

Galileo Wireless SDK
Reference Manual
Software Development Kit

Software Release

1.00

Document Revision

March 2009

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

Europe

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

Table of Contents

1. OVERVIEW	6
1.1. GALILEO COMPONENTS	7
1.1.1. Coordinator (USB device)	7
1.1.2. Sync Device (Camera)	8
1.1.3. Sync Device (Light)	9
1.1.4. Trigger	10
1.1.5. Range Extender	11
1.1.6. Galileo Network	12
1.2. NOTE ABOUT THE WORD “TRIGGER”	13
1.3. DIRECTORIES STRUCTURE	14
1.4. REDISTRIBUTABLE FILES	15
1.5. GALILEO TEST APPLICATION	16
2. USING THE GALILEO™ SDK	17
2.1. OVERVIEW	17
2.2. PROGRAMMING LANGUAGES	18
2.3. LOAD/UNLOAD THE DRIVER	19
2.4. ENUMERATE/OPEN A DEVICE	20
2.5. PARAMETERS	22
2.6. COMMANDS	23
3. GALILEO SDK REFERENCE	24
3.1. INITIALIZATION FUNCTIONS	24
3.1.1. Overview: Initialization functions	24
3.1.2. WLGetVersion	25
3.1.3. WLLoadDriver	26
3.1.4. WLUnloadDriver	27
3.1.5. WLGetCoordinatorInfo	28
3.1.6. WLEnumDevices	29
3.1.7. WLOpenDevice	30
3.1.8. WLCloseDevice	31
3.2. INFO AND CONFIGURATION FUNCTIONS	32
3.2.1. Overview: Info and Configuration functions	32
3.2.2. WLGetDeviceInfo	33
3.2.3. WLGetParameterAttribute	34
3.2.4. WLGetParameter	35
3.2.5. WLSetParameter	36
3.3. DEVICE CONTROL FUNCTIONS	37
3.3.1. Overview: Device Control functions	37
3.3.2. WLSendCommand	38
3.3.3. WLReadData	39
3.3.4. WLGetStatus	40
4. GALILEO ACTIVEX CONTROL	41
4.1. OVERVIEW	41
4.1.1. How to add the control to a Visual Studio project	41
5. GALILEO LABVIEW™ REFERENCE	42
5.1. OVERVIEW	42

5.2.	INITIALIZATION VIS	43
5.2.1.	Overview: Initialization VIs	43
5.2.2.	Enum Devices	44
5.2.3.	Open Device	45
5.2.4.	Close Device	46
5.3.	INFO AND CONFIGURATION VIS	47
5.3.1.	Overview: Configuration VIs	47
5.3.2.	Get Info	48
5.3.3.	Get Parameter	49
5.3.4.	Set Parameter	50
5.3.5.	Get Parameter Attribute	51
5.4.	DEVICE CONTROL VIS	52
5.4.1.	Overview: Device Control functions	52
5.4.2.	Send Command	53
5.4.3.	Read Trigger Delay	54
5.4.4.	Get Status	55
6.	GALILEO MATLAB™ REFERENCE	56
6.1.	OVERVIEW	56
6.2.	INITIALIZATION FUNCTIONS	57
6.2.1.	Overview: Initialization functions	57
6.2.2.	Version	58
6.2.3.	EnumDevices	59
6.2.4.	OpenDevice	60
6.2.5.	CloseDevice	61
6.3.	INFO AND CONFIGURATION FUNCTIONS	62
6.3.1.	Overview: Info and Configuration functions	62
6.3.2.	GetDeviceInfo	63
6.3.3.	GetParameterAttribute	64
6.3.4.	GetParameter	65
6.3.5.	SetParameter	66
6.4.	DEVICE CONTROL FUNCTIONS	67
6.4.1.	Overview: Device Control functions	67
6.4.2.	SendCommand	68
6.4.3.	ReadTriggerDelay	69
6.4.4.	GetStatus	70
7.	APPENDIX	71
7.1.	APPENDIX A - RETURN CODES	71
7.2.	APPENDIX B – INFO	72
7.3.	APPENDIX C – PARAMETERS	73
7.4.	APPENDIX D – COMMANDS	74
7.5.	APPENDIX E – DATA TYPES	75
7.5.1.	WL_DEV_MODEL	75
7.5.2.	WL_REVISION	75
7.5.3.	WL_GAIN	75
7.5.4.	WL_CMD_OP	76
7.5.5.	WL_STATUS	76
7.5.6.	WL_ERROR	76
7.5.7.	WL_INFO	76
7.5.8.	WL_PARAM	76
7.5.9.	WL_COMMAND	76
7.6.	APPENDIX F – STRUCTURES	77

7.6.1. WL_ENUMITEM77

1. Overview

The on-line documentation of the Galileo Software Development Kit (SDK) and its components is divided into the following parts:

Using the SDK

This section shows how to use the SDK.

SDK Reference

This section contains a detailed description of the SDK functions.

ActiveX Reference

This section contains a detailed description of the ActiveX control interface.

LabVIEW™ Interface Reference

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

MATLAB™ Interface Reference

This section contains a detailed description of the camera MATLAB™ Drivers.

Appendix

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

1.1. Galileo components

The Galileo Wireless System is made of different components.

1.1.1. Coordinator (USB device)

The USB key forms the root of the ZigBee network and allows the communication between the Synchronization devices and the Computer.



1.1.2. Sync Device (Camera)

The camera synchronization device generates the signals for the synchronization of the camera (external frequency and trigger pulse). The light sensor controls the duty cycle of the square wave and reacts to any change of the environmental light.

The STP camera device supports STP synchronization signals but does not have the light sensor.



1.1.3. Sync Device (Light)

The light synchronization device generates the signals for the synchronization of the light sources (external frequency). The trigger output is usually not used.



1.1.4. Trigger

The trigger device generates the trigger pulse that is sent to the synchronization devices and computes the delay between the trigger detection and the pulse issued to the cameras.

The trigger pulse can be generated with a “switch” connected to the Control LEMO connector with a special cable.



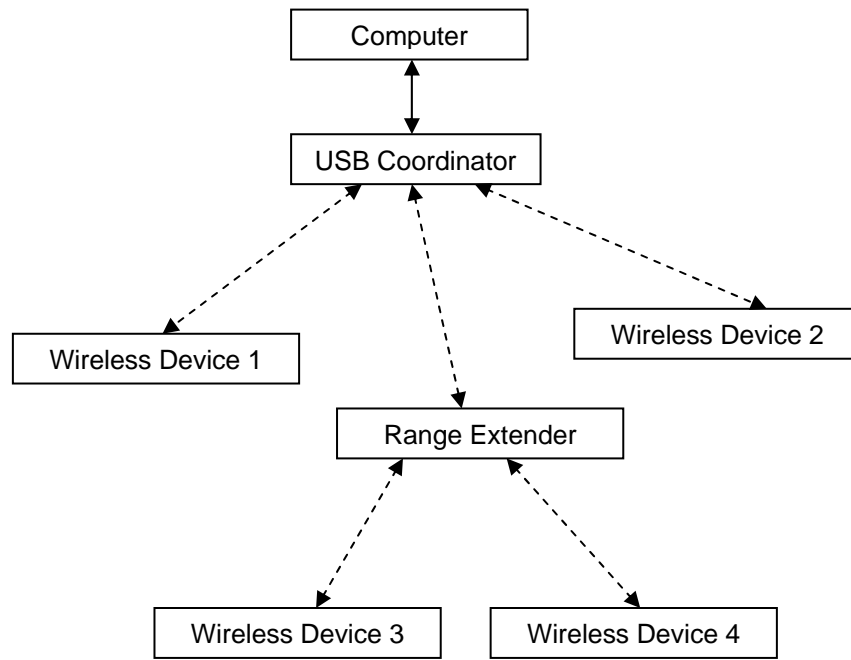
1.1.5. Range Extender

Those devices extend the ZigBee network. This device does not require a GPS antenna.



1.1.6. Galileo Network

The image below shows an example of a Galileo Wireless Network



1.2. Note about the word “Trigger”

In this manual the word “trigger” refers to an external event (usually a single pulse) that ends a camera acquisition in a “circular” or “round-robin” array of buffers. The external square wave source of synchronization is referred as “Sync In”.

1.3. Directories structure

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

BIN32: it contains the 32-bit files (DLLs, INF and drivers) that can be re-distributed with the devices and your application.

BIN64: it contains the 64-bit files (DLLs, INF and drivers) that can be re-distributed with the devices and your application.

DOCS: it contains the SDK documentation.

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

LIB: it contains the SDK lib files.

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

1.4. Redistributable Files

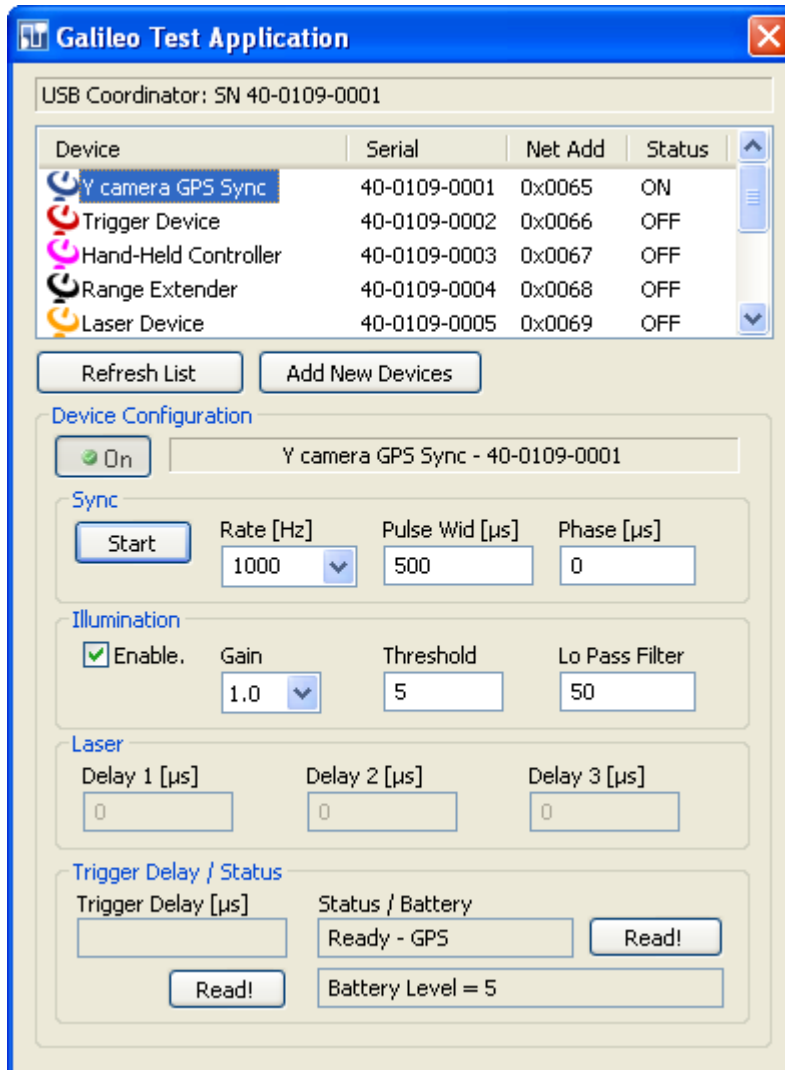
This section outlines the options available to third-party vendors for distributing Galileo drivers for Windows 2000/XP/Vista. The files that can be redistributed are in the BIN32/BIN64 subdirectories of the installation folder (C:\Program Files\IDT\Galileo).

File	Description
WlDevDrv32.dll	SDK main interface driver
SiUSBXP.dll	Support DLL for the USB To ZigBee bridge
IDTusb2ZP.inf	INF file for installation of the Bridge
SiLib.sys, SiUSBXP.sys	sys drivers 32/64-bit

1.5. Galileo Test Application

The Galileo Test Application lists the devices and gives the user a set of configuration parameters to edit.

The user may open the device, change rate, pulse width, as well as illumination parameters. The commands can be sent and the status can be read, as well as the trigger delay.

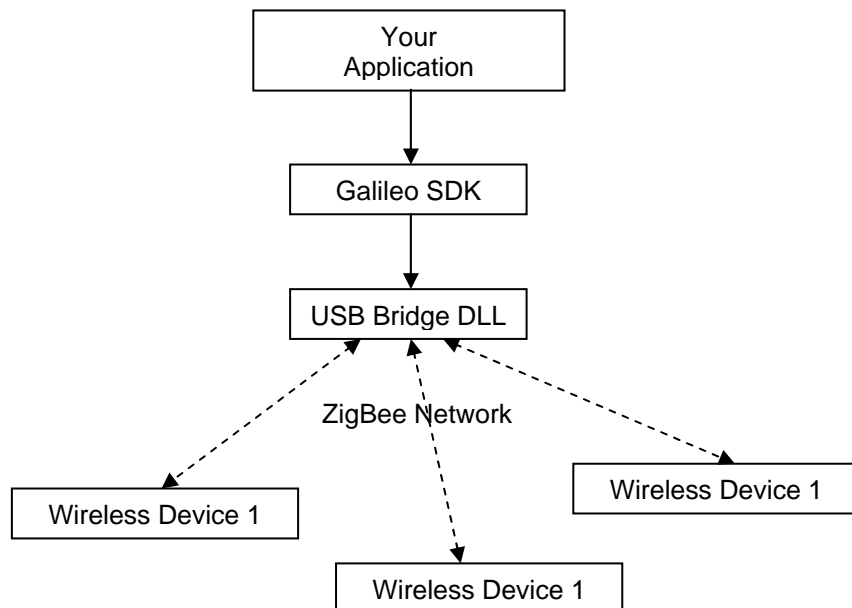


2. Using the Galileo™ SDK

2.1. Overview

The Galileo SDK provides a consistent programmatic interface for the control of the wireless devices. The picture below shows the different layers involved in the communication.

The wireless devices communicate to the USB Bridge in a ZigBee network. ZigBee is a low-power wireless network standard based on the IEEE 802.15.4-2006 standard.



2.2. Programming Languages

A C/C++ header file is included in the SDK (**WLDevAPI.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.

The Windows driver is a DLL (WLDevDrv32.dll) that resides in the system32 directory. It may be found also in the Bin32 sub-directory.

MS Visual C++™: A Visual C++ 6.0 stub COFF library is provided (**WLDevDrv32.lib** in the Lib sub-directory); if you are using Visual C++, link to WLDevDrv32.lib. The DLL uses Windows standard calling conventions (`_stdcall`).

Borland C++ Builder™: the WLDevDrv32.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 WLDevDrv32.lib WLDevDrv32.dll".

MS Visual C#: the **WLDevAPI.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 "**WLDevice**" class members.

MS Visual Basic: a Visual Basic module is included in the SDK (WLDevAPI.bas) in the Include subdirectory.

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

2.3. Load/Unload the Driver

The first call into the MV driver must be **WLoadDriver**. Call **WUnloadDriver** when you are finished.

2.4. Enumerate/Open a Device

To get the list of available devices, call **WLEnumDevices**. Once the devices are enumerated, use the **nDeviceID** field of the device list in your call to **WLOpenDevice**. Here is a simple example of opening the first available device:

```

WL_ENUMITEM wl[10];
unsigned long nListLen = sizeof(wl)/sizeof(WL_ENUMITEM);

// Load the driver
WLLoadDriver();

// nListLen is the length of your WL_ENUMITEM array
WLEnumDevices(&wl[0], &nListLen );
// nListLen is now the number of devices available. It may be
// larger than your WL_ENUMITEM array length!
if (( nListLen > 0 ) && (wl[0].bIsOpen == FALSE ))
{
    WL_HANDLE hDev;
    // Open the first device in the list.
    WLOpenDevice(wl[0].nCamID, &hDev );
    // Do something...
    ...
    // Close the device.
    WLCloseDevice( hDev );
}

// Unload the driver
WLUnloadDriver();

```

The device list contains a unique ID which identifies each particular device.

The wireless devices have different models and capabilities:

Y camera GPS device: camera device that provides synchronization signal and trigger to Y cameras. It synchronizes with other devices via GPS antenna. It has light sensor for illumination adjustment.

Y camera GPS/STP device: camera device that provides synchronization signal and trigger to Y cameras. It synchronizes with other devices via GPS antenna or STP cable. It has light sensor for illumination adjustment.

Generic camera GPS device: camera device that provides synchronization signal and trigger to generic cameras. It synchronizes with other devices via GPS antenna or STP cable. It has light sensor for illumination adjustment.

LED Light Device: light device that provides synchronization signal to LED light sources. It synchronizes with other devices via GPS antenna or STP cable. It doesn't have the light sensor.

Trigger Device: device that provides trigger event to other devices. It synchronizes with other devices via GPS antenna.

Hand-Held controller: controller that can be used to configure other devices.

Range-Extender: device that extends the ZigBee network (repeater).

2.5. Parameters

The parameters that can be configured in each device are described below.

Period: the period of the square wave generated by the device (in microseconds).

Pulse Width: the pulse width of the square wave generated by the device (in microseconds).

Phase (delay): the phase of the square wave (in microseconds).

Illumination Gain: the illumination gain or attenuation (from 0.2 to 1.8, the average is 1.0).

Illumination Threshold: the illumination sensor threshold (0 to 100). The value is a percentage and it determines the minimum change of the light intensity required to change the pulse width.

Example 1: the value of threshold is 5; if the external light intensity changes by less of 5%, nothing changes; if the light intensity changes by 5% or more, then the pulse width of the sync signal is automatically changed by the same value (5% or more).

Illumination Low Pass Filter: the illumination reaction parameter (0: no change 100: immediate change). The value is a percentage and it determines the importance of the value of previous illumination measurement in the computation of the next pulse width (weighted sum).

Example 1: the value is 100. The new value of pulse width is function of the previous light measurement only, the previous value of pulse width does not affect it.

Example 2: the value is 5. The previous measurement of light determines only 1% of the pulse width and the previous pulse width value determines the remaining 99%.

Example 3: the value is 0. The light measurement does not influence the computation of next pulse width.

Laser Delays: the phase of three sync signals generated by the laser device.

2.6. Commands

Once the parameters are configured the user can send the device the following commands:

WLC_SYNC: the device starts or stops the synchronization signal.

WLC_I_ADJUST: the device starts or stops the illumination adjustment.

3. Galileo SDK Reference

3.1. Initialization Functions

3.1.1. Overview: Initialization functions

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

WLGetVersion returns the DLL version numbers.

WLLoadDriver loads the driver and initializes it.

WLUnloadDriver unloads the driver.

WLGetCoordinatorInfo reads information parameters from the coordinator USB key.

WLEnumDevices enumerates the device connected to the computer.

WLOpenDevice opens a device.

WLCloseDevice closes a device previously open.

3.1.2. WLGetVersion

WL_ERROR WLGetVersion (unsigned long *pVerMajor, unsigned long *pVerMinor)

Return values

WL_SUCCESS if successful, otherwise

WL_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

Remarks

This function must be called to retrieve the DLL version.

See also:

3.1.3. WLLoadDriver

WL_ERROR WLLoadDriver (void)

Return values

WL_SUCCESS if successful, otherwise

WL_E_NO_USB_DEV if the USB Bridge is not installed or detected

WL_E_USB_DEV_OPEN if the USB Bridge cannot be initialized

Parameters

None

Remarks

The routine loads the driver DLL and initializes it. It must be called before any other routine, except **WLGetVersion**.

See also: **WUnloadDriver**

3.1.4. WLUnloadDriver

void WLUnloadDriver (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 driver and unloads it.

See also: **WLLoadDriver**

3.1.5. WLGetCoordinatorInfo

WL_ERROR WLGetCoordinatorInfo (WL_INFO nInfoKey, unsigned long *pLoValue, unsigned long *pHiValue)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

WL_E_NOT_SUPPORTED, if the nInfoKey is not supported.

Parameters

nInfoKey

Specifies which parameter the function has to return

pLoValue

Specifies the pointer to the variable that receives the LS part of the info value

pHiValue

Specifies the pointer to the variable that receives the MS part of the info value

Remarks

This function returns device specific information from the coordinator, such as model and serial number. See the **Appendix B** for a list of all the available nInfoKey values.

See also:

3.1.6. WLEnumDevices

WL_ERROR WLEnumDevices (**PWL_ENUMITEM** *pItemList*, **unsigned long** **pItemNr*, **unsigned long** *nResetList*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_ARGUMENTS, if any of the parameters is not valid.

Parameters

pItemList

Specifies the pointer to an array of WL_ENUMITEM structures

pItemNr

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

nResetList

Specifies if the list of detected devices should be erased before starting a new enumeration

Remarks

The routine enumerates the active devices and fills the **WL_ENUMITEM** structures with information about them. This routine must be called before **WLOpenDevice** 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 enumerated devices. If the *nResetList* parameter is set to 0, the routine detects only if new devices has been added to the network and does not check if the devices already in the list have been disconnected. If the parameter is set to 1, the routine erases the list and enumerates all the devices.

See also: **WLOpenDevice**

3.1.7. WLOpenDevice

WL_ERROR WLOpenDevice (unsigned long *nDeviceId*, **WL_HANDLE*** *pHandle*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_DEV_ID, if the device ID is not valid.

WL_E_DEV_ALREADY_OPEN, if the device is already open.

Parameters

nDeviceId

Specifies the ID of the device to be opened

pHandle

Specifies the pointer to the variable that receives the handle

Remarks

The routine opens the device with the *nDeviceId* identifier. The value can be retrieved by calling the **WLEnumDevices** routine (see the **WL_ENUMITEM** structure).

See also: **WLCloseDevice**

3.1.8. WLCloseDevice

WL_ERROR WLCloseDevice (WL_HANDLE *hDevice*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the device handle is not valid.

Parameters

hDevice

Specifies the handle to an open device

Remarks

Closes an open device

See also: **WLOpenDevice**

3.2. Info and configuration Functions

3.2.1. Overview: Info and Configuration functions

The info and configuration functions allow the user to read information from the devices and configure them.

WLGetDeviceInfo gets information from the device, such as model, serial number, network address, etc.

WLGetParameterAttribute gets the parameters attributes (minimum and maximum values).

WLGetParameter gets a parameter from the device.

WLSetParameter sets a parameter.

3.2.2. WLGetDeviceInfo

WL_ERROR WLGetDeviceInfo (**WL_HANDLE** *hDevice*, **WL_INFO** *nInfoKey*, **unsigned long** **pLoValue*, **unsigned long** **pHiValue*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the device handle is not valid.

WL_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

WL_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

pLoValue

Specifies the pointer to the variable that receives the LS part of the info value

pHiValue

Specifies the pointer to the variable that receives the MS part of the info value

Remarks

This function returns device specific information, such as model and serial number. See the **Appendix B** for a list of all the available *nInfoKey* values.

See also:

3.2.3. WLGetParameterAttribute

WL_ERROR WLGetParameterAttribute (WL_HANDLE *hDevice*, WL_PARAM *nParamKey*, unsigned long **pMinLo*, unsigned long **pMinHi*, unsigned long **pMaxLo*, unsigned long **pMaxHi*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the Device handle is not valid.

WL_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

WL_E_NOT_SUPPORTED, if the *nParamKey* is not supported.

Parameters

hDevice

Specifies the handle to an open Device

nParamKey

Specifies which parameter the function returns.

pMinLo, *pMinHi*

Specifies the pointer to the parameter's minimum value (LS and MS parts).

pMaxLo, *pMaxHi*

Specifies the pointers to the parameter's maximum value (LS and MS parts).

Remarks

This function reads the parameter's minimum and maximum values.

See also: WLGetParameter

3.2.4. WLGetParameter

WL_ERROR WLGetParameter (**WL_HANDLE** *hDevice*, **WL_PARAM** *nParamKey*, **unsigned long** **pValueLo*, **unsigned long** **pValueHi*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the Device handle is not valid.

WL_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

WL_E_NOT_SUPPORTED, if the nParamKey is not supported.

Parameters

hDevice

Specifies the handle to an open Device

nParamKey

Specifies which parameter the function returns

pValueLo

Specifies the pointer to the variable that receives the LS part of the parameter's value

pValueHi

Specifies the pointer to the variable that receives the MS part of the parameter's value

Remarks

This function reads a parameter from the Device.

See also: WLSetParameter

3.2.5. WLSetParameter

WL_ERROR WLSetParameter (**WL_HANDLE** *hDevice*, **WL_PARAM** *nParamKey*, **unsigned long** *nValueLo*, **unsigned long** *nValueHi*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the Device handle is not valid.

WL_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

WL_E_NOT_SUPPORTED, if the nParamKey is not supported.

WL_E_INVALID_VALUE, if the parameter value is not valid

Parameters

hDevice

Specifies the handle to an open Device

nParamKey

Specifies which parameter the function sets.

nValueLo

Specifies the LS part of the parameter's value

nValueHi

Specifies the MS part of the parameter's value

Remarks

This function writes a parameter to the Device.

See also: **WLGetParameter**

3.3. Device Control Functions

3.3.1. Overview: Device Control functions

The Device control functions allow the user to send commands to the Device and retrieve data and status from the device.

WLSendCommand sends a specific command to the device.

WLReadData reads data from the device such as the trigger delay.

WLGetStatus gets the device status and the battery level.

3.3.2. WLSendCommand

WL_ERROR WLSendCommand (WL_HANDLE *hDevice*, WL_COMMAND *nCommandKey*, unsigned long *nParamLo*, unsigned long *nParamHi*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the Device handle is not valid.

WL_E_NOT_SUPPORTED, if the command key is not supported.

WL_E_INVALID_VALUE, if the parameter value is not valid

WL_E_OFFLINE, if the device is disconnected

Parameters

hDevice

Specifies the handle to an open Device

nCmdKey

Specifies the command code

nParamLo

Specifies the LS part of the optional command parameter

nParamHi

Specifies the MS part of the optional command parameter

Remarks

This function sends a command to the device. If the device handle is set to **WLHND_BROADCAST** the command is sent to all the connected devices. For a list of the commands please refer to the Appendix.

See also:

3.3.3. WLReadData

WL_ERROR WLReadData (WL_HANDLE hDevice, WL_DATA nDataKey, void * pData, unsigned long nSize)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the device handle is not valid.

WL_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

WL_E_BUSY, if the device is busy and cannot answer

Parameters

hDevice

Specifies the handle to an open device

nDataKey

Specifies the type of data to read

pData

Specifies the pointer to the buffer that receives the data

nSize

Specifies the size of the data buffer

Remarks

This function retrieves data from a device.

See also:

3.3.4. WLGetStatus

WL_ERROR WLGetStatus (**WL_HANDLE** *hDevice*, **unsigned long** **pnStatus*, **unsigned long** **pnBatterySts*)

Return values

WL_SUCCESS if successful, otherwise

WL_E_INVALID_HANDLE, if the Device handle is not valid.

WL_E_INVALID_ARGUMENTS, if the pointer to the buffer is not valid.

WL_E_BUSY, if the device is busy and cannot answer

Parameters

hDevice

Specifies the handle to an open Device

pnStatus

Specifies the pointer to a variable that receives the device status

pnBatterySts

Specifies the pointer to a variable that receives the battery status

Remarks

This function retrieves the device status and the battery status. For a complete list of device status values, see the WL_STATUS enumeration list in the appendix. The battery status is a combination of bits.

BIT 0 to 3: they show the battery level from a minimum of 0 to a maximum of 5

BIT4: the bit is set to 1 if the device is externally powered

BIT5: the bit is set to 1 if there is a battery fault.

See also:

4. Galileo ActiveX control

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

The control may be inserted in any application and may be controlled using the software interface.

The routines exported by the ActiveX correspond to the SDK API.

4.1.1. How to add the control to a Visual Studio project

The control may be added to a Visual C++ dialog box or a Visual Basic form.

- Add the ActiveX control to the toolbox. Right click on the Toolbox, and select "Add/Remove Items..."
- From the "Customize Toolbox" dialog, select the "COM components" tab.
- Locate the XWLDevX control, select it and click OK.
- Drag the control from the Toolbox to the dialog box or form in your project.

5. Galileo LabVIEW™ Reference

5.1. Overview

The LabVIEW™ Interface allows controlling the devices from inside a National Instruments LabVIEW application. It works with LabVIEW 6 and above, on Windows 2000/XP/Vista. Windows NT and MAC OS are not supported.

The LabVIEW™ Interface includes the **VIs (Virtual Instruments)** for controlling the devices and some sample VIs showing how to use the interface: the VIs are packaged in a library called **IDTWL.LLB**) located in the **user.lib\Galileo** subdirectory of the LabVIEW folder. The examples are located in the **LabVIEW** subdirectory of the installation folder (C:\Program Files\IDT\Galileo).

The 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 “Galileo Wireless VIs” button.

5.2. Initialization VIs

5.2.1. Overview: Initialization VIs

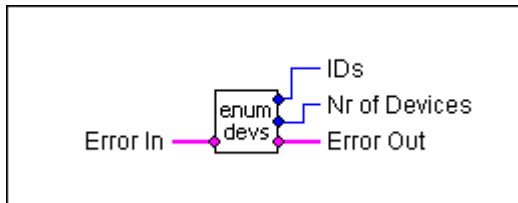
Initialization Virtual Instruments allow the user to enumerate the available devices, open and close.

Enum Devices enumerates the devices connected to the ZigBee network.

Open Device opens a device.

Close Device closes a device, previously open.

5.2.2. Enum Devices



Inputs

Error

Specifies a standard error cluster input terminal

Outputs

Error

Specifies the return error code of the function (0 if it's successful, non 0 otherwise)

IDs

Specifies the array containing the IDs of the enumerated devices

Nr of Devices

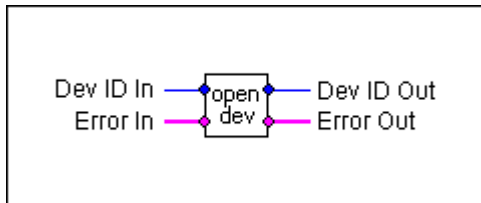
Specifies the number of enumerated devices

Remarks

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

See also: “Open Device”

5.2.3. 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

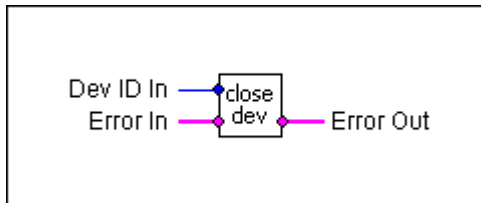
Specifies the ID of the opened device

Remarks

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

See also: “Close Device”

5.2.4. 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 an error code.

See also: “Open Device”

5.3. Info and Configuration Vis

5.3.1. Overview: Configuration VIs

Configuration Virtual Instruments allow the user to read information from the devices, read parameters, read parameters minimum and maximum values and write parameters.

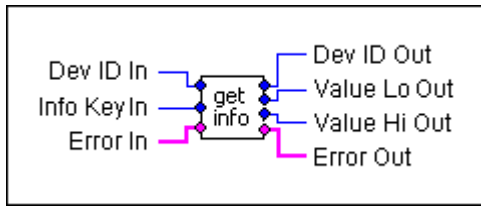
Get Info reads information from the device, such as model, serial number, network address, etc.

Get Parameter reads a specific parameter from the device.

Set Parameter writes a single specific parameter to the configuration.

Get Parameter Attribute reads the minimum and the maximum values of a parameter.

5.3.2. 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

ValueLo

Specifies the value of the info parameter (LS value)

ValueHi

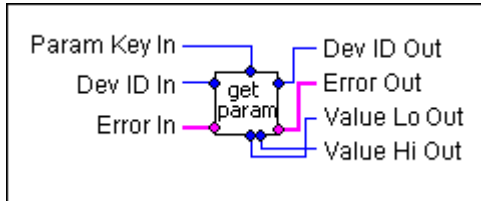
Specifies the value of the info parameter (MS value)

Remarks

This VI returns Device specific information, such as serial number of network address, 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 an error code.

See also: “Get Parameter”

5.3.3. 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

Outputs

Device ID

Specifies the Device ID

Error

Specifies the return error condition

ValueLo

Specifies the current value of the parameter (LS value)

ValueHi

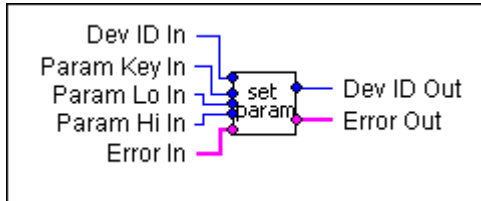
Specifies the current value of the parameter (MS value)

Remarks

This VI reads a specific configuration parameter from the device and returns its value. 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 an error code.

See also: "Set Parameter"

5.3.4. 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 Lo

Specifies the value of the parameter (LS value)

Param Hi

Specifies the value of the parameter (MS value)

Outputs

Device ID

Specifies the Device ID

Error

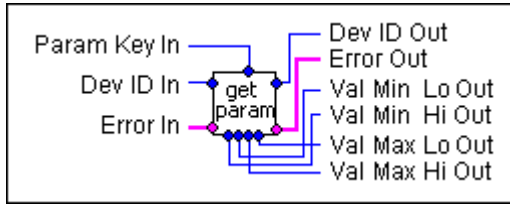
Specifies the return error condition

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 is available in Appendix C. If any error occurs during the operation, the Error Out terminal signals an error code.

See also: "Get Parameter"

5.3.5. Get Parameter Attribute



Inputs

Device ID

Specifies a valid Device ID

Error

Specifies a standard error cluster input terminal

Param Key

Specifies the index of the parameter

Outputs

Device ID

Specifies the Device ID

Error

Specifies the return error condition

Value Min Lo and Hi

Specifies the minimum value of the parameter (LS and MS values)

Value Max Lo and Hi

Specifies the maximum value of the parameter (LS and MS values)

Remarks

This VI reads the minimum and the maximum values of a parameter. If any error occurs during the operation, the Error Out terminal signals an error code.

See also: "Get Parameter", "Set Parameter"

5.4. Device Control VIs

5.4.1. Overview: Device Control functions

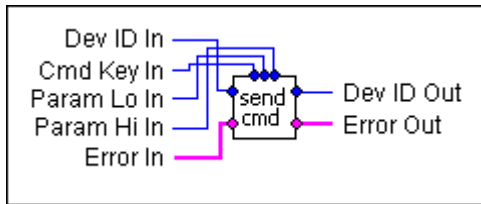
The Device control VIs allow the user to send commands to the Device and retrieve data and status from the device.

Send Command sends a specific command to the device.

Read Trigger Delay reads the trigger delay from the device.

Get Status gets the device status and the battery level.

5.4.2. Send Command



Inputs

Device ID

Specifies a valid Device ID

Error

Specifies a standard error cluster input terminal

Cmd Key

Specifies the index of the command

Param Lo

Specifies the value of the parameter (LS value)

Param Hi

Specifies the value of the parameter (MS value)

Outputs

Device ID

Specifies the Device ID

Error

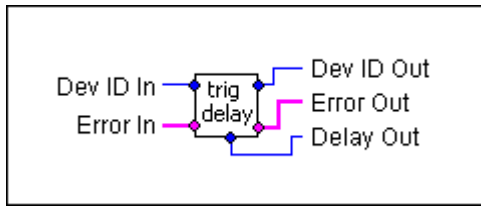
Specifies the return error condition

Remarks

This VI sends a command with a parameter to the device. The command key is one of the input parameters. A list of the commands is available in Appendix. If any error occurs during the operation, the Error Out terminal signals an error code.

See also:

5.4.3. Read Trigger Delay



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

Delay

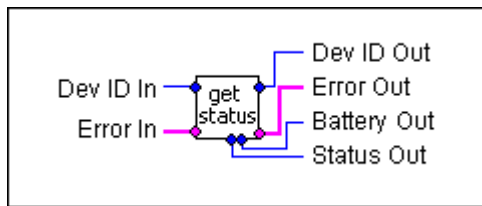
Specifies the value of the trigger delay

Remarks

This VI reads the trigger delay the device and returns its value. If any error occurs during the operation, the Error Out terminal signals an error code.

See also:

5.4.4. Get Status



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

Status

Specifies the status of the device

Battery

Specifies the status and level of the battery

Remarks

This VI reads the status and the battery level from the device. If any error occurs during the operation, the Error Out terminal signals an error code.

See also:

6. Galileo MATLAB™ Reference

6.1. Overview

The MATLAB™ Interface allows to control the devices from inside the Mathworks™ MATLAB application. The interface works with MATLAB 6.5 and greater, on Windows 2000, XP Professional and Vista.

The interface includes the 'MEX' file for controlling the device (packaged in a library called WLDevML.dll) and a few 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 ...] = WLDevML [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. 'EnumDevices') and output1 is the result of the operation (0 = SUCCESS, otherwise ERROR).

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

The MATLAB interface reflects the SDK API with a few exceptions. The MATLAB interface is listed below.

6.2. Initialization Functions

6.2.1. Overview: Initialization functions

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

Version returns the DLL version numbers.

EnumDevices enumerates the device connected to the computer.

OpenDevice opens a device.

CloseDevice closes a device previously open.

6.2.2. Version

`[strVersion] = XStreamML ('Version')`

Inputs

None

Outputs

strVersion

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

Remarks

This function returns the MATLAB interface version string.

See also:

6.2.3. EnumDevices

[*nResult*, *nItems*, *wlArray*] = WLDevML ('EnumDevices')

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

wlArray

Specifies the array containing the IDs of the detected devices

Remarks

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

See also: OpenDevice

6.2.4. OpenDevice

[*nResult*, *nDeviceId*] = WLDevML ('OpenDevice', *nInputId*)

Inputs

nInputId

Specifies the ID of the device to open (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 open device

Remarks

The routine opens the device with unique ID *nInputId*. The user may supply a specific device ID or 0: in this case the first available device is open. If any error occurs, the routine returns a code in the *nResult* variable, otherwise it returns 0.

See also: CloseDevice

6.2.5. CloseDevice

[nResult] = WLDevML ('**CloseDevice**', *nDeviceId*)

Inputs

nDeviceId

Specifies the ID of the device

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. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: OpenDevice

6.3. Info and configuration Functions

6.3.1. Overview: Info and Configuration functions

The info and configuration functions allow the user to read information from the devices and configure them.

GetDeviceInfo gets information from the device, such as model, serial number, network address, etc.

GetParameterAttribute gets the parameters attributes (minimum and maximum values).

GetParameter gets a parameter from the device.

SetParameter sets a parameter.

6.3.2. GetDeviceInfo

[*nResult*, *nValueLo*, *nValueHi*] = WLDevML ('GetDeviceInfo', *nDeviceId*, *nInfoKey*)

Inputs

nDeviceId

Specifies a valid device ID

nInfoKey

Specifies the info parameter key

Outputs

nResult

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

nValueLo

Specifies the returned value (LS part)

nValueHi

Specifies the returned value (MS part)

Remarks

This function returns device specific information, such as serial number or network address. See the Appendix B for a list of all the available *nInfoKey* values.

See also:

6.3.3. GetParameterAttribute

[*nResult*, *nMinValueLo*, *nMinValueHi*, *nMaxValueLo*, *nMaxValueHi*] = WLDDevML ('GetParameterAttribute', *nDeviceId*, *nParamKey*)

Inputs

nDeviceId

Specifies a valid device ID

nParamKey

Specifies the parameter index

Outputs

nResult

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

nMaxValueLo, *nMinValueHi*

Specifies the minimum value of the parameter (MS and LS parts)

nMaxValueLo, *nMaxValueHi*

Specifies the maximum value of the parameter (MS and LS parts)

Remarks

This function reads parameter minimum and maximum values from the device. 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.

See also: GetParameter, SetParameter

6.3.4. GetParameter

[nResult, nValueLo, nValueHi] = WLDevML ('GetParameter', *nDeviceId*, *nParamKey*)

Inputs

nDeviceId

Specifies a valid device ID

nParamKey

Specifies the parameter index

Outputs

nResult

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

nValueLo

Specifies the value of the parameter (LS part)

nValueHi

Specifies the value of the parameter (MS part)

Remarks

This function reads a parameter from the device. 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.

See also: SetParameter

6.3.5. SetParameter

[nResult] = WLDevML ('SetParameter', *nDeviceId*, *nParamKey*, *nValueLo*, *nValueHi*)

Inputs

nDeviceId

Specifies a valid device ID

nParamKey

Specifies the index of the parameter

nValueLo

Specifies the value of the parameter (LS part)

nValueHi

Specifies the value of the parameter (MS part)

Outputs

nResult

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

Remarks

This function writes a specific configuration parameter to the device. A list of the parameters indexes 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.

See also: GetParameter

6.4. Device Control Functions

6.4.1. Overview: Device Control functions

The Device control functions allow the user to send commands to the Device and retrieve data and status from the device.

SendCommand sends a specific command to the device.

ReadTriggerDelay reads the trigger delay from the device.

GetStatus gets the device status and the battery level.

6.4.2. SendCommand

[*nResult*] = WLDevML ('SndCommand, *nDeviceId*, *nCmdKey*, *nValueLo*, *nValueHi*)

Inputs

nDeviceId

Specifies a valid device ID

nCmdKey

Specifies the index of the command

nValueLo

Specifies the value of the command parameter (LS part)

nValueHi

Specifies the value of the command parameter (MS part)

Outputs

nResult

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

Remarks

This function sends a command to the device. A list of the commands indexes is available in Appendix. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0.

See also:

6.4.3. ReadTriggerDelay

[*nResult*, *nTrigDelay*] = WLDevML ('ReadTriggerDelay', *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)

nTrigDelay

Specifies the value of the trigger delay in microseconds

Remarks

This function reads the trigger delay from a device. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0.

See also:

6.4.4. GetStatus

[*nResult*, *nStatus*, *nBatterySts*] = WLDevML ('GetStatus', *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)

nStatus

Specifies the value of the status

nBatterySts

Specifies the value of the battery status and level

Remarks

This function reads the device status and the battery status and level. If any error occurs during the operation, the routine returns an error code in the *nResult* variable, otherwise it returns 0.

See also:

7. Appendix

7.1. Appendix A - Return Codes

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

Code	Value	Notes
WL_SUCCESS	0	OK – No errors
WL_E_GENERIC_ERROR	1	Generic Error
WL_E_NO_USB_DEV	2	The USB key has not been detected
WL_E_USB_DEV_OPEN	3	Unable to open the USB Key
WL_E_NOT_SUPPORTED	4	The function is not supported
WL_E_INVALID_DEV_ID	5	Invalid device ID used in WLOpenDevice. The ID is retrieved from the WLEnumDevices routine
WL_E_INVALID_HANDLE	6	Invalid WL_HANDLE handle
WL_E_INVALID_VALUE	7	Invalid parameter value
WL_E_INVALID_ARGUMENTS	8	Invalid function arguments
WL_E_CAM_ALREADY_OPEN	9	Cannot open the device because it's already open.
WL_E_BUSY	10	The device is busy and cannot answer
WL_E_OFFLINE	11	The device is offline (not connected to the network)

7.2. Appendix B – Info

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

Parameter	Description
WLI_DEVICE_ID	Device ID (see WL_ENUMITEM structure in WLDevAPI.h)
WLI_DEVICE_MODEL	Device Model (see WL_DEV_MODEL in WLDevAPI.h)
WLI_FW_VERSION	The device firmware version
WLI_SERIAL	The device serial number (10 digits decimal value)
WLI_REVISION	The device revision
WLI_NET_ADDRESS	The device ZigBee network Address

7.3. Appendix C – Parameters

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 **WLDevAPI.h** header file in the **Include** subdirectory.

Parameter	Rd/Write	Description
WLP_PERIOD	R/W	The sync signal period in microseconds [us]
WLP_PULSE_WID	R/W	The sync signal pulse width in microseconds [us]
WLP_PHASE	R/W	The sync signal phase in microseconds [us]
WLP_I_GAIN	R/W	The illumination adjustment gain
WLP_I_THRESHOLD	R/W	The illumination adjustment threshold
WLP_I_LPF	R/W	The illumination adjustment Low Pass Filter
WLP_L_DELAY_1	R/W	The phase delay is first laser sync signal
WLP_L_DELAY_2	R/W	The phase delay is second laser sync signal
WLP_L_DELAY_3	R/W	The phase delay is third laser sync signal

7.4. Appendix D – Commands

The following table shows the values and a brief description of the command that can be sent to the device. The numeric values of the parameters can be found in the **WLDevAPI.h** header file in the **Include** subdirectory. The “Hi Param” is ignored. It has been introduced for future use.

Parameter	Value	Lo Param	Hi Param	Description
WLC_SYNC	0	0/1	Not used	Activate/Deactivate sync
WLC_I_ADJUST	1	0/1	Not used	Activate/Deactivate Illumination adjustment

7.5. Appendix E – Data types

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

7.5.1. WL_DEV_MODEL

The **WL_CAM_MODEL** type enumerates the device models.

- **WL_DM_UNKNOWN**: Unknown Device model
- **WL_DM_Y_GPS**: Y camera GPS device.
- **WL_DM_Y_GPS_STP**: Y camera GPS/STP device.
- **WL_DM_CAM_GPS**: generic camera GPS device.
- **WL_DM_LED_LIGHT**: LED Light device.
- **WL_DM_SNS_TRG**: Trigger device.
- **WL_DM_HHELD_CTR**: Hand-Held controller.
- **WL_DM_RANGE_EXT**: Range Extender (repeater).
- **WL_DM_LASER_CTR**: Laser controller.

7.5.2. WL_REVISION

The **WL_REVISION** type enumerates the hardware revisions.

- **WL_REV_A**: Revision A.
- **WL_REV_B**: Revision B.
- **WL_REV_C**: Revision C.
- **WL_REV_D**: Revision D.
- **WL_REV_E**: Revision E.
- **WL_REV_F**: Revision F.
- **WL_REV_G**: Revision G.
- **WL_REV_H**: Revision H.
- **WL_REV_I**: Revision I.
- **WL_REV_J**: Revision J.

7.5.3. WL_GAIN

The **WL_GAIN** enumerates the illumination gains:

- **WLG_0_PT_2**: gain 0.2.
- **WLG_0_PT_4**: gain 0.4.
- **WLG_0_PT_6**: gain 0.6.

- **WLG_0_PT_8**: gain 0.8.
- **WLG_1_PT_0**: gain 1.0.
- **WLG_1_PT_2**: gain 1.2.
- **WLG_1_PT_4**: gain 1.4.
- **WLG_1_PT_6**: gain 1.6.
- **WLG_1_PT_8**: gain 1.8.

7.5.4. WL_CMD_OP

The WL_CMD_OP enumerates the operations with commands:

- **WL_CMD_STOP**: stop the execution of the command.
- **WL_CMD_START**: start the execution of the command.

7.5.5. WL_STATUS

The WL_STATUS enumerates the device status:

- **WLST_UNKNOWN**: unknown status (error reading).
- **WLST_READY_NOGPS**: the device is ready and the GPS signal is not detected.
- **WLST_READY_GPS**: the device is ready and the GPS signal is detected.
- **WLST_SYNC_NOGPS**: the device is generating sync and the GPS signal is not detected.
- **WLST_SYNC_GPS**: the device is generating sync and the GPS signal is detected.
- **WLST_OFFLINE**: the device is offline (not connected to the network).

7.5.6. WL_ERROR

The WL_ERROR enumerates the return codes. See Appendix A.

7.5.7. WL_INFO

The WL_INFO enumerates the information index. See Appendix B.

7.5.8. WL_PARAM

The WL_PARAM enumerates the parameters. See Appendix C.

7.5.9. WL_COMMAND

The WL_COMMAND enumerates the commands. See Appendix D.

7.6. Appendix F – Structures

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

7.6.1. WL_ENUMITEM

The WL_ENUMITEM structure contains information about a device. It must be used in the enumeration procedure with the WLEnumDevices routine.

```
typedef struct
{
    unsigned long cbSize;

    unsigned long nDeviceId;
    unsigned long nDeviceModel;
    unsigned long nSerial;
    unsigned long nNetAddress;
    unsigned long nFwVersion;
    unsigned long nRevision;
    unsigned long bIsOpen;

} WL_ENUMITEM, *PWL_ENUMITEM;
```

Members

cbSize

It specifies the size of the WL_ENUMITEM structure.

nDeviceId

It specifies the ID which identifies the device. The user must use this ID to open the device with WLOpenDevice.

nDeviceModel

It specifies the device model.

nSerial

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

nNetAddress

It specifies the ZigBee network address of the device.

nFwVersion

It specifies the device firmware version.

nRevision

It specifies the device revision number.

bIsOpen

It specifies whether the device is currently open or not.