variable.

## 2.4. Load/Unload the Driver

The first call into the Timing Hub driver must be **ThLoadDriver**. The routine loads the driver and initialize the environment. Then call **ThUnloadDriver** when you are finished.

## 2.5. Enumerate/Open a device

To get the list of available timing devices, call **ThEnumDevices**. Use the *nDeviceId* field of the devices list in your call to **ThOpenDevice**. Here is a simple example of opening the first available device:

```
TH_ENUMITEM thList[10];
unsigned long nListLen = sizeof(thList)/sizeof(TH_ENUMITEM);
ThLoadDriver();
// nListLen is the length of your TH_ENUMITEM array
ThEnumDevices( &thList[0], &nListLen );
// nListLen is now the number of devices available. It may be
// larger than your TH_ENUMITEM array length!
if (( nListLen > 0 ) && ( thList[0].bIsOpen == FALSE ))
{
    TH_HANDLE hDevice;
    // Open the first device in the list.
    ThOpenDevice( thList[0].nDeviceId, &hDevice );
    // Do something...
    ...
    // Close the device.
    ThCloseDevice( hDevice );
}
// Unload the driver
ThUnloadDriver();
```

The devices list contains a unique ID which identifies each particular device. Many developers use the unique ID to associate a meaningful name string with a timing device.

## 2.6. Configuring a device

The device state is represented by the opaque **TH\_SETTINGS** structure. You can read the default state, read the state from the device, or send the state to the device. Parameters are read and written to a **TH\_SETTINGS** structure with functions **ThGetParameter** and **ThSetParameter**. The function **ThGetParameterAttribute** provides information on a parameter's range and whether the parameter is read-only or not. When **ThSetParameter** is called to set a parameter of one of the 8 available outputs the output index is required. When all needed parameters have been changed in the **TH\_SETTINGS** structure, you can download the new configuration set to the device and activate the new settings calling the **ThRefreshDeviceSettings** function.

Here is an example of setting the channel 0 output to 1 ms high time and 1 ms low time.

```
TH_SETTINGS thCfg;
thCfg.cbSize = sizeof(TH_SETTINGS);           // Don't forget this!

// Read default settings from the device.
ThReadDefaultSettings( hDevice, &thCfg );

// Set channel 0 output high time and low time to 1 ms
// we obtain a 500 Hz signal
ThSetParameter(hDevice, &thCfg, TH_OUTPUT_CHN_0, THP_HIGH,
               NS_2_STEPS(1000000) );
ThSetParameter(hDevice, &thCfg, TH_OUTPUT_CHN_0, THP_LOW,
               NS_2_STEPS(1000000) );

// Send settings to the device
ThRefreshDeviceSettings( hDevice, &thCfg );
```

The macro **NS\_2\_STEPS** can be used to transform nanoseconds value to internal time base clocks number (1 internal step = 20 ns).

## 2.7. Opening and Closing output channels

Each output channel may be opened and closed without affecting the state of the others. The user may call **ThOpenOutputs** to open one or more channels simultaneously and call **ThCloseOutputs** to close them. One of the input parameters is the output mask: in the mask each bit controls a single channel. See the table below.

Mask Value	Controlled Output Channel
0x01	0
0x02	1
0x04	2
0x08	3
0x10	4
0x20	5
0x40	6
0x80	7

The example below shows how to open the output channels 0 and 1 simultaneously, and then close channel 1.

```
// Open channel 0 and 1
ThOpenOutputs ( hDevice, TH_OUTPUT_MSK_0 + TH_OUTPUT_MSK_1 );

// Do something
...

// Close channel 1 only
ThCloseOutputs ( hDevice, TH_OUTPUT_MSK_1 );
```

## 2.8. Measurement

The timing hub has two external inputs and 8 outputs. External signals and outputs frequency or pulse width may be computed. Two types of measurements are available:

### Synchronous measurement

The **ThSynchMeasure** returns after the measurement is completed or any error occurs. The user must specify the external input or output to be measured, the measurement type (frequency or pulse width), the inversion of the input signal (useful to compute the width of the low level portion of the signal), the time out value in ms and a valid pointer to a variable which receives the measurement result.

### Asynchronous measurement

The **ThStartMeasure** routine returns immediately. One of the parameters of the routine is a pointer to a callback routine which is called by the driver when the measurement is completed or when an error occurs. The measurement may be aborted by calling the **ThAbortMeasure** routine.

## 2.9. Signals Time-Base

The Timing Hub has 8 eight 32-bit counter output channels individually configurable. Another 32 bit counter is used to make measurements from external signals or outputs. Two external signal inputs may be used as external triggers or time-base for output waveforms.

The timing hub has an internal clock running at **50 MHz** that may be used to generate square-waves or pulses at lower frequency. External signals may be used as waveform generators.

If the **Time-base** for output signal generation is the internal clock, low level time, high level time and delay of output signal are set with a 20 ns resolution, referred as **internal clock units or steps**.

If the time-base for output signal generation is an external periodic signal, the resolution depends on the external signal rate. Example: if the external signal is a 100 KHz signal connected to the External Input 0, the time-base clock unit is  $1/100000 = 10 \mu\text{s}$ . A value of 3 for the High Level Time corresponds to  $30 \mu\text{s}$  and a value of 7 for the Low Level Time corresponds to  $70 \mu\text{s}$ . The output signal will have a period of  $100 \mu\text{s}$  (e.g. 10,000 Hz), with a duty cycle of 30%.



## 2.10. Internal Mode

In Internal mode you may generate a square waveform with an initial delay (THP\_DELAY), followed by a continuous pulse train with a programmable high time part (THP\_HIGH) and low time part (THP\_LOW). Each of these parameters is expressed in internal time-base units or steps (each step is 20 ns).

The initial delay may be executed at high or low level (THP\_DELAY\_STATE). Another signal (external input or a channel output) may be selected as a gate (THP\_GATE\_TRG) for the output signal. The timing hub executes a logical AND between the gate and the internal square waveform before generating the output signal. The gate signal may be inverted (TH\_GATE\_TRG\_INV), as well as the output signal (THP\_OUTPUT\_INV).

## 2.11. External Mode

In External mode you may generate a square waveform with an initial delay (THP\_DELAY), followed by a continuous pulse train with a programmable high time part (THP\_HIGH) and low time part (THP\_LOW). Each of these parameters is expressed in external time-base units or steps (each step is a period of the external signal).

The external source may be selected through THP\_SOURCE parameter. The initial delay may be executed at high or low level (THP\_DELAY\_STATE). Another signal (external input or an output channel) may be selected as a gate (THP\_GATE\_TRG) for the output. The timing hub executes a logical AND between the gate and the internal square waveform before generating the output signal. The gate signal may be inverted (TH\_GATE\_TRG\_INV), as well as the output signal (THP\_OUTPUT\_INV) and the source signal (THP\_SOURCE\_INV).

## 2.12. Start/Stop Mode

In Start/Stop mode you may generate a square waveform with an initial delay (THP\_DELAY), followed by a continuous pulses train with a programmable high time part (THP\_HIGH) and low time part (THP\_LOW). Each of these parameters is expressed in internal time-base units (20 ns). The initial delay can be executed at high or low level (THP\_DELAY\_STATE).

The generation of the waveform is controlled by two signals (triggers) selected among external inputs or other outputs. The leading edge of the first trigger (selected by the THP\_GATE\_TRG parameter) starts the output generation, while the leading edge of the second trigger (selected by the THP\_GATE\_TRG\_2 parameter) stops it. The procedure is repeated if the first and second triggers are periodic.

Both the triggers may be inverted (TH\_GATE\_TRG\_INV and TH\_GATE\_TRG\_INV\_2), as well as the output (THP\_OUTPUT\_INV).

## 2.13. Rate Switch Mode

In Rate Switch mode you may generate two square waveforms (main and alternate) with an initial delay (respectively THP\_DELAY and THP\_DELAY\_2), followed by a pulse train with a programmable high time part (THP\_HIGH and THP\_HIGH\_2) and low time part (THP\_LOW and THP\_LOW\_2). Each of these parameters is expressed in internal time-base units or steps (each step is 20 ns). The initial delay can be executed at high or low level (THP\_DELAY\_STATE).

The generation of the waveforms is controlled by two signals (triggers) selected among external inputs or other outputs. The leading edge of the first trigger (selected by the THP\_GATE\_TRG parameter) starts the generation of the main waveform, while the leading edge of the second trigger (selected by the THP\_GATE\_TRG\_2 parameter) starts the generation of the alternate one. The procedure is repeated if the first and second triggers are periodic.

Both the trigger may be inverted (TH\_GATE\_TRG\_INV and TH\_GATE\_TRG\_INV\_2), as well as the output (THP\_OUTPUT\_INV).

## 2.14. Burst Mode

In Burst mode you may generate a programmable number of pulses (THP\_PULSES\_COUNT) after an initial delay (THP\_DELAY). The pulses have a programmable high time part (THP\_HIGH) and low time part (THP\_LOW). Each of these parameters is expressed in internal time-base units or steps (each step is 20 ns). The initial delay can be executed at high or low level (THP\_DELAY\_STATE).

The pulses generation may be controlled by a signal (trigger) selected among external inputs or other outputs. The leading edge of the trigger (selected by the THP\_GATE\_TRG parameter) starts the pulses generation. The pulses generation is immediate if the trigger source parameter is set to TH\_INPUT\_NONE. The user may select if the pulses generation is done once or at any trigger leading edge (THP\_PULSES\_REPEAT parameter).

The trigger may be inverted (TH\_GATE\_TRG\_INV), as well as the output (THP\_OUTPUT\_INV).

## 3. MotionPro Timing Hub SDK Reference

### 3.1. Initialization Functions

#### 3.1.1. Overview: Initialization functions

Initialization functions allow the user to initialize the Timing Hub, enumerate the available devices, open and close them.

**ThGetVersion** returns the DLL version numbers and the demo flag.

**ThLoadDriver** loads the driver and initializes it.

**ThUnloadDriver** unloads the driver.

**ThEnumDevices** enumerates the Timing Hubs connected to the computer.

**ThOpenDevice** opens a timing device.

**ThCloseDevice** closes a timing device previously open.

### 3.1.2. ThGetVersion

**TH\_ERROR ThGetVersion (unsigned short \*pVerMajor, unsigned short \*pVerMinor, unsigned short \*plsDemo)**

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_GENERIC\_ERROR if the version numbers could not be extracted from the driver.

#### Parameters

*pVerMajor*

Specifies the pointer to the variable that receives the major version number

*pVerMinor*

Specifies the pointer to the variable that receives the minor version number

*plsDemo*

Specifies the pointer to the variable that receives the demo flag; If 1, the driver is demo, if 0 it isn't.

#### Remarks

This function must be called to retrieve the Timing Hub DLL version number and demo flag. If the demo flag is returned TRUE, the currently installed driver does not require the presence of the hub to operate.

#### See also:

### 3.1.3. ThLoadDriver

**TH\_ERROR ThLoadDriver (void)**

**Return values**

TH\_SUCCESS if successful, otherwise

TH\_E\_HARDWARE\_FAULT if any error occurs during the initialization.

**Parameters**

None

**Remarks**

The routine loads the Timing Hub driver DLL and initializes it. It must be called before any other routine, except **ThGetVersion**. If any error occurs, the routine returns TH\_E\_HARDWARE\_FAULT. The user may retrieve the hardware error code by calling the **ThGetHardwareError** routine.

See also: **ThUnloadDriver, ThGetHardwareError**

### 3.1.4. ThUnloadDriver

**void ThUnloadDriver (void)**

**Return values**

None

**Parameters**

None

**Remarks**

This function must be called before terminating the application. This function frees any memory and resource allocated by the device driver and unloads it.

See also: **ThLoadDriver**

### 3.1.5. ThEnumDevices

**TH\_ERROR** ThEnumDevices (PTH\_ENUMITEM *pItemList*, unsigned long *\*pItemNr*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_HARDWARE\_FAULT if any error occurs during the devices enumeration.

TH\_E\_INVALID\_ARGUMENTS, if any of the parameters is not valid.

#### Parameters

*pItemList*

Specifies the pointer to an array of TH\_ENUMITEM structures

*pItemNr*

Specifies the pointer to the variable that receives the number of detected devices

#### Remarks

The routine enumerates the active devices and fills the **TH\_ENUMITEM** structures with information about them. This routine must be called before **ThOpenDevice** to find out which devices are available. The *pItemNr* variable must specify the number of structures in the *pItemList* array and receives the number of detected devices. If any error occurs during the devices enumeration, the routine returns TH\_E\_HARDWARE\_FAULT. The user may retrieve the hardware error code by calling the **ThGetHardwareError** routine.

See also: **ThOpenDevice**, **ThGetHardwareError**

### 3.1.6. ThOpenDevice

**TH\_ERROR ThOpenDevice (unsigned long *nDeviceId*, TH\_HANDLE\* *pHandle*)**

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_DEV\_ID, if the device ID is not valid.

TH\_E\_HARDWARE\_FAULT if any error occurs during the device opening.

#### Parameters

*nDeviceId*

Specifies the ID of the device to be opened

*pHandle*

Specifies the pointer to the variable that receives the device handle

#### Remarks

The routine opens the device whose ID is in the variable *nDeviceId*. The value can be retrieved calling the **ThEnumDevices** (see the TH\_ENUMITEM structure). If any error occurs during the device opening, the routine returns TH\_E\_HARDWARE\_FAULT. The user may retrieve the hardware error code by calling the **ThGetHardwareError** routine

See also: **ThCloseDevice**, **ThGetHardwareError**



### 3.1.7. ThCloseDevice

**TH\_ERROR ThCloseDevice (TH\_HANDLE *hDevice*)**

#### **Return values**

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_DEVICE\_ID, if the device ID is not valid.

#### **Parameters**

*hDevice*

Specifies the handle to an open device

#### **Remarks**

Closes an open Device

See also: **ThOpenDevice**

## 3.2. Configuration Functions

### 3.2.1. Overview: Configuration functions

The configuration functions allow the user to control the parameters of the timing hub device.

**ThGetDeviceInfo** gets information from the timing device, such as model, firmware version, revision, etc.

**ThReadDefaultSettings** reads the default settings from the device and fills the TH\_SETTINGS opaque structure.

**ThReadDeviceSettings** reads the current settings from the device and fills the TH\_SETTINGS opaque structure.

**ThRefreshDeviceSettings** sends an updated TH\_SETTINGS structure to the device and refreshes the device settings.

**ThValidateDeviceSettings** validates and updates a device TH\_SETTINGS structure.

**ThSetParameter** sets one of the device parameters in the TH\_SETTINGS opaque structure.

**ThGetParameter** gets one of the parameters from the TH\_SETTINGS opaque structure.

**ThGetParameterAttribute** gets a parameter's attribute, such as minimum value, maximum value, default value, read-only attribute.

### 3.2.2. ThGetDeviceInfo

**TH\_ERROR** ThGetDeviceInfo (TH\_HANDLE *hDevice*, TH\_INFO *nInfoKey*, unsigned long \**pValueLo*, unsigned long \**pValueHi*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_NOT\_SUPPORTED, if the *nInfoKey* is not supported.

#### Parameters

*hDevice*

Specifies the handle to an open device

*nInfoKey*

Specifies which parameter the function has to return

*pValueLo*

Specifies the pointer to the variable that receives the least significant long part of the value

*pValueHi*

Specifies the pointer to the variable that receives the most significant long part of the value

#### Remarks

This function returns device specific information, such as device type or version numbers, generally state-independent information. If the value range exceeds a 32 bit value, the most significant long value is filled. See the **Appendix B** for a list of all the available *nInfoKey* values.

**See also:** ThGetParameter

### 3.2.3. ThReadDefaultSettings

**TH\_ERROR** ThReadDefaultSettings (TH\_HANDLE *hDevice*, PTH\_SETTINGS *pSettings*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

#### Parameters

*hDevice*

Specifies the handle to an open device

*pSettings*

Specifies the pointer to the structure to be filled with the device settings

#### Remarks

This function reads the default settings of the specified device and fills the TH\_SETTINGS structure. The structure is opaque and can be accessed only through the ThGetParameter and ThSetParameter functions. To change a parameter on the device, the entire structure must be sent to the driver, using the ThRefreshDeviceSettings function. The default state is specific to each individual device.

**See also:** ThGetParameter, ThSetParameter, ThRefreshDeviceSettings

### 3.2.4. ThReadDeviceSettings

**TH\_ERROR ThReadDeviceSettings** (**TH\_HANDLE** *hDevice*, **PTH\_SETTINGS** *pSettings*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

#### Parameters

*hDevice*

Specifies the handle to an open device

*pSettings*

Specifies the pointer to the structure to be filled with the device settings

#### Remarks

This function reads the current settings of the specified device and fills the TH\_SETTINGS structure. The structure is opaque and can be accessed only through the ThGetParameter and ThSetParameter functions. To change a parameter on the device, the entire structure must be sent to the driver, using the ThRefreshDeviceSettings function.

**See also:** ThGetParameter, ThSetParameter, ThRefreshDeviceSettings

### 3.2.5. ThRefreshDeviceSettings

**TH\_ERROR ThRefreshDeviceSettings** (**TH\_HANDLE** *hDevice*, **PTH\_SETTINGS** *pSettings*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_INVALID\_CFG, if the TH\_SETTINGS structure is not valid.

#### Parameters

*hDevice*

Specifies the handle to an open device

*pSettings*

Specifies the pointer to the structure that contains the device settings

#### Remarks

The state contained in the TH\_SETTINGS structure is validated, modified if necessary, and then sent to the device. The structure is opaque and can be accessed only through the ThGetParameter and ThSetParameter functions.

**See also:** ThReadDefaultSettings, ThReadDeviceSettings

### 3.2.6. ThValidateDeviceSettings

**TH\_ERROR ThValidateDeviceSettings** (**TH\_HANDLE** *hDevice*, **PTH\_SETTINGS** *pSettings*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_INVALID\_CFG, if the TH\_SETTINGS structure is not valid.

#### Parameters

*hDevice*

Specifies the handle to an open device

*pSettings*

Specifies the pointer to the structure that contains the device settings

#### Remarks

The state contained in the TH\_SETTINGS structure is validated and modified if necessary. The structure is opaque and can be accessed only through the ThGetParameter and ThSetParameter functions.

**See also:** [ThReadDefaultSettings](#), [ThReadDeviceSettings](#)

### 3.2.7. ThSetParameter

**TH\_ERROR** ThSetParameter (**TH\_HANDLE** *hDevice*, **PTH\_SETTINGS** *pSettings*, **unsigned long** *nOutput*, **TH\_PARAM** *nParamKey*, **unsigned long** *nValue*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_NOT\_SUPPORTED, if the *nParamKey* is not supported.

TH\_E\_READONLY, if the parameter is read-only and cannot be changed

#### Parameters

*hDevice*

Specifies the handle to an open device

*pSettings*

Specifies the pointer to the TH\_SETTINGS structure the parameter is written to.

*nOutput*

Specifies the index of the output channel to configure

*nParamKey*

Specifies which parameter the function sets.

*nValue*

Specifies the parameter's value

#### Remarks

This function writes a parameter to the TH\_SETTINGS structure. The user must specify the output channel which receives the new parameter value.

**See also:** ThGetParameter



### 3.2.8. ThGetParameter

**TH\_ERROR** ThGetParameter (**TH\_HANDLE** *hDevice*, **PTH\_SETTINGS** *pSettings*, **unsigned long** *nOutput*, **TH\_PARAM** *nParamKey*, **unsigned long** *\*pValue*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_NOT\_SUPPORTED, if the *nParamKey* is not supported.

#### Parameters

*hDevice*

Specifies the handle to an open device

*pSettings*

Specifies the pointer to the TH\_SETTINGS structure the parameter is read from

*nOutput*

Specifies the index of the output channel the parameter is read from.

*nParamKey*

Specifies which parameter the function returns

*pValue*

Specifies the pointer to the parameter's value

#### Remarks

This function reads a parameter from the TH\_SETTINGS structure. The user must specify the output channel which owns the parameter value.

**See also:** ThSetParameter

### 3.2.9. ThGetParameterAttribute

**TH\_ERROR** ThGetParameterAttribute (**TH\_HANDLE** *hDevice*, **PTH\_SETTINGS** *pSettings*, **TH\_PARAM** *nParamKey*, **TH\_ATTRIBUTE** *nParamAttr*, **unsigned long** *\*pValue*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_NOT\_SUPPORTED, if the nParamKey is not supported.

#### Parameters

*hDevice*

Specifies the handle to an open device

*pSettings*

Specifies the pointer to the TH\_SETTINGS structure the parameter is read from.

*nParamKey*

Specifies which parameter the function returns.

*nParamAttr*

Specifies which attribute the function returns.

*pValue*

Specifies the pointer to the parameter's attribute value.

#### Remarks

This function reads a parameter attribute depending on the nParamAttr value. It may be: minimum value, maximum value, default value, read-only attribute (see Appendix D).

**See also:** ThGetParameter

## 3.3. Outputs Enable/Disable Functions

### 3.3.1. Overview: Outputs Enable/Disable Functions

These functions allow the user to open or close one or more output channels.

**ThOpenOutputs** turns on one or more output channels.

**ThCloseOutputs** turns off/resets one or more output channels.

### 3.3.2. ThOpenOutputs

**TH\_ERROR** ThOpenOutputs(**TH\_HANDLE** *hDevice*, **unsigned long** *nOutputMask*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_HARDWARE\_FAULT if any error occurs during the device opening.

#### Parameters

*hDevice*

Specifies the handle of an open device

*nOutputMask*

Specifies the mask of the output channels to open

#### Remarks

The routine opens one or more output channel according to the *nOutputMask* parameter. In the output mask parameter, each bit controls a single output channel (BIT0 for channel 0, BIT1 for channel 1, etc.). If other output channels have been opened before, the routine does not affect their state. If any error occurs during the device opening, the routine returns TH\_E\_HARDWARE\_FAULT. The user may retrieve the hardware error code by calling the **ThGetHardwareError** routine.

See also: **ThCloseOutputs**, **ThGetHardwareError**

### 3.3.3. ThCloseOutputs

**TH\_ERROR** ThCloseOutputs(**TH\_HANDLE** *hDevice*, **unsigned long** *nOutputMask*)

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_HARDWARE\_FAULT if any error occurs during the device opening.

#### Parameters

*hDevice*

Specifies the handle of an open device

*nOutputMask*

Specifies the mask of the output channels to close

#### Remarks

The routine closes/resets one or more output channel according to the *nOutputMask* parameter. In the output mask parameter, each bit controls a single output channel (BIT0 for channel 0, BIT1 for channel 1, etc.). If other output channels have been opened before, the routine does not affect their state. If any error occurs during the device opening, the routine returns TH\_E\_HARDWARE\_FAULT. The user may retrieve the hardware error code by calling the **ThGetHardwareError** routine.

See also: **ThOpenOutputs**, **ThGetHardwareError**

## 3.4. Measure Functions

### 3.4.1. Overview: Measure functions

Measurement functions allow the user to measure external frequency and pulse width from one of the timing hub inputs.

**ThSynchMeasure** measures from one of the external inputs or channels outputs synchronously.

**ThStartMeasure** starts a measurement from one of the input or channels outputs and returns immediately.

**ThAbortMeasure** aborts a measure previously started.

### 3.4.2. ThSynchMeasure

**TH\_ERROR ThSynchMeasure (TH\_HANDLE *hDevice*, unsigned long *nMeasureInput*, unsigned long *nMeasureType*, unsigned long *nInvertInput*, unsigned long\* *pnMeasuredValue*, unsigned long *nTimeOut*)**

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_TIMEOUT, if a time out occurred.

TH\_E\_HARDWARE\_FAULT, if any error occurs while calling the driver.

#### Parameters

*hDevice*

Specifies the handle to an open device

*nMeasureInput*

Specifies the measurement input channel.

*nMeasureType*

Specifies the measurement type (frequency or pulse width)

*nInvertInput*

Specifies if the signal has to be inverted before the measurement

*pnMeasuredValue*

Specifies the pointer to the variable which receive the measured value

*nTimeOut*

Specifies the measurement time out

#### Remarks

This function reads a value from one of the device channels (two external inputs and eight outputs). The input channel and the measurement type (frequency or pulse width) must be specified. The input signal may be inverted before the measurement. This feature is useful to measure the duration of the low level portion of the signal. The routine returns when the measurement is done or any error occurs (synchronous measurement).

See also: **ThStartMeasure**

### 3.4.3. ThStartMeasure

**TH\_ERROR ThStartMeasure (TH\_HANDLE *hDevice*, unsigned long *nMeasureInput*, unsigned long *nMeasureType*, unsigned long *nInvertInput*, unsigned long\* *pnMeasuredValue*, TH\_AsyncCallback *pfnCallback*, unsigned long *nFlags*, unsigned long *nUserData*)**

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_HARDWARE\_FAULT, if any error occurs while calling the driver.

TH\_E\_GENERIC\_ERROR, if any other error occurred.

#### Parameters

*hDevice*

Specifies the handle to an open device

*nMeasureInput*

Specifies the measurement input channel.

*nMeasureType*

Specifies the measurement type (frequency or pulse width)

*nInvertInput*

Specifies if the signal has to be inverted before the measurement

*pnMeasuredValue*

Specifies the pointer to the variable which receive the measured value

*pfnCallback*

Specifies the pointer to the callback routine; the routine is called by the driver when the measurement is done or any error occurred. See the TH\_AsyncCallback in the Appendix.

*nFlags*

Specifies the flags; see Appendix.

*nUserData*

Specifies the value of user data; the value is passed to the callback routine when it's called.

#### Remarks

This function starts a measurement and returns immediately. When the measure has been completed or any error occurred, the *pfnCallback* routine is called.

**See also:** ThSynchMeasure, ThAbortMeasure



### 3.4.4. ThAbortMeasure

**TH\_ERROR ThAbortMeasure (TH\_HANDLE *hDevice*)**

**Return values**

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_HARDWARE\_FAULT, if any error occurs while calling the driver.

**Parameters**

*hDevice*

Specifies the handle to an open device

**Remarks**

This function aborts any measurement previously started.

**See also:** ThStartMeasure

## 3.5. Miscellaneous Functions

### 3.5.1. Overview: Miscellaneous functions

Miscellaneous functions allow the user to read hardware error codes and strings.

**ThGetHardwareError** reads the hardware error code and returns the error string related to that code.

### 3.5.2. ThGetHardwareError

**TH\_ERROR ThGetHardwareError (TH\_HANDLE *hDevice*, unsigned long\* *pnHwError*, char\* *pszBuffer*, unsigned long *nSize*)**

#### Return values

TH\_SUCCESS if successful, otherwise

TH\_E\_INVALID\_HANDLE, if the device handle is not valid.

TH\_E\_INVALID\_ARGUMENTS, if one of the arguments is not valid.

TH\_E\_GENERIC\_ERROR, if the hardware error code is not correct.

#### Parameters

*hDevice*

Specifies the handle to an open device

*pnHwError*

Specifies the pointer to the variable which receives the error code

*pszBuffer*

Specifies the char buffer which receives the error string

*nSize*

Specifies the size in bytes of the char buffer

#### Remarks

If any of the driver's API returns TH\_E\_HARDWARE\_FAULT, the hardware related error may be retrieved by calling **ThGetHardwareError** function. The function returns the hardware error occurred after the latest device operation. Also, the function fills the *pszBuffer* buffer with a message that describes the returned error code.

#### See also:

## 4. MotionPro Timing Hub ActiveX Control Reference

### 4.1. Overview

ActiveX is a set of technologies that enable software components to interact with one another in a networked environment, regardless of the language in which the components were created. An ActiveX control is a user interface element created using ActiveX technology. ActiveX controls are small, fast, and powerful, and make it easy to integrate and reuse software components.

The **XStreamTHX** ActiveX control includes all the capabilities of the Timing Hub in a simple windowless control that can be inserted in any application. The ActiveX technology is supported in Windows Operating Systems only.

## 4.2. Timing Control Functions

### 4.2.1. Overview: Timing Control functions

Device Control functions allows the user to control device.

**Open** opens a device.

**Close** closes a device previously open.

**GetInfo** gets information from the timing device, such as model, firmware version, revision, etc.

**SetParameter** sets one of the parameters in the configuration set.

**GetParameter** gets one of the parameters from the configuration set.

**GetParameterAttribute** gets a parameter's attribute, such as minimum value, maximum value, default value, read-only.

**RefreshDeviceSettings** sends the new configuration settings to the device and activates them.

**ReadDefaultSettings** loads the default configuration into the configuration set.

**OpenOutputs** turns on one or more output channels.

**CloseOutputs** turns off/resets one or more output channels.

**Measure** measures from one of the inputs channels.

## 4.2.2. Open

**BOOL Open** (long *nDeviceId*)

### Return values

TRUE if successful, otherwise

FALSE if any error occurs.

### Parameters

*nDeviceId*

It specifies the ID of the device to be opened.

### Remarks

The routine opens the device whose ID is in the variable *nDeviceId*. Use 0 to open first enumerated device, 1 to open the second, and so on.

**See also:** **Close**

### 4.2.3. Close

**void Close (void)**

**Return values**

None

**Parameters**

None

**Remarks**

This function closes a device previously open.

**See also:** Open

#### 4.2.4. GetInfo

**long GetInfo** (**long** *nInfoKey*)

##### **Return values**

The info value, if successful, otherwise 0

##### **Parameters**

*nInfoKey*

Specifies which parameter the function has to return

##### **Remarks**

This function returns device specific information, such as device type or version numbers, generally state-independent information. See the **Appendix B** for a list of all the available *nInfoKey* values.

**See also:** GetParameter



### 4.2.5. SetParameter

**short SetParameter** (**long** *nOutput*, **long** *nParamKey*, **long** *nValue*)

#### Return values

TRUE if successful, otherwise

FALSE if any error occurs.

#### Parameters

*nOutput*

Specifies the index of the output channel to configure

*nParamKey*

Specifies which parameter the function sets.

*nValue*

Specifies the parameter's value

#### Remarks

This function writes a specific configuration parameter to the configuration set. The parameter key is one of the input parameters. A list of the parameters indexes is available in Appendix C. The user may call SetParameter several times to set different parameters, and then call RefreshDeviceSettings to download the configuration to the device and activate it.

**See also:** [GetParameter](#), [RefreshDeviceSettings](#)

## 4.2.6. RefreshDeviceSettings

### **BOOL RefreshDeviceSettings (void)**

#### **Return values**

TRUE if successful, otherwise

FALSE if any error occurs.

#### **Parameters**

#### **Remarks**

This function downloads the current configuration to the device and activates it. The user may call `SetParameter` several times to set different parameters, and then call `RefreshDeviceSettings` to download the configuration to the device.

**See also:** `GetParameter`, `SetParameter`, `ReadDefaultSettings`

## 4.2.7. ReadDefaultSettings

### **BOOL ReadDefaultSettings (void)**

#### **Return values**

TRUE if successful, otherwise

FALSE if any error occurs.

#### **Parameters**

#### **Remarks**

This function loads the default parameters values from the device and stores them to a local hidden structure. The user may call RefreshDeviceSettings to download the new configuration to the device and activate it.

**See also:** GetParameter, SetParameter, RefreshDeviceSettings

### 4.2.8. GetParameter

**long GetParameter (long *nOutput*, long *nParamKey*)**

#### **Return values**

The parameter's value, if successful, otherwise 0

#### **Parameters**

*nOutput*

Specifies the index of the output channel the parameter is read from.

*nParamKey*

Specifies which parameter the function returns

#### **Remarks**

This function reads a device parameter.

**See also:** SetParameter, RefreshDeviceSettings

### 4.2.9. GetParameterAttribute

**long GetParameterAttribute (long *nParamKey*, long *nParamAttr*)**

#### **Return values**

The parameter's attribute, if successful, otherwise 0

#### **Parameters**

*nParamKey*

Specifies which parameter the function returns.

*nParamAttr*

Specifies which attribute the function returns.

#### **Remarks**

This function reads a parameter attribute depending on the *nParamAttr* value. It may be: minimum value, maximum value, default value, read-only attribute (see Appendix D).

**See also:** GetParameter

### 4.2.10. OpenOutputs

**BOOL** ThOpenOutputs (**long** *nOutputMask*)

#### Return values

TRUE if successful, otherwise

FALSE if any error occurs.

#### Parameters

*nOutputMask*

Specifies the mask of the output channels to open

#### Remarks

The routine opens one or more output channel according to the *nOutputMask* parameter. If other output channels have been opened before, the routine does not affect their state.

**See also:** CloseOutputs

### 4.2.11. CloseOutputs

**BOOL CloseOutputs** (**long** *nOutputMask*)

**Return values**

TRUE if successful, otherwise

FALSE if any error occurs.

**Parameters**

*nOutputMask*

Specifies the mask of the output channels to close

**Remarks**

The routine closes/resets one or more output channel according to the *nOutputMask* parameter. If other output channels have been opened before, the routine does not affect their state.

**See also:** OpenOutputs

## 4.2.12. Measure

**long Measure** (**long** *nInput*, **long** *nType*, **long** *nTimeOut*)

### Return values

The measured value, if successful, otherwise 0

### Parameters

*nInput*

Specifies the measurement input channel.

*nType*

Specifies the measurement type (frequency or pulse width)

*nTimeOut*

Specifies the measurement time out

### Remarks

This function measures the frequency or the pulse width of one the two external input channels or one of the outputs. The routine returns when the measurement is done or any error occurs.

### See also:



## 5. MotionPro Timing Hub LabVIEW™ Interface Reference

### 5.1. Overview

The Timing Hub LabVIEW™ Interface allows generating triggering signals and measuring external input from inside National Instruments LabVIEW application. It works with LabVIEW 6 and greater, on Windows 2000/XP. Windows NT is not supported.

The Timing Hub LabVIEW™ Interface includes the **VIs (Virtual Instruments)** for controlling the timing hub and some example VIs to show how to use the interface: the Timing Hub VIs are packaged in a library called **IDTTH.LLB**) located in the **IdtTH** directory in the **user.lib** subdirectory of the LabVIEW folder. The examples are located in the **LabVIEW** subdirectory of the installation folder (C:\Program Files\IDTXsTH).

The Timing Hub VIs may be accessed by selecting the “Show Functions Palette” menu item from the Window” menu, then by clicking the “User Libraries” button and the “IDT Timing Hub VIs” button.

The interface is not supported by the MAC OS/X version of the SDK.

The VI interface and examples are listed below.

## 5.2. Initialization VIs

### 5.2.1. Overview: Initialization VIs

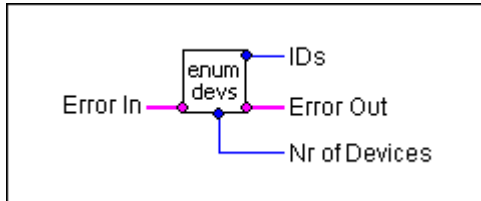
Initialization Virtual Instruments allow the user to enumerate the available devices, initialize, open and close them.

**IDT TH Enum Devices** enumerates the Timing Hub devices currently connected to the computer.

**IDT TH Open Device** opens a Timing Hub.

**IDT TH Close Device** closes a Timing Hub previously open.

## 5.2.2. IDT TH Enum Devices



### Inputs

*Error*

Specifies a standard error cluster input terminal

### Outputs

*Error*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*IDs*

Specifies the array containing the IDs of the detected devices

*Nr of Devices*

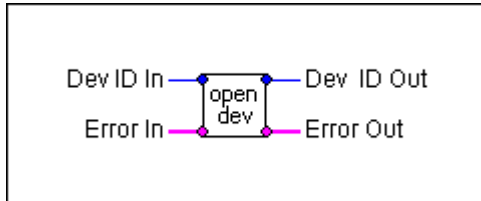
Specifies the number of detected timing hub devices

### Remarks

The VI enumerates the active devices and returns a list of the detected device IDs. This VI must be set before “**IDT TH Open Device**” to find out which devices are available. The “Nr of devices” output contains the number of detected devices. If any error occurs during the devices enumeration, the Error Out terminal signals the error condition.

**See also:** “IDT TH Open Device”

### 5.2.3. IDT TH Open Device



#### Inputs

##### *Device ID*

Specifies the ID of the device to be opened, or 0 for the first available device

##### *Error*

Specifies a standard error cluster input terminal

#### Outputs

##### *Error*

Specifies the return error condition

##### *Device ID*

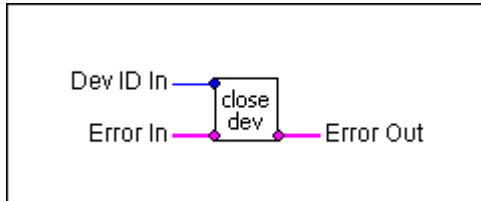
Specifies the ID of the opened device

#### Remarks

The VI opens the device with a specific ID. The value can be retrieved by calling the “**IDT TH Enum Devices**” VI. The user may supply a specific device ID or 0: in this case the first available device is opened. If any error occurs during the device opening, the Error Out terminal signals this error. The VI also returns the ID of the open device.

**See also:** “IDT TH Close Device”

## 5.2.4. IDT TH Close Device



### Inputs

#### *Device ID*

Specifies the ID of the device to be closed

#### *Error*

Specifies a standard error cluster input terminal

### Outputs

#### *Error*

Specifies the return error condition

### Remarks

This VI closes a device previously open. If any error occurs during the operation, the Error Out terminal signals this error.

**See also:** "IDT TH Open Device"

## 5.3. Configuration VIs

### 5.3.1. Overview: Configuration VIs

Configuration Virtual Instruments allow the user to read information from the device, read configuration parameters from the device and write them to the device.

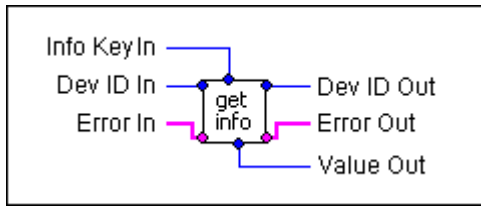
**IDT TH Get Info** reads information from the device, such as device model, firmware version, etc.

**IDT TH Get Parameter** reads a single specific parameter from the configuration and gets its minimum and maximum value.

**IDT TH Set Parameter** writes a single specific parameter to the configuration.

**IDT TH Send Config** downloads the updated configuration to the device and activates it.

### 5.3.2. IDT TH Get Info



#### Inputs

*Device ID*

Specifies a valid device ID

*Info Key*

Specifies which parameter has to be returned by the VI

*Error*

Specifies a standard error cluster input terminal

#### Outputs

*Device ID*

Specifies the device ID

*Error*

Specifies the return error condition

*Value*

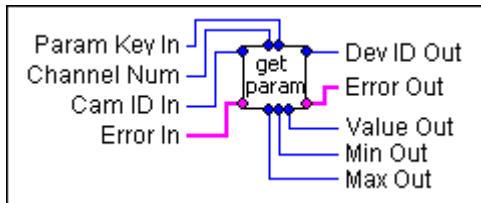
Specifies the value of the info parameter

#### Remarks

This VI returns device specific information, such as device model and serial number, generally state-independent information. See the Appendix B for a list of all the available Info Key values. If any error occurs during the operation, the Error Out terminal signals this error.

**See also:** "IDT TH Get Parameter"

### 5.3.3. IDT TH Get Parameter



#### Inputs

##### *Device ID*

Specifies a valid device ID

##### *Error*

Specifies a standard error cluster input terminal

##### *Param Key*

Specifies the index of the parameter

##### *Channel Num*

Specifies the index of the output channel

#### Outputs

##### *Device ID*

Specifies the device ID

##### *Error*

Specifies the return error condition

##### *Value*

Specifies the current value of the parameter

##### *Min*

Specifies the minimum value of the parameter

##### *Max*

Specifies the maximum value of the parameter

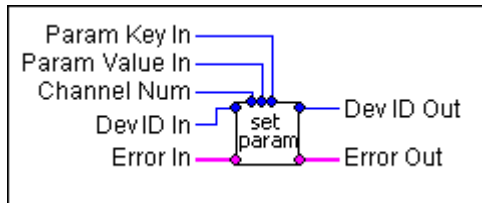
#### Remarks

This VI reads a specific configuration parameter from the device and returns the parameter value, the minimum and the maximum. The parameter key is one of the input parameters. A list of the parameters constants is available in Appendix C. If any error occurs during the operation, the Error Out terminal signals this error.

**See also:** "IDT TH Set Parameter", "IDT TH Send Config"



### 5.3.4. IDT TH Set Parameter



#### Inputs

##### *Device ID*

Specifies a valid device ID

##### *Error*

Specifies a standard error cluster input terminal

##### *Param Key*

Specifies the index of the parameter

##### *Param Value*

Specifies the value of the parameter

##### *Channel Num*

Specifies the index of the output channel

#### Outputs

##### *Device ID*

Specifies the device ID

##### *Error*

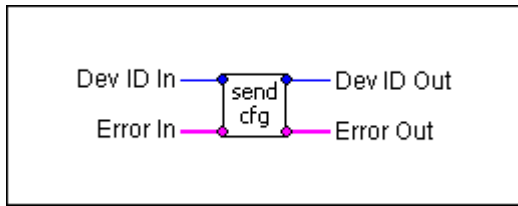
Specifies the return error condition

#### Remarks

This VI writes a specific configuration parameter to the device. The parameter key is one of the input parameters. A list of the parameters indexes is available in Appendix C. If any error occurs during the operation, the Error Out terminal signals this error. The user may call the **"IDT TH Set Parameter"** VI several times to set different parameters, and then call the **"IDT TH Send Config"** VI to download the configuration to the device.

**See also:** "IDT TH Get Parameter", "IDT TH Send Config"

### 5.3.5. IDT TH Send Config



#### Inputs

##### *Device ID*

Specifies a valid device ID

##### *Error*

Specifies a standard error cluster input terminal

#### Outputs

##### *Device ID*

Specifies the device ID

##### *Error*

Specifies the return error condition

#### Remarks

This VI sends the current configuration to the device and activates it. The user may call the “**IDT TH Set Parameter**” VI several times to set different parameters, and then call the “**IDT TH Send Config**” VI to download the configuration to the device. If any error occurs during the operation, the Error Out terminal signals this error.

**See also:** “IDT TH Get Parameter”, “IDT TH Set Parameter”

## 5.4. Outputs Enable/Disable VIs

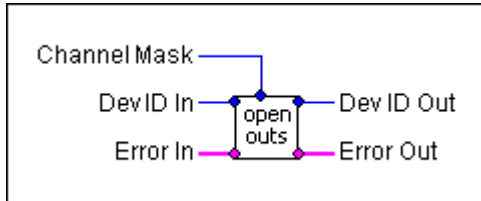
### 5.4.1. Overview: Outputs enable/disable VIs

These VIs allow the user to open or close one or more output channels.

**IDT TH Open Outputs** turns on one or more output channels.

**IDT TH Close Outputs** turns off/resets one or more output channels.

## 5.4.2. IDT TH Open Outputs



### Inputs

*Device ID*

Specifies a valid device ID

*Channels Mask*

Specifies the mask of the output channels to open/enable

*Error*

Specifies a standard error cluster input terminal

### Outputs

*Error*

Specifies the return error condition

*Device ID*

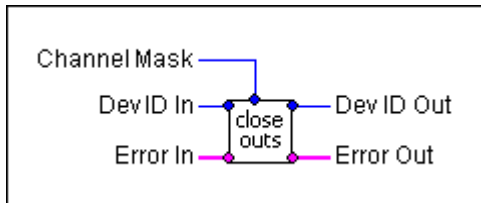
Specifies the device ID

### Remarks

The VI opens one or more output channel according to the Channels Mask parameter. In the output mask parameter, each bit controls a single output channel (BIT0 for channel 0, BIT1 for channel 1, etc.). If other output channels have been opened before, the routine does not affect their state. If any error occurs, the Error Out terminal signals this error.

**See also:** "IDT TH Close Outputs"

### 5.4.3. IDT TH Close Outputs



#### Inputs

*Device ID*

Specifies a valid device ID

*Channels Mask*

Specifies the mask of the output channels to close/disable

*Error*

Specifies a standard error cluster input terminal

#### Outputs

*Error*

Specifies the return error condition

*Device ID*

Specifies the device ID

#### Remarks

The VI closes/resets one or more output channel according to the Channel Mask parameter. In the output mask parameter, each bit controls a single output channel (BIT0 for channel 0, BIT1 for channel 1, etc.). If other output channels have been opened before, the routine does not affect their state. If any error occurs, the Error Out terminal signals this error.

**See also:** "IDT TH Open Outputs"

## 5.5. Measurement VIs

### 5.5.1. Overview: Measurement VIs

These VIs allow the user to measure the frequency or the pulse width of a signal (one of the two external inputs or one of the eight outputs).

**IDT TH Synch Measure** measures frequency or pulse width synchronously.

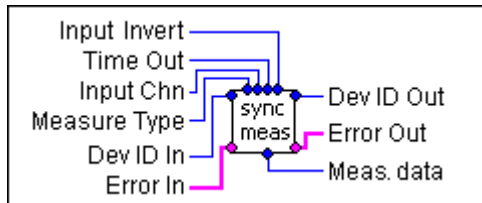
**IDT TH Measure Start** starts an asynchronous measure of frequency or pulse width.

**IDT TH Measure Abort** aborts the current asynchronous measurement.

**IDT TH Measure Read** reads the value of the current asynchronous measured data.

**IDT TH Measure is Ready** signals if the current asynchronous measure has been completed.

## 5.5.2. IDT TH Synch Measure



### Inputs

#### *Device ID*

Specifies a valid device ID

#### *Error*

Specifies a standard error cluster input terminal

#### *Time Out*

Specifies the measure time out in ms

#### *Measure Input*

Specifies the input channel

#### *Measure Type*

Specifies the type of value to measure (frequency, pulse width)

#### *Input Invert*

Specifies if the input needs to be inverted before the measurement

### Outputs

#### *Device ID*

Specifies the device ID

#### *Error*

Specifies the return error condition

#### *Measured Value*

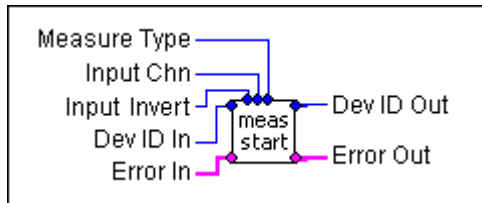
Specifies the measured data output

### Remarks

This VI measures a signal frequency or pulse width synchronously. The input may be one of the two external inputs or one of the eight outputs. The measurement is synchronous and the function exits when the measurement has been completed or a time out occurs.

### See also:

### 5.5.3. IDT TH Measure Start



#### Inputs

##### *Device ID*

Specifies a valid device ID

##### *Error*

Specifies a standard error cluster input terminal

##### *Measure Input*

Specifies the input channel

##### *Measure Type*

Specifies the type of value to measure (frequency or pulse width)

##### *Input Invert*

Specifies if the input needs to be inverted before the measurement

#### Outputs

##### *Device ID*

Specifies the device ID

##### *Error*

Specifies the return error condition

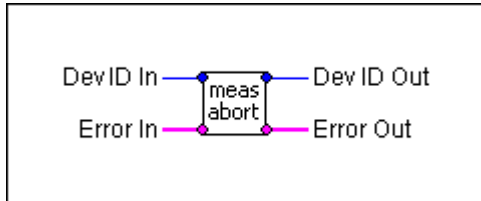
#### Remarks

This VI starts a measurement and returns immediately. The user may know when measurement has been completed by calling the **"IDT TH Measure is Ready"** VI. If any error occurs during the operation, the Error Out terminal signals this error.

**See also:** "IDT TH Measure Abort", "IDT TH Measure is Ready", "IDT TH Measure Read"



### 5.5.4. IDT TH Measure Abort



#### Inputs

*Device ID*

Specifies a valid device ID

*Error*

Specifies a standard error cluster input terminal

#### Outputs

*Device ID*

Specifies the device ID

*Error*

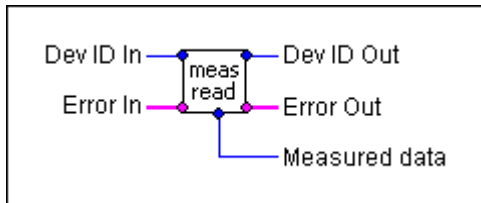
Specifies the return error condition

#### Remarks

This VI aborts the current asynchronous measurement. If any error occurs during the operation, the Error Out terminal signals this error.

**See also:** "IDT TH Measure Start", "IDT TH Measure is Ready", "IDT TH Measure Read"

### 5.5.5. IDT TH Measure Read



#### Inputs

*Device ID*

Specifies a valid device ID

*Error*

Specifies a standard error cluster input terminal

#### Outputs

*Device ID*

Specifies the device ID

*Error*

Specifies the return error condition

*Measured Data*

Specifies the measured data

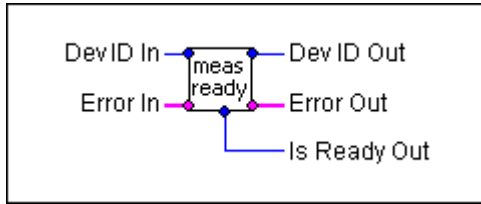
#### Remarks

This VI reads the value of the current asynchronous measured data.

.

**See also:** "IDT TH Measure Start", "IDT TH Measure is Ready", "IDT TH Measure Abort"

### 5.5.6. IDT TH Measure is Ready



#### Inputs

*Device ID*

Specifies a valid device ID

*Error*

Specifies a standard error cluster input terminal

#### Outputs

*Device ID*

Specifies the device ID

*Error*

Specifies the return error condition

*Is Ready*

Specifies whether the measurement is finished (1) or not (0).

#### Remarks

This VI returns the status of the current measurement. If the “Is Ready” returned value is 1 the current measurement has been completed, otherwise not.

**See also:** “IDT TH Measure Start“, “IDT TH Measure Read“, “IDT TH Measure Abort“

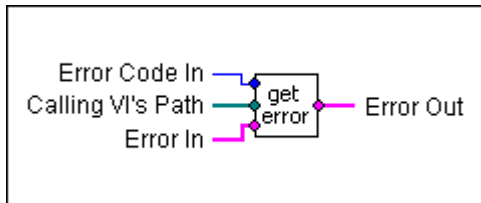
## 5.6. Miscellaneous VIs

### 5.6.1. Overview: Miscellaneous VIs

Miscellaneous Virtual Instruments allow the user to manage the error conditions in the Timing Hub VIs.

**IDT TH Get Error** manages the error conditions in the other Timing Hub VIs (this VI is for internal use only).

## 5.6.2. IDT TH Get Error



### Inputs

*Error Code*

Specifies the device specific error code

*Calling VI's Path*

Specifies the path of the VI which generates the error

*Error*

Specifies a standard error cluster input terminal

### Outputs

*Error*

Specifies the return error condition

### Remarks

This VI manages the error conditions in the other Timing Hub VIs (this VI is for internal use only).

### See also:

## 5.7. How to use the VIs

### 5.7.1. Opening and closing a device

A device must be opened before using its functions and then it must be closed. To open a specific device you have to supply to the Open VI the unique ID of that device. You may also supply 0 to open the first available device. To obtain the list of all available devices you may use the “IDT TH Enum Devices” VI.

### 5.7.2. Configuring a device

Before configuring a device, several calls to the “IDT TH Set Parameter” VI may be done. When the parameters have been set, a call to the “IDT TH Send Config” VI downloads the new configuration activates it. If you want to read a parameter value you may call the “IDT TH Get Parameter” VI.

### 5.7.3. Measurement

Measurement Virtual Instruments allow the user to measure the frequency or the pulse width of signal (one of the two external inputs or one of the eight outputs). The measurement may be synchronous or asynchronous.

**Synchronous:** after opening the device, the “IDT TH Synch Measure” VI may be called. This VI measures an input trigger frequency/pulse synchronously, so that the function exits when the measurement has been completed or a time out occurs.

**Asynchronous:** after opening the device, the “IDT TH Start Measure” VI may be called. The VI starts a measurement and returns immediately. The user may know when measurement has been completed by calling the “IDT TH Measure is Ready” VI. If the current measurement has been completed, the VI returns 1, otherwise 0. When the measurement is finished, the measured value may be read by calling the “IDT TH Measure Read” VI.

### 5.7.4. Error handling

The IDT LabVIEW interface uses the standard error cluster found in many LabVIEW VIs. The error cluster includes status, code and source parameters. When an error occurs, status is set to TRUE, source is set to the VI that caused the error, and code is set to one of the values shown in the table of Appendix D.

## 5.8. Examples VIs

### 5.8.1. 1\_enum\_devices

This VI shows how to display the result of a devices enumeration. The output of the “IDT TH Enum Devices” VI is displayed in a group of four LED and four edit boxes. If a device is enumerated the corresponding LED is turned on and the device ID is displayed in the edit box.

### 5.8.2. 2\_getinfo

This VI shows how to retrieve information from the device (device model, firmware version, etc.). The first available device is opened and the following information is retrieved and displayed: device model, firmware version and serial number.

### 5.8.3. 3\_ch0\_generator

This example shows how to generate a trigger signal to the output channel 0. The example opens the first available device, sets Internal mode (default), configures the device to output a 1 Hz signal (high level and low level durations are set to 25000000 clocks, and each clock is 20 ns), and then opens the output channel 0. The output channel 0 LED on the Timing Hub will blink at 1 Hz rate (50% duty cycle).

### 5.8.4. 4\_internal\_mode

This example shows how to use the timing box in Internal mode and set some of the parameters. You may configure the High Level Time, Low Level Time and Delay parameter of the selected channel in nanoseconds. Also, you may select a gate and decide to invert it and/or invert the output signal. When you have finished setting the parameters, click on the “Update Configuration” button to send the new configuration to the device and activate the new settings. If some error occurs, it will be show in the error status cluster.

### 5.8.5. 5\_external\_mode

This example shows how to use the timing box in External mode and set some of the parameters. You may configure the High Level Time, Low Level Time and Delay parameter of the selected channel in external time-base clocks.

If the external signal (set in Source Select control) is a 100 KHz signal connected to the External Input 0, the time-base clock unit is  $1/100000 = 10 \mu\text{s}$ . A value of 3 for the High Time corresponds to  $30 \mu\text{s}$  and a value of 7 for the Low Time corresponds to  $70 \mu\text{s}$ . The output signal will have a period of  $100 \mu\text{s}$  (e.g. 10,000 Hz), with a duty cycle of 30%.

Also, you may select a gate and decide to invert the source and/or the output signal. When you are finished setting the parameters, click on the “Update Configuration” button and send the new configuration to the device. If some error occurs, it will be show in the error status cluster.

### 5.8.6. 6\_startstop\_mode

This example shows how to use the timing box in Start/Stop mode and set some of the parameters. You may configure the High Time, Low Time and Delay parameter of the selected channel in nanoseconds. Also, you may select the start and stop trigger and decide to invert the output signal. When you have finished setting the parameters, press the “Update Configuration” button and send the new configuration to the device. If some error occurs, it will be show in the error status cluster.

### 5.8.7. 7\_rateswitch\_mode

This example shows how to use the timing box in Rate Switch mode and set some of the parameters. You may configure the High Time, Low Time and Delay parameter of the main and alternate waveform in nanoseconds. Also, you may select the start and stop (change rate) trigger and decide to invert the output signal. When you are finished setting the parameters, press the “Update Configuration” button and send the new configuration to the device. If some error occurs, it will be show in the error status cluster.

### 5.8.8. 8\_burst\_mode

This example shows how to use the timing box in Burst mode and set some of the parameters. You may configure the High Time, Low Time of the pulse in nanoseconds. You may set the number of pulses to generate and the initial delay before them. You may select the signal to use as a trigger and decide to invert the output signal or the trigger signal. When you are finished setting the parameters, press “Update Configuration” button to send the new configuration to the device. If some error occurs, it will be show in the error status cluster.

### 5.8.9. 9\_synch\_measure

This example shows how to execute a synchronous measurement. Connect to input 1 or 2 an external trigger signal, then configure the parameters in the “Measure Parameters” frame. Then run the example VI and see the result on the “Measured Value” frame box. The “Frequency” field in the measured value frame is valid only when ‘frequency’ is selected as “Measure Type”. Click on “Restart Measure” button to do another measure with new parameters. If some error occurs, it will be shown in the error status cluster. Press the “STOP VI” button to exit and reset the error.

### 5.8.10. 10\_asynch\_measure

This example shows how to execute an asynchronous measurement. Connect input 1 or 2 to an external trigger signal, and then configure the parameters in the “Measure Parameters” frame. Then run the example VI and see the result on the “Measured Value” frame box. The “Frequency” field in the measured value frame is valid only when ‘frequency’ is selected as “Measure Type”. Click on “Restart Measure” button to run another measure with new parameters. If some error occurs, it will be shown in the error status cluster. Press the “STOP VI” button to exit and reset the error.



## 6. MotionPro Timing Hub MATLAB™ Interface Reference

### 6.1. Overview

The MATLAB™ Interface allows the user to operate the Timing Hub from inside the Mathworks™ MATLAB application. The interface works with MATLAB 6.5 and greater, on Windows 2000/XP Professional. Windows NT is not supported.

The Timing Hub MATLAB™ Interface includes the 'MEX' file for controlling the device (packaged in a library called TimHubML.dll) and some example .m files to show how to use the interface.

Every routine may be called from a MATLAB™ script file in the form:

**[output1, output2 ...] = TimHubML [input1, input2 ...]**

The number of inputs and outputs depends on the function selected. In any function call input1 is the name of the requested command (for ex. 'IdtThEnumDevices') and output1 is the result of the operation (0 = SUCCESS, otherwise ERROR).

More details on the commands syntax may be retrieved by typing "help TimHubML" at MATLAB command prompt or opening the file **TimHubML.m** with a text editor.

The MATLAB interface reflects the SDK Application Program Interface (see Timing Hub SDK reference section) with a few exceptions. The MATLAB interface and examples are listed below.

## 6.2. Initialization Functions

### 6.2.1. Overview: Initialization functions

Initialization functions allow the user to initialize the Timing Hub device, enumerate the available devices, open and close them.

**IdtThGetVersion** retrieves the driver version.

**IdtThEnumDevices** enumerates the IDs of the Timing Hub devices connected to the computer.

**IdtThOpenDevice** opens an Timing Hub device.

**IdtThCloseDevice** closes an Timing Hub device previously open.

## 6.2.2. IdtThGetVersion

`[strVersion] = TimHubML ('IdtThGetVersion')`

### Inputs

None

### Outputs

*strVersion*

Specifies the driver version string (for example, '2.03')

### Remarks

This function must be called to retrieve the Timing Hub MATLAB interface version string.

### See also:

### 6.2.3. IdtThEnumDevices

[*nResult*, *nItems*, *thArray*] = TimHubML ('**IdtThEnumDevices**')

#### Inputs

None

#### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*nItems*

Specifies the number of detected devices

*thArray*

Specifies the array containing the IDs of the detected devices

#### Remarks

The routine enumerates the active devices and returns an array filled with the detected devices IDs. This routine must be called before **IdtThOpenDevice** to find out which devices are available. The *nItems* variable contains the number of detected devices. If any error occurs during the devices enumeration, the *nResult* variable contains an error code.

**See also:** IdtThOpenDevice

## 6.2.4. IdtThOpenDevice

**[*nResult*, *nDeviceId*] = TimHubML ('IdtThOpenDevice', *nInputId*)**

### Inputs

*nInputId*

Specifies the ID of the device to be opened, or 0 for the first available device

### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*nDeviceId*

Specifies the ID of the opened device

### Remarks

The routine opens the device whose ID is in the variable *nInputId*. The value can be retrieved by calling the **IdtThEnumDevices** enumeration function. The user may supply a specific device ID or 0: in this case the first available device is opened. If any error occurs during the device opening, the routine returns an error code in the *nResult* variable, otherwise it returns 0. The function also returns the device Id.

**See also:** IdtThCloseDevice

### 6.2.5. IdtThCloseDevice

**[nResult]** = TimHubML ('**IdtThCloseDevice**', *nDeviceId*)

#### **Inputs**

*nDeviceId*

Specifies the ID of the device to be closed

#### **Outputs**

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

#### **Remarks**

This function closes a device previously open. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0.

**See also:** IdtThOpenDevice

## 6.3. Configuration functions

### 6.3.1. Overview: Configuration functions

Configuration functions allow the user to read information from the device, read configuration parameters from the device and write them to the device.

**IdtThGetDeviceInfo** reads information from the device, such as device model, firmware version, etc.

**IdtThGetParameter** reads a single specific parameter from the configuration and gets its minimum and maximum value.

**IdtThSetParameter** writes a single specific parameter to the configuration.

**IdtThSendCfg** downloads the updated configuration to the device and activates it.

### 6.3.2. IdtThGetDeviceInfo

**[*nResult*, *nInfoValue*] = TimHubML ('IdtThGetDeviceInfo', *nDeviceId*, *nInfoKey*)**

#### **Inputs**

*nDeviceId*

Specifies a valid device ID

*nInfoKey*

Specifies which parameter the function has to return

#### **Outputs**

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*nInfoValue*

Specifies the value of the info parameter

#### **Remarks**

This function returns device specific information, such as device type or version numbers, generally state-independent information. See the Appendix B for a list of all the available *nInfoKey* values.

**See also:** IdtThGetParameter



### 6.3.3. IdtThGetParameter

*[nResult, nValue, nMinValue, nMaxValue]* = TimHubML ('**IdtThGetParameter**',  
*nDeviceId, nParamKey*)

#### Inputs

*nDeviceId*

Specifies a valid device ID

*nParamKey*

Specifies the index of the parameter

#### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*nValue*

Specifies the current value of the parameter

*nMinValue*

Specifies the minimum value of the parameter

*nMaxValue*

Specifies the maximum value of the parameter

#### Remarks

This function reads a specific parameter from the current configuration and returns its value, the minimum and the maximum. The parameter key is one of the input parameters. A list of the parameters constants is available in Appendix C. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0. The user may call the **IdtThSetParameter** function several times to set different parameters, and then call the **IdtThSendCfg** to download the configuration to the device.

**See also:** `IdtThSetParameter`, `IdtThSendCfg`

### 6.3.4. IdtThSendCfg

**[nResult]** = TimHubML ('**IdtThSendCfg**', *nDeviceId*)

#### Inputs

*nDeviceId*

Specifies a valid device ID

#### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

#### Remarks

This function downloads the configuration to the device and activates it. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0. The user may call the **IdtThSetParameter** function several times to set different parameters, and then call the **IdtThSendCfg** to download the configuration to the device.

**See also:** IdtThGetParameter, IdtThSetParameter

## 6.4. Outputs Enable/Disable Functions

### 6.4.1. Overview: Outputs enable/disable Functions

These functions allow the user to open or close one or more output channels.

**IdtThOpenOutputs** turns on one or more output channels.

**IdtThCloseOutputs** turns off/resets one or more output channels.

## 6.4.2. IdtThOpenOutputs

**[*nResult*] = TimHubML ('IdtThOpenOutputs', *nDeviceId*, *nOutputMask*)**

### Inputs

*nDeviceId*

Specifies the ID of the device to be enabled

*nOutputMask*

Specifies the mask of the output channels to open

### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

### Remarks

The routine opens one or more output channel according to the *nOutputMask* parameter. In the output mask parameter, each bit controls a single output channel (BIT0 for channel 0, BIT1 for channel 1, etc.). If other output channels have been opened before, the routine does not affect their state. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0.

**See also:** IdtThCloseOutputs

### 6.4.3. IdtThCloseOutputs

**[*nResult*]** = TimHubML ('**IdtThCloseOutputs**', *nDeviceId*, *nOutputMask*)

#### Inputs

*nDeviceId*

Specifies the ID of the device to be disabled

*nOutputMask*

Specifies the mask of the output channels to close

#### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

#### Remarks

The routine closes/resets one or more output channel according to the *nOutputMask* parameter. In the output mask parameter, each bit controls a single output channel (BIT0 for channel 0, BIT1 for channel 1, etc.). If other output channels have been opened before, the routine does not affect their state. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0.

**See also:** IdtThOpenOutputs

## 6.5. Measurement Functions

### 6.5.1. Overview: Measurement Functions

Measurement functions allow the user to measure the frequency or the pulse width of a signal (one of the two external inputs or one of the eight outputs).

**IdtThSynchMeasure** measures frequency or pulse width synchronously.

**IdtThStartMeasure** starts an asynchronous measure of frequency or pulse width.

**IdtThAbortMeasure** aborts the current asynchronous measurement.

**IdtThReadMeasure** reads the value of the current asynchronous measured data.

**IdtThMeasureIsReady** signals if the current asynchronous measure has been completed.

## 6.5.2. IdtThSynchMeasure

**[*nResult*, *nMeasuredData*] = TimHubML ('IdtThSynchMeasure', *nDeviceId*, *nMeasureInput*, *nMeasureType*, *nInvertInput*, *nTimeOut*)**

### Inputs

*nDeviceId*

Specifies a valid device ID

*nMeasureInput*

Specifies the input channel to measure

*nMeasureType*

Specifies the type of value to measure

*nInvertInput*

Specifies if the input signal has to be inverted before the measurement

*nTimeOut*

Specifies the measure time out in ms

### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*nMeasuredData*

Specifies the value of the measured data

### Remarks

This function measures a signal frequency or pulse width synchronously. The input may be one of the two external inputs or one of the eight outputs. The measurement is synchronous and the function exits when the measurement has been completed or a time out occurs.

### See also:

### 6.5.3. IdtThStartMeasure

**[nResult]** = TimHubML (**'IdtThStartMeasure'**, *nDeviceId*, *nMeasureInput*, *nMeasureType*, *nInvertInput*)

#### Inputs

*nDeviceId*

Specifies a valid device ID

*nMeasureInput*

Specifies the input channel to measure (0=Input1, 1=Input2)

*nMeasureType*

Specifies the type of value to measure (0=Frequency, 1=Pulse Width)

*nInvertInput*

Specifies if the input signal has to be inverted before the measurement

#### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

#### Remarks

This function starts a measurement and returns immediately. The user may know when measurement has been completed by calling the "**IdtThMeasureIsReady**" function.

**See also:** IdtThAbortMeasure, IdtThReadMeasure, IdtThMeasureIsReady



### 6.5.4. IdtThAbortMeasure

**[*nResult*]** = TimHubML ('**IdtThAbortMeasure**', *nDeviceId*)

#### **Inputs**

*nDeviceId*

Specifies a valid device ID

#### **Outputs**

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

#### **Remarks**

This function stops a previously started asynchronous measurement.

**See also:** IdtThStartMeasure, IdtThReadMeasure, IdtThMeasureIsReady

### 6.5.5. IdtThReadMeasure

[*nResult*, *nMeasuredData*] = TimHubML ('**IdtThReadMeasure**', *nDeviceId*)

#### Inputs

*nDeviceId*

Specifies a valid device ID

#### Outputs

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*nMeasuredData*

Specifies the value of the measured data

#### Remarks

This function reads the value of the current asynchronous measured data.

**See also:** IdtThStartMeasure, IdtThAbortMeasure, IdtThMeasureIsReady

### 6.5.6. IdtThMeasureIsReady

**[*nResult*, *nIsReady*] = TimHubML ('IdtThMeasureIsReady', *nDeviceId*)**

#### **Inputs**

*nDeviceId*

Specifies a valid device ID

#### **Outputs**

*nResult*

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

*nIsReady*

Specifies whether the measurement is finished (1) or not (0).

#### **Remarks**

This function returns the status of the current acquisition. If the returned value *nIsReady* is 1 the current measurement has been completed, otherwise not.

**See also:** IdtThStartMeasure, IdtThAbortMeasure, IdtThReadMeasure

## 6.6. How to use the Interface functions

### 6.6.1. Opening and closing a device

A device must be opened before using its functions and then it must be closed. To open a specific device you have to supply to the “IdtThOpenDevice” function the unique ID of that device. You may also supply 0 to open the first available device. To obtain the list of all available devices you may call the “IdtThEnumDevices” function.

### 6.6.2. Configuring a device

Before configuring a device, several calls to the “IdtThSetParameter” function may be done. When the parameters have been set, a call to the “IdtThSendCfg” function downloads the new configuration activates it. If you want to read a parameter value you may call the “IdtThGetParameter” function.

### 6.6.3. Measurement

Measurement functions allow the user to measure the frequency or the pulse width of signal (one of the two external inputs or one of the eight outputs). The measurement may be synchronous or asynchronous.

**Synchronous:** after opening the device, the “IdtThSynchMeasure” function may be called. This function measures an input signal frequency or pulse width synchronously, and exits when the measurement has been completed or a time out occurs.

**Asynchronous:** after opening the device, the “IdtThStartMeasure” function may be called. The VI starts a measurement and returns immediately. The user may know when measurement has been completed by calling the “IdtThMeasureIsReady” function. If the current measurement has been completed, the function returns 1, otherwise 0. When the measurement is finished, the measured value may be read by calling the “IdtThReadMeasure” function.

### 6.6.4. Error handling

The Timing Hub MATLAB interface returns the same error codes displayed in the Appendix D.

## 6.7. Examples

### 6.7.1. IdtThEnumEx

This example shows how to obtain the list of all available devices.

### 6.7.2. IdtThInfoEx

This example shows how to obtain some information from a device.

### 6.7.3. IdtThReadParmEx

This example shows how to read specific parameter from a device.

### 6.7.4. IdtThSynchMeasEx

This example shows how to execute a synchronous measurement of a signal.

### 6.7.5. IdtThAsynchMeasEx

This example shows how to execute an asynchronous measurement of an external input.

### 6.7.6. IdtThWaveGenEx

This example shows how to generate an output signal. The example opens the first available device, configures the output channel 0 to generate a 1 Hz signal with a duty cycle of 50%. The high level time and low level time of channel 0 are set to 50 ms. The output 0 LED on the Timing Hub blinks at 1 Hz rate for 10 seconds.

### 6.7.7. IdtThWaveDelayEx

This example shows how to generate two signals: output channel 0 generates a 1 Hz signal with 50% duty cycle, while output channel 1 generates a signal with same rate and duty cycle and a delay of 500 ms. The output LED blink in an alternate way.

### 6.7.8. IdtThInternalEx

This example shows how to use the timing box in Internal mode and set some of the parameters. You may configure the High Level Time, Low Level Time and Delay parameter of the selected channel in nanoseconds. Also, you may select a gate and decide to invert it and/or invert the output signal. When you have finished setting the parameters, click on the "Send Configuration Settings to Device" button to send the new configuration to the device and activate the new settings.

### 6.7.9. IdtThExternalEx

This example shows how to use the timing box in External mode and set some of the parameters. You may configure the High Level Time, Low Level Time and Delay parameter of the selected channel in external time-base clocks.

If the external signal (set in Source Select control) is a 100 KHz signal connected to the External Input 0, the time-base clock unit is  $1/100000 = 10 \mu\text{s}$ . A value of 3 for the High Time corresponds to  $30 \mu\text{s}$  and a value of 7 for the Low Time corresponds to  $70 \mu\text{s}$ . The output signal will have a period of  $100 \mu\text{s}$  (e.g. 10,000 Hz), with a duty cycle of 30%.

Also, you may select a gate and decide to invert the source and/or the output signal. When you are finished setting the parameters, click on the “Send Configuration Settings to Device” button and send the new configuration to the device.

### 6.7.10. IdtThStartStopEx

This example shows how to use the timing box in Start/Stop mode and set some of the parameters. You may configure the High Time, Low Time and Delay parameter of the selected channel in nanoseconds. Also, you may select the start and stop trigger and decide to invert the output signal. When you have finished setting the parameters, press the “Send Configuration Settings to Device” button and send the new configuration to the device.

### 6.7.11. IdtThRateSwitchEx

This example shows how to use the timing box in Rate Switch mode and several of its related parameters. You may configure the High Time, Low Time and Delay parameter of the main and alternate waveform in nanoseconds. Also, you can select the start and stop (change rate) trigger and decide to invert the output channel output. When you have finished setting the parameters, click on the “Send Configuration Settings to Device” button to send the new configuration to the device and activate the new settings.

### 6.7.12. IdtThBurstEx

This example shows how to use the timing box in Burst mode and set some of the parameters. You may configure the High Time, Low Time of the pulse in nanoseconds. You may set the number of pulses to generate and the initial delay before them. You may select the signal to use as a trigger and decide to invert the output signal or the trigger signal. When you are finished setting the parameters, press “Send Configuration Settings to Device” button to send the new configuration to the device.

### 6.7.13. IdtThMeasureEx

This example shows how to execute a measure of a signal. You may select the source signal to measure and the type of measure (frequency or pulse width). When you have finished setting the parameters, click on the “Measure External Input” button to send the new configuration to the device and run the new measurement.

## 7. Appendix

### 7.1. Appendix A - Return Codes

The following table shows the values of the codes returned by the Timing Hub APIs. The values can be found in the **TimHubAPI.h** header file in the **Include** subdirectory.

Code	Value	Notes
TH_SUCCESS	0	OK – No errors
TH_E_GENERIC_ERROR	1	Generic Error
TH_E_NOT_SUPPORTED	2	The function is not supported for this device
TH_E_INVALID_VALUE	3	Invalid parameter value
TH_E_INVALID_HANDLE	4	Invalid TH_SETTINGS structure
TH_E_INVALID_HANDLE	5	Invalid TH_HANDLE handle
TH_E_INVALID_DEV_ID	6	Invalid device id used in ThOpenDevice. The ID is retrieved calling the ThEnumDevices routine
TH_E_INVALID_ARGUMENTS	7	Invalid function arguments
TH_E_READONLY	8	The parameter is read-only and cannot be modified
TH_E_DEV_ALREADY_OPEN	9	The device is already open.
TH_E_HARDWARE_FAULT	10	Hardware error. To retrieve the hardware error code call the ThGetHardwareError routine.
TH_E_BUSY	11	The device is busy and the operation cannot be performed
TH_E_TIMEOUT	12	Operation time out.

## 7.2. Appendix B – Information Parameters

The following table shows the values and a brief description of the parameters that can be read calling the ThGetDeviceInfo routine. The numeric values of the parameters can be found in the **TimHubAPI.h** header file in the **Include** subdirectory.

Parameter	Description
THI_DEVICE_MODEL	Device Model ( see TH_DEV_MODEL)
THI_DEVICE_ID	Device ID (see TH_ENUMITEM structure)
THI_FW_VERSION	Firmware version
THI_SERIAL	The device serial number (10 decimal digits value)
THI_REVISION	The Timing Hub hardware revision (A, B, C, D, etc.)
THI_LINK_VER	The Link (USB) version: it may be 20 (USB 2.0) or 11 (USB 1.1)



### 7.3. Appendix C – Device Settings

The following table shows the values and a brief description of the parameters that can be read and written in the device. The numeric values of the parameters can be found in the **TimHubAPI.h** header file in the **Include** subdirectory.

Parameter	R/W	Description
THP_OP_MODE	R/W	Operational mode [internal, external, start/stop, rate switch, burst]
THP_DELAY	R/W	Main output initial delay [time-base steps]
THP_HIGH	R/W	Main output high state [time-base steps]
THP_LOW	R/W	Main output low state [time-base steps]
THP_OUTPUT_INV	R/W	Invert output. See TH_INVERT.
THP_DELAY_STATE	R/W	Initial delay and offset state. See TH_STATE.
THP_GATE_TRG	R/W	Gate/Trigger select. See TH_INPUT_CHANNEL.
THP_GATE_TRG_INV	R/W	Invert gate/trigger. See TH_INVERT.
THP_GATE_TRG_2	R/W	Second Gate/Trigger select. See TH_INPUT_CHANNEL.
THP_GATE_TRG_INV_2	R/W	Invert second Gate/Trigger. See TH_INVERT.
THP_DELAY_2	R/W	Alternate output initial delay [time-base steps]
THP_HIGH_2	R/W	Alternate output high state [time-base steps]
THP_LOW_2	R/W	Alternate output low state [time-base steps]
THP_SOURCE	R/W	Source select. See TH_INPUT_CHANNEL.
THP_SOURCE_INV	R/W	Invert source. See TH_INVERT.
THP_PULSES_COUNT	R/W	Number of pulses to generate (1 to 2 <sup>16</sup> -1)
THP_PULSES_REPEAT	R/W	Generate pulses only once (0) or every next trigger edge (1)

## 7.4. Appendix D – LabVIEW / MATLAB Error Codes

This appendix describes the error codes used in the LabVIEW error cluster and in the MATLAB interface.

Error Code	Description
1	Driver Fault
2	Device not found
3	Invalid parameter value
4	Default device parameters can not be loaded
5	The device could not be closed
6	A parameter value could not be read
7	A parameter value could not be written
8	A info parameter could not be read
9	Failed to enable/disable device or channel output
10	Failed to measure
11	Failed to abort measure
12	Measurement Timeout
13	Error reading hardware register
100	Generic error

## 7.5. Appendix E – Data types

This appendix describes the data types defined in the **TimHubAPI.h** header file.

### 7.5.1. TH\_DEV\_MODEL

The TH\_DEV\_MODEL type enumerates the device models.

- **TH\_DM\_UNKNOWN**: Unknown device model
- **TH\_DM\_USB\_1**: MotionPro Timing Hub Model 1 (Hi-speed USB link).

### 7.5.2. TH\_REVISION

The TH\_REVISION type enumerates the devices revision numbers.

- **TH\_REV\_A**: revision A (original).
- **TH\_REV\_B, C, D**: revision B, C, D, etc.

### 7.5.3. TH\_OP\_MODE

The TH\_OP\_MODE enumerates the device operational modes:

- **TH\_OP\_INTERNAL**: internal mode.
- **TH\_OP\_EXTERNAL**: external mode.
- **TH\_OP\_START\_STOP**: start/stop mode.
- **TH\_OP\_RATE\_SWITCH**: rate switch mode.
- **TH\_OP\_BURST**: synchronous burst mode.

### 7.5.4. TH\_OUTPUT\_CHANNEL

The TH\_OUTPUT\_CHANNEL enumerates the output channels:

- **TH\_OUTPUT\_CHN\_0**: output channel 0.
- **TH\_OUTPUT\_CHN\_1**: output channel 1.
- **TH\_OUTPUT\_CHN\_2**: output channel 2.
- **TH\_OUTPUT\_CHN\_3**: output channel 3.
- **TH\_OUTPUT\_CHN\_4**: output channel 4.
- **TH\_OUTPUT\_CHN\_5**: output channel 5.
- **TH\_OUTPUT\_CHN\_6**: output channel 6.
- **TH\_OUTPUT\_CHN\_7**: output channel 7.
- **TH\_OUTPUT\_CHN\_ALL**: all the output channels (only for SetParameter).

### 7.5.5. TH\_INPUT\_CHANNEL

The TH\_OUTPUT\_CHANNEL enumerates the input channels:

- **TH\_INPUT\_NONE**: no input.
- **TH\_INPUT\_EXT\_0**: select external input 0.
- **TH\_INPUT\_EXT\_1**: select external input 1.
- **TH\_INPUT\_CHN\_0**: select output channel 0 as input.
- **TH\_INPUT\_CHN\_1**: select output channel 1 as input.
- **TH\_INPUT\_CHN\_2**: select output channel 2 as input.
- **TH\_INPUT\_CHN\_3**: select output channel 3 as input.
- **TH\_INPUT\_CHN\_4**: select output channel 4 as input.
- **TH\_INPUT\_CHN\_5**: select output channel 5 as input.
- **TH\_INPUT\_CHN\_6**: select output channel 6 as input.
- **TH\_INPUT\_CHN\_7**: select output channel 7 as input.

### 7.5.6. TH\_OUTPUT\_MASK

The TH\_OUTPUT\_MASK enumerates the output channels mask used for open/close the channels:

- **TH\_OUTPUT\_MSK\_0**: output channel 0.
- **TH\_OUTPUT\_MSK\_1**: output channel 1.
- **TH\_OUTPUT\_MSK\_2**: output channel 2.
- **TH\_OUTPUT\_MSK\_3**: output channel 3.
- **TH\_OUTPUT\_MSK\_4**: output channel 4.
- **TH\_OUTPUT\_MSK\_5**: output channel 5.
- **TH\_OUTPUT\_MSK\_6**: output channel 6.
- **TH\_OUTPUT\_MSK\_7**: output channel 7.
- **TH\_OUTPUT\_MSK\_ALL**: all the output channels.

### 7.5.7. TH\_INPUT\_INVERT

The TH\_INPUT\_INVERT enumerates the external input inversion:

- **TH\_INVERT\_NO**: do not invert the input.
- **TH\_INVERT\_YES**: invert the input.

### 7.5.8. TH\_INPUT\_INVERT

The TH\_REPEAT enumerates the pulse repetition:

- **TH\_NO**: do not repeat the pulse after the first.
- **TH\_YES**: repeat the pulse after the first.

### 7.5.9. TH\_STATE

The TH\_STATE enumerates the levels (used for delay start level).

- **TH\_LOW**: level is low.
- **TH\_HIGH**: level is high.

### 7.5.10. TH\_MEASURE\_TYPE

The TH\_MEASURE\_TYPE enumerates the measure type.

- **TH\_MT\_FREQUENCY**: measure the frequency of the input channel.
- **TH\_MT\_PULSE\_WID**: measure the pulse width of the input channel.

### 7.5.11. TH\_CALLBACK\_FLAGS

The TH\_CALLBACK\_FLAGS enumerates the Queue callback flags:

- **TH\_CF\_DONE**: callback is called only when the operation is completed.
- **TH\_CF\_FAIL**: callback is called only when the operation fails.

### 7.5.12. TH\_ATTRIBUTE

The TH\_ATTRIBUTE enumerates the attribute type:

- **TH\_ATTR\_MIN**: the minimum value.
- **TH\_ATTR\_MAX**: the maximum value.
- **TH\_ATTR\_DEFAULT**: the default value.
- **TH\_ATTR\_READONLY**: the read-only flag

### 7.5.13. TH\_ERROR

The TH\_ERROR enumerates the return codes. See Appendix A.

### 7.5.14. TH\_INFO

The TH\_INFO enumerates the device information index. See Appendix B.

### 7.5.15. TH\_PARAM

The TH\_PARAM enumerates the device parameters. See Appendix C.

## 7.6. Appendix F – Structures

This appendix describes the structures defined in the **TimHubAPI.h** header file.

### 7.6.1. TH\_ENUMITEM

The TH\_ENUMITEM structure contains information about a device. It must be used in the device enumeration procedure with the ThEnumDevices routine.

```
typedef struct
{
    unsigned long cbSize;
    unsigned long nDeviceModel;
    unsigned long nDeviceId;
    unsigned long nSerial;
    unsigned long nRevision;
    unsigned long bIsOpen;
    unsigned long nLinkVer;
} SV_ENUMITEM, *PSV_ENUMITEM;
```

#### Members

##### *cbSize*

It specifies the size of the structure.

##### *nDeviceModel*

It specifies the device model.

##### *nDeviceId*

It specifies the ID which identifies a device among others. The user must use this id to open the device with ThOpenDevice.

##### *nSerial*

it specifies the device serial number (10 decimal digits value).

##### *nRevision*

it specifies the device hardware revision number (A, B, C, etc.).

##### *bIsOpen*

It specifies whether the device is currently open or not.

##### *nLinkVer*

It specifies the link (USB) version. It may be 20 (USB 2.0) or 11 (USB 1.1)

## 7.6.2. TH\_SETTINGS

The TH\_SETTINGS structure is an opaque structure that contains all the device parameters in compact format. The user may access the structure using the ThSetParameter and ThGetParameter routines.

```
typedef struct
{
    unsigned long cbSize;
    unsigned long nData[ 256 ];
} TH_SETTINGS, *PTH_SETTINGS;
```

### Members

#### *cbSize*

It specifies the size of the structure. Must be set to sizeof (TH\_SETTINGS), otherwise the related functions don't work.

#### *nData*

It specifies the opaque structure data, an array of 256 unsigned long values.



### 7.6.3. TH\_AsyncCallback

The TH\_AsyncCallback is the prototype of the callback function passed to the ThStartMeasure routine. The callback is called by the driver when the operation is completed.

```
typedef void (TIMHUBAPI *TH_AsyncCallback)
(
    unsigned long nUserData,
    TH_ERROR      nErrCode,
    unsigned long nFlags
);
```

#### Members

##### *nUserData*

Specifies a user defined data value, passed by the user.

##### *nErrCode*

It specifies the operation return code.

##### *nFlags*

It specifies a combination of the TH\_CALLBACK\_FLAGS values.