IMPORTANT INFORMATION

The APT product range covers a large number of controller types and enclosure formats (Cubes, benchtop units and 19" rack modules). The range includes stepper and DC servo controllers, piezo controllers, solenoid controllers, strain gauge readers and auto-alignment controllers (NanoTrak).

The software functionality associated with these controllers has been implemented within the various ActiveX Controls provided by the APT support software. The following list highlights the various ActiveX Controls provided and the associated controller products covered by each:-

Motor ActiveX Control

BSC001 – 1 Channel Benchtop Stepper Driver

BSC002 – 2 Channel Benchtop Stepper Driver

BMS001 – 1 Channel Benchtop Low Power Stepper Driver

BMS002 – 2 Channel Benchtop Low Power Stepper Driver

MST601 – 2 Channel Modular Stepper Driver

BSC101 – 1 Channel Benchtop Stepper Driver (2006 onwards)

BSC102 – 2 Channel Benchtop Stepper Driver (2006 onwards)

BSC103 – 3 Channel Benchtop Stepper Driver (2006 onwards)

BBD101 – 1 Channel Benchtop Brushless DC Motor Driver

BBD102 – 2 Channel Benchtop Brushless DC Motor Driver

BBD103 – 3 Channel Benchtop Brushless DC Motor Driver

OST001 – 1 Channel Cube Stepper Driver

ODC001 – 1 Channel Cube DC Servo Driver

TST001 – 1 Channel T-Cube Stepper Driver

TDC001 – 1 Channel T-Cube DC Servo Driver

TSC001 – 1 Channel T-Cube Solenoid Driver

Piezo ActiveX Control

BPC001 – 1 Channel Benchtop Piezo Driver

BPC002 – 2 Channel Benchtop Piezo Driver

MPZ601 – 2 Channel Modular Piezo Driver

BPC101 – 1 Channel Benchtop Piezo Driver (2006 onwards)

BPC102 – 2 Channel Benchtop Piezo Driver (2006 onwards)

BPC103 – 3 Channel Benchtop Piezo Driver (2006 onwards)

TPZ001 – 1 Channel T-Cube Piezo Driver

TSG001 – 1 Channel T-Cube Strain Gauge Reader

NanoTrak ActiveX Control

BNT001 – 2 Channel Benchtop NanoTrak Controller

MNA601 – 2 Channel Modular NanoTrak Controller

APTLaser ActiveX Control

TLS001 – T-Cube Laser Source

TLD001 – T-Cube Laser Diode Driver

APTTEC ActiveX Control

TTC001 – T-Cube TEC Controller

APTQuad ActiveX Control

TQD001 – T-Cube Quad Detector Reader

Introduction_Programming_Guide

The APT controller family comprises a range of USB equipped hardware units, with the associated support software running on a host controller PC. This guide describes the PC software supplied to support operation of the APT controller units.
Typically two or more APT controller units will be connected together on the same USB bus in order to achieve multiple channels of operation, e.g. a particular alignment or positioning application may require more than the two motor channels of operation offered by a single APT stepper controller.
To support the handling of one or more APT controllers, a main APT software server system is supplied for use on the host controller PC. This is referred to as the APT Server (or Server) in the rest of this guide. The APT Server consists of a collection of ActiveX Controls (or Controls) together with associated support files and libraries.
The APT ActiveX Controls collection provides a rich set of graphical user interface (GUI) panels and programmable interfaces which can be incorporated easily into custom client applications (i.e. programs written to use the APT Server functionality) using a wide variety of Windows development environments (e.g. Visual Basic). Each type of APT hardware unit has an associated ActiveX Control, which provides all the interfacing requirements for the user to interact with the unit, either by GUI or programmable means.
System wide ActiveX Controls are also provided, which allow useful functionality, (e.g. event logging and system hardware enumeration), to be added to client applications. The various Active Controls supplied with the APT Server are described in more detail later in the 'APT ActiveX Controls' section.
In addition to the main APT Server software, a number of other user programs are supplied, which allow immediate use of the APT controllers without the need for custom software development. These various utilities are described further in the 'APT Utilities' section.
Installation instructions for the USB Drivers, APT Server and various utilities, are contained in the Software Installation section.

Software Installation

Download the software from www.thorlabs.com and follow the on-screen instructions. ***APT Utilities***

APT User.exe

The APT.exe application allows the user to interact with any number of APT hardware control units connected to the PC USB Bus. This program allows multiple graphical instrument panels to be displayed so that multiple APT units can be controlled. All basic operating parameters can be set through this program, and all basic operations (such as motor moves) can be initiated. This application is described in more detail in the *APT User helpfile*.

In many cases, the APT.exe application provides all of the functionality necessary to operate the APT hardware, without the need for users to develop any further custom software themselves. For those users who do need to further customise and automate usage of the APT controllers (e.g. to implement an alignment algorithm) this application provides a very good example of how the APT ActiveX Controls are used.

In fact, the program comprises a thin application framework around the functionality already contained within the APT Controls. Consequently, the bulk of the functionality provided by the APT.exe application is available for incorporation in any user custom application whenever the APT ActiveX Controls are used. To illustrate this fully, the complete Visual Basic source project for the APT.exe application is provided as a useful aid to client application developers. More details on using the APT ActiveX Controls when developing client application are given in the section 'APT ActiveX Controls '.

APT Config Utility

The APT Config utility is used to adjust settings and modes of operation for the APT Server itself. The first release of this utility allows settings to be made that relate to simulator and event logging operation of the APT server. As the APT software is developed further, this utility will allow other APT server adjustments and settings to be made.

The Server settings made using the config utility are stored on the host control PC and are accessed by the APT Server when it is running (i.e. when a client application that uses the APT Server is in execution). The 'Config' utility is a program that allows the settings to be altered off-line, prior to running a client application (such as the APT.exe application).

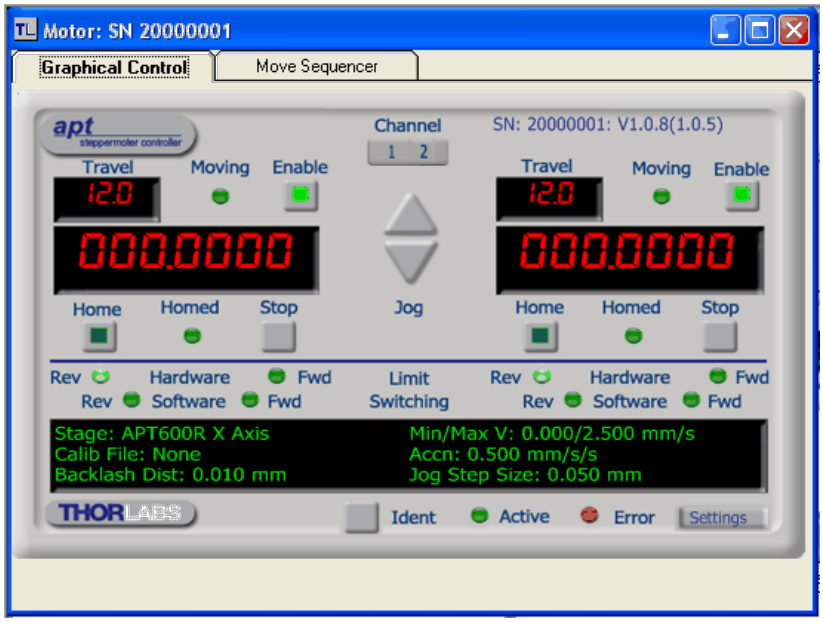
APT ActiveX Controls

In a general sense, ActiveX Controls are re-usable software components that supply both a graphical user interface and a programmable interface. Many such Controls are available for Windows applications development, providing a large range of re-usable functionality. For example, there are Controls available that can be used to manipulate image files, connect to the internet or simply provide user interface components such as buttons and list boxes. With the APT system, ActiveX Controls are deployed to allow direct control over (and also reflect the status of) the range of electronic controller units.

Based on ActiveX interfacing technology, an ActiveX Control is a language independent software component. In this way, ActiveX Controls can be incorporated into a wide range of software development environments for use by client application software developers. Development environments supported include Visual Basic, Labview, Visual C++, C++ Builder, VB.NET and C#.NET.

In the case of the APT system, ActiveX Controls provide a very convenient package of software functionality, whereby a single Control can supply all of the user interface and associated programmable functions relating to a particular hardware unit.

Consider the ActiveX Control supplied for the APT stepper controller unit [see screen shot below].



This Control provides a complete user graphical instrument panel to allow the motor unit to be manually operated, as well as a complete set of software functions to allow all parameters to be set and motor operations to be automated by a client application. The instrument panel reflects the current operating state of the controller unit to which it is associated (e.g. such as motor position).

Updates to the panel take place automatically when a client application is making software calls into the same control. For example, if a client application instructs the associated stepper motor control to move a motor, the progress of that move is reflected automatically by changing position readouts on the graphical interface, without the need for further programming intervention.

Note. Visual Basic is a relatively easy development system to use and is recommended for those users who are new to Windows applications development. In this documentation, implementation specific explanations and code samples are written using Visual Basic syntax and some familiarity of software development using Visual Basic is therefore assumed. However, these code samples also provide a useful illustration as to how the APT Server is accessed in other development environments. Refer to the documentation supplied with other development environments for further information on accessing and programming ActiveX objects.

ActiveX Control Usage

Typically, a Windows development system (such as Visual Basic) will allow a third party ActiveX Control to be incorporated into the design time environment for ease of access by the software developer. For Visual Basic 6, this is achieved as follows:

Run Visual basic, then open a new project.

From the top menu bar, select Project/Components

Select the appropriate box for the control required

e.g. MG17Motor ActiveX Control module

The control should then appear in the Toolbox and is used on a form in the same way as any other control

Within an application, each occurrence of an ActiveX Control is called an 'instance'. At ‘design time’ (i.e. when authoring a software application), developers will ‘drag and drop’ many instances of Controls onto one or more application windows (often referred to as forms), positioning and re-sizing as required to achieve a particular layout. Control settings (i.e. properties, described below) are also altered at design time as required.

At ‘run time’ (i.e. when running the software application), the Controls are then available for direct user interaction (if they have a user interface, some do not), often with little extra programming required by the software developer.

The APT Controls perform this process by allowing complete instrument control panels to be added to client application with very little requirement for extra programming (although some minimal software administration is required as described further below).

The programmable interfaces of an ActiveX control basically comprise three different entities, Methods, Properties and Events.

Methods

A ‘Method’ is the ActiveX terminology for a software function, (i.e. Methods are called passing any necessary parameters in the same way a function would be called). In the APT system, Methods can be used to set and retrieve hardware settings, initiate hardware actions (e.g. moving motors, switching feedback loop modes etc), and determine the configuration of hardware units attached to the host controller PC. A number of APT Method calls are shown in the following code sample:

```
'sets motor serial number and starts control

Public Function SetSerialNum(ByVal lSerialNum As Long) As Long

MG17Motor1.HWSerialNum = lSerialNum

MG17Motor1.StartCtrl

End Function

' return number of NanoTraks in system

MG17System.GetNumHWUnits(USB_NANOTRAK, lNumNanoTraks)

'sets piezo controller's channel 1 to closed loop mode

MG17Piezo1.SetControlMode(CHAN1_ID, CLOSED_LOOP)
```

Properties

A ‘Property’ is essentially a parameter setting associated with the ActiveX Control. Properties can be 'read only' or 'read/write' and can be accessed at design time (depending on the development environment, often in the form of a properties window), as well as run time.

The APT system has few properties relating to hardware specific settings, most relate to generic property values found on all Controls such as Left, Top, Width and Height.

However, the HWSerialNum property is found on many APT Controls, and is the primary setting that relates a specific ActiveX Control instance with a particular physical hardware unit. Further details on using this property can be found in the reference section of this software guide. An example source code listing is given below.

```
Public Property Get SerialNum() As Long

SerialNum = MG17NanoTrak1.HWSerialNum

End Property

Public Property Set SerialNum(ByVal lSerialNum)

MG17NanoTrak1.HWSerialNum = lSerialNum

End Property
```

Note that many APT hardware parameters that would normally be exposed as properties are actually exposed as Get/Set method pairs within the APT Server. This is because, unlike most ActiveX Controls, the APT Controls are in communication with physical hardware, and these parameters, when altered, often cause a change in how the actual hardware unit operates. Typically, Properties should not initiate activity and so Get/Set method pairs are used in the APT system instead.

Events.

An ‘Event’ is the name given to a notification that is made to the host client application by an ActiveX Control (i.e. a call made by the Control to a Method implemented by the client application). This Control callable Method is more usually referred to as an ‘event handler’ and the process of calling this event handler is more usually referred to as ‘firing an event’.

An ActiveX Control will fire an event when it wants to inform the client application that some action, state change or other system event has occurred. In the case of a simple button Control, an event might be fired when the user clicks on the button. An event handler (implemented in the client application) will typically contain code that has been written to handle the fired event, e.g. in the case of a button Control, the ‘Click’ event handler may display some form of message to the user.

The various Controls within the APT Server system are designed to fire events when certain important hardware events have occurred, e.g. when a motor has finished moving, a MoveComplete event will be fired. The associated event handler will then handle this change of state, e.g. measure a power level and then initiate a move to the next required position).

Using event handlers can lead to a very efficient client application implementation. Instead of a client program polling for system events (such as end of moves) and thereby preventing other application processing taking place, it can initiate some activity (e.g. a motor move) and then return to other processing or user interface handling until such time as the MoveComplete event is fired. To illustrate the difference between the ‘polling’ and ‘event driven’ approach, the following two code samples illustrate both approaches for carrying out the same activity (i.e. moving a motor).

Example of polling code

```
'call to home motor

fPosition = 0

MG17Motor1.SetAbsMovePos CHAN1_ID, 5

lRetVal = MG17Motor1.MoveAbsolute CHAN1_ID, False

...

Do While lPosition <> 5

MG17Motor1.GetPosition CHAN1_ID, fPosition

DoEvents

Loop
```

Example of event driven code...

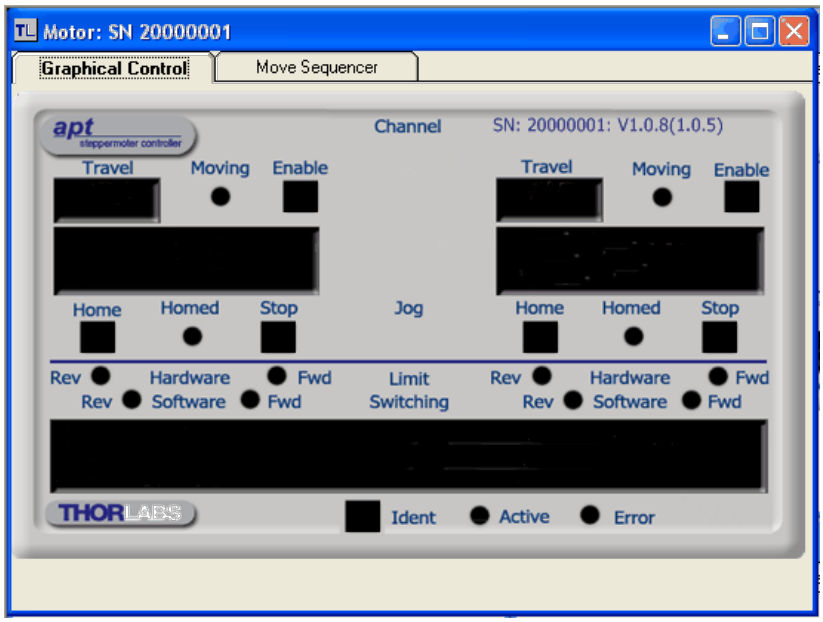
```
'call to move motor absolute 5mm

MG17Motor1.SetAbsMovePos CHAN1_ID, 5

lRetVal = MG17Motor1.MoveAbsolute CHAN1_ID, False

...

' when server fires event to say that home is completed, then call function to Private Sub MG17Motor1_MoveComplete(ByVal lHWChannel As Long)

Dim strtemp$

strtemp = "Motor1 Move Absolute Completed Ch " & Format$(lHWChannel)

DoEvents

If lHWChannel = CHAN1_ID Then
```

```
MoveActionCompleted 1

ElseIf

...

End If

End Sub

' event handler

Private Function MoveActionCompleted(ByVal lAxis As Long) As Long

Select Case m_lMotorState

Case 0 ' move absolute

m_lMotorState = m_lMotorState + 1

...required next move

Case ...

End Select

End Function
```

ActiveX Control Activation

As previously described in the ActiveX Control Usage section, an ActiveX Control is typically 'dragged and dropped' in the design time environment in order to add an instance to the client application. Design time properties are then set, and at run time, calls are made to methods and properties, and any required event handlers are executed when the associated events are fired.

The APT Server exposes a number of ActiveX Controls that implement a complete user instrument panel to allow direct manual interaction with the associated hardware units (after dragging and dropping them within the client development environment). The following screen shot shows the typical appearance of an APT ActiveX Control after being dragged and dropped within the Visual Basic 6 environment.



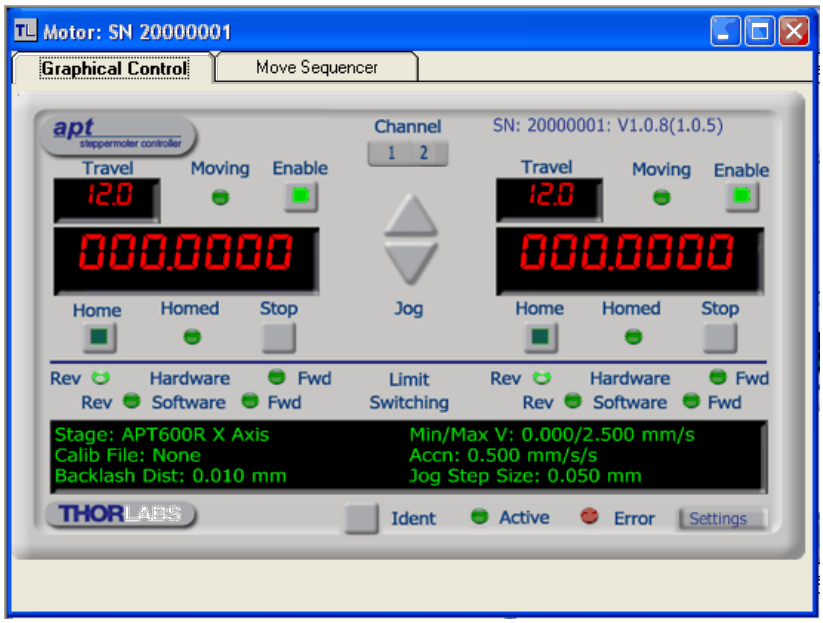
Note how the Controls' appearance is inactive.

There are two client program administration tasks that must be performed by a client application before an APT Control can be

activated for user interaction.

First, as discussed in the ActiveX Control Usage section, the HWSerialNumber property must be set to the correct (factory programmed) hardware unit serial number. This is usually found on a label attached to the unit itself. Secondly, at run time, a single call to the StartCtrl method must be made, in order to activate the control.

These two steps are the same for any of the APT graphical instrument panel ActiveX Controls. The following screen shot shows the appearance of an APT ActiveX Control after these two steps have been taken.



EnableEventDlg

This method has global effect, and will show the Event Diaolog box for all ActiveX controls.

Visual Basic Syntax
Function **EnableEventDlg**(bEnable As Boolean) As Long

Parameters
bEnable– specifies whether the Event Dialog message box is enabled

Returns
MG Return Code

Details
The Event Log assists in the development of client applications by combining traditional error and call logs into one object.
As a tool used during client application development the Event Log is useful for checking that the sequence of calls made into the server and the values of parameters passed are as intended, however, there may be situations where the continual 'pop up' of the log is inconvenient
This method enables or disables the event dialog box. If the *bEnable* parameter is set to TRUE, the event dialog is enabled and will be displayed each time an event (normal, error or warning) occurs.
If *bEnable* is set to FALSE, the event dialog is disabled, and will not be displayed unless the ShowEventDlg method is called.
For more information on the event dialog, see the MG17EventLog, within the MG17LoggerControl section.

ShowEventDlg

This method has global effect, and will show the Event Diaolog box for all ActiveX controls.

Visual Basic Syntax
Function **ShowEventDlg**(Void) As Long

Returns
MG Return Code

Details
In situations where it is inconvenient for the event dialog box to be displayed on the occurrence of each event, it can be disabled

using the EnableEventDlg method. This method can then be called to display the log only when necessary.
For more information on the event dialog, see the MG17EventLog section, within the MG17LoggerControl.

LLSetGetDigOPs

See Also Example Code

Visual Basic Syntax

Function **LLSetGetDigOPs**(bSet As Variant_Bool, plBits As Long) As Long

Parameters

bSet – specifies whether the method is setting or returning

plBits - the status of the digital outputs

Returns

MG Return Code

Details

The CONTROL IO connector on the rear panel of the unit exposes a number of digital outputs. The number of outputs available depends on the type of unit.

This method sets or gets the status of the digital outputs on the APT unit associated with the ActiveX control instance. If the *bSet* parameter is set to TRUE, the method sets the output state. If *bSet* is set to FALSE, the method returns the output state.

The outputs are set (and returned) in the least significant bits of the *plBits* parameter. The number of bits used is dependent on the number of digital outputs present on the associated hardware unit.

For example, to turn on the digital output on a BSC101 motor controller, the least significant bit of the *plBits* parameter should be set to 1. Similarly, to turn on all four digital outputs on a BNT001 NanoTrak unit, the least significant bits of the *plBits* parameter should be set to 1111 (15), and to turn the same outputs off, *plBits* should be set to 0000.

LLGetHostStatusBits

See Also Example Code

Visual Basic Syntax

Function **LLGetHostStatusBits**(plStatusBits As Long) As Long

Parameters

plStatusBits – the 32-bit integer containing the status flags.

Returns

MG Return Code

Details

This method is applicable only to 2- and 3-channel card slot type controllers such as the BSC103 and BPC202.

The USER IO connector on the rear panel of these units exposes a number of digital inputs.

This low level method returns a number of status flags pertaining to the status of the inputs on the host controller unit for the single channel controller card associated with the ActiveX control instance.

These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description

1. x00000001	1.	Digital input 1 state (1 - logic high, 0 - logic low).
2. x00000002	2	Digital input 2 state (1 - logic high, 0 - logic low).
0x00000004	3.	Digital input 3 state (1 - logic high, 0 - logic low).
0x00000008	4.	Digital input 4 state (1 - logic high, 0 - logic low).

LLSetGetHostDigOPs

See Also Example Code

Visual Basic Syntax

Function **LLSetGetHostDigOPs**(bSet As Variant_Bool, plBits As Long) As Long

Parameters

bSet – specifies whether the method is setting or returning

plBits - the status of the digital outputs

Returns

MG Return Code

Details

This method is applicable only to 2- and 3-channel card slot type controllers such as the BSC103 and BPC202.

The USER IO connector on the rear panel of these units exposes four digital outputs.

This method sets or gets the status of the digital outputs on the host controller unit for the single channel controller card associated with the ActiveX control instance. If the *bSet* parameter is set to TRUE, the method sets the output state. If *bSet* is set to FALSE, the method returns the output state.

The outputs are set (and returned) in the least significant bits of the *plBits* parameter.

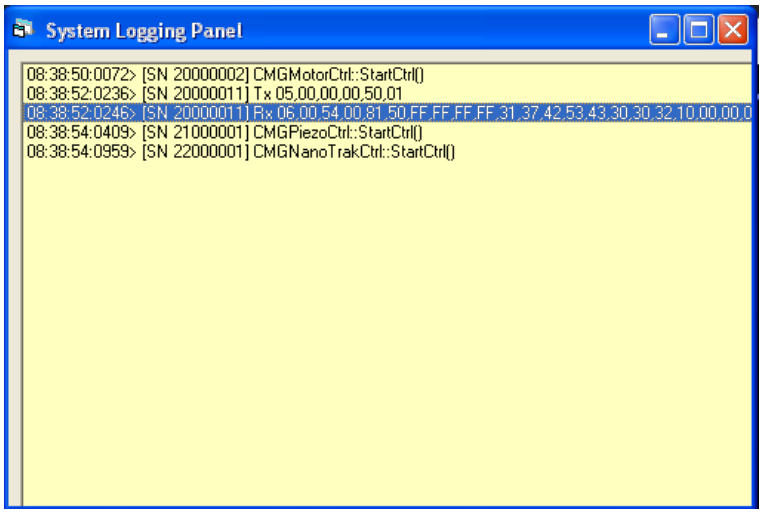
For example, to turn on all four digital outputs on a BSC103, the least significant bits of the *plBits* parameter should be set to 1111 (15). Similarly, to turn on digital outputs 1 and 3 of a BPC203, the 4 least significant bits of the *plBits* parameter should be set to 0101 (5).

Introduction to MG17Logger Control Programming

The Event Log assists in the development of client applications by combining traditional error and call logs into one object. Logged information can be displayed either in the MG17Logger control window or in the MG17EventLog dialog box, or can be saved to a text file.

Routine information, such as method calls, is displayed by default, but the log can also be set to log hardware communications and status messages as required. This is achieved using the APT Config utility.

The MG17Logger control can be dragged onto the development workspace to provide a running account of system occurrences as client application development progresses. A typical screen shot is shown below.



See the individual methods and properties for further information.

MG17Logger Control Property List

See Also Example Code

Visual Basic Syntax

Property **List**(lIndex As Long) As String

Returns

MG Return Code

Details

This read only property lists the items contained in the Event Log.

MG17Logger Control Property ListCount

See Also Example Code

Visual Basic Syntax

Property **ListCount** As Long

Returns

MG Return Code

Details

This property returns the number of items currently displayed in the Event Log.

MG17Logger Control Property ListIndex

See Also Example Code

Visual Basic Syntax

Property **ListIndex** As Long

Returns

MG Return Code

Details

This property returns and sets the index of the currently selected item in the Event Log.

Introduction - MG17EventLog Programming

The MG17EventLog is a member object to the MG17Logger Control, which allows event log manipulation without the need to create and display a permanent logger control within the application.

The Event Log assists in the development of client applications by combining traditional error and call logs into one object. Logged information can be displayed either by the MG17EventLog dialog box or in the MG17Logger control window, or can be saved to a text file.

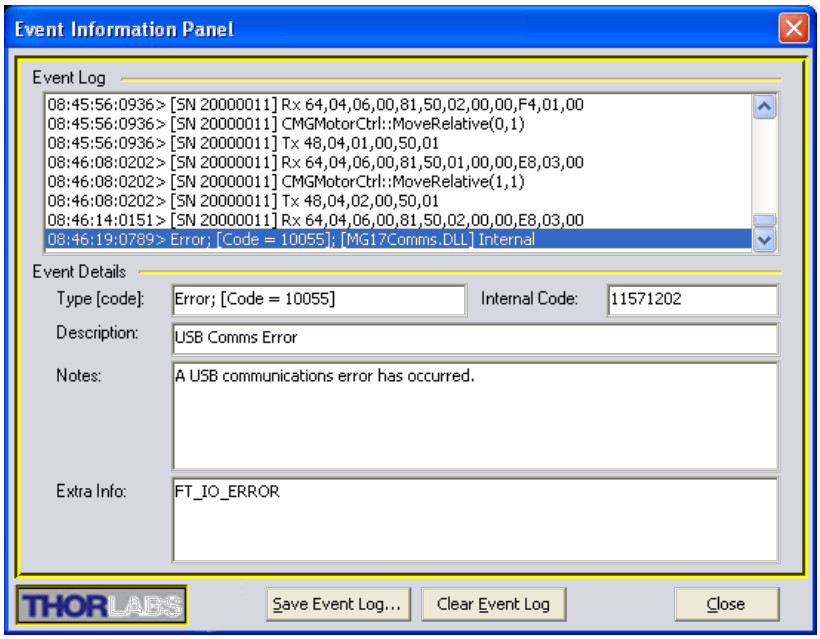
Routine information, such as method calls, is displayed by default, but the log can also be set to log hardware communications and status messages as required. This is achieved using the APT Config utility.

When using the MG17EventLog object to assist in client application development, the ShowEventInfoDlg method typically is set to TRUE, so that event information is displayed automatically when an event occurs. If required, the ShowInfoDlgOnEventFlags property can be called to display the dialog box only on the occurrence of specific types of event, e.g. errors.

Events are categorized as normal, error, warning or custom in order to aid development and debugging of applications.

The information stored in the Event Log can be viewed by calling the ShowEventInfoDlg method or saved to a text file using the SaveLog method. The maximum number of items logged can be specified using the MaxLogSize property.

The screen shot below shows typical information displayed after a sequence of calls into the server.



As a tool used during client application development the Event Log is useful for checking that the sequence of calls made into the server and the values of parameters passed are as intended,

e.g., if a motorised stage does not move over the expected distance the reason for the error can be traced using the call log to view the method parameters which initiated the move.

See the individual methods and properties for further information.

MethodAddItem

See Also Example Code

Visual Basic Syntax

Sub AddItem(bstrItem As String)

IDL Syntax

??????????

Returns

MG return code

Details

This property adds an item to the list.

MG17EventLog Method AddErrorItem

See Also**Example Code**

Visual Basic Syntax

Function **AddErrorItem**(lCode As Long, bstrFuncName As String, bstrExtraInfo As String, lInternalCode As Long, bSuppressInfoDlg As Boolean) As Long

IDL Syntax

HRESULT ([in] BSTR bstrFuncName, [in] BSTR bstrExtraInfo, [in] LONG lInternalCode, [in] VARIANT_BOOL bSuppressInfoDlg, [out, retval] LONG* plRetVal)

Parameters

bstrFuncName - the name of the function in the custom application calling the AddNormalItem method

bstrExtraInfo - extra information applicable to the event

lInternalCode - unique identifier specifying the position in the application code where the event was called

bSuppressInfoDlg - information dialog box status

Returns

MG Return Code

Details

This method adds an error event item to the event log, with minimal parameters specified. To add an event with all parameters, see the AddItem method.

The ActiveX Drivers have been designed to include full event logging capabilities with the details of each event being stored in an Event Log. The event log can prove useful during the development of a client application to confirm the sequence of actions carried out by the ActiveX Drivers and check that this occurs as expected. It can also prove useful to confirm the integrity of any parameters that are passed through to the Drivers.

Traceability is provided by the *lInternalCode* parameter, which allows a unique code to be annotated, identifying in the application software, the exact point from which the event was called. The name of the function in the client application that raised the event is entered in the *bstrFuncName* parameter. Associated information can be added in the *bstrExtraInfo* parameter as required.

The information stored in the Event Log can be saved to disk in the form of a text file, by using the SaveLog method. Furthermore, specific types of event can be saved automatically to disk whenever they occur by using the SaveOnEventFlags method. If a save facility only is required, the Events Log dialog box can be hidden by setting the *bSuppressInfoDlg* parameter

to 'TRUE'.

The maximum number of Items that can be stored in the Event Log is limited (for resource reasons) to the value specified using the MaxLogSize property. If the number of events exceeds this maximum value then older entries are discarded to free up memory, allowing new entries to be added. The Event log is cleared using the ClearLog method and the information stored can be viewed at any time during execution of a client application by calling the ShowEventInfoDlg method.

MG17EventLog Method AddNormalItem

See Also **Example Code**

Visual Basic Syntax

Function **AddNormalItem**(bstrFuncName As String, bstrExtraInfo As String, lInternalCode As Long, bSuppressInfoDlg As Boolean) As Long

IDL Syntax

HRESULT AddNormalItem ([in] BSTR bstrFuncName, [in] BSTR bstrExtraInfo, [in] LONG lInternalCode, [in] VARIANT_BOOL bSuppressInfoDlg, [out, retval] LONG* plRetVal)

Parameters

bstrFuncName - the name of the function in the custom application calling the AddNormalItem method

bstrExtraInfo - extra information applicable to the event

lInternalCode - unique identifier specifying the position in the application code where the event was called

bSuppressInfoDlg - information dialog box status

Returns

MG Return Code

Details

This method adds a normal event item to the event log, with minimal parameters specified. To add an event with all parameters, see the AddItem method.

The ActiveX Drivers have been designed to include full event logging capabilities with the details of each event being stored in an Event Log. The event log can prove useful during the development of a client application to confirm the sequence of actions carried out by the ActiveX Drivers and check that this occurs as expected. It can also prove useful to confirm the integrity of any parameters that are passed through to the Drivers.

Traceability is provided by the *lInternalCode* parameter, which allows a unique code to be annotated, identifying in the application software, the exact point from which the event was called. The name of the function in the client application that raised the event is entered in the *bstrFuncName* parameter. Associated information can be added in the *bstrExtraInfo* parameter as required.

The information stored in the Event Log can be saved to disk in the form of a text file, by using the SaveLog method. Furthermore, specific types of event can be saved automatically to disk whenever they occur by using the SaveOnEventFlags method. If a save facility only is required, the Events Log dialog box can be hidden by setting the *bSuppressInfoDlg* parameter to 'TRUE'.

The maximum number of Items that can be stored in the Event Log is limited (for resource reasons) to the value specified using the MaxLogSize property. If the number of events exceeds this maximum value then older entries are discarded to free up memory, allowing new entries to be added. The Event log is cleared using the ClearLog method and the information stored can be viewed at any time during execution of a client application by calling the ShowEventInfoDlg method.

MG17EventLog Method AddWarningItem

See Also Example Code

Visual Basic Syntax

Function **AddWarningItem**(lCode As Long, bstrFuncName As String, bstrExtraInfo As String, lInternalCode As Long, bSuppressInfoDlg As Boolean) As Long

IDL Syntax

HRESULT AddWarningItem ([in] BSTR bstrFuncName, [in] BSTR bstrExtraInfo, [in] LONG lInternalCode, [in] VARIANT_BOOL bSuppressInfoDlg, [out, retval] LONG* plRetVal)

Parameters

bstrFuncName - the name of the function in the custom application calling the AddNormalItem method

bstrExtraInfo - extra information applicable to the event

lInternalCode - unique identifier specifying the position in the application code where the event was called

bSuppressInfoDlg - information dialog box status

Returns

MG Return Code

Details

This method adds a warning event item to the event log, with minimal parameters specified. To add an event with all parameters, see the AddItem method.

The ActiveX Drivers have been designed to include full event logging capabilities with the details of each event being stored in an Event Log. The event log can prove useful during the development of a client application to confirm the sequence of actions carried out by the ActiveX Drivers and check that this occurs as expected. It can also prove useful to confirm the integrity of any parameters that are passed through to the Drivers.

Traceability is provided by the *lInternalCode* parameter, which allows a unique code to be annotated, identifying in the application software, the exact point from which the event was called. The name of the function in the client application that raised the event is entered in the *bstrFuncName* parameter. Associated information can be added in the *bstrExtraInfo* parameter as required.

The information stored in the Event Log can be saved to disk in the form of a text file, by using the SaveLog method. Furthermore, specific types of event can be saved automatically to disk whenever they occur by using the SaveOnEventFlags method. If a save facility only is required, the Events Log dialog box can be hidden by setting the *bSuppressInfoDlg* parameter to 'TRUE'.

The maximum number of Items that can be stored in the Event Log is limited (for resource reasons) to the value specified using the MaxLogSize property. If the number of events exceeds this maximum value then older entries are discarded to free up memory, allowing new entries to be added. The Event log is cleared using the ClearLog method and the information stored can be viewed at any time during execution of a client application by calling the ShowEventInfoDlg method.

MG17EventLog Method ClearLog

See Also Example Code

Visual Basic Syntax

Function **ClearLog**() As Long

IDL Syntax

HRESULT ClearLogFile ([out, retval] LONG* plRetVal)

Returns

MG Return Code

Details

Clears the event log of all items

Refer to the ShowEventInfoDlg method for a detailed description of the Event Log.

MG17EventLog Method ClearLogFile

See Also **Example Code**

Visual Basic Syntax

Function **ClearLogFile**() As Long

IDL Syntax

HRESULT ClearLogFile ([out, retval] LONG* plRetVal)

Returns

MG Return Code

Details

Deletes all information stored in the event log disk file.

Refer to the ShowEventInfoDlg method for a detailed description of the Event Log.

MG17EventLog Method GetEventInfo

See Also **Example Code**

Visual Basic Syntax

Function **GetEventInfo**(lIndex As Long, lCode As Long, bstrFuncName As String, bstrDesc As String, bstrNotes As String, bstrExtraInfo As String, lInternalCode As Long) As Long

IDL Syntax

HRESULT GetEventInfo ([in] LONG lIndex, [out] LONG* lCode, [out] BSTR* bstrFuncName, [out] BSTR* bstrDesc, [out] BSTR* bstrNotes, [out] BSTR* bstrExtraInfo, [out] LONG* lInternalCade, [out, retval] LONG* plRetVal)

Parameters

lIndex - the index number in the event log, of the event to be returned

lCode - event identification number

bstrFuncName - the name of the function in the custom application calling the AddNormalItem method

bstrDesc - description of the event

bstrNotes - notes applicable to the event

bstrExtraInfo - extra information applicable to the event

lInternalCode - unique identifier specifying the position in the application code where the event was called

Returns

MG Return Code

Details

Returns relevant information about the event item specified by the *lIndex* parameter. The number of events in the log is obtained by the LogSize property.

For more details on the information contained in the event log, see the AddItem, AddErrorItem, AddWarningItem or AddNormalItem methods.

MG17EventLog Method GetLastEventInfo

See Also **Example Code**

Visual Basic Syntax

Function **GetLastEventInfo**(lCode As Long, bstrFuncName As String, bstrDesc As String, bstrNotes As String, bstrExtraInfo As String, lInternalCode As Long) As Long

IDL Syntax

HRESULT GetLastEventInfo ([out] LONG* lCode, [out] BSTR* bstrFuncName, [out] BSTR* bstrDesc, [out] BSTR* bstrNotes, [out] BSTR* bstrExtraInfo, [out] LONG* lInternalCade, [out, retval] LONG* plRetVal)

Parameters

lCode - event identification number

bstrFuncName - the name of the function in the custom application calling the AddNormalItem method

bstrDesc - description of the event

bstrNotes - notes applicable to the event

bstrExtraInfo - extra information applicable to the event

lInternalCode - unique identifier specifying the position in the application code where the event was called

Returns

MG Return Code

Details

Returns relevant information about the most recent event item in the event log.

For more details on the information contained in the event log, see the AddItem, AddErrorItem, AddWarningItem or AddNormalItem methods.

MG17EventLog Method SaveLog

See Also **Example Code**

Visual Basic Syntax

Function **SaveLog**() As Long

IDL Syntax

HRESULT SaveLog ([out, retval] LONG* plRetVal)

Returns

MG Return Code

Details

Saves all items in the event log to a disk file specified by the LogFileNamePath property.

Refer to the ShowEventInfoDlg method for a detailed description of the Event Log.

MG17EventLog Method ShowEventInfoDlg

See Also!AL("EventItem",0,`',`') **Example Code**

Visual Basic Syntax

Function **ShowEventInfoDlg**() As Long

IDL Syntax

HRESULT ShowEventInfoDlg ([out, retval] LONG* plRetVal)

Returns

MG Return Code

Details

Shows the event log information panel.

The ActiveX Drivers have been designed to include full event logging capabilities with the details of each event being stored in an Event Log. The event log can prove useful during the development of a client application to confirm the sequence of actions carried out by the ActiveX Drivers and check that this occurs as expected. It can also prove useful to confirm the integrity of any parameters that are passed through to the Drivers.

The information stored in the Event Log can be saved to disk in the form of a text file, by using the SaveLog method. Furthermore, specific types of event can be saved automatically to disk whenever they occur by using the SaveOnEventFlags method.

The maximum number of Items that can be stored in the Event Log is limited (for resource reasons) to the value specified using the MaxLogSize property. If the number of events exceeds this maximum value then older entries are discarded to free up memory, allowing new entries to be added. The Event log is cleared using the ClearLog.

For details on adding events to the event log, see the AddItem, AddErrorItem, AddWarningItem or AddNormalItem methods.

MG17EventLog Property LogFileNamePath

See Also Example Code

Visual Basic Syntax

Property **LogFileNamePath** As String

IDL Syntax

HRESULT LogFileNamePath ([out, retval] BSTR* pVal)

HRESULT LogFileNamePath ([in] BSTR newVal)

Returns

MG Return Code

Details

This property returns and sets the name of the event log disk file used by the SaveLog method.

MG17EventLog Property LogSize

See Also Example Code

Visual Basic Syntax

Property **LogSize** As Long

IDL Syntax

HRESULT LogSize ([out, retval] LONG* pVal)

Returns

MG Return Code

Details

This read only property returns the number of event items currently stored in the event log.

See the MaxLogSize property for details on setting the size of the event log.

MG17EventLog Property MaxLogSize

See Also Example Code

Visual Basic Syntax

Property **MaxLogSize** As Long

IDL Syntax

HRESULT MaxLogSize ([out, retval] LONG* pVal)

HRESULT MaxLogSize ([in] LONG newVal)

Returns

MG Return Code

Details

This property returns and sets the maximum number of items that can be stored in the event log.

See the LogSize property for details on obtaining the number of entries in the event log.

MG17EventLog Property SaveOnEventFlags

See Also Example Code

Visual Basic Syntax

Property **SaveOnEventFlags** As Long

IDL Syntax

HRESULT SaveOnEventFlags ([in] LONG newVal)

Returns

MG Return Code

Details

This property returns and sets flags that automatically save the event log to disk file when a specified type of event item is added to the log.

Flags can be set as follows:

0 - None

1. - Save on normal events
2. - Save on warning events
3. - Save on normal or warning events
4. - Save on error events
5. - Save on error or normal events
6. - Save on error or warning events
7. - Save on error, warning or normal events

MG17EventLog Property ShowInfoDlgOnEventFlags

See Also Example Code

Visual Basic Syntax

Property **ShowInfoDlgOnEventFlags** As Long

IDL Syntax

HRESULT ShowInfoDlgOnEventFlags ([out, retval] LONG* pVal)

Returns

MG Return Code

Details

This property returns and sets flags that determine if the event information panel is displayed when a specified type of event item is added to the event log.

Flags can be set as follows:

0 - None

1. - Show on normal events
2. - Show on warning events
3. - Show on normal or warning events
4. - Show on error events
5. - Show on error or normal events
6. - Show on error or warning events
7. - Show on error, warning or normal events

Introduction to Motor Control Programming

The 'Motor' ActiveX Control provides the functionality required for a client application to control one or more of the APT series of motor controller units. This range of motor controllers covers DC servo and stepper drivers in a variety of formats including compact Cube type controllers, benchtop units and 19" rack based modular drivers.

The Solenoid Controller units (KSC101 and TSC001) are also accessed using the Motor ActiveX Control.

A single Motor ActiveX Control instance can be used to control any of the aforementioned controllers. To specify the particular controller being addressed, every unit is factory programmed with a unique 8-digit serial number. This serial number is key to the operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus. The serial number must be specified using the HWSerialNum property before an ActiveX control instance can communicate with the hardware unit. This can be done at design time or at run time. Note that the appearance of the ActiveX Control GUI (graphical user interface) will change to the required format when the serial number has been entered.
The Methods and Properties of the Motor ActiveX Control can be used to perform activities such as homing stages, absolute and relative moves, changing velocity profile settings and operation of the solenoid state (TSC001 T-Cube). With a few exceptions, these methods are generic and apply equally to both single and dual channel units.
Where applicable, the target channel is identified in the *lChanID* parameter and on single channel units, this must be set to CHAN1_ID. On dual channel units, this can be set to CHAN1_ID, CHAN2_ID or CHANBOTH_ID as required. See the HWCHANNEL enumeration for more details.
For details on the operation of the stepper motor controller, refer to the manual for the unit, which is available from www.thorlabs.com. **Motor Control Method Summary**

This topic contains a brief description of the methods contained in the motor ActiveX control. A detailed description of the methods, properties and associated enumerations is given in subsequent topics (see contents list). Each method or property name and associated parameter list is written using Visual Basic syntax.

Note. Some methods use boolean parameters to specify or return certain settings. Boolean parameters in the ActiveX Drivers have been implemented as long (32bit) Integers. In this way when passing boolean parameters a TRUE setting is defined as a non zero value and a FALSE setting is defined as a zero value. When methods return boolean parameters a TRUE setting will be indicated by the value 1 and a FALSE setting by the value 0.

Method Name	Description	Applicable APT Controllers (model part number 3 letter prefix)
CalibrateEnc	Calibrates encoder equipped stage.	BSC, BMS, MST
DeleteParamSet	Deletes stored settings for specific controller.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
DisableHWChannel	Disables the drive output.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
DoEvents	Allows client application to process other activity.	BSC, BMS, MST, ODC, OST, TDC, TST, TSC
EnableEventDlg	Enables the Event Dialog Box for all methods	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
EnableHWChannel	Enables the drive output.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetAbsMovePos	Gets the absolute move position.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetAbsMovePos_AbsPos	Gets the absolute move position (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST

GetBLashDist	Gets the backlash distance.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetBLashDist_BLashDist	Gets the backlash distance (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetButtonParams	Gets the front panel button settings (Cube drivers).	ODC, OST, TDC, TST
GetChannelSwitch	Gets the GUI channel switch position.	BSC, BMS, MST (Dual Channel Units Only)
GetCtrlStarted	Gets the ActiveX Control started flag.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
GetDCCurrentLoopParams	Gets the present settings for the current PI loop	BBD
GetDCMotorOutputParams	Gets the settings for the DC Motor Output	BBD
GetDCPositionLoopParams	Gets the present settings for the position PID loop	BBD
GetDCProfileModeParams	Gets the settings for the DC Motor Move Profile	BBD
GetDCTrackSettleParams	Gets the settings for the Track/Settle function	BBD
GetDispMode	Gets the GUI display mode.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetEncCalibTableParams	Gets the encoder calibration table parameters for encoder equipped stages.	BSC, BMS, MST
GetEncPosControlParams	Gets the encoder position control parameters for encoder equipped stages.	BSC, BMS, MST
GetEncPosCorrectParams	Gets the encoder position correction parameters for encoder equipped stages.	BSC, BMS, MST
GetHomeParams	Gets the homing sequence parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetHomeParams_HomeVel	Gets the homing velocity parameter (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetHomeParams_ZeroOffset	Gets the homing zero offset parameter (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetHWCommsOK	Gets the hardware communications OK flag.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetHWLimSwitches	Gets the limit switch configuration settings.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetIndicatorLEDMode	Gets the front panel indication LED operating modes (Cube drivers).	ODC, OST, TDC, TST
GetJogMode	Gets the jogging button operating modes.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetJogMode_Mode	Get the jogging button operating mode (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetJogMode_StopMode	Gets the jogging button stopping mode (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetJogStepSize	Gets the jogging step size.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetJogStepSize_StepSize	Gets the jogging step size (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetJogVelParams	Gets the jogging velocity profile parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetJogVelParams_Accn	Gets the jogging acceleration parameter (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetJogVelParams_MaxVel	Gets the jogging maximum velocity parameter (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetMotorParams	Gets the motor gearing parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetParentHWInfo ;	Gets the identification information of the host controller.	BBD, BSC102/3, MST
GetPhaseCurrents	Gets the coil phase currents.	BSC, BMS, MST, OST, TST
GetPIDParams_Deriv	Gets the servo control loop derivative parameter (DC servo controllers - returned by value).	ODC, TDC
GetPIDParams_Int	Gets the servo control loop integration parameter (DC servo controllers - returned by value).	ODC, TDC
GetPIDParams_Prop	Gets the servo control loop proportional parameter (DC servo controllers - returned by value).	ODC, TDC
GetPosition	Gets the current motor position.	BBD, BSC, BMS, MST, ODC,

		OST, TDC, TST
GetPosition_Position	Gets the current motor position (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetPositionEx	Gets the current motor position.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetPositionEx_UncalibPosition	Gets the current uncalibrated motor position (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetPositionOffset	Gets the motor position offset.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetPotParams	Gets the velocity control potentiometer parameters (Cube drivers).	ODC, OST, TDC, TST
GetRelMoveDist	Gets the relative move distance.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetRelMoveDist_RelDist	Gets the relative move distance (returned by reference).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetRotStageModes	Gets the move direction for rotational stages	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetStageAxis	Gets the stage type information associated with the motor under control.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetStageAxisInfo	Gets the stage axis parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetStageAxisInfo_MaxPos	Gets the stage maximum position (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetStageAxisInfo_MinPos	Gets the stage minimum position (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetStatusBits_Bits	Gets the controller status bits encoded in 32 bit integer (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetTriggerParams	Gets the move triggering parameters.	BSC, BMS, MST
GetVelParamLimits	Gets the maximum velocity profile parameter limits.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetVelParams	Gets the velocity profile parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetVelParams_Accn	Gets the move acceleration (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
GetVelParams_MaxVel	Gets the move maximum velocity (returned by value).	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
Identify	Identifies the controller by flashing unit LEDs.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
LLGetDigIPs	Gets digital input states encoded in 32 bit integer.	BBD, BSC, BMS, MST
LLGetEncoderCount	Gets the encoder counts set for the associated position	BBD, TDC
LLGetHostStatusBits	Gets the base unit status bits encoded in 32 bit integer.	BBD, BSC,
LLGetStatusBits	Gets the status bits encoded in 32 bit integer for the specified channel	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
LLSetEncoderCount	Sets the encoder counts for the associated position	BBD, TDC
LLSetGetDigOPs	Sets or Gets the channel card digital output bits encoded in 32 bit integer.	BBD, BSC, MST
LLSetGetHostDigOPs	Sets or Gets the base unit user digital output bits encoded in 32 bit integer.	BBD, BSC,
LLSetGetPIDParams	Sets or Gets the servo control loop PID parameters (DC servo controllers).	ODC, TDC
LoadParamSet	Loads stored settings for specific controller.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
MoveAbsolute	Initiates an absolute move.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveAbsoluteEnc	Initiates an absolute move with specified positions for encoder equipped stages.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveAbsoluteEx	Initiates an absoloute move with specified positions.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveAbsoluteRot	Initiates an absolute move with specified positions for rotary stages.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveHome	Initiates a homing sequence.	BBD, BSC, BMS, MST, ODC,

		OST, TDC, TST
MoveJog	Initiates a jog move.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveRelative	Initiates a relative move.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveRelativeEnc	Initiates a relative move with specified distances for encoder equipped stages.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveRelativeEx	Initiates a relative move with specified distances.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
MoveVelocity	Initiates a move at constant velocity with no end point.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SaveParamSet	Saves settings for a specific controller.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
SC_Disable	Disables (turns off) the solenoid.	TSC
SC_Enable	Enables (turns on) the solenoid.	TSC
SC_GetCycleParams	Gets solenoid parameters for automated on/off operation.	TSC
SC_GetOperatingMode	Gets the solenoid operating mode.	TSC
SC_GetOPState	Gets the activity state of the solenoid.	TSC
SC_SetCycleParams	Sets solenoid parameters for automated on/off operation.	TSC
SC_SetOperatingMode	Sets the solenoid operating mode.	TSC
SetAbsMovePos	Sets the absolute move position.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetBLashDist	Sets the backlash distance.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetButtonParams	Sets the front panel button settings (Cube drivers).	ODC, OST, TDC, TST
SetChannelSwitch	Sets the GUI channel switch position.	BSC, BMS, MST (Dual Channel Units Only)
SetDispMode	Sets the GUI display mode.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetDCCurrentLoopParams	Specifies the settings for the current PI loop	BBD
SetDCMotorOutputParams	Specifies the settings for the DC Motor Output	BBD
SetDCPositionLoopParams	Specifies the settings for the position PID loop	BBD
SetDCProfileModeParams	Specifies the settings for the DC Motor Move Profile	BBD
SetDCTrackSettleParams	Specifies the settings for the Track/Settle function	BBD
SetEncCalibTableParams	Sets the encoder calibration table parameters for encoder equipped stages.	BSC, BMS, MST
SetEncPosControlParams	Sets the encoder position control parameters for encoder equipped stages.	BSC, BMS, MST
SetEncPosCorrectParams	Sets the encoder position correction parameters for encoder equipped stages.	BSC, BMS, MST
SetHomeParams	Sets the homing sequence parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetHWLimSwitches	Sets the limit switch configuration settings.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetIndicatorLEDMode	Sets the front panel indication LED operating modes (Cube drivers).	ODC, OST, TDC, TST
SetJogMode	Sets the jogging button operating modes.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetJogStepSize	Sets the jogging step size.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetJogVelParams	Sets the jogging velocity profile parameters.	BBD, BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetMotorParams	Sets the motor gearing parameters.	BSC, BMS, MST, ODC, OST, TDC, TST
SetPhaseCurrents	Sets the coil phase currents.	BSC, BMS, MST, OST, TST
SetPositionOffset	Sets the motor position offset.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetPotParams	Sets the velocity control potentiometer parameters (Cube drivers).	ODC, OST, TDC, TST
SetRelMoveDist	Sets the relative move distance.	BBD, BSC, BMS, MST, ODC,

		OST, TDC, TST
SetRotStageModes	Sets the move direction for rotational stages	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetStageAxisInfo	Sets the stage axis parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
SetTriggerParams	Sets the move triggering parameters.	BSC, BMS, MST
SetVelParams	Sets the velocity profile parameters.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
ShowEventsDlg	Display the Event Dialog panel.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
ShowSettingsDlg	Display the GUI Settings panel.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
StartCtrl	Starts the ActiveX Control (starts communication with controller)	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
StopCtrl	Stops the ActiveX Control (stops communication with controller)	BBD, BSC, BMS, MST, ODC, OST, TDC, TST, TSC
StopImmediate	Stops a motor move immediately.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST
StopProfiled	Stops a motor move in a profiled (decelleration) manner.	BBD, BSC, BMS, MST, ODC, OST, TDC, TST

Piezo Enumeration APT_PARENT_HWTYPES

This enumeration contains constants that specify the hardware type of the enclosure associated with the ActiveX control.

Constant	Name	Purpose
28	USB_BPC003	3. Channel, 75V Benchtop Piezo Controller
29	ETHNET_MMR601	6 Bay Modular Rack, Ethernet
30	USB_MMR601	6 Bay Modular Rack, USB

Motor Enumeration *BRAKE_STATE*

This enumeration contains constants that specify the state of the brake.

Constant	Name	Purpose
1.	BRAKE_ON	Brake ON
2.	BRAKE_OFF	Brake OFF

Motor Enumeration *BUTTONMODE*

This enumeration contains constants that specify the operating mode of the front panel buttons.

Constant	Name	Purpose
1.	BUTTONMODE_JOG	Sets the operating mode to ‘Jogging’
2.	BUTTONMODE_POSITION	Sets the operating mode to ‘Go To Position’

Motor Enumeration *DISPLAYMODE*

This enumeration contains constants that specify the display mode of the virtual display panel.

Constant	Name	Purpose
1.	DISPMODE_PANEL	Display set to panel view
2.	DISPMODE_GRAPH	Display set to graph view
3.	DISPMODE_POSITION	Display set to position view

Motor Enumeration *ENCPOSCONTROLMODE*

This enumeration contains constants that specify the control mode for position correction when a move is initiated on the channel specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	ENC_POSCTRL_DISABLED	Encoded position correction disabled
2.	ENC_POSCTRL_POSITION	'Position (& Correct)' mode selected
3.	ENC_POSCTRL_POSITIONSTOPSHORT	'Position (Stop Short & Correct)' mode selected

Motor Enumeration *ENCPOSSOURCEMODE*

This enumeration contains constants that specify the source for positional information when a move is initiated on the channel specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	ENC_POSSRC_MICROSTEP	Microstep count
2.	ENC_POSSRC_ENCODER	Encoder count

Motor Enumeration *ENCQEPSENSE*

This enumeration contains constants that specify the sense of the encoder signals being read by the control electronics when an encoded move is initiated on the channel specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	ENC_QEP_POSITIVE	Positive
-1	ENC_QEP_NEGATIVE	Negative

Motor Enumeration *HOMEDIR*

This enumeration contains constants that specify the direction moved when the channel, specified in the HWCHANNEL enumeration, is homing.

Constant	Name	Purpose
1.	HOME_FWD	Homes in the forward direction
2.	HOME_REV	Homes in the reverse direction

Motor Enumeration *HOMELIMSWITCH*

This enumeration contains constants that specify the limit switch associated with the home position, for the channel, specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	HOMELIMSW_REV_HW	Identifies the reverse hardware limit switch as the home limit
4	HOMELIMSW_FWD_HW	Identifies the forward hardware limit switch as the home limit

Motor Enumeration *HWCHANNEL*

Note. This enumeration is applicable only to APT units that are channel based.

This enumeration contains constants that specify individual channels on an APT hardware unit.

Constant	Name	Purpose
0	CHAN1_ID	Selects channel 1
1.	CHAN2_ID	Selects channel 2 (dual channel units only)
10	CHANBOTH_ID	Selects both channels (dual channel units only)

Notes.

The methods are generic and apply equally to both single and dual channel units. On single channel units, the *lChanID* parameter must be set to CHAN1_ID. On dual channel units, this can be set to CHAN1_ID, CHAN2_ID or CHANBOTH_ID as required.

Some methods that contain an lChanID parameter do not accept CHANBOTH_ID as a valid value. In this case, an error value is returned.

Motor Enumeration *HWLIMSWITCH*

This enumeration contains constants that specify the action that the hardware limit switches for the channel specified in the HWCHANNEL enumeration, will take when the switch is contacted.

Constant	Name	Purpose
1.	HWLIMSW_IGNORE	The limit is ignored
2.	HWLIMSW_MAKES	The limit switch closes on contact
3.	HWLIMSW_BREAKS	The limit switch opens on contact

Motor Enumeration *INDICATORLEDMODE*

This enumeration contains constants that specify the operating mode of the front panel LEDs.

Constant	Name	Purpose
1.	LEDMODE_IDENT	The LED will flash when the 'Ident' button is clicked on the APT Software GUI panel.
2.	LEDMODE_LIMITSWITCH	The LED will flash when the motor reaches a forward or reverse limit switch.
8	LEDMODE _MOVING	The LED is lit when the motor is moving.

Motor Enumeration *JOGMODE*

Jog commands can be issued via the Motor Control GUI panel or by using a remote handset.
This enumeration contains constants that specify the jog mode applicable to the channels specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	JOG_CONTINUOUS	Only applies to remote handset operation. When a jog command is received, the motor continues to move until the jog signal is removed (i.e the jog button is released).
2.	SINGLE_STEP	When a jog command is received, the motor moves by the step size specified in the SetJogStepSize method (or the GUI).

Motor Enumeration *KMOTTRIGMODE*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating mode of the trigger.

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a relative, absolute or home move as follows:

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Constant	Name	Purpose
0X00	KMOT_TRIG_DISABLE	The trigger IO is disabled
0X01	KMOT_TRIG_IN_GPI	General purpose logic input (read through status bits using the LLGetStatusBits method).
0X02	KMOT_TRIG_IN_RELMOVE	Input trigger for relative move.
0X03	MOT_TRIG_IN_ABSMOVE	Input trigger for absolute move.
0X04	MOT_TRIG_IN_HOME	Input trigger for home move.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output, or to indicate motion status or to produce a trigger pulse at configurable positions as follows:

Constant	Name	Purpose
0X0A	KMOT_TRIG_OUT_GPO	General purpose logic output (set using the LLSetGetDigOPs method).
0X0B	KMOT_TRIG_OUT_INMOTION	Trigger output active (level) when motor 'in motion'. The output trigger goes high (5V) or low (0V) (as set in the SetKCubeTriggerParams method.
0X0C	KMOT_TRIG_OUT_MAXVELOCITY	Trigger output active (level) when motor at 'max velocity'.
0X0D	MOT_TRIG_OUT_POSSTEPSFWD	Trigger output active (pulsed) at pre-defined positions moving forward (set using the SetKCubePosTriggerParams method). Only one Trigger port at a time can be

		set to this mode.
0X0E	MOT_TRIG_OUT_POSSTEPSREV	Trigger output active (pulsed) at pre-defined positions moving backwards (set using the SetKCubePosTriggerParams method). Only one Trigger port at a time can be set to this mode.
0X0F	MOT_TRIG_OUT_POSSTEPSBOTH	Trigger output active (pulsed) at pre-defined positions moving forwards and backward. Only one Trigger port at a time can be set to this mode.

Motor Enumeration KMOTTRIGPOLARITY

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating polarity of the trigger.

Constant	Name	Purpose
0X01	KMOT_TRIGPOL_HIGH	The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).
0X02	KMOT_TRIGPOL_LOW	The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

Motor Enumeration KMOTWHEELDIRSENSE

This enumeration contains constants used by the SetKCubePanelParams method, that specify the direction sense of a move initiated by the velocity wheel on the top panel of the unit.

Constant	Name	Purpose
0	KMOT_WHEEL_DIRSENSE_DISABLED	The wheel is disabled.
1	KMOT_WHEEL_DIRSENSE_POS	Upwards rotation of the wheel results in a positive motion (i.e. increased position count).
2	KMOT_WHEEL_DIRSENSE_NEG	The following option applies only when the Wheelmode is set to Velocity Control Mode (1) the SetKCubePanelParams method. If set to Jog Mode (2) or Go to Position Mode (3), the following option is ignored. Upwards rotation of the wheel results in a negative motion (i.e. decreased position count).

Motor Enumeration KMOTWHEELMODE

This enumeration contains constants used by the SetKCubePanelParams method, that specify the operating mode of the wheel on the top panel of the unit.

Constant	Name	Purpose
1	KMOT_WHEEL_MODE_VEL	Deflecting the wheel starts a move with the velocity proportional to the deflection.
2	KMOT_WHEEL_MODE_JOG	Deflecting the wheel initiates a jog move, using the parameters specified by the SetJogStepSize and SetJogVelParams methods. Keeping the wheel deflected repeats the move automatically after the current move has completed.
3	KMOT_WHEEL_MODE_GOTOPOS	Deflecting the wheel starts a move from the current position to one of the two predefined “teach” positions.

Motor Enumeration *MOTORJOGDIRECTION*

This enumeration contains constants that specify the direction moved when a jog command is initiated on the channel, specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	JOG_FWD	Moves in the forward direction
2.	JOG_REV	Moves in the reverse direction

Motor Enumeration *MOVEVELDIR*

This enumeration contains constants that specify the direction the channel specified in the HWCHANNEL enumeration will move, when the MoveVelocity method is called.

Constant	Name	Purpose
1.	MOVE_FWD	Move in a forward direction.
2.	MOVE_REV	Move in a reverse direction.

Motor Enumeration *ROTMOVEMODE*

This enumeration contains constants that specify the move mode of a rotation stage.

Constant	Name	Purpose
1.	ROT_MOVE_POS	Rotate Positive
2.	ROT_MOVE_NEG	Rotate Negative
3.	ROT_MOVE_SHORT	Rotate Quickest

Motor Enumeration *ROTPOSDISPMODE*

This enumeration contains constants that specify the way in which rotation stages report position information. It is called by the SetRotStageModes method.

Constant	Name	Purpose
1.	ROT_POSDISP_360	The stage position is in the range 0 to 360. The degree count is reset on each rotation.
2.	ROT_POSDISP_TOTAL	The count is the total number of degrees rotated. The count is not reset.

Solenoid Controller Enumeration *SOLOPERATINGMODE*

This enumeration contains constants that specify operating mode of the solenoid controller.

Constant	Name	Purpose
1.	SOLENOID_MANUAL	Selects manual mode
2.	SOLENOID_SINGLE	Selects Single Shot mode
3.	SOLENOID_AUTO	Selects Auto mode
4.	SOLENOID_TRIGGER	Selects Trigger mode

Solenoid Controller Enumeration *SOLOUTPUTSTATE*

This enumeration contains constants that specify the output state of the controller.

Constant	Name	Purpose
1.	OUTPUTSTATE_ON	TSC_OUTPUT_ON
2.	OUTPUTSTATE_OFF	TSC_OUTPUT_OFF

Motor Enumeration *STAGEAXIS*

This enumeration contains constants that specify the stage axis applicable to the channels specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	AXIS_UNKNOWN	
2.	AXIS_SINGLE	

3.	AXIS_ROTARY	Identifies a rotary stage
16	AXIS_X	Identifies the X axis
17	AXIS_Y	Identifies the Y axis
18	AXIS_Z	Identifies the Z axis
19	AXIS_PITCH	Identifies the Pitch axis
20	AXIS_ROLL	Identifies the Roll axis
21	AXIS_YAW	Identifies the Yaw axis

Motor Enumeration *STAGETYPE*

This enumeration contains constants that specify the stage TYPE applicable to the channels specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	TYPE_UNKNOWN	
2.	TYPE_17APT600R	
3.	TYPE_17APT600L	
16	TYPE_NANOMAX300	

Motor Enumeration *STAGEDIRSENSE*

This enumeration contains constants that specify the direction in which the motor associated with the channel specified in the HWCHANNEL enumeration, will rotate when a positive move is demanded.

Constant	Name	Purpose
1.	MOVE_POS_IS_CW	Clockwise is positive
2.	MOVE_POS_IS_CCW	Counter clockwise is positive

Motor Enumeration *STAGEUNITS*

This enumeration contains constants that specify the units of travel for the stage associated with the channel specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	UNITS_MM	Travel is in mm
2.	UNITS_DEG	Travel is in degrees

Motor Enumeration *STOPMODE*

This enumeration contains constants that specify the way in which the motor associated with the channel specified in the HWCHANNEL enumeration, will stop when the jog signal is removed.

Constant	Name	Purpose
1.	STOP_IMMEDIATE	Moves are stopped abruptly in a non-profiled manner, (i.e. the parameter *fAccn*, set using the SetVelParams method, is ignored) and there is a risk that steps, and therefore positional integrity, could be lost.
2.	STOP_PROFILED	Moves are stopped in a profiled manner using the velocity profile parameters set using the SetVelParams method.

Motor Enumeration *SWLIMITMODE*

This enumeration contains constants that specify the way in which the channels specified in the HWCHANNEL enumeration will stop when the software limit is reached

Constant	Name	Purpose
1.	SWLIMIT_IGNORE	Ignores the software limit
2.	SWLIMIT_IMMEDIATESTOP	Stops the motor abruptly, in a non-profiled manner
3.	SWLIMIT_PROFILEDSTOP	Stops the motor in a profiled manner using the velocity profile parameters set using the SetVelParams method.

Note. It is good programming practice to use 'PROFILEDSTOP'. If IMMEDIATESTOP is used, the motor could slip and positional integrity could be lost.

Motor Enumeration *TRIGMODE*

This enumeration contains constants that specify the trigger mode on the channel specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	TRIGMODE_DISABLED	Triggering operation is disabled
2.	TRIGMODE_IN	A move is initiated on the specified channel when a trigger input signal (i.e. rising edge) is received on the Trig In connector (either BNC or D-type pin depending on the unit).
3.	TRIGMODE_INOUT	As for TRIGMODE_IN, but now a trigger output signal will also be generated on the Trig Output connector when the move begins.
4.	TRIGMODE_OUT	A trigger output signal is generated on the Trig Out connector when a move (relative, absolute or home) is initiated via software (either through the GUI panel or ActiveX method call).

Motor Enumeration *TRIGMOVE*

This enumeration contains constants that specify the type of move to be initiated when a trigger input signal is detected on the Trig In connector of the channel specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	TRIGMOVE_REL	A triggered relative move. Relative move distance and velocity profile parameters can be set prior to triggering using the SetRelMoveDist and SetVelParams methods respectively.
2.	TRIGMOVE_ABS	A triggered absolute move. Absolute move position and velocity profile parameters can be set prior to triggering using the SetAbsMovePos and SetVelParams methods respectively.
3.	TRIGMOVE_HOME	A triggered home sequence. Homing parameters can be set prior to triggering using the SetHomeParams method.

_

Motor Event EncCalibComplete

See Also Example Code

Visual Basic Syntax

Event **EncCalibComplete**(lChanID As Long)

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This event is fired by the system whenever an encoder calibration sequence, initiated using the CalibrateEnc method, has been completed.

Motor Event HomeComplete

See Also Example Code

Visual Basic Syntax

Event **HomeComplete**(lChanID As Long)

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This event is fired by the system whenever the channel specified by the *lChanID* parameter has completed its 'Home' sequence.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: If both channels are homing and complete their moves at the same time, then the *lChanID* parameter will be set to CHANBOTH_ID

Motor Event HWResponse

See Also Example Code

Visual Basic Syntax

Event **HWResponse**(lHWCode As Long, lMsgIdent As Long)

Parameters

lHWCode - Value identifying the hardware condition encountered

lMsgIdent - The internal identifier of the command that caused the problem condition to occur

Returns

MG Return Code

Details

This event is fired if the hardware unit encounters a fault, failure or warning condition. In normal operation the HWResponse event will not be fired. It is good programming practice to handle this event in case hardware problems occur.

Motor Event MoveComplete

See Also Example Code

Visual Basic Syntax

Event **MoveComplete**(lChanID As Long)

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This event is fired by the system whenever the channel specified by the *lChanID* parameter has completed its move sequence.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: If both channels are moving and complete their moves at the same time, then the *lChanID* parameter will be set to CHANBOTH_ID

Motor Event MoveStopped

See Also Example Code

Visual Basic Syntax

Event **MoveStopped**(lChanID As Long)

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This event is fired by the system whenever the move for the channel specified by the *lChanID* parameter has been stopped by using the StopImmediate or StopProfiled methods.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: If both channels are stopped simultaneously, then the *lChanID* parameter will be set to CHANBOTH_ID

Motor Event SettingsChanged

See Also Example Code

Visual Basic Syntax

Event **SettingsChanged**(lFlags As Long)

Parameters

lFlags - for future use,

Returns

MG Return Code

Details

This event is fired by the system whenever the user has altered settings via the 'Settings' button on the user GUI interface. This is useful for informing the client application of settings changes.

The event is also fired when the motor channel is enabled or disabled by using the Enable button on the GUI panel.

Note. The *lFlags* parameter is for future use and currently resides only as a placeholder.

Motor Method *CalibrateEnc*

See Also Example Code

Visual Basic Syntax

Function **CalibrateEnc**(lChanID As Long, bWait As Boolean) As Long

Parameters

lChanID – the channel identifier

bWait – wait for calibration sequence to complete if TRUE

Returns

MG Return Code

Details

When ‘Encoder Positioning’ is selected, all position displays and motor moves are based on positions derived from the encoder system fitted to the stage/actuator.

Example. The encoders currently fitted to our stages are set to 10000 counts per mm (i.e. 0.1micron per count). Therefore, to move 1 mm, the controller will drive out the appropriate number of microsteps to result in an encoder count change of 10000. To display position values, the software converts the encoder count into a 'real world' position by dividing by 10000.

For a perfect 1mm pitch lead screw, a microstep count of 25600 would equate exactly to an encoder count of 10000. However, due to lead screw pitch, non-linearity and other cyclic errors, this is not achievable in the real world. It is the purpose of the encoder feedback handling within the APT software to accommodate for this and achieve the required encoder position (a more accurate position reading).

One way to accommodate for this lead screw non-linearity is for the system to acquire a look up table of microstep count verses encoder count readings. Using this 'calibration' table the system is then able to adjust the microstep count required (to drive the motor) to achieve the required encoder count.

When this method is called (or the 'Calibrate' button on the Settings panel is clicked), the system begins the calibration sequence on the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

If the *bWait* parameter is set to TRUE, the method waits for the calibration sequence to complete before returning. In either mode, an EncCalibComplete event is fired once the calibration has been completed.

The positions at which the encoder counts are read, can be set using the SetEncCalibTableParams method.

Method *DeleteParamSet*

See Also Example Code

Visual Basic Syntax

Function **DeleteParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be deleted

Returns

MG Return Code

Details

This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed.

Motor Method DisableHWChannel

See Also Example Code

Visual Basic Syntax

Function **DisableHWChannel**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This method disables the channel(s) specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

When the method is called, the phase currents used to drive the motor are de-energised, which then allows the motor to be

rotated by hand.

If a motor is rotated whilst the channel is disabled, positional information is lost, and the channel should be homed after it is re-enabled.

To enable a particular channel, see the EnableHWChannel method.

Motor Method *DoEvents*

See Also Example Code

Visual Basic Syntax

Function **DoEvents**() As Long

Returns

MG Return Code

Details

This method enables the APT server to allow message processing to take place within a client application. For example, if the client application is processing a lengthy loop and is making multiple calls to the server, this can often cause the client application to become non-responsive because other user interface message handling, such as button clicks, cannot be processed. Calling the DoEvents method allows such client application message handling to take place.

Note. The DoEvents method works in the same way as the DoEvents keyword found in Visual Basic.

Example

```
Private Sub cmdDoEvents_Click()

' Set the global looping flag to begin looping.

m_blooping = True

' show hour glass mouse pointer

MousePointer = 11

' Begin lengthy processing loop.

Do while m_blooping

' Initiate a relative move and wait until complete.

MG17Motor1.MoveRelativeEx CHAN1_ID, 0.1, 0.1, True

' Before looping to initiate relative move,

' call DoEvents to allow other client application

' message handling (e.g. button click handling to

' allow m_blooping to be set to false) to take place.

MG17Motor1.DoEvents

Loop

' Show standard mouse pointer.

MousePointer = 0
```

Motor Method *EnableHWChannel*

See Also Example Code

Visual Basic Syntax

Function **EnableHWChannel**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This method enables the channel(s) specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

After a channel is enabled, it is good practice to home the motor, thereby restoring positional integrity.

To disable a particular channel, see the DisableHWChannel method.

Motor Method *GetAbsMovePos*

See Also Example Code

Visual Basic Syntax

Function **GetAbsMovePos**(lChanID As Long, pfAbsPos As Single) As Long

Parameters

lChanID - the channel identifier

pfAbsPos - the absolute position associated with the specified channel

Returns

MG Return Code

Details

This method retrieves the absolute position to which the channel specified by the *lChanID* parameter will move the next time the *MoveAbsolute* method is called, and returns a value in the *pfAbsPos* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot return the absolute position of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To set the absolute position for a particular channel, see the SetAbsMovePos method.

Motor Method *GetAbsMovePos_AbsPos*

See Also Example Code

Visual Basic Syntax

Function **GetAbsMovePos_AbsPos** (lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetAbsMovePos method, and returns the absolute position parameter value that is returned by reference in the GetAbsMovePos method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetBLashDist*

See Also Example Code

Visual Basic Syntax

Function **GetBLashDist**(lChanID As Long, pfBLashDist As Single) As Long

Parameters

lChanID - the channel identifier

pfBLashDist - the distance by which the motor will overshoot the demanded position, in order to overcome backlash.

Returns

MG Return Code

Details

This method gets the distance which the channel specified by the *lChanID* parameter will overshoot to overcome backlash, and returns a value (in millimeters or degrees) in the *pfBLashDist* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot return the backlash distance of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To set the relative distances for a particular channel, see the SetBLashDist method.

Motor Method *GetBLashDist_BLashDist*

See Also Example Code

Visual Basic Syntax

Function **GetBLashDist_BLash**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetBLashDist method, and returns the backlash distance parameter value that is returned by reference in the GetBLashDist method,

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method GetBowIndex

See Also Example Code

Visual Basic Syntax

Function G**etBowIndex**(lChanID As Long, plBowIndex As Long) As Long

Parameters

lChanID - the channel identifier

plBowIndex – the move profile to be used

Returns

MG Return Code

Details.

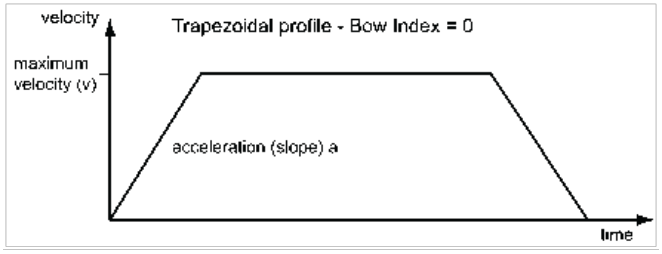
The specific move profile created by the system trajectory generator depends the profile mode and profile parameters presently selected. This method returns the present setting for the profile mode in the *plBowIndex* parameter, which accepts values as follows:

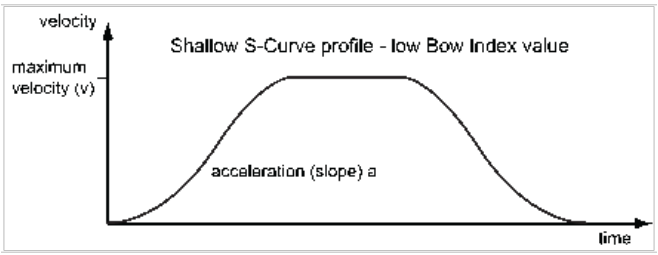
Trapezoidal Point to Point Profile	0
S-curve Point to Point Profile	1 to 18

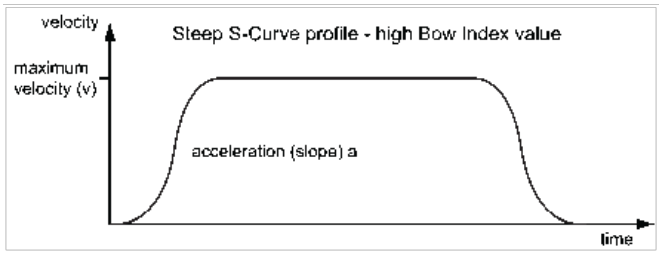
In either case, the velocity, acceleration and end position of the profile are specified using the SetVelParamsSetVelParams_Mot and the MoveRelativeMoveRelative_Mot or MoveAbsoluteMoveAbsolute_Mot methods.

The *Trapezoidal* profile is a standard, symmetrical acceleration/deceleration motion curve, in which the start velocity is always zero. To select a trapezoidal profile, set the BowIndex parameter to '0'.

The *S-curve* profile is a trapezoidal curve with an additional '*BowIndex'* value (1 to 18), which limits the rate of change of acceleration and smooths out the contours of the motion profile. The higher the Bow Index value, then the steeper the acceleration and deceleration phases as shown below. **Note**. High values of Bow Index may cause a move to overshoot:







The Bow Index can be set by calling the SetBowIndexGetBowIndex_Mot method..

Motor Method *GetBrakeState*

See Also Example Code

Note. This method is applicable only to the BDC series of benchtop DC Servo Controllers

Visual Basic Syntax

Function **GetBrakeState** (lChanID As Long, plState As Long) As Long

Parameters

lChanID - the channel identifier

plState – the state of the brake (on or off)

Returns

MG Return Code

Details

The ‘CONTROL IO’ connector on the rear panel exposes a TTL logic output that can be used to control a stage equipped with an electronic logic controlled brake mechanism. This method is used to obtain the current setting of the brake function (ON or OFF).

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. The BDC series features a separate control instance for each channel and therefore the *lChanID* parameter should always be set to *Chan1_ID*.

The brake state is returned in the plState parameter and takes values from the BRAKE_STATE enumeration as follows:

1. BRAKE_ON
2. BRAKE_OFF

Note. Careful use of the brake/enable servo must be considered when using this function. It is important to note that until the motor is disabled, the controller will continue to servo the motor to its current position and therefore possible brake/servo error conflict can occur if the motor shaft is moved when applying the brake. Furthermore, if the motor is mounted in a vertical position with the brake applied and the motor disabled, the motor will fall under gravity to its lower end stop when the brake is removed unless the motor is enables prior to removing the brake.

The brake state can be set by calling the SetBrakeState method.

Motor Method*GetButtonParams*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube Driver products ODC001, OST001, TDC001 and TST001

Function **SetButtonParams**((lChanID As Long, plButMode As Long, pfLeftButPos As Single, pfRightButPos As Single) As Long

Parameters
lChanID – the channel identifier
plButMode – the operating mode of the front panel buttons
pfLeftButPos – the absolute position associated with the left hand button
pfRightButPos – the absolute position associated with the right hand button

Returns
MG Return Code

Details
This method returns the operation mode of the front panel buttons on the motor controller unit.
Note. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the t-Cube units, *lChanID* should always be set to 'CHAN1_ID'.
The buttons on the front of the unit can be used either to jog the motor, or to perform moves to absolute positions.
The mode of operation is specified in the *plButMode* parameter which in turn takes valyues from the BUTTONMODE enumeration as follows:
BUTTON_MODEJOG: Once set to this mode, the move parameters for the buttons are taken from the 'Jog' parameters on the 'Move/Jogs' settings tab.
BUTTON_MODEPOSITION: In this mode, each button can be programmed with a different position value, such that the controller will move the motor to that position when the specific button is pressed.
Note. The following parameters are applicable only if 'Go to Position is selected in the 'Button Mode' field.
fLeftButPos: The position to which the motor will move when the left hand button is pressed.
fRightButPos: The position to which the motor will move when the right hand button is pressed.
Note. A 'Home' move can be performed by pressing and holding both buttons for 2 seconds. This function is irrespective of the 'Button Mode' setting. **Motor Method *GetCtrlStarted***

See Also Example Code

Visual Basic Syntax
Function **GetCtrlStarted**(pbStarted As Boolean) As Long

Parameters
pbStarted – the control serial number

Returns
MG Return Code

Details
This method checks that the associated control has been started (i.e. the StartCtrl method has been called). If the *pbStarted* parameter returns True, the control has been started. **Motor Method *GetChannelSwitch***

See Also Example Code

Note. This method is applicable only to BSC002 units.

Visual Basic Syntax

Function **GetChannelSwitch**(plChanID As Long) As Long

Parameters

plChanID – The channel identifier

Returns

MG Return Code

Details

Some methods (such as the ShowSettingsDlg method) have no channel selection (ChanID) parameter, and apply to whichever channel is currently selected. This method is for use with older 2-channel units (BSC002) and allows the channel number currently selected to be returned.

The *plChanID* parameter takes values specified by the HWCHANNEL enumeration.

Brushless DC Motor Method *GetDCCurrentLoopParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **GetDCCurrentLoopParams**(lChanID As Long, plProp As Long, plInt As Long, plIntLim As Long, plIntDeadBand As Long, plFFwd) As Long

Parameters

plChanID - the channel identifier

plProp - the value of the proportional constant

plInt - the value of the integration constant

plIntLim – the max value for the integration constant

plIntDeadBand – the value of the Integral Dead Band

plFFwd – the Integral Feed Forward value

Returns

MG Return Code

Details.

The motion processors within the BBD series controllers use digital current control as a technique to control the current through each phase winding of the motors. In this way, response times are improved and motor efficiency is increased. This is achieved by comparing the required (demanded) current with the actual current to create a current error, which is then passed through a digital PI-type filter. The filtered current value is used to develop an output voltage for each motor coil.

This method obtains the present values of the proportional and integration feedback loop constants for the current feedback in the *plProp* and *plInt* parameters respectively. These parameters determine the system response characteristics and accept values in the range 0 to 32767.

The *plIntLim* parameter is used to cap the output value of the integrator and prevent runaway of the integral sum at the output. It returns values in the range 0 to 32767.

Note. The following two parameters assist in fine tuning the motor drive current and help reduce audible noise and/or oscillation when the stage is in motion. A certain amount of trial and error may be experienced in order to obtain the optimum settings.

The *plIntDeadBand* parameter allows an integral dead band to be set, such that when the error is within this dead band, the integral action stops, and the move is completed using the proportional term only. It returns values in the range 0 to 32767.

The *plFFwd* parameter is a feed-forward term that is added to the output of the PI filter. It returns values in the range 0 to 32767.

The lChanID parameter is not valid for the BBD series brushless DC motor controller and is preset at the factory to '1'.

To set values for the Current Loop PI constants, see the SetDCCurrentLoopParams method.

Brushless DC Motor Method GetDCJoystickParams

See Also Example Code

Visual Basic Syntax

Function **GetDCJoystickParams**(lChanID As Long, pfMaxVelLO As Float, pfMaxVelHI As Float, pfAccnLO As Float, pfAccnHI As Float, plDirSense As Long) As Long

Parameters

lChanID - the channel identifier

pfMaxVelLO – the max velocity of a move when low gear mode is selected

pfMaxVelHI – the max velocity of a move when high gear mode is selected

pfAccnLO – the acceleration of a move when low gear mode is selected

pfAccnHI – the acceleration of a move when high gear mode is selected

plDirSense – the direction sense of the move

Returns

MG Return Code

Details.

The MJC001 joystick console has been designed for use by microscopists to provide intuitive, tactile, manual positioning of the stage. The console consists of a two axis joystick for XY control which features both low and high gear modes. This method is used to return the present settings of the max velocity and acceleration values for these modes.

The *pfMaxVelLO* and *pfAccnLO* parameters return the max velocity and acceleration of a move when low gear mode is selected.

Similarly, the *pfMaxVelHI* and *pfAccnHI* parameters return the max velocity and acceleration of a move when high gear mode is selected.

The actual direction sense of any joystick initiated move is dependent upon the application. The *plDirSense* parameter returns the present setting for the direction sense. This is useful when matching joystick direction sense to actual stage direction sense.

Brushless DC Motor Method *GetDCMotorOutputParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **GetDCMotorOutputParams**(lChanID As Long, pfContCurrLim As Float, pfEnergyLim As Float, pfMotorLim As Float, pfMotorBias As Float) As Long

Parameters

lChanID - the channel identifier

pfContCurrLim - the continuous current limit, as a percentage of max current.

pfEnergyLim - the accumulated energy limit, as a percentage of the default maximum.

pfMotorLim – the limit for the motor drive signal

pfMotorBias – the value of the motor bias

Returns

MG Return Code

Details.

This method contains parameters which define the limits that are presently applied to the motor drive signal.

The system incorporates a current 'foldback' facility, whereby the continuous current level can be capped. The continuous current limit is returned in the *pfContCurrLim* parameter, and returns values as a percentage of maximum peak current, in the range 0% to 100%, which is the default maximum level set at the factory (this maximum value cannot be altered).

When the current output of the drive exceeds the limit returned in the *pfContCurrLim* parameter, accumulation of the excess current energy begins. The *pfEnergyLim* parameter returns the limit set for this accumulated energy, as a percentage of the factory set default maximum, in the range 0% to 100%. When the accumulated energy exceeds the value returned in the *fEnergyLim* parameter, a 'current foldback' condition is said to exist, and the commanded current is limited to the value returned in the *fContCurrLim* parameter. When this occurs, the Current Foldback status bit (bit 25) is set in the Status Register. When the accumulated energy above the *fContCurrLim* value falls to 0, the limit is removed and the status bit is cleared.

The *pfMotorLim* parameter returns the limit set for the motor drive signal, and accepts values in the range 0 to 32767 (100%). If the system produces a value greater than the limit set, the motor command takes the limiting value. For example, if *pfMotorLim* is set to 30000 (91.6%), then signals greater than 30000 will be output as 30000 and values less than -30000 will be output as -30000.

When an axis is subject to a constant external force in one direction (such as a vertical axis pulled downwards by gravity) the servo filter can compensate by adding a constant DC bias to the output. The present setting for this bias is returned in the *pfMotorBias* parameter, which accepts values in the range -32767 to 32768. The default value is 0. Once set, the motor bias is applied while the position loop is enabled.

To make settings for the motor parameters and limits, see the SetDCMotorOutputParams method.

Brushless DC Motor Method *GetDCPositionLoopParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **GetDCPositionLoopParams**(lChanID As Long, plProp As Long, plInt As Long, plIntLim As Long, plDeriv As Long, plDerivTime As Long, plLoopGain As Long, plVelFFwd As Long, plAccFFwd As Long, plPosErrLim As Long) As Long

Parameters

plChanID - the channel identifier

plProp - the value of the proportional constant

plInt - the value of the integration constant

plIntLim – the max value for the integration sum

plDeriv – the value of the derivative constant

plDerivTime – the sampling rate for calculating the derivative term, in cycles

plLoopGain – a scaling factor applied to the output of the PID loop

plVelFFwd – A velocity term, added to the output of the PID filter to assist tuning

plAccFFwd – An acceleration term, added to the output of the PID filter to assist tuning

plPosErrLim – the value of the position error limit

Returns

MG Return Code

Details

The motion processors within the BBD series controllers use a position control loop to determine the motor command output. The purpose of the position loop is to match the actual motor position and the demanded position. This is achieved by comparing the demanded position with the actual encoder position to create a position error, which is then passed through a digital PID-type filter. The filtered value is the motor command output.

This method obtains the present settings for the operating parameters of the PID position in the *plProp, plInt* and *plDeriv* parameters respectively. These parameters determine the system response characteristics and accept values in the range 0 to 32767.

The *plIntLim* parameter is used to cap the value of the *Integrator* to prevent runaway of the integral sum at the output. It returns values in the range 0 to 32767. If 0 is returned, then the integration term in the PID loop is ignored.

Under normal circumstances, the derivative term of the PID loop is recalculated at every servo cycle. However, it may be desirable to reduce the sampling rate to a lower value, in order to increase stability or simplify tuning. The *plDerivTime* parameter returns the sampling rate, for example if 10 is returned, the derivative term is calculated every 10 servo cycles. The value is returned in cycles, in the range 1 to 32767.

The *plLoopGain* parameter is a scaling factor applied to the output of the PID loop. It returns values in the range 0 to 65535, where 0 is 0% and 65535 is 100%.

The *plVelFFwd* and *plAccFFwd* parameters are velocity and acceleration feed-forward terms that are added to the output of the PID filter to assist in tuning the motor drive signal. They return values in the range 0 to 32767.

Under certain circumstances, the actual encoder position may differ from the demanded position by an excessive amount. Such a large position error is often indicative of a potentially dangerous condition such as motor failure, encoder failure or excessive mechanical friction. To warn of, and guard against this condition, a maximum position error can be returned in the *plPosErrLim* parameter, in the range 0 to 65535. The actual position error is continuously compared against the limit returned, and if exceeded, the Motion Error bit (bit 15) of the Status Register is set and the associated axis is stopped.

The plChanID parameter is not valid for the BBD series brushless DC motor controller and is preset at the factory to '1'.

To set values for the Position Loop PID constants, see the SetDCPositionLoopParams method.

Brushless DC Motor Method *GetDCProfileModeParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **GetDCProfileModeParams**(lChanID As Long, plProfMode As Long, pfJerk As Long) As Long

Parameters

lChanID - the channel identifier

plProfMode – the move profile to be used

pfJerk – the jerk value to be used if S-curve mode is selected

Returns

MG Return Code

Details.

The system incorporates a trajectory generator, which performs calculations to determine the instantaneous position, velocity and acceleration of each axis at any given moment. During a motion profile, these values will change continuously. Once the move is complete, these parameters will then remain unchanged until the next move begins.

The specific move profile created by the system depends on several factors, such as the profile mode and profile parameters presently selected, and other conditions such as whether a motion stop has been requested. This method is used to return the present setting for the profile mode in the *plProfMode* parameter, which accepts values as follows:

Trapezoidal Point to Point Profile 0

S-curve Point to Point Profile 2

In either case, the velocity, acceleration and end position of the profile are specified using the SetVelParams and the MoveRelative or MoveAbsolute methods.

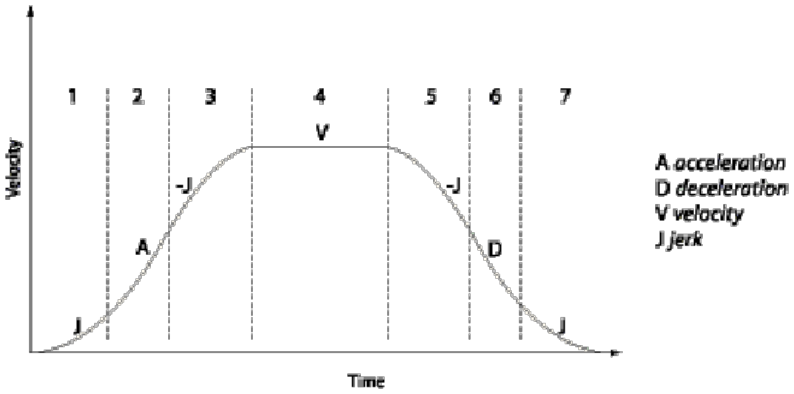
The *Trapezoidal* profile is a standard, symmetrical acceleration/deceleration motion curve, in which the start velocity is always zero.

The *S-curve* profile is a trapezoidal curve with an additional '*Jerk'* parameter, which limits the rate of change of acceleration and smooths out the contours of the motion profile.

The Jerk value is returned in mm/s3 in the *pfJerk* parameter, which and accepts values in the range 0 to 46566139. The returned value relates to the maximum rate of change in acceleration in a single cycle of the basic trapezoidal curve.

In this profile mode, the acceleration increases gradually from 0 to the specified acceleration value, then decreases at the same rate until it reaches 0 again at the specified velocity. The same sequence in reverse brings the axis to a stop at the programmed destination position.

Example



The figure above shows a typical S-curve profile. In segment (1), the S-curve profile drives the axis at the specified jerk (J) until the maximum acceleration (A) is reached. The axis continues to accelerate linearly (jerk = 0) through segment (2). The profile then applies the negative value of jerk to reduce the acceleration to 0 during segment (3). The axis is now at the maximum velocity (V), at which it continues through segment (4). The profile then decelerates in a similar manner to the acceleration phase, using the jerk value to reach the maximum deceleration (D) and then bring the axis to a stop at the destination.

To set values for the DC Profile Mode parameters, see the SetDCProfileModeParams method.

Brushless DC Motor Method *GetDCSettledCurrentLoopParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **GetDCSettledCurrentLoopParams**(lChanID As Long, plSettledProp As Long, plSettledInt As Long, plSettledIntLim As Long, plSettledIntDeadBand As Long, plSettledFFwd) As Long

Parameters

plChanID - the channel identifier

plSettledProp - the value of the proportional constant

plSettledInt - the value of the integration constant

plSettledIntLim – the max value for the integration constant

plSettledIntDeadBand – the value of the Integral Dead Band

plSettledFFwd – the Integral Feed Forward value

Returns

MG Return Code

Details.

The current needed to hold a motor in a fixed (settled) position is much smaller than that required for a move. It is good practise to decrease the current in a stationary motor, in order to reduce heating, and thereby minimize thermal movements caused by expansion.

This method assists in maintaining stable operation and is applicable only when the stage is settled. It obtains the present values of the proportional and integration feedback loop constants for the current feedback in the *plSettledProp* and *plSettledInt* parameters respectively. These parameters determine the system response characteristics and accept values in the range 0 to 32767.

The *plSettledIntLim* parameter is used to cap the output value of the integrator and prevent runaway of the integral sum at the output. It returns values in the range 0 to 32767.

Note. The following two parameters assist in fine tuning the motor drive current and help reduce audible noise and/or oscillation when the stage is near the target position. A certain amount of trial and error may be experienced in order to obtain the optimum settings.

The *plSettledIntDeadBand* parameter allows an integral dead band to be set, such that when the error is within this dead band, the integral action stops, and the move is completed using the proportional term only. It returns values in the range 0 to 32767.

The *pSettledIFFwd* parameter is a feed-forward term that is added to the output of the PI filter. It returns values in the range 0 to 32767.

The lChanID parameter is not valid for the BBD series brushless DC motor controller and is preset at the factory to '1'.

To set values for the Current Loop PI constants, see the SetDCSettledCurrentLoopParams method

Brushless DC Motor Method*GetDCTrackSettleParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **GetDCTrackSettleParams**(lChanID As Long, plSettleTime As Long, plSettleWnd As Long, plTrackWnd As Long) As Long

Parameters

lChanID - the channel identifier

plSettleTime - the No of cycles that the axis must be settled before the 'Settled' status bit is set

plSettleWnd - the No of encoder counts that the position error must be less than or equal to, before the axis is considered 'settled'

plTrackWnd - the maximum allowable position error

Returns

MG Return Code

Details.

The system incorporates a monitoring function, which continuously indicates whether or not the axis has 'settled'. The 'Settled' indicator is bit 14 in the Status Register and is set when the associated axis is settled. Note that the status bit is controlled by the processor, and cannot be set or cleared manually.

The axis is considered to be 'settled' when the following conditions are met:

the axis is at rest (i.e. not performing a move),

the error between the demanded position and the actual motor position is less than or equal to a specified number of encoder counts (0 to 65535) returned in the *plSettleWnd* parameter (Settle Window),

the above two conditions have been met for a specified number of cycles (settle time), returned in the *plSettleTime* parameter (range 0 to 32767).

The processor also provides a 'tracking window', which is used to monitor servo performance outside the context of motion error. The tracking window is a programmable position error limit within which the axis must remain, but unlike the position error limit set in the SetDCPositionLoopParams method, the axis is not stopped if it moves outside the specified tracking window. This function is useful for processes that rely on the motor's correct tracking of a set trajectory within a specific range. The tracking window may also be used as an early warning for performance problems that do not yet qualify as motion error.

The size of the tracking window (i.e. the maximum allowable position error while remaining within the tracking window) is returned in the *plTrackWnd* parameter, in the range 0 to 65535. If the position error of the axis exceeds this value, the Tracking Indicator status bit (bit 13) is set to 0 in the Status Register. When the position error returns to within the window boundary, the status bit is set to 1.

To make settings for the Track and Settle parameters, see the SetDCTrackSettleParams method

Brushless DC Motor Method GetDCTriggerParams

See Also Example Code

Note. This method is applicable only to brushless DC driver units BBDxxx and TBD001.

Visual Basic Syntax

Function **GetDCTriggerParams**(lChanID As Long, plTrigInMode As Long, plTrigOutMode As Long) As Long

Parameters

lChanID - the channel identifier

plTrigInMode – gets the input trigger mode for the specified channel

plTrigOutMode – gets the output trigger mode for the specified channel

Returns

MG Return Code

Details

This method is used to obtain the present trigger mode settings for the brushless DC motor controller. It is possible to configure a particular controller to respond to trigger inputs, generate trigger outputs or both respond to and generate a trigger output. When a trigger input is received, the unit can be set to initiate a move (relative, absolute or home). Similarly the unit can be set to generate a trigger output signal when a specified event (e.g move initiated) occurs. For those units configured for both input and output triggering, a move can be initiated via a trigger input while at the same time, a trigger output can be generated to

initiate a move on another unit.

The trigger settings can be used to configure multiple units in a master – slave set up, thereby allowing multiple channels of motion to be synchronized. Multiple moves can then be initiated via a single software or hardware trigger command.

The channel to be interrogated is specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration. For single channel units such as the TDB001, this parameter is locked at '0' (Channel 1) and cannot be changed.

The Trigger In input can be configured to initiate a relative, absolute or homing home, either on the rising or falling edge of the signal driving it. As the trigger input is edge sensitive, it needs to see a logic LOW to HIGH transition ("rising edge") or a logic HIGH to LOW transition ("falling edge") for the move to be started. Additionally, the move parameters must be downloaded to the unit prior to the move using the relevant relative move or absolute move methods as described below. A move already in progress will not be interrupted; therefore external triggering will not work until the previous move has been completed.

The *plTrigInMode* parameter gets the input trigger mode for the specified channel. The various trigger modes are specified by the DCTRIGINMODE enumeration as follows:

DCTRIGIN_DISABLED – triggering operation is disabled

DCTRIGINRISE_RELMOVE – a relative move (specified using the latest *MoveRelative* or MoveRelativeEx settings) is initiated on the specified channel when a rising edge input signal is received on the TRIG IN connector.

DCTRIGINFALL_RELMOVE – as above, but the relative move is initiated on receipt of a falling edge signal.

DCTRIGINRISE_ABSMOVE – an absolute move (specified using the latest MoveAbsolute or MoveAbsoluteEx settings) is initiated on the specified channel when a rising edge input signal is received on the TRIG IN connector.

DCTRIGINFALL_ABSMOVE – as above, but the absolute move is initiated on receipt of a falling edge signal.

DCTRIGINRISE_HOMEMOVE – a home move (specified using the latest *MoveHome* settings) is initiated on the specified channel when a rising edge input signal is received on the TRIG IN connector.

DCTRIGINFALL_HOMEMOVE – as above, but the home move is initiated on receipt of a falling edge signal.

The Trigger Out output can be configured to be asserted to either logic HIGH or LOW as a function of certain motion-related conditions, such as when a move is in progress (In Motion), complete (Move Complete) or reaches the constant velocity phase on its trajectory (Max Vel). The logic state of the output will remain the same for as long as the chosen condition is true. The logic state associated with the condition can be selected to be either LOW or HIGH.

The *plTrigOutMode* parameter gets the output trigger mode for the specified channel. The various trigger modes are specified by the DCTRIGOUTMODE enumeration as follows:

DCTRIGOUT_DISABLED – triggering operation is disabled

DCTRIGOUTHIGH_INMOTION – The output trigger goes high (5V) when the stage is in motion.

DCTRIGOUTLOW_INMOTION – The output trigger goes low (0V) when the stage is in motion.

DCTRIGOUTHIGH_MOTIONCOMPLETE – The output trigger goes high (5V) when the current move is completed.

DCTRIGOUTLOW_MOTIONCOMPLETE – The output trigger goes low (0V) when the current move is completed.

DCTRIGOUTHIGH_MAXVELOCITY – The output trigger goes high (5V) when the stage reaches max velocity (as set using the SetVelParams method).

DCTRIGOUTLOW_MAXVELOCITY – The output trigger goes low (0V) when the stage reaches max velocity (as set using the SetVelParams method).

The trigger parameters are obtained using the SetDCTriggerParams method.

Please refer to the handbook supplied with your unit for further information on identifying the trigger connections.

Motor Method *GetDispMode*

See Also Example Code

Visual Basic Syntax

Function **GetDispMode**(plDispMode As Long) As Long

Parameters

plDispMode – The display mode selected

Returns

MG Return Code

Details

This method returns the display mode for the associated GUI panel. The *plDispMode* parameter takes values from the DISPLAYMODE enumeration as follows:

1. DISPMODE_PANEL
2. DISPMODE_GRAPH
3. DISPMODE_POSITION

When set to DISPMODE_PANEL, the display shows the standard GUI panel.

If DISPMODE_GRAPH is seleceted, the display is changed to a graphical display, showing the position of the motor channel(s). Moves to absolute positions can then be initiated by positioning the mouse within the display and clicking, see Graphical Control of Motor Positions for more information.

When set to DISPMODE_POSITION, only the 'Position' display is shown.

The display mode can be set by calling the SetDispMode method.

Motor Method *GetEncCalibTableParams*

See Also Example Code

Visual Basic Syntax

Function **GetEncCalibTableParams**(lChanID As Long, plEncCalib As Long, pfCalibStep As Float, plCalibDwell As Long, plQEPSense As Long) As Long

Parameters

lChanID – the channel identifier

plEncCalib – the Encoder Calibration factor (counts/mm or counts/degree)

pfCalibStep – the stepping stepping distance to use when a calibration table is acquired

plCalibDwell – returns the dwell time (in ms) to wait before reading the encoder count

plQEPSense – returns the sense of the encoder signals being read by the controller electronics, (positive or negative)

Returns

MG Return Code

Details

When 'Encoder Positioning' is selected, all position displays and motor moves are based on positions derived from the encoder system fitted to the stage/actuator.

Example. The encoders currently fitted to Our stages are set to 10000 counts per mm (i.e. 0.1micron per count). Therefore, to move 1 mm, the controller will drive out the appropriate number of microsteps to result in an encoder count change of 10000. To display position values, the software converts the encoder count into a 'real world' position by dividing by 10000.

For a perfect 1mm pitch lead screw, a microstep count of 25600 would equate exactly to an encoder count of 10000. However, due to lead screw pitch, non-linearity and other cyclic errors, this is not achievable in the real world. It is the purpose of the

encoder feedback handling within the APT software to accommodate for this and achieve the required encoder position (a more accurate position reading).

One way to accommodate for this lead screw non-linearity is for the system to acquire a look up table of microstep count verses encoder count readings. Using this 'calibration' table the system is then able to adjust the microstep count required (to drive the motor) to achieve the required encoder count.

When the 'Calibrate' button in the encoder Settings panel is clicked, (or the CalibrateEnc method is called), the system begins the calibration sequence by first homing the channel and then moving through a series of microstep position points taking the encoder reading at each point.

This method obtains settings which detail the positions at which the encoder count, associated with the channel specified by the *lChanID* parameter, is read. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The *plEncCalib* parameter returns the overall encoder calibration factor in counts/mm or counts/degree. On encoder equipped stages currently available, this factor is factory set to 10000 (i.e. 10000 counts/mm or 0.1micron per count).

The p*fCalibStep* parameter obtains the stepping distance to used between positions, e.g. a stepping distance of 0.2mm instructs the system to stop and read the encoder count every 0.2mm.

In many cases it is good practice to allow the stage mechanics to settle for a short period (allowing vibrations and other transients to die down) before sampling the encoder reading. The *plCalibDwell* parameter obtains the dwell time (in ms) to wait at each position point visited, before reading the encoder count.

The *plQEPSense* parameter takes values specified by the ENCQEPSENSE Enumeration, and returns the sense of the encoder signals being read by the controller electronics, i.e. positive or negative. There is only one setting applicable for a particular stage/encoder system and when set should not be altered.

Values for the Calibration Table Parameters can be set or changed, by calling the SetEncCalibTableParams method.

Note. Before it can be used, the calibration table must be enabled by calling the SetEncPosControlParams method (UseCal parameter) or via the Encoder Control Panel.

Motor Method *GetEncPosControlParams*

See Also Example Code

Visual Basic Syntax

Function **GetEncPosControlParams** (lChanID As Long, plPosSrcMode As Long, plPosCorrMode As Long, pbUseCalib As variant_Bool) As Long

Parameters

lChanID – the channel identifier

plPosSrcMode – the source of positional information, microstep count or encoder count.

plPosCorrMode – the position correction mode.

pbUseCalib – use calibration table if True

Returns

MG Return Code

Details

This method returns the position source and the correction mode for a move initiated on the channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The positioning mode is returned in the *plPosSrcMode* parameter, which in turn takes values from the ENCPOSSOURCEMODE enumeration as follows:

ENC_POSSRC_MICROSTEP - By default the software is set to 'Microstep Positioning' mode. In this mode, all position displays and motor moves are based on positions derived from the internal microstep counters within the controller (i.e. are not based on the encoder count).

For example, the APT system can control to a resolution of 25600 microsteps per revolution. For a stage or actuator with a 1 mm pitch lead screw, the controller will generate 25600 microsteps to move 1 mm. To display position values, the software will convert the microstep count into a 'real world' position by dividing by 25600.

ENC_POSSRC_ENCODER - In this mode the position displays and all motor moves are based on positions derived from the encoder system fitted to the stage/actuator. So for example, the encoders currently fitted to Our stages are set to 10000 counts per mm (i.e. 0.1micron per count). To move 1 mm, the controller will drive out the appropriate number of microsteps to result in an encoder count change of 10000. To display position values, the software will convert the encoder count into a 'real world' position by dividing by 10000.

Note that the positions values returned by the GetPosition and GetPositionEx methods are derived from the microstep count in 'Microstep Positioning' mode and from the encoder count in 'Encoder Positioning' mode.

The position correction mode to be used is returned in the *lPosCorrMode* parameter, which in turn, takes values from the ENCPOSCONTROLMODE Enumeration as follows:

ENC_POSCTRL_POSITION – This setting selects 'Position (& Correct)' mode, whereby the system first attempts to move to the required encoder based position (with or without calibration table support), and then adjusts the motor position, using a series of very small correction steps, until the required encoder position is achieved.

The various correction stepping parameters that affect this operation, are set using the SetEncPosCorrectParams method or via the 'Encoder' tab of the settings tabbed dialog accessed using the 'Settings' button on the main graphical panel.

Note that depending on the nature of the lead screw, the stage/actuator may end up either side of the required position. The correction moves will be applied in both forward or reverse directions as required in order to achieve the required encoder position.

ENC_POSCTRL_POSITIONSTOPSHORT - In some applications it is desirable to always 'reach' the required final position from the same direction (conventionally using positive moves on Our stages). To support this, the 'Position (Stop Short & Correct)' correction mode is available.

When this mode is selected the system stops short of the intended position by a user specified distance and then issues small position correction steps to achieve the required final encoder position. Again the user adjustable parameters associated with this operation are set using the SetEncPosCorrectParams method or via the 'Encoder' tab of the settings tabbed dialog accessed using the 'Settings' button on the main graphical panel.

ENC_POSCTRL_DISABLED – Position correction is switched off.

Note. Before using an encoded position correction mode, the backlash correction distance should be set to zero – see the SetBLashDist method.

If a calibration table has been acquired using the SetEncCalibTableParams method, it must be enabled by setting the *UseCalib* parameter to True.

Motor Method *GetEncPosCorrectParams*

See Also Example Code

Visual Basic Syntax

Function **GetEncPosCorrectParams** (lChanID As Long, plPosSetPtWnd As Long, plSnguStepWnd As Long, plStopShortDist As Long, plCorrMoveStep As Long) As Long

Parameters

lChanID – the channel identifier

plPosSetPtWnd – the tolerance (in encoder counts) within which the system must achieve the required position setpoint when an encoded move is corrected.

plSnguStepWnd – the window within which the system attempts to reach the required position setpoint by single microstep stepping.

plStopShortDist –the distance (in microsteps) to stop short of an encoded move (when the 'Position (Stop Short & Correct)' position correction mode is enabled

plCorrMoveStep –the size of correction step (in microsteps) to be used when the system issues correction moves outside of the

distance set in the *lSnguStepWnd* parameter

Returns

MG Return Code

Details

In addition to the calibration table described in the SetEncCalibTableParams method, the APT software can be set to invoke further positioning correction at the end of an encoded move. This can sometimes be required when there has been thermal drift in the mechanics since the time the calibration table was acquired. There are two position correction modes available, 'Position (& Correct)' and 'Position (Stop Short & Correct)' selected using the SetEncPosControlParams method (or by using the 'Position Correct Mode' drop down list in the 'Encoder Control Panel').

This method returns settings for an encoded move initiated on the channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The *plPosSetPtWnd* parameter returns a window of acceptability or tolerance (in encoder counts) that the system must achieve.

For example, when set to a value of 1, the system must get to within 1 encoder count either side of the required position. This setting is useful for noisy systems (i.e. mechanical noise causing jitter on the encoder count) where achieving a precise encoder count may prove difficult. For an encoder calibration factor of 10000/mm, a value of 1 for this setting means the system will position to within +-0.1micron of the required position.

When 'Position (& Correct)' mode is selected (see the SetEncPosControlParams method), the system first attempts to move to the required encoder based position (with or without calibration table support), and then adjusts the motor position, using a series of very small correction steps, until the required encoder position is achieved. The *plSnguStepWnd* returns the window (or tolerance), within which the system attempts to reach the required position setpoint by single micro stepping.

For example, if this parameter is set to a value of 10, and if the encoded move initially ends up within 10 counts (1 micron for a calibration factor of 10000) either side of the required position, the system will single microstep to reach the required position

If, at the end of the initial encoded move, the system detects that the encoder position is outside of this window, then the system steps at whatever multiple of microsteps is returned by the *plCorrMoveStep* parameter. When the encoder position falls within the window specified by the *plSnguStepWnd* parameter, single microstepping can take place.

Assuming 25600 microsteps per revolution of the stepper motor and a 1mm pitch lead screw, a value of 5 equates to approx. 0.2 micron step size.

In some applications it is desirable to always 'reach' the required final position from the same direction (conventionally using positive moves on Our stages). When the 'Position (Stop Short & Correct) position correction mode is selected (see the SetEncPosControlParams method), the *plStopShortDist* parameter specifies the distance (in microsteps) to stop short of an encoded move.

Assuming 25600 microsteps per revolution of the stepper motor and a 1mm pitch lead screw, a value of 250 equates to approx. 10 microns stop short distance.

These parameters can be set by calling the SetEncPosCorrectParams method.

Motor Method *GetHomeParams*

See Also Example Code

Visual Basic Syntax

Function **GetHomeParams**(lChanID As Long, plDirection As Long, plLimSwitch As Long, pfHomeVel As Single, pfZeroOffset As Single) As Long

Parameters

lChanID - the channel identifier

plDirection - the direction sense to move when homing

plLimSwitch - The limit switch associated with the home position

pfHomeVel - the velocity at which the motors should move when Homing

pfZeroOffset - the distance (in mm or degrees) of the limit switch from the Home position

Returns

MG Return Code

Details

This method retrieves the homing parameters for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The direction sense for a move to Home (either forward or reverse) is returned in the *plDirection* parameter, which in turn, takes values from the HOMEDIR enumeration,

The limit switch associated with the home position (hardware forward, hardware reverse, software forward or software reverseis returned in the *plLimSwitch* parameter, which in turn, takes values from the HOMELIMSWITCH enumeration,.

The velocity of a homing move is returned in the *pfHomeVel* parameter.

The distance of the Home position from the Home Limit Switch is returned in the *pfZeroOffset* parameter.

The system is notified to the homing parameters of a particular stage by calling the SetHomeParams method.

Motor Method GetHomeParams_HomeVel

See Also Example Code

Visual Basic Syntax

Function **GetHomeParams_HomeVel**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetHomeParams method, and returns the home velocity parameter value that is returned by reference in the GetHomeParams

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method GetHomeParams_ZeroOffset

See Also Example Code

Visual Basic Syntax

Function **GetHomeParams_ZeroOffset**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetHomeParams method, and returns the Zero offset parameter value that is returned by reference in the GetHomeParams

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetHWCommsOK*

See Also Example Code

Visual Basic Syntax

Function **GetHWCommsOK**(pbCommsOK As Boolean) As Long

Parameters

pbCommsOK – Hardware comms OK if true

Returns

MG Return Code

Details

This method checks communications between the associated hardware unit and the control PC. If the *pbCommsOK* parameter returns True, communication is OK. If *pbCommsOK* returns False, a comms error exists, e.g. a faulty USB cable, a device has become disconnected from the USB bus, the device was connected after boot up etc.

Motor Method *GetHWLimSwitches*

See Also Example Code

Visual Basic Syntax

Function **GetHWLimSwitches**(lChanID As Long, plRevLimSwitch As Long, plFwdLimSwitch As Long) As Long

Parameters

lChanID - the channel identifier

plRevLimSwitch - the action of the reverse limit switch when contact is made

plFwdLimSwitch - the action of the forward limit switch when contact is made

Returns

MG Return Code

Details

This method returns the settings of the hardware limit switches associated with the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The action that the forward and reverse hardware limit switches make on contact, is returned in the *lFwdLimSwitch* and *lRevLimSwitch* parameters respectively, which in turn, take values from the HWLIMSWITCH enumeration.

The action can be either Make, Break or Ignore.

The system is notified of the limit switch action by calling the SetHWLimSwitches method.

Motor Method *GetIndicatorLEDMode*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube Driver products ODC001, OST001, TDC001 and TST001

Visual Basic Syntax

Function **GetIndicatorLEDMode**(lChanID As Long, plLEDMode As Long) As Long

Parameters

lChanID – the channel identifier

plLEDMode – the operating mode of the front panel LED

Returns

MG Return Code

Details

This method returns the operation mode of the front panel LED on the motor controller unit.

Note. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the OptoSTDriver and OptoDCDriver, *lChanID* should always be set to 'CHAN1_ID'.

A single LED is fitted to the front panel of the unit. The configuration state of this LED is returned in the *plLEDMode* parameter, which in turn, takes values from the INDICATORLEDMODE enumeration as follows:

1. LEDMODE_IDENT: The LED will flash when the 'Ident' button is clicked on the APT Software GUI panel.
2. LEDMODE_LIMITSWITCH: The LED will flash when the motor reaches a forward or reverse limit switch.

4 LEDMODE_BUTTONMODECHANGE: The LED will flash when the button mode is changed.

9 LEDMODE_MOVING: The LED is lit when the motor is moving.

All modes are disabled by default.

It is recognised that, in a light sensitive environment stray light from the LED could be undesirable. Therefore it is possible to disable selectively, one or all of the LED indicator modes described above by setting the appropriate value in the *lLEDMode* parameter.

For example, if the *lLEDMode* parameter returns '15' (binary 1111) all modes will be enabled. Similarly, if the *lLEDMode* parameter returns '6' (binary 0110), only the LEDMODE_LIMITSWITCH and LEDMODE_BUTTONMODECHANGE modes are enabled.

Motor Method *GetJogMode*

See Also Example Code

Visual Basic Syntax

Function **GetJogMode**(lChanID As Long, plMode As Long, plStopMode As Long) As Long

Parameters

lChanID - the channel identifier

plMode - the jogging mode

plStopMode - the way in which the motor stops when the jog button is released

Returns

MG Return Code

Details

This method retrieves the jogging mode for the channel(s) specified by the *lChanID* parameter, and returns the value in the *plMode* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot return the jogging mode of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

The *plMode* parameter takes values specified by the JOGMODE enumeration.

The *plStopMode* parameter takes values specified by the STOPMODE enumeration and returns the way in which the motor stops when the jog signal is removed.

For further details on jogging modes and how to set the jogging mode for a particular channel, see the SetJogMode method.

Motor Method *GetJogMode_Mode*

See Also Example Code

Visual Basic Syntax

Function **GetJogMode_Mode**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetJogMode method, and returns the jog mode parameter value that is returned by reference the GetJogMode method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetJogMode_StopMode*

See Also Example Code

Visual Basic Syntax

Function **GetJogMode_StopMode**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetJogMode method, and returns the stop mode parameter value that is returned by reference in the GetJogMode method.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetJogStepSize*

See Also Example Code

Visual Basic Syntax

Function **GetJogStepSize**(lChanID As Long, pfStepSize As Single) As Long

Parameters

lChanID - the channel identifier

pfStepSize - the size of step taken when the jog signal is initiated

Returns

MG Return Code

Details

Note: This method is applicable only if the Jog Mode is set to 'Step' in the SetJogMode method.

This method retrieves the distance to move when a jog command is initiated. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The step size is returned in real world units (mm or degrees) in the *pfStepSize* parameter:

To set the jogging mode for a particular channel, see the SetJogStepSize method.

Motor Method GetJogStepSize_StepSize

See Also Example Code

Visual Basic Syntax

Function **GetJogStepSize_StepSize** (lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetJogStepSize method, and returns the jog step size parameter value that is returned by reference the GetJogStepSize method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetJogVelParams*

See Also Example Code

Visual Basic Syntax

Function **GetJogVelParams**(lChanID As Long, pfMinVel As Single, pfAccn As Single, pfMaxVel As Single) As Long

Parameters

lChanID - the channel identifier

pfMinVel - the returned minimum velocity at which to start jog moves

pfAccn - the returned acceleration at which to climb to the maximum velocity

pfMaxVel - the returned maximum velocity at which to perform jog moves

Returns

MG Return Code

Details

This method obtains the trapezoidal velocity profile parameters. The values are returned in the appropriate parameter as real

world units (mm or degrees) dependent upon stage type.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The velocity parameters associated with a particular channel can be set by calling the SetJogVelParams method.

Note that for short jog steps it may not be possible for the motor to reach the maximum velocity before decelerating to a stop.

Motor Method GetJogVelParams_Accn

See Also Example Code

Visual Basic Syntax

Function **GetJogVelParams_Accn**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetJogVelParams method, and returns the acceleration parameter value that is returned by reference the GetJogVelParams method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method GetJogVelParams_MaxVel

See Also Example Code

Visual Basic Syntax

Function **GetJogVelParams_MaxVel**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetJogVelParams method, and returns the maximum velocity parameter value that is returned by reference in the GetJogVelParams method.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

KCube Motor Method GetKCubePanelParams

This method is applicable only to K-Cube Series motor controllers

See Also Example Code

Visual Basic Syntax
Function **GetKCubePanelParams** (lChanID As Long, plWheelMode As Long, pfWheelVel As Float, pfWheelAccn As Float,

plWheelDirSense As Long, pfPresetPos1 As Float, pfPresetPos2 As Long, plDispBrightness As Long, plDispTimeout As Long, plDispDimLevel As Long) As Long

Parameters
lChanID - the channel identifier
plWheelMode - The operating mode of the top panel velocity wheel
pfWheelVel - The max velocity of a move initiated by the top panel velocity wheel
pfWheelAccn - The max acceleration of a move initiated by the top panel velocity wheel
plWheelDirSense - The direction sense of a move initiated by the top panel velocity wheel
pfPresetPos1 - The preset position 1 when operating in go to position mode
pfPresetPos2 - The preset position 2 when operating in go to position mode
plDispBrightness - The display brightness when the unit is active
plDispTimeout - The display time out (in minutes). A static display will be dimmed after this time period has elapsed.
plDispDimLevel - The display brightness after time out.

Returns
MG Return Code

Details
This method returns the operating parameters of the velocity wheel on the top panel of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode is returned in the plWheelMode parameter , which in turn takes values from the KMOTWHEELMODE enumeration as follows:

1 Velocity Control Mode - Deflecting the wheel starts a move with the maximum velocity proportional to the deflection. The maximum velocity (i.e. velocity corresponding to the full deflection of the joystick wheel) and acceleration are returned in the pfWheelVel and pfWheelAccn parameters.

2 Jog Mode - Deflecting the wheel initiates a jog move, using the parameters specified by the SetJogStepSize and SetJogVelParams methods. Keeping the wheel deflected repeats the move automatically after the current move has completed.

3 Go To Position Mode - Deflecting the wheel starts a move from the current position to one of the two predefined “teach” positions. The teach positions are returned in the pfPresetPos1 and pfPresetPos2 parameters.

The direction of a move initiated by the velocity wheel is returned in the lWheelDirSense parameter, which takes values from the KMOTWHEELDIRSENSE enumeration as follows:

0 The wheel is disabled

1 Upwards rotation of the wheel results in a positive motion (i.e. increased position count).

The following option applies only when the Wheelmode is set to Velocity Control Mode (1). If set to Jog Mode (2) or Go to Position Mode (3), the following option is ignored.

2 Upwards rotation of the wheel results in a negative motion (i.e. decreased position count).

In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit. The brightness is returned in the plDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Furthermore, 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after the time interval returned in the plDispTimeout parameter has elapsed. Time out is returned in minutes in the range 0 (never) to 480. The dim level is returned in the plDispDimLevel parameter, as a value from 0 (Off) to 10 (brightest) but is also limited by the plDispBrightness parameter.

The current setting for panel parameters may be obtained by calling the SetKCubePanelParams method

KCube Motor Method GetKCubeTriggerParams

This method is applicable only to K-Cube Series motor controllers

See Also Example Code

Visual Basic Syntax
Function **GetKCubeTriggerParams** (lChanID As Long, plTrig1Mode As Long, plTrig1Polarity As Long, plTrig2Mode As Long, plTrig2Polarity As Long) As Long

Parameters
lChanID - the channel identifier
plTrig1Mode - TRIG1 operating mode
plTrig1Polarity - The active state of TRIG1 (i.e. logic high or logic low)
plTrig2Mode - TRIG2 operating mode
plTrig2Polarity - The active state of TRIG2 (i.e. logic high or logic low)

Returns
MG Return Code

Details

The K-Cube motor controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be used to read an external logic signal or output a logic level to control external equipment. Either of them can be independently configured as an input or an output and the active logic state can be selected High or Low to suit the requirements of the application. Electrically the ports output 5 Volt logic signals and are designed to be driven from a 5 Volt logic.

When the port is used in the input mode, the logic levels are TTL compatible, i.e. a voltage level less than 0.8 Volt will be recognised as a logic LOW and a level greater than 2.4 Volt as a logic HIGH. The input contains a weak pull-up, so the state of the input with nothing connected will default to a logic HIGH. The weak pull-up feature allows a passive device, such as a mechanical switch to be connected directly to the input.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method returns the current operating parameters of the TRIG1 and TRIG2 connectors on the front panel of the unit.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode of the trigger is returned in the plTrig1Mode and plTrig2Mode parameters, which takes values from the KMOTTRIGMODE enumeration as follows:

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a relative, absolute or home move as follows:

0x00 The trigger IO is disabled

0x01 General purpose logic input (read through status bits using the LLGetStatusBits method).

0x02 Input trigger for relative move.

0x03 Input trigger for absolute move.

0x04 Input trigger for home move.

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output, or to indicate motion status

or to produce a trigger pulse at configurable positions as follows:

0x0A General purpose logic output (set using the LLSetGetDigOPs method).

0x0B Trigger output active (level) when motor 'in motion'. The output trigger goes high (5V) or low (0V) (as set in the lTrig1Polarity and lTrig2Polarity parameters) when the stage is in motion.

0x0C Trigger output active (level) when motor at 'max velocity'.

0x0D Trigger output active (pulsed) at pre-defined positions moving forward (set using wStartPosFwd, lIntervalFwd, wNumPulsesFwd and lPulseWidth members). Only one Trigger port at a time can be set to this mode.

0x0E Trigger output active (pulsed) at pre-defined positions moving backwards (set using wStartPosRev, lIntervalRev, wNumPulsesRev and lPulseWidth members). Only one Trigger port at a time can be set to this mode.

0x0F Trigger output active (pulsed) at pre-defined positions moving forwards and backward. Only one Trigger port at a time can be set to this mode.

Trigger Out Position Steps

In the last three modes described above, the controller outputs a configurable number of pulses, of configurable width, when the actual position of the stage matches the position values configured as the Start Position and Position Interval - see SetKCubePosTriggerParams method. These mode allow external equipment to be triggered at exact position values. The position pulses are generated by dedicated hardware, allowing a very low latency of less than 1 usec.The low latency of this triggering mode provides a very precise indication of a position match (assuming a stage velocity of 10 mm/sec, the less than 1 usec latency would in itself only result in a 10 nm position uncertainty, which is normally well below the accuracy limitations of the mechanics.)

Position triggering can be configured to be unidirectional (forward or revese only) or bidirectional (both). In bidirectional mode the forward and reverse pulse sequences can be configured separately.A cycle count setting (set in the SetKCubePosTriggerParams method, *lNumCycles* parameter) allows the uni- or bidirectional position triggering sequence to be repeated a number of times.

(A nice picture would be useful here.)

Please note that position triggering can only be used on one TRIG port at a time, as there is only one set of position trigger parameters.

The operation of the position triggering mode is described in more detail in the SetKCubePosTriggerParams method.

The polarity of the trigger pulse is returned in the plTrig1Polarity and plTrig2Polarity parameters, which takes values from the KMOTTRIGPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

For details on setting the trigger parameters, see the SetKCubeTriggerParams method.

KCube Motor Method GetKCubePosTriggerParams

This method is applicable only to K-Cube Series motor controllers

See Also Example Code

Visual Basic Syntax
Function SetKCubePosTriggerParams (lChanID As Long, pfStartPosFwd As Float, pfPosIntervalFwd As Float, plNumPulsesFwd As Long, pfStartPosRev As Float, pfPosIntervalRev As Float, plNumPulsesRev As Long, pfPulseWidth As Float, plNumCycles As Long) As Long

Parameters
lChanID - the channel identifier
pfStartPosFwd - Trigger pulse output position start forward [in position counts - encoder counts or microsteps].
pfPosIntervalFwd - Trigger pulse output position interval forward [in position counts - encoder counts or microsteps].
plNumPulsesFwd - Number of forward output pulses during a move.
pfStartPosRev - Trigger pulse output position start backward [in position counts - encoder counts or microsteps].
pfPosIntervalRev - Trigger pulse output position interval backward [in position counts - encoder counts or microsteps].
plNumPulsesRev - Number of backward output pulses during a move.
pfPulseWidth - Trigger output pulse width (from 1 µs to 1000000 µs).
plNumCycles - Number of forward/reverse move cycles.

Returns
MG Return Code

Details

The K-Cube motor controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be set to be used as input or output triggers. This method returns operating parameters used when the triggering mode is set to a trigger out position steps mode by calling the SetKCubeTriggerParams method.

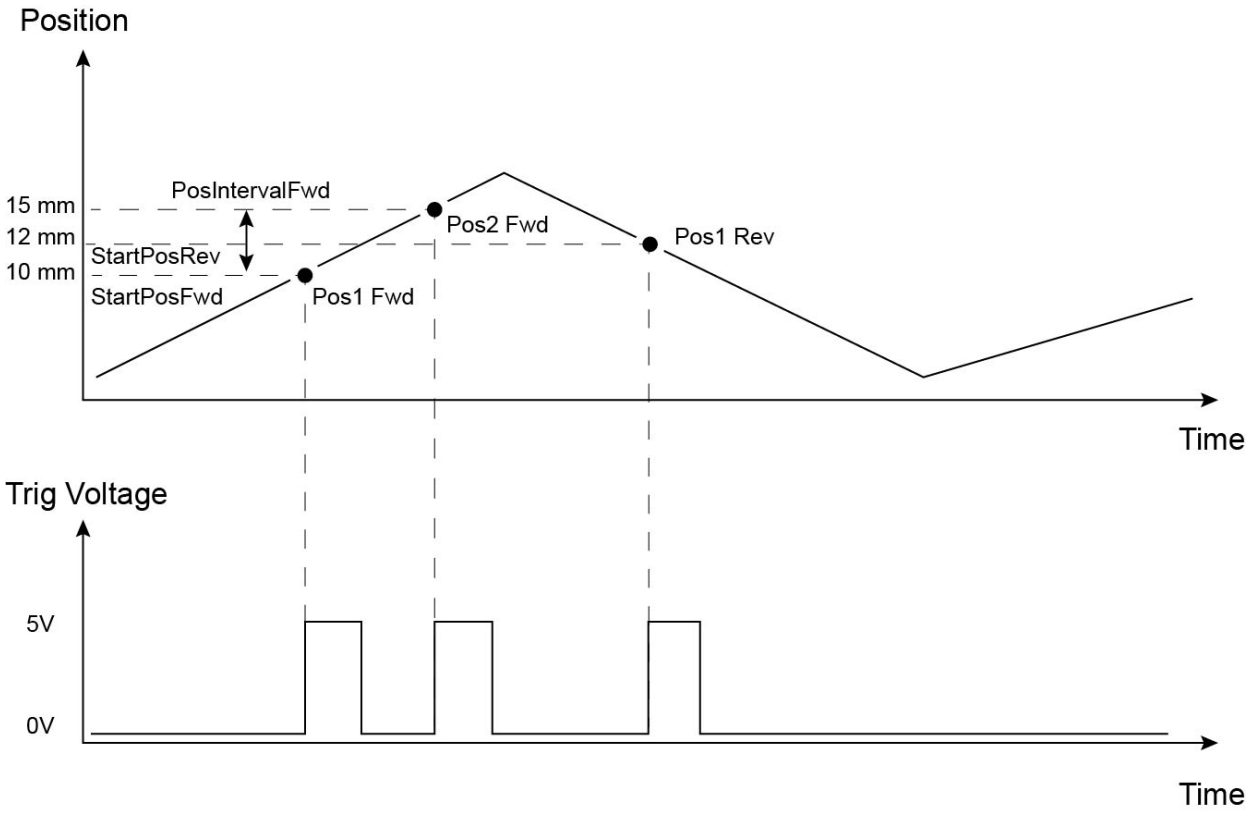
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

As soon as position triggering is selected on either of the TRIG ports, the port will assert the inactive logic state, set in the SetKCubeTriggerParams method. As the stage moves in its travel range and the actual position matches the position returned in the pfStartPosFwd parameter, the TRIG port will output its active logic state. The active state will be output for the length of time returned by the pfPulseWidth parameter, then return to its inactive state and schedule the next position trigger point at the "pfStartPosFwd value plus the value returned in the pfPosIntervalFwd parameter. Thus when this second position is reached, the TRIG output will be asserted to its active state again. The sequence is repeated the number of times returned in the plNumPulsesFwd parameter.

When the number of pulses returned in the plNumPulsesFwd parameter has been generated, the trigger engine will schedule the next position to occur at the position returned in the pfStartPosRev parameter. The same sequence as the forward direction is now repeated in reverse, except that the pfPosIntervalRev and plNumPulsesRev parameters apply. When the number of pulses has been output, the entire forward-reverse sequence will repeat the number of times returned by plNumCycles parameter. This means that the total number of pulses output will be plNumCycles x *(plNumPulsesFwd + plNumPulsesRev).*

Once the total number of output pulses have been generated, the trigger output will remain inactive.

When a unidirectional sequence is selected, only the forward or reverse part of the sequence will be activated.



Example for a move from 0 to 20 mm and back.

In forward direction: The first trigger pulse occurs at 10 mm (StartPosFwd), the next trigger pulse occurs after another 5 mm (PosIntervalFwd), the stage then moves to 20 mm.

In reverse direction: The next trigger occurs when the stage gets to 12 mm.

Note that the position triggering scheme works on the principle of always triggering at the next scheduled position only, regardless of the actual direction of movement. If, for example, a position trigger sequence is set up with the forward start position at 10 mm, but initially the stage is at 15 mm, the first forward position trigger will occur when the stage is moving in the reverse direction. Likewise, if the stage does not complete all the forward position trigger points, the reverse triggering will not activate at all. For normal operation it is assumed that all trigger points will be reached during the course of the movement.

The position triggering sequence can be stopped at any time by changing the triggering function of the port to one of the other options in the SetKCubeTriggerParams method (for example, general purpose output). If the function of the port is then changed back to position triggering, the sequence will re-start from the beginning.

To set the position trigger parameters, see the SetKCubePosTriggerParams method.

Motor Method *GetMotorParams*

See Also Example Code

Visual Basic Syntax

Function **SetMotorParams**(lChanID As Long, plStepsPerRev As Long, plGearBoxRatio As Long) As Long

Parameters

lChanID - the channel identifier

plStepsPerRev - returns the number of full steps per resolution of the stepper motor

plGearBoxRatio - returns the ratio of the motor's gearbox (if fitted)

Returns

MG Return Code

Details

This method is used to obtain the 'resolution' characteristics of the stepper motor connected to the channel specified by the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. The resolution of the motor, combined with other characteristics (such as lead screw pitch) of the associated actuator or stage, determines the overall resolution.

The number of full steps per revolution of the stepper motor is returned in the *plStepsPerRev* parameter, which has minimum and maximum limits of '1' and '1000' respectively.

If the motor has a gearbox, the ratio of the gearbox is returned in the *plGearboxRatio* parameter. For example, if the gearbox has a reduction ratio of X:1 (i.e. every 1 turn at the output of the gearbox requires X turns of the motor shaft) then the *lStepsPerRev* returns the value 'X'. This parameter has minimum and maximum limits of '1' and '1000' respectively.

Note. The stepper motors used on the majority of Our stages/actuators have 200 full steps per rev and no gearbox fitted. For these motors the *lStepsPerRev* and *lGearboxRatio* parameters have values of 200 and 1 respectively. As an exception to this, the ZST family of actuators use 24 steps per rev stepper motors fitted with a 76:1 reduction gearbox. In this case, the *lStepsPerRev* and *lGearboxRatio* should be set to '24' and '76' respectively.

Note. The correct default values for *lStepsPerRev* and *lGearboxRatio* are applied automatically when the APTConfig.exe utility is used to associate a specific stage or actuator type with a motor channel. See the APTConfig helpfile for more details.

The resolution characteristics of the motor are set using the SetMotorParams method.

Motor Method *GetParentHWInfo*

See Also Example Code

Visual Basic Syntax

Function **GetParentHWInfo** (plHWSerialNum As Long, plHWType As Long) As Long

Parameters

plHWSerialNum– the serial number of the host enclosure

plHWType – the hardware type of the host enclosure

Returns

MG Return Code

Details

This method returns the serial number and hardware type identifier of the parent (or host) controller enclosure.

In certain APT products, individual daughter cards or modules are fitted to a common enclosure (e.g. BSC103 three channel benchtop stepper motor controller) which is programmed at the factory with a unique serial number. The *plHWSerialNum* parameter is used to retrieve this serial number and is useful when trying to establish which daughter cards are located within a specific enclosure.

The *plHWType* parameter returns values specified by the APT_PARENT_HWTYPES enumeration as follows:

27 USB_BMC003 3 Channel, 48V Benchtop Motor Controller

29 ETHNET_MMR601 6 Bay Modular Rack, Ethernet (

30 USB_MMR601 6 Bay Modular Rack, USB

Motor Method *GetPIDParams_Deriv*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube DC Driver products ODC001 and TDC001

Visual Basic Syntax

Function **GetPIDParams_Int**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

The OptoDCDriver implements a full servo control loop for motor velocity and position control. The loop response to demanded position moves is determined via Proportional, Integration and Derivative settings. These settings can be altered using the 'Servo Loop (PID) Control Settings' parameters in the settings panel.

Proportional – This term provides the 'restoring' force used to drive the motor to the demand position, reducing the positional error. It accepts values in the range 0 to 32767.

Integral – This term provides the 'restoring' force that grows with time, ensuring that the static position error is zero under a constant torque loading. It accepts values in the range 0 to 32767.

Derivative – This term provides the 'damping' force proportional to the rate of change of the position error. It accepts values in the range 0 to 32767.

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the LLGetPIDParams method, and returns the 'Derivative' parameter value that is returned by reference in the LLGetPIDParams

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Methods *GetPIDParams_Int*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube DC Driver products ODC001 and TDC001

Visual Basic Syntax

Function **GetPIDParams_Int**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

The OptoDCDriver implements a full servo control loop for motor velocity and position control. The loop response to demanded position moves is determined via Proportional, Integration and Derivative settings. These settings can be altered using the 'Servo Loop (PID) Control Settings' parameters in the settings panel.

Proportional – This term provides the 'restoring' force used to drive the motor to the demand position, reducing the positional error. It accepts values in the range 0 to 32767.

Integral – This term provides the 'restoring' force that grows with time, ensuring that the static position error is zero under a constant torque loading. It accepts values in the range 0 to 32767.

Derivative – This term provides the 'damping' force proportional to the rate of change of the position error. It accepts values in the range 0 to 32767.

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the LLGetPIDParams method, and returns the 'Integral' parameter value that is returned by reference in the LLGetPIDParams

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetPIDParams_Prop*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube DC Driver products ODC001 and TDC001

Visual Basic Syntax

Function **GetPIDParams_Prop**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

The OptoDCDriver implements a full servo control loop for motor velocity and position control. The loop response to demanded position moves is determined via Proportional, Integration and Derivative settings. These settings can be altered using the 'Servo Loop (PID) Control Settings' parameters in the settings panel.

Proportional – This term provides the 'restoring' force used to drive the motor to the demand position, reducing the positional error. It accepts values in the range 0 to 32767.

Integral – This term provides the 'restoring' force that grows with time, ensuring that the static position error is zero under a constant torque loading. It accepts values in the range 0 to 32767.

Derivative – This term provides the 'damping' force proportional to the rate of change of the position error. It accepts values in the range 0 to 32767.

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the LLGetPIDParams method, and returns the 'Proportional' parameter value that is returned by reference in the LLGetPIDParams

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetPhaseCurrents*

See Also Example Code

Visual Basic Syntax
Function **GetPhaseCurrents**(lChanID As Long, plRestVal As Long, plMoveVal As Long) As Long

Parameters
lChanID - the channel identifier
plRestVal - the phase current value when the motor is at rest
plMoveVal - the phase current value when the motor is moving

Returns
MG Return Code

Details
The current needed to hold a motor in a fixed position is much smaller than that required for a move. It is good practice to decrease the current in a stationary motor in order to reduce heating, and thereby minimize thermal movements caused by expansion.

This method returns the phase current settings for the motor associated with the channel specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration.
The resting current and the moving current are returned in the *plRestVal* and *plMoveVal* parameters respectively.

Note. For BSC20x series users.
The moving phase current is set automatically by the unit and cannot be adjusted. The *plMoveVal* parameter returns the default value set in the .ini file. The *plRestVal* parameter returns the resting phase current value, which can be set by calling the SetPhaseCurrents method. **Motor Method *GetPosition***

See Also Example Code

Visual Basic Syntax
Function **GetPosition**(lChanID As Long, pfPosition As Single) As Long

Parameters
lChanID - the channel identifier
pfPosition - the current position of the associated channel

Returns
MG Return Code

Details
This method obtains the present position for the channel(s) specified by the *lChanID* parameter, and returns a value in the *pfPosition* parameter.
If a calibration file has been associated with the channel using the APT Config utility, the method returns the calibrated position. If no calibration file has been associated, then the returned position is uncalibrated. Refer to the helpfile in the APT Config utility for further details on using position calibration files.
Note that the position values returned are derived from either the 'Microstep Count' or the Encoder Count, dependent on the positioning mode selected. See the encoder operation section for more information.
The position of the stage associated with the specified channel is determined by its displacement from the 'Home' position.
The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. **Motor Method *GetPosition_Position***

See Also Example Code

Visual Basic Syntax

Function **GetPosition_Position**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetPosition method, and returns the position parameter value that is returned by reference in the GetPosition method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetPositionEx*

See Also Example Code

Visual Basic Syntax

Function **GetPositionEx**(lChanID As Long, pfCalibPosition As Single, pfUncalibPosition As Single) As Long

Parameters

lChanID - the channel identifier

pfCalibPosition - the current calibrated position of the associated channel

pfUncalibPosition - the current uncalibrated position of the associated channel

Returns

MG Return Code

Details

This method obtains the present position for the channel(s) specified by the *lChanID* parameter, and returns values in the *pfCalibPosition and pfUncalibPosition* parameters.

If a calibration file has been associated with the channel using the APT Config utility, the method returns the calibrated position in the *pfCalibPosition* parameter and the uncalibrated position in the *pfUncalibPosition* parameter. If no calibration file has been assigned, then the uncalibrated position is returned in both parameters. Refer to the helpfile in the APT Config utility for further details on using position calibration files.

Note that the position values returned are derived from either the 'Microstep Count' or the Encoder Count, dependent on the positioning mode selected. See the encoder operation section for more information.

The position of the stage associated with the specified channel is determined by its displacement from the 'Home' position.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Motor Method GetPositionEx_UncalibPosition

See Also Example Code

Visual Basic Syntax

Function **GetPositionEx_UncalibPosition**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetPositionEx method, and returns the uncalibrated position parameter value that is returned by reference in the GetPositionEx method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetPositionOffset*

See Also Example Code

Visual Basic Syntax

Function **GetPositionOffset**(lChanID As Long, pfPosOffset As Float) As Long

Parameters

lChanID – The channel identifier

pfPosOffset –The zero offset value

Returns

MG Return Code

Details

This method returns a position offset (previously set using the SetPositionOffset method) for the channel specified by the *lChanID* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the OptoSTDriver and OptoDCDriver, *lChanID* should always be set to 'CHAN1_ID'.

For example, in an application using a 50mm travel linear stage, it may be advantageous to display zero at mid travel, with movement either side of the mid point being displayed as positive or negative. If the *pfPosOffset* parameter returns –25, the GUI position display will show zero at 25mm. A movement of 2mm in a positive direction will be displayed as 2.00 and similarly, a movement of 3mm in a negative direction will be displayed as –3.00.

Motor Method*GetPotParams*

See Also Example Code

Visual Basic Syntax

Function **GetPotParams**((lChanID As Long, plVel1PotVal As Long, pfVel1 As Single, plVel2PotVal As Long, pfVel2 As Single, plVel3PotVal As Long, pfVel3 As Single, plVel4PotVal As Long, pfVel4 As Single,) As Long

Parameters

lChanID – the channel identifier

plVel1PotVal – the Velocity 1 Pot Deflection Value

pfVel1 – the velocity associated with deflection value 1

plVel2PotVal – the Velocity 2 Pot Deflection Value

pfVel2 – the velocity associated with deflection value 2

plVel3PotVal – the Velocity 3 Pot Deflection Value

pfVel3 – the velocity associated with deflection value 3

plVel4PotVal – the Velocity 4 Pot Deflection Value

pfVel4 – the velocity associated with deflection value 4

Returns

MG Return Code

Details

This method returns a number of potentiometer deflection values and associated velocites pertaining to the operation of the motor controller unit.

Note. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the OptoSTDriver and OptoDCDriver, *lChanID* should always be set to 'CHAN1_ID'.

The potentiometer slider is sprung such that when released it returns to it's central position. In this central position the motor is stationary. As the slider is moved away from the center, the motor begins to move; the speed of this movement increases as the slider deflection is increased. Bidirectional control of motor moves is possible by moving the slider in both directions. The speed of the motor increases by discrete amounts rather than continuously, as a function of slider deflection. These speed settings can be altered via the 'Potentiometer Control Settings' parameters.

There are 4 pairs of parameters, each pair specifies a pot deflection value (in the range 0 to 127) together with an associated velocity (set in real world units, mm or degrees) to be applied at or beyond that deflection. As each successive deflection is reached by moving the pot slider, the next velocity value is applied. These settings are applicable in either direction of pot deflection, i.e. 4 possible velocity settings in the forward or reverse motion directions.

These parameters can be set by calling the SetPotParams method.

Motor Method *GetRelMoveDist*

See Also Example Code

Visual Basic Syntax

Function **GetRelMoveDist**(lChanID As Long, pfRelDist As Single) As Long

Parameters

lChanID - the channel identifier

pfRelDist - the relative distance associated with the specified channel

Returns

MG Return Code

Details

This method obtains the relative distance which the channel specified by the *lChanID* parameter will move, the next time the *MoveRelative* method is called, and returns a value in the *pfRelDist* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot return the relative distance of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To set the relative distances for a particular channel, see the SetRelMoveDist method.

Motor Method *GetRelMoveDist_RelDist*

See Also Example Code

Visual Basic Syntax

Function **GetRelMoveDist_RelDist**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetRelMoveDist method, and returns the relative distance parameter value that is returned by reference the GetRelMoveDist

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetRotStageModes*

See Also Example Code

Note. This method is applicable only to Rotational Stages

Visual Basic Syntax

Function **GetRotStageModes** (plMoveMode As Long, plPosReportMode As Long) As Long

Parameters

plMoveMode - the move direction, positive, negative or quickest

plPosReportMode – the position reporting mode, 360 or total.

Returns

MG Return Code

Details

This method is applicable only to rotational stages. It returns values which specify the direction in which a move is performed, and the way in which positional information is reported.

The *plMoveMode* parameter returns the move direction and takes values from the ROTMOVEMODE enumeration as follows:

1. ROT_MOVE_POS Rotate Positive
2. ROT_MOVE_NEG Rotate Negative
3. ROT_MOVE_SHORT Rotate Quickest

Note. The *lMoveMode* parameter is applicable only if the Absolute Position Reporting Mode is set to 'Equivalent Angle 0 to 360 degrees' in the Rotation Stages settings panel – see the Operation section, Rotation Stages tab, OR the *lPosReportMode* parameter (see below) is set to ROT_POSDISP_360.

The *lPosReportMode* parameter returns the position reporting mode. It takes values from the ROTPOSDISPMODE enumeration as follows:

1. ROT_POSDISP_360 Position reporting is 0 to 360°
2. ROT_POSDISP_TOTAL Position reporting is the total degree count

Motor Method *GetStageAxisInfo*

See Also Example Code

Visual Basic Syntax

Function **GetStageAxisInfo**(lChanID As Long, pfMinPos As Single, pfMaxPos As Single, plUnits As Long, pfPitch As Single, plDirSense As Long) As Long

Parameters

lChanID - the channel identifier

pfMinPos - the minimum limit position

pfMaxPos - the maximum limit position

plUnits - the units (mm or degrees)

pfPitch - the pitch (in mm) of the motor lead screw (i.e. the distance to travel per revolution of the motor)

plDirSense - reserved for future use

Returns

MG Return Code

Details

This method retrieves information describing the properties of the stage and axis associated with the channel specified by the *lChanID.* These settings are specific to the design of the associated stage.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The *plUnits* parameter takes values specified by the STAGEUNITS enumeration and contains the units of measure for stage motion, either millimeters or degrees, whichever is relevant.

The positions of the minimum and maximum travel limits are returned in the *pfMinPos* and *pfMaxPos* parameters respectively, and are determined by their displacement from the Home position. pfMinPos is usually set to zero.

The *pfZeroOffset* parameter returns the distance between the Home position and the minimum limit - see Motor Controller Operators Guide for further information.

The *pfPitch* parameter returns the pitch of the motor leadscrew (in mm or degrees per rev), which is used to calculate the distance moved for each revolution of the motor.

The stage axis information for a particular channel can be set using the SetStageAxisInfo method.

Motor Method *GetStageAxisInfo_MaxPos*

See Also Example Code

Visual Basic Syntax

Function **GetStageAxisInfo_MaxPos** (lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetStageAxisInfo method, and returns the maximum position parameter value that is returned by reference the GetStageAxisInfo method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetStageAxsInfo_MinPos*

See Also Example Code

Visual Basic Syntax

Function **GetStageAxisInfo_MinPos** (lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetStageAxisInfo method, and returns the minimum position parameter value that is returned by reference in the GetStageAxisInfo method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetStatusBits_Bits*

See Also Example Code

Visual Basic Syntax

Function **GetStatusBits_Bits** (lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the LLGetStatusBits method, and returns the status bits parameter value that is returned by reference in the LLGetStatusBits method.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetTriggerParams*

See Also Example Code

Note for BSC001 or BSC002 Users. On units to modification state B or earlier, triggering functionality is enabled by fitting a trigger enable plug. These plugs are available from the company on request.

A label indicating the build and modification state of the hardware is located on the underside of the unit. Modification states are 'scored out' as they become current.

Note. This method is not applicable to BSC101, BSC103, OptoSTDriver (OST001) and OptoDCDriver (ODC001) units.

Visual Basic Syntax

Function **GetTriggerParams**(lChanID As Long, plTrigMode As Long, plTrigMove As Long) As Long

Parameters

lChanID - the channel identifier

plTrigMode – gets the trigger mode setting for the specified channel

plTrigMove – gets the type of move to be initiated on receipt of a trigger signal

Returns

MG Return Code

Details

This method is used to return the configuration of the APT stepper controller for triggered move operation. It is possible to configure a particular controller to respond to trigger inputs, generate trigger outputs or both respond to and generate a trigger output. When a trigger input is received, the stepper unit can be set to initiate a move (relative, absolute or home). Similarly the unit can be set to generate a trigger output signal when instructed to move via a software (USB) command. For those units configured for both input and output triggering, a move can be initiated via a trigger input while at the same time, a trigger output can be generated to initiate a move on another unit.

The trigger settings can be used to configure multiple units in a master/slave set up, thereby allowing multiple channels of motion to be synchronized. Multiple moves can then be initiated via a single command.

The channel whose settings are to be returned is specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration.

The *plTrigMode* parameter gets the trigger mode for the specified channel. The various trigger modes are specified by the TRIGMODE enumeration as follows:

TRIGMODE_DISABLED – triggering operation is disabled

TRIGMODE_IN – a move (specified using the *lTrigMove* parameter below) is initiated on the specified channel when a trigger input signal (i.e. rising edge) is received on the Trig In connector (either BNC or D-type pin depending on the unit).

TRIGMODE_INOUT – As for TRIGMODE_IN, but now a trigger output signal will also be generated on the Trig Output

connector when the move begins.

TRIGMODE_OUT – A trigger output signal is generated on the Trig Out connector when a move (relative, absolute or home) is initiated via software (either through the GUI panel or ActiveX method call).

The *plTrigMove* parameter determines the type of move to be initiated on the specified channel when a trigger input signal is detected on the Trig In connector. The move types are specified by the TRIGMOVE enumeration as follows:

TRIGMOVE_REL – triggered relative move. Relative move distance and velocity profile parameters can be set prior to triggering using the SetRelMoveDist and SetVelParams methods respectively.

TRIGMOVE_ABS – triggered absolute move. Absolute move position and velocity profile parameters can be set prior to triggering using the SetAbsMovePos and SetVelParams methods respectively.

TRIGMOVE_HOME – triggered home sequence. Homing parameters can be set prior to triggering using the SetHomeParams method

The trigger parameters are set using the SetTriggerParams method.

Please refer to the handbook supplied with your unit for further information on identifying the trigger connections.

Motor Method *GetVelParams*

See Also Example Code

Visual Basic Syntax

Function **GetVelParams**(lChanID As Long, pfMinVel As Single, pfAccn As Single, pfMaxVel As Single) As Long

Parameters

lChanID - the channel identifier

pfMinVel - the returned minimum velocity at which to start a move

pfAccn - the returned rate at which the velocity climbs from minimum to maximum, and slows from maximum to minimum

pfMaxVel - the returned maximum velocity at which to perform a move

Returns

MG Return Code

Details

This method obtains the trapezoidal velocity profile parameters for all moves other than jogs. The values are returned in the appropriate parameter as real world units (mm or degrees) dependent upon stage type.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The velocity parameters associated with a particular channel can be set by calling the SetVelParams method.

Motor Method *GetVelParams_Accn*

See Also Example Code

Visual Basic Syntax

Function **GetVelParams_Accn**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetVelParams method, and returns the acceleration parameter value that is returned by reference in the GetVelParams method

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetVelParams_MaxVel*

See Also Example Code

Visual Basic Syntax

Function **GetVelParams_MaxVel**(lChanID As Long) As Float

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

Certain development environments, e.g. MatLab, can only support returns by value. This method acts as a wrapper for the GetVelParams method, and returns the maximum velocity parameter value that is returned by reference in the GetVelParams method.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Motor Method *GetVelParamLimits*

See Also Example Code

Visual Basic Syntax

Function **GetVelParamLimits**(lChanID As Long, pfMaxAccn As Single, pfMaxVel As Single) As Long

Parameters

lChanID - the channel identifier

pfMaxAccn - the returned maximum allowable acceleration (and deceleration) for the APT controller

pfMaxVel - the returned maximum allowable velocity for the APT controller.

Returns

MG Return Code

Details

This method returns the maximum values that can be set for the velocity profile parameters. These maximum values are dependent upon the stage and axis associated with the specified channel, and are returned in the appropriate parameter as real world units (mm or degrees).

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Example

```
Private Sub cmdGetVelLimits_Click()
```

```
' Get velocity limits

Dim sngMaxVelLimit As Single, sngMaxAccnLimit As Single

MG17Motor1.GetVelParamLimits CHAN1_ID, sngMaxAccnLimit, sngMaxVelLimit

MsgBox "Maximum velocity allowed = " & sngMaxVelLimit & vbCrLf & _

"Maximum acceleration allowed = " & sngMaxAccnLimit

End Sub
```

Method *Identify*

See Also Example Code

Visual Basic Syntax

Function **Identify**() As Long

Returns

MG Return Code

Details

This method allows the hardware unit associated with the ActiveX control instance to be identified. The associated unit is specified by the HWSerialNum property.

When the method is called, the front panel LEDs on the relevant hardware unit will flash for a short period.

Method *LoadParamSet*

See Also Example Code

Visual Basic Syntax

Function **LoadParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be loaded

Returns

MG Return Code

Details

This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

Motor Method *LLGetADCInputs*

See Also Example Code

Visual Basic Syntax

Function **GetADCInputs** (plADCVal1 As Long, plADCVal2 As Long) As Long

Parameters

plADCVal1 - the value on the analog input

plADCVal2 – reserved for future use.

Returns

MG Return Code

Details

This method reads the voltage applied to the analog input on the rear panel CONTROL IO connector, and returns a value in the *plADCVal1* parameter. The returned value is in the range 0 to 32768, which corresponds to zero to 5 V.

Note. The *plADCVal2* parameter is reserved for future use.

In this way, a 0 to 5V signal generated by a client system could be read in by calling this method and monitored by a custom client application. When the signal reaches a specified value, the application could instigate further actions, such as a motor move.

T-Cube Motor Method LLGetEncoderCount

See Also Example Code

Visual Basic Syntax

Function **LLGetEncoderCount**(lChanID As Long, plEncCount As Long) As Long

Parameters

lChanID – the channel identifier

plEncCount – the encoder count.

Returns

MG Return Code

Details

In certain applications (e.g. where the time taken to perform a home move takes an inconveniently long time), it may be advantageous to move to a particular absolute position and then set the number of encoder counts to a previously measured value.

This low level method returns the encoder count value for the channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The encoder count value is returned in the *plEncCount* parameter.

Note. Ensure that the motor is at the required absolute position before calling this method.

The encoder count value for a particular position may be set by calling the LLSetEncoderCount method.

Motor Method *LLGetStatusBits*

See Also Example Code

Visual Basic Syntax

Function **LLGetStatusBits**((lChanID As Long, plStatusBits As Long) As Long

Parameters

lChanID – the channel identifier

plStatusBits – the 32-bit integer containing the status flags.

Returns

MG Return Code

Details

This low level method returns a number of status flags pertaining to the operation of the motor controller channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description
1. x00000001	1.	CW hardware limit switch (0 - no contact, 1 - contact).
2. x00000002	2	CCW hardware limit switch (0 - no contact, 1 - contact).
0x00000004	3.	CW software limit switch (0 - no contact, 1 - contact). *Not applicable to Part Number ODC001 and TDC001 controllers*
0x00000008	4.	CCW software limit switch (0 - no contact, 1 - contact). *Not applicable to Part Number ODC001 and TDC001 controllers*
0x00000010	5.	Motor shaft moving clockwise (1 - moving, 0 - stationary).
0x00000020	6.	Motor shaft moving counterclockwise (1 - moving, 0 - stationary)
0x00000040	7. 22)	Shaft jogging clockwise (1 - moving, 0 - stationary).
0x00000080	8. 21)	Shaft jogging counterclockwise (1 - moving, 0 - stationary).
0x00000100	9.	Motor connected (1 - connected, 0 - not connected). *Not applicable to Part Number BMS001 and BMS002 controllers* *Not applicable to Part Number ODC001 and TDC001 controllers*
0x00000200	10.	Motor homing (1 - homing, 0 - not homing).
0x00000400	11.	Motor homed (1 - homed, 0 - not homed).
0x00000800	12.	For Future Use

Note. *Bits 13 to 20 are applicable only to the BBD10x series brushless DC controllers*

0x00001000	13.	Trajectory within tracking window (1 – within window, 0 – not within window)
0x00002000	14.	Axis within settled window (1 – settled within window, 0 – not settled within window)
0x00004000	15.	Axis exceeds position error limit (1 – limit exceeded, 0 – within limit)
0x00008000	16.	Set when position module instruction error exists (1 – instruction error exists, 0 – no error)
0x00010000	17.	Interlock link missing in motor connector (1 – missing, 0 – present)
0x00020000	18.	Position module over temperature warning (1 – over temp, 0 – temp OK)
0x00040000	19.	Position module bus voltage fault (1 – fault exists, 0 – OK)
0x00080000	20.	Axis commutation error (1 – error, 0 – OK)

***Note**. Bits 21 to 26 (Digital Input States) are only applicable if the associated digital input is fitted to your controller – see the relevant handbook for more details*

0x00100000	21.	Digital input 1 state (1 - logic high, 0 - logic low).
0x00200000	22.	Digital input 2 state (1 - logic high, 0 - logic low).
0x00400000	23.	Digital input 3 state (1 - logic high, 0 - logic low).
0x00800000	24.	Digital input 4 state (1 - logic high, 0 - logic low).
0x01000000	25.	*BBD10x Controllers* Axis phase current limit (1 – current limit exceeded, 0 – below limit) *Other Controllers* Digital input 5 state (1 - logic high, 0 - logic low).
0x02000000	26.	Digital input 6 state (1 - logic high, 0 - logic low).
	27. to 29	For Future Use
0x20000000	30	Active (1 – indicates unit is active, 0 – not active)
0x40000000	31	For Future Use

0x80000000	32	Channel enabled (1 – enabled, 0- disabled)

Motor Method *LLGetStatusBits (BBD)*

Note. The status bits described here are applicable only to the BBD10x series of Brushless DC Motor Controllers

See Also Example Code

Visual Basic Syntax

Function **LLGetStatusBits**((lChanID As Long, plStatusBits As Long) As Long

Parameters

lChanID – the channel identifier

plStatusBits – the 32-bit integer containing the status flags.

Returns

MG Return Code

Details

This low level method returns a number of status flags pertaining to the operation of the motor controller channel associated with the ActiveX control.

The *lChanID* parameter is not valid for the BBD series brushless DC motor controller and is preset at the factory to '1'.

These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description
1. x00000001	1.	CW or +VE hardware limit switch (0 - no contact, 1 - contact).
2. x00000002	2	CCW or –VE hardware limit switch (0 - no contact, 1 - contact).
0x00000004	3.	CW or +VE software limit switch (0 - no contact, 1 - contact).
0x00000008	4.	CCW or -VE software limit switch (0 - no contact, 1 - contact).
0x00000010	5.	Motor shaft moving clockwise or forwards (1 - moving, 0 - stationary).
0x00000020	6.	Motor shaft moving counterclockwise or backwards (1 - moving, 0 - stationary)
0x00000040	7.	Shaft jogging clockwise (1 - moving, 0 - stationary).
0x00000080	8.	Shaft jogging counterclockwise (1 - moving, 0 - stationary).
0x00000100	9.	Motor connected (1 - connected, 0 - not connected).
0x00000200	10.	Motor homing (1 - homing, 0 - not homing).
0x00000400	11.	Motor homed (1 - homed, 0 - not homed).
0x00000800	12.	For Future Use
0x00001000	13.	Trajectory within tracking window (1 – within window, 0 – not within window) See SetDCTrackSettleParams method for more details
0x00002000	14.	Axis within settled window (1 – settled within window, 0 – not settled within window) See SetDCTrackSettleParams method for more details
0x00004000	15.	Axis exceeds position error limit (1 – limit exceeded, 0 – within limit) See SetDCPositionLoopParams method for more details
0x00008000	16.	Set when position module instruction error exists

		(1 – instruction error exists, 0 – no error) Typically, this error is generated when a move has been demanded which is impossible to execute, e.g. the stage has hit a limit switch, yet a command exists to continue moving in the same direction.
0x00010000	17.	Interlock link missing in motor connector (1 – missing, 0 – present) This error usually means that the motor has been disconnected.
0x00020000	18.	Position module over temperature warning (1 – over temp, 0 – temp OK) The internal temperature sensor senses an overtemperature condition, and the output is disabled.
0x00040000	19.	Position module bus voltage fault (1 – fault exists, 0 – OK)
0x00080000	20.	Axis commutation error (1 – error, 0 – OK) The motor commutation block has detected a fault.
0x00100000	21.	Digital input 1 state (1 - logic high, 0 - logic low).
	22. to 24	Not Used
0x01000000	25	Axis phase current limit (1 – current limit exceeded, 0 – below limit) The motor current has exceeded the predefined safe limit, so the current limiter has been activated to prevent damage to the motor coils – see SetDCMotorOutputParams
0x02000000	26 to 29	Not Used
0x20000000	30	Active (1 – indicates unit is active, 0 – not active)
0x40000000	31	For Future Use
0x80000000	32	Channel enabled (1 – enabled, 0- disabled)

Piezo Method *LLSaveHWDefaults*

See Also Example Code

Visual Basic Syntax

Function **LLSaveHWDefaults**() As Long

Returns

MG Return Code

Details

This method allows the current settings of the operation parameters to be saved into the onboard 'Flash' memory of the hardware unit. These parameters then become the default settings when the unit is next powered up.

Note. These defaults are overridden when the APT server is booted and begins communication with the hardware unit.

There may be occasions when the hardware unit is operated in the absence of the APT server, in which case the parameter defaults saved using this method then apply. For example, it may be desired to operate the unit as a stand alone piezo amplifier.

Typically however, the APT units will be operated using the PC server program.

T-Cube Motor Method LLSetEncoderCount

See Also Example Code

Visual Basic Syntax

Function **LLSetEncoderCount**(lChanID As Long, lEncCount As Long) As Long

Parameters

lChanID – the channel identifier

lEncCount – the encoder count.

Returns

MG Return Code

Details

In certain applications (e.g. where the time taken to perform a home move takes an inconveniently long time), it may be advantageous to move to a particular absolute position and then set the number of encoder counts to a previously measured value.

This low level method allows the encoder counts to be set for the channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The required encoder count value is set in the *lEncCount* parameter.

Note. Ensure that the motor is at the required absolute position before calling this method.

The encoder count value at a particular position may be obtained by calling the LLGetEncoderCount method.

Motor Method *LLSetPosition*

See Also Example Code

Visual Basic Syntax
Function **LLSetPosition**(lChanID As Long, lPosition As Long) As Long

Parameters
lChanID – the channel identifier
lPosition – the position to be set.

Returns
MG Return Code

Details
This method sets the position for the channel specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration.

In normal circumstances, the stage is homed immediately after power-up (at this stage the position is unknown as the stage is free to move when the power is off); and after the homing process is completed the position counter is automatically updated to show the actual position. From this point onwards the position counter always shows the actual absolute position.

The method is used to set the 'live' position count in the controller. The required position is set in the *lPosition* parameter in hardware units. The new value of the position counter as a 32-bit signed integer, encoded in the Intel format. The scaling between real time values and this parameter is dependent on the stage being driven and is detailed below.

Conversion between position, velocity and acceleration values in standard physical units and their equivalent APT parameters.

To convert between the position and encoder counters in the stage being driven, and real world units, (e.g. mm) the system uses certain conversion (scaling) factors. These conversion factors differ depending on the stage being driven and the controller being used.

Background

The principle described below is the same for all APT motion stepper and brushed or brushless DC controllers and stages, but the individual distance and time conversion factors will be typically different for each stage and/or controller.

In real life, the physical units needed to describe position, velocity and acceleration are related to position and time measurement units (millimetres/degrees and seconds). In motion controllers, however, normally the system only knows the distance travelled in encoder counts (pulses) as measured by an encoder fitted to the motor shaft. In most cases the motor shaft rotation is also scaled down further by a gearbox and a leadscrew. In any case, the result is a scaling factor between encoder counts and position. The value of this scaling factor depends on the stage. In the section below this scaling factor will be represented by the symbol EncCnt.

Time is related to the sampling interval of the system, and as a result, it depends on the motion controller. Therefore, this value is the same for all stages driven by a particular controller. In the sections below the sampling interval will be denoted by T.

The sections below describe the position, velocity and acceleration scaling factors for all the controllers and stages that are used with these controllers. The symbols POS_{APT}, VEL_{APT} and ACC_{APT} are used to denote the position, velocity and acceleration values used in APT commands, whereas the symbols Pos, Vel and Acc denote physical position, velocity and acceleration values in mm, mm/sec and mm/sec^2 units for linear stages and degree, degree/sec and degree/sec^2 for rotational stages.

As APT parameters are integer values, the APT values calculated from the equations need to be rounded to the nearest integer.

Brushed DC Controller (TDC001) driven stages

Mathematically:

$POS_{APT} = EncCnt \times Pos$

$VEL_{APT} = EncCnt \times T \times 65536 \times Vel$

$ACC_{APT} = EncCnt \times T^2 \times 65536 \times Acc$

where $T = 2048 / 6 \times 10^6$

The value of EncCnt and the resulting conversion factors are listed below for each stage:

Stage	EncCnt	Scaling Factor		
		Position	Velocity	Acceleration
MTS25-Z8	34304	34304	767367.49	261.93
MTS50-Z8	34304	34304	767367.49	261.93
PRM1-Z8	1919.64	1919.64	42941.66	14.66
Z8xx	34304	34304	767367.49	261.93
Z6xx	24600	24600	550292.68	187.83

Brushless DC Controller (TBD001, BBD10X and BBD20X) driven stages

Mathematically:

$POS_{APT} = EncCnt \times Pos$

$VEL_{APT} = EncCnt \times T \times 65536 \times Vel$

$ACC_{APT} = EncCnt \times T^2 \times 65536 \times Acc$

where $T = 102.4 \times 10^{-6}$

The value of EncCnt and the resulting conversion factors are listed below for each stage:

Stage	EncCnt	Scaling Factor		
		Position	Velocity	Acceleration
DDSM100	2000	2000	13421.77	1.374
DDS220	20000	20000	134217.73	13.744
DDS300	20000	20000	134217.73	13.744
DDS600	20000	20000	134217.73	13.744
MLS203	20000	20000	134217.73	13.744

Stepper Motor Controller (TST001 BSC00x, BSC10x, MST601) Driven Stages

For these stepper controllers the server sends absolute micro-steps to the controllers. Depending on the stage and the stepper motor concerned there are different micro step values required to move either a linear distance in millimetres or a rotational distance in degrees.

In general for 200 full step motors (the majority of our motors) the above range of stepper controllers is designed to insert 128 micro steps for every full step of the stepper.

So for a 200 full step motor the number of micro steps per full turn is defined as follows

Full turn micro steps = Motor full steps per turn x Number of Micro steps per full step

For a 200 full step motor this is given by :-

Full turn micro steps = 200 x 128 = 25600

Each stage can either be a direct drive or driven through a gear box. The table below indicates the relationship between absolute micro steps and a positional output in millimetres or degrees

This table is relevant for the range of controllers listed above. Note that micro step values are for a position is 1mm, a velocity of 1mm/sec and an acceleration of 1mm/sec/sec

tage	Gearing	Position	Micro Step Values		
			Position(µs)	Velocity(µs/sec)	Acceleration(µs/sec^2)
RV001	0.5mm/turn	1mm	51200	51200	51200
RV013	1mm/turn	1mm	25600	25600	25600
RV014	1mm/turn	1mm	25600	25600	25600
RV113	1.25mm/turn	1mm	20480	20480	20480
RV114	1.25mm/turn	1mm	20480	20480	20480
W103*	No gear	0.998deg	71	71	71
R360**	5.4546deg/turn	0.999deg	4693	4693	4693

*Note that there is no exact value of micro steps to get to exactly 1 degree this is because 1 turn represents 360 degrees which is 25600 micro steps. So actual resolution is

360/25600 = 0.0140625 degrees per micro step.

**Note that there is no exact value of micro steps to get to exactly 1 degree this is because 1 turn represents 5.4546 degrees which is 25600 micro steps. So actual resolution is

5.4546/25600 = 0.0002131 degrees

Stepper Motor Controller (BSC20x, MST602) Driven Stages

The BSC20x series and MST602 stepper controllers include a Trinamics encoder with a resolution of 409600 micro-steps per revolution

This table is relevant only for the Trinamic-based range of controllers listed above. Note that micro step values are for a position is 1mm, a velocity of 1mm/sec and an acceleration of 1mm/sec/sec

Or in degrees, deg/sec or deg/sec/sec

tage	Gearing	Position	Trinamic converted Values		
			Position(µs)	Velocity(µs/sec)	Acceleration(µs/sec^2)
RV001	0.5mm/turn	1mm	819200	43974656	9012
RV013	1mm/turn	1mm	409600	21987328	4506
RV014	1mm/turn	1mm	409600	21987328	4506
RV113	1.25mm/turn	1mm	327680	17589862	3605
RV114	1.25mm/turn	1mm	327680	17589862	3605
W103*	No gear	1.0002deg	1138	61088	13
R360**	5.4546deg/turn	0.99997deg	75091	4030885	826

In the above table the numbers that need to be sent to the controllers are based upon the Trinamics chip set conversions. The position is just the absolute number of micro-steps as before, as compared with the BSC10X range, the only difference is the 16 times greater resolution. However for velocity and acceleration now need different conversion factors to get to correct motion profiles. For example, if a velocity of 409600 micro-steps per sec is required, then multiply by 53.68 i.e. 409600*53.68 gives 21987328 which for a 1mm lead screw would give 1mm/sec.

To accelerate at a rate of 409600 micro-steps/sec/sec (1mm/sec/sec), divide 409600 by 90.9 which gives 4506.

Motor Method *MoveAbsolute*

See Also Example Code

Visual Basic Syntax

Function **MoveAbsolute**(lChanID As Long, bWait As Boolean) As Long

Parameters

lChanID - the channel identifier

bWait - specifies the way in which the MoveAbsolute method returns

Returns

MG Return Code

Details

This method initiates a motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The absolute position to move is specified by the SetAbsMovePos method.

If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveAbsolute returns only after the motors have completed their moves. In either mode, a MoveComplete event is fired once the motor moves have been completed.

When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the MoveComplete event. This event driven approach allows a client application to service other tasks while the motors are moving.

Note that in order to specify completely an absolute move, the *SetAbsMovePos* method must be called prior to calling the *MoveAbsolute* method.

Note: The MoveAbsoluteEx method allows the absolute position values to be specified in the parameter list.

To get the absolute position for a particular channel, see the GetAbsMovePos method.

Motor Method *MoveAbsoluteEnc*

See Also Example Code

Visual Basic Syntax

Function **MoveAbsoluteEnc**(lChanID As Long, fAbsPosCh1 As Float, fAbsPosCh2 As Float, lTimeInc As Long, bWait As Variant_Bool) As Long

Parameters

lChanID - the channel identifier

fAbsPosCh1 - the absolute position associated with channel 1

fAbsPosCh2 - the absolute position associated with channel 2

lTimeInc – specifies the time out delay

bWait - specifies the way in which the MoveAbsoluteEnc method returns

Returns

MG Return Code

Details

This method initiates an absolute motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Encoder position correction is supported if 'Position (& Correct)' or 'Position (Stop Short & Correct)' is selected. (see the SetEncPosControlParams method or the Encoder Operation section).

The absolute position to move is specified by the *fAbsPosCh1* and *fAbsPosCh2* parameters. **Note**. If the *lChanID* parameter is set to a single channel then the unspecified *fAbsPos* parameter is ignored.

If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveAbsoluteEnc returns only after the motors have completed their moves. In either mode, a MoveComplete event is fired once the motor moves have been completed.

When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the MoveComplete event. This event driven approach allows a client application to service other tasks while the motors are moving.

When a position correction mode is enabled, it is possible in some circumstances, for final position correction to take a considerable time (e.g. mechanically noisy systems or excessive stop short distance parameters used). The *lTimeInc* parameter specifies an extra timeout delay (in milliseconds) that the system adds to the internally calculated move time out normally applied to a non-encoded move.

Note. The timeout is only applied if the *bWait* parameter is set to TRUE, causing the software call to wait until the move has been completed.

If it is found that encoded moves are timing out before the move completes, then the value of the *lTimeInc* parameter should be increased (empirically) until the timeouts are eliminated.

Note: If an encoder calibration table has been acquired (see SetEncCalibTableParams) and enabled (see SetEncPosControlParams), then all absolute motor moves (MoveAbsolute, MoveAbsoluteEx and MoveAbsoluteEnc) will operate using the encoder calibration table. However, the MoveAbsoluteEnc method must be used if position correction functionality is to be invoked at the end of an encoded move.

Motor Method *MoveAbsoluteEx*

See Also Example Code

Visual Basic Syntax

Function **MoveAbsoluteEx**(lChanID As Long, dAbsPosCh1 As Double, dAbsPosCh2 As Double, bWait As Boolean) As Long

Parameters

lChanID - the channel identifier

dAbsPosCh1 - the absolute position associated with channel 1

dAbsPosCh2 - the absolute position associated with channel 2

bWait - specifies the way in which the MoveAbsoluteEx method returns

Returns

MG Return Code

Details

This method initiates a motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The absolute position to move is specified by the *dAbsPosCh1* and *dAbsPosCh2* parameters. **Note**. If the *lChanID* parameter is set to a single channel then the unspecified *dAbsPos* parameter is ignored.

If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveAbsoluteEx returns only after the motors have completed their moves. In either mode, a MoveComplete event is fired once the motor moves have been completed.

When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the MoveComplete event. This event driven approach allows a client application to service other tasks while the motors are moving.

Note: See the MoveAbsolute method for an alternative way of moving to an absolute position.

Motor Method *MoveAbsoluteRot*

See Also Example Code

Note. This method is applicable only to Rotational Stages.

Visual Basic Syntax

Function **MoveAbsoluteRot**(lChanID As Long, fAnglePosCh1 As Float, fAnglePosCh2 As Float, lMoveMode As Long, bWait As Boolean) As Long

Parameters

lChanID - the channel identifier

fAnglePosCh1 – the absolute position to move for channel 1

fAnglePosCh2 - the absolute position to move for channel 2

lMoveMode – the move direction, positive, negative or quickest

bWait - specifies the way in which the MoveAbsoluteRot method returns.

Returns

MG Return Code

Details

This method is applicable only to rotational stages. It initiates a motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The absolute position to move for channel 1 and channel 2 is specified by the *fAnglePosCh1* and *fAnglePosCh2* parameters respectively. If the *lChanID* parameter is set to channel 1, the *fAnglePosCh2* parameter is ignored, and vice versa.

The *lMoveMode* parameter specifies the move direction and takes values from the ROTMOVEMODE enumeration as follows:

1. ROT_MOVE_POS Rotate Positive
2. ROT_MOVE_NEG Rotate Negative
3. ROT_MOVE_SHORT Rotate Quickest

Note. The *lMoveMode* parameter is applicable only if the Absolute Position Reporting Mode is set to ‘Equivalent Angle 0 to 360 degrees’ in the Rotation Stages settings panel – see the Operation section, Rotation Stages tab.

If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True',

MoveAbsoluteRot returns only after the motors have completed their moves. In either mode, a MoveComplete event is fired once the motor moves have been completed.

When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the MoveComplete event. This event driven approach allows a client application to service other tasks while the motors are moving.

Motor Method *MoveHome*

See Also **Example Code**

Visual Basic Syntax

Function **MoveHome**(lChanID As Long, bWait As Boolean) As Long

Parameters

lChanID - the channel identifier

bWait - specifies the way in which the MoveHome method returns

Returns

MG Return Code

Details

This method initiates the homing sequence on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Each axis on the associated stage has settings for the parameters Zero Offset and Minimum Position. The first value is the distance between the negative limit switch and the end of travel. The second value is the minimum absolute position that can be set for the stage axis. Typically, when *MoveHome* is called, the stage axis moves to its negative limit and then moves forward by a set distance (zero offset). The absolute position count is then reset to zero to provide the reference point for all subsequent absolute moves. If position is lost on a stage axis, the *MoveHome* method should be called to re-establish the zero (home) position.

Note. For stages without limit switches, (e.g. the CR1-Z6 or T25X) a zero reference point can be established by moving the stage manually to its zero position and then clicking the Home/Zero button. Set the Minimum and Maximum positions as required , either in the settings panel, or by calling the SetStageAxisInfo method.

If the *bWait* parameter is set to 'False', the method returns as soon as the homing sequence has been initiated. If *bWait* is set to 'True', *MoveHome* returns only after the motors have finished homing. In either mode, a HomeComplete event is fired once the homing sequence has been completed.

When a client application needs to perform a homing sequence, it is more efficient programming practice to set *bWait* to 'False' and respond to the HomeComplete event. This event driven approach allows a client application to service other tasks while the motors are homing.

Example

```
Private Sub cmdHomeMotors_Click()

' Home both motors

MG17Motor1.MoveHome CHANBOTH_ID, True

End Sub

Private Sub cmdMoveAbsolute_Click()

' Move absolute to user-specified position

MG17Motor1.SetAbsMovePos CHAN1_ID, CSng(Val(txtAbsPos))

MG17Motor1.MoveAbsolute CHAN1_ID, True
```

```
End Sub
```

Motor Method *MoveJog*
See Also Example Code

Visual Basic Syntax
Function **MoveJog**(lChanID As Long, lJogDir As Long,) As Long

Parameters
lChanID - the channel identifier
lJogDir - the direction in which the motor is jogged

Returns
MG Return Code

Details
This method is used to initiate a jog move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. The move profile (step size) is specified by the SetJogStepSize method.

Note. The move can also be initiated from the associated motor GUI panel, or the jog buttons on the hand held control unit which connects to the 'Motor Control' port on the back of the hardware unit.
The *lJogDir* parameter specifies the direction of the move, either 'Forward' or 'Reverse' and takes values from the MOTORJOGDIRECTION enumeration **Motor Method *MoveRelative***

See Also Example Code

Visual Basic Syntax

Function **MoveRelative**(lChanID As Long, bWait As Boolean) As Long

Parameters

lChanID - the channel identifier

bWait - specifies the way in which the MoveRelative method returns

Returns

MG Return Code

Details

This method initiates a motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The distance to move, relative to the present position, is specified by the SetRelMoveDist method.

If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveRelative returns only after the motors have completed their moves. In either mode, a MoveComplete event is fired once the motor moves have been completed.

When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the MoveComplete event. This event driven approach allows a client application to service other tasks while the motors are moving.

Note that in order to specify completely arelative move, the *SetRelMoveDist* method must be called prior to calling the *MoveRelative* method.

Note: The MoveRelativeEx method allows the relative position values to be specified in the parameter list.

To get the relative move distance for a particular channel, see the GetRelMoveDist method.

Example

```
Private Sub cmdMoveRelative_Click()

' Move relative by user-specified amount
```

```
MG17Motor1.SetRelMoveDist CHAN2_ID, CSng(Val(txtRelMove))

MG17Motor1.MoveRelative CHAN2_ID, True

End Sub
```

Motor Method *MoveRelativeEnc*

See Also Example Code

Visual Basic Syntax

Function **MoveRelativeEnc**(lChanID As Long, fRelDistCh1 As Float, fRelDistCh2 As Float, lTimeInc As Long, bWait As Variant_Bool) As Long

Parameters

lChanID - the channel identifier

fRelDistCh1 - the relative distance associated with channel 1

fRelDistCh2 - the relative distance associated with channel 2

lTimeInc – specifies the time out delay

bWait - specifies the way in which the MoveRelativeEnc method returns

Returns

MG Return Code

Details

This method initiates a relative motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Encoder position correction is supported if 'Position (& Correct)' or 'Position (Stop Short & Correct)' is selected. (see the SetEncPosControlParams method or the Encoder Operation section).

The distance to move, relative to the present position, is specified by the f*RelDistCh1* and f*RelDistCh2* parameters. **Note**. If the *lChanID* parameter is set to a single channel then the unspecified f*RelDist* parameter is ignored.

If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveRelativeEnc returns only after the motors have completed their moves. In either mode, a MoveComplete event is fired once the motor moves have been completed.

When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the *MoveComplete* event. This event driven approach allows a client application to service other tasks while the motors are moving.

When a position correction mode is enabled, it is possible in some circumstances, for final position correction to take a considerable time (e.g. mechanically noisy systems or excessive stop short distance parameters used). The *lTimeInc* parameter specifies an extra timeout delay (in milliseconds) that the system adds to the internally calculated move time out normally applied to a non-encoded move.

Note. The timeout is only applied if the *bWait* parameter is set to TRUE, causing the software call to wait until the move has been completed.

If it is found that encoded moves are timing out before the move completes, then the value of the *lTimeInc* parameter should be increased (empirically) until the timeouts are eliminated.

Note: If an encoder calibration table has been acquired (see SetEncCalibTableParams) and enabled (see SetEncPosControlParams), then all relative motor moves (MoveRelative, MoveRelativeEx and MoveRelativeEnc) will operate using the encoder calibration table. However, the MoveRelativeEnc method must be used if position correction functionality is to be invoked at the end of an encoded move.

Motor Method *MoveRelativeEx*

See Also Example Code

Visual Basic Syntax

Function **MoveRelativeEx**(lChanID As Long, dRelDistCh1 As Double, dRelDistCh2 As Double, bWait As Boolean) As Long

Parameters

lChanID - the channel identifier

dRelDistCh1 - the relative distance associated with channel 1

dRelDistCh2 - the relative distance associated with channel 2

bWait - specifies the way in which the MoveRelativeEx method returns

Returns

MG Return Code

Details

This method initiates a motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The distance to move, relative to the present position, is specified by the *dRelDistCh1* and *dRelDistCh2* parameters. **Note**. If the *lChanID* parameter is set to a single channel then the unspecified *dRelDist* parameter is ignored.

If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveRelativeEx returns only after the motors have completed their moves. In either mode, a MoveComplete event is fired once the motor moves have been completed.

When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the *MoveComplete* event. This event driven approach allows a client application to service other tasks while the motors are moving.

Note: See the MoveRelative method for an alternative way of moving relative distances.

Motor Method *MoveVelocity*

See Also Example Code

Visual Basic Syntax

Function **MoveVelocity**(lChanID As Long, lDirection As Long) As Long

Parameters

lChanID - the channel identifier

lDirection - the direction sense to move

Returns

MG Return Code

Details

This method initiates a motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The direction to move is specified by the *lDirection* parameter, which in turn takes values from the MOVEVELDIR enumeration.

When this method is called, the motor will move continuously in the specified direction, using the velocity parameters set in the SetVelParams method, until either a stop method (either StopImmediate or StopProfiled) is called, or a limit switch is reached.

T-Cube Solenoid Controller Method *SC_Disable*

See Also Example Code

Visual Basic Syntax

Function **SC_Disable**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details.

This method de-activates the output to the solenoid operated device.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The output may be activated by calling the SC_Enable method.

T-Cube Solenoid Controller Method *SC_Enable*

See Also Example Code

Visual Basic Syntax

Function **SC_Enable**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details.

This method activates the output to the solenoid-operated device.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The output may be de-activated by calling the SC_Disable method.

T-Cube Solenoid Controller Method *SC_GetCycleParams*

See Also Example Code

Visual Basic Syntax

Function **SC_GetCycleParams**(lChanID As Long, pfOnTime As Float, pfOffTime As Float, plNumCycles As Long) As Long

Parameters

lChanID - the channel identifier

pfOnTime – The time that the solenoid is activated

pfOffTime – The time that the solenoid is de-activated

plNumCycles – The number of Open/Close cycles to perform

Returns

MG Return Code

Details.

This method returns the cycle parameters that are applicable when the solenoid controller is operating in one of the non-manual modes – see the SetOperatingMode method for more details.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The time which the solenoid is activated and de-activated (100ms to 100s) is returned in the *fOnTime* and *fOffTime* parameters.

If the unit is operating in 'Auto' mode, the number of Open/Close cycles to perform. (0 to 1,000,000) is returned in the *fNumCycles* parameter. If set to '0' the unit cycles indefinitely.

T-Cube Solenoid Controller Method SC_GetOperatingMode

See Also Example Code

Visual Basic Syntax

Function **SC_GetOperatingMode** (lChanID As Long, plMode As Long) As Long

Parameters

lChanID - the channel identifier

plMode – the operating mode

Returns

MG Return Code

Details.

This method returns the operating mode of the unit.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The operating mode is returned in the plMode parameter, which in turn, accepts values from the SOLOPERATINGMODE enumeration as follows:.

1. *SOLENOID_MANUAL* - In this mode, operation of the solenoid is via the front panel 'Enable' button, or by the 'Output' buttons on the GUI panel.
2. *SOLENOID_SINGLE* - In this mode, the solenoid will open and close each time the front panel 'Enable' button is pressed, or the 'Output ON' button on the GUI panel is clicked. The ON and OFF times are specified by calling the SC_SetCycleParams method.
3. *SOLENOID_AUTO* - In this mode, the solenoid will open and close continuously after the front panel 'Enable' button is pressed, or the 'Output ON' button on the GUI panel is clicked. The ON and OFF times, and the number of cycles performed, are specified by calling the SC_SetCycleParams method.
4. *SOLENOID_TRIGGER* - In Triggered mode, a rising edge on rear panel TRIG IN BNC input will start execution of the parameters programmed on the unit (On Time, Off Time, Num Cycles - see SC_SetCycleParams method). The unit must be primed (i.e. the ENABLE button pressed and the ENABLED LED lit) before the unit can respond to the external trigger.

The operating mode can be set by calling the SC_SetOperatingMode method.

T-Cube Solenoid Controller Method *SC_GetOPState*

See Also Example Code

Visual Basic Syntax

Function **SC_GetOPState**(lChanID As Long, plState As Long) As Long

Parameters

lChanID - the channel identifier

plState – the output state of the controller

Returns

MG Return Code

Details.

This method returns the output (ON or OFF) of the solenoid controller.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The output state is returned as a value in the *plState* parameter, which in turn, takes values from the SOLOUTPUTSTATE enumeration.

1. OUTPUTSTATE_ON Controller Output ON
2. OUTPUTSTATE_OFF Controller Output OFF

T-Cube Solenoid Controller Method *SC_SetCycleParams*

See Also Example Code

Visual Basic Syntax

Function **SC_SetCycleParams**(lChanID As Long, fOnTime As Float, fOffTime As Float, lNumCycles As Long) As Long

Parameters

lChanID - the channel identifier

fOnTime – The time that the solenoid is activated

fOffTime – The time that the solenoid is de-activated

lNumCycles – The number of Open/Close cycles to perform

Returns

MG Return Code

Details.

This method sets the cycle parameters that are applicable when the solenoid controller is operating in one of the non-manual modes – see the SetOperatingMode method for more details.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The time which the solenoid is activated and de-activated (100ms to 100s) is specified in the *fOnTime* and *fOffTime* parameters.

If the unit is operating in ‘Auto’ mode, the number of Open/Close cycles to perform. (0 to 1,000,000) is specified in the *fNumCycles* parameter. If set to ‘0’ the unit cycles indefinitely. If the unit is not operating in ‘Auto’ mode, the *fNumCycles* parameter is ignored.

T-Cube Solenoid Controller Method *SC_SetOperatingMode*

See Also Example Code

Visual Basic Syntax

Function **SC_SetOperatingMode** (lChanID As Long, lMode As Long) As Long

Parameters

lChanID - the channel identifier

lMode – the operating mode

Returns

MG Return Code

Details.

This method sets the operating mode of the unit.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The operating mode is specified in the lMode parameter, which in turn, accepts values from the SOLOPERATINGMODE enumeration as follows:.

1. *SOLENOID_MANUAL* - In this mode, operation of the solenoid is via the front panel 'Enable' button, or by the 'Output' buttons on the GUI panel.
2. *SOLENOID_SINGLE* - In this mode, the solenoid will open and close each time the front panel 'Enable' button is pressed, or the 'Output ON' button on the GUI panel is clicked. The ON and OFF times are specified by calling the SC_SetCycleParams method.
3. *SOLENOID_AUTO* - In this mode, the solenoid will open and close continuously after the front panel 'Enable' button is pressed, or the 'Output ON' button on the GUI panel is clicked. The ON and OFF times, and the number of cycles performed, are specified by calling the SC_SetCycleParams method.
4. *SOLENOID_TRIGGER* - In Triggered mode, a rising edge on rear panel TRIG IN BNC input will start execution of the parameters programmed on the unit (On Time, Off Time, Num Cycles - see SC_SetCycleParams method). The unit must be primed (i.e. the ENABLE button pressed and the ENABLED LED lit) before the unit can respond to the external trigger.

The current setting for the operating mode can be obtained by calling the SC_GetOperatingMode method.

Method *SaveParamSet*

See Also Example Code

Visual Basic Syntax

Function **SaveParamSet**(bstrName As String) As Long

Parameters

bstrName – the name under which the parameter set is to be saved

Returns

MG Return Code

Details

This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved.

Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

Motor Method *SetAbsMovePos*

See Also Example Code

Visual Basic Syntax

Function **SetAbsMovePos**(lChanID As Long, fAbsPos As Single) As Long

Parameters

lChanID - the channel identifier

fAbsPos - the absolute position associated with the specified channel

Returns

MG Return Code

Details

This method sets the absolute position to which the channel specified by the *lChanID* parameter will move, the next time the *MoveAbsolute* method is called. The position is set to the value specified in the *fAbsPos* parameter and is measured from the home position, in real world units (mm or degrees).

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the absolute position of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To get the absolute position for a particular channel, see the GetAbsMovePos method.

Motor Method *SetBLashDist*

See Also Example Code

Visual Basic Syntax

Function **SetBLashDist**(lChanID As Long, fBLashDist As Single) As Long

Parameters

lChanID - the channel identifier

fBLashDist - the distance by which the motor will overshoot the demanded position, in order to overcome backlash.

Returns

MG Return Code

Details

This method allows the backlash compensation distance to be set for use during motor moves (absolute and relative moves) by the channel specified by the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The system compensates for lead screw backlash on reverse direction moves, by moving passed the demanded position by the amount (in millimeters or degrees) set in the *fBLashDist* parameter, and then reversing, which ensures that positions are always approached from a forward direction.

If the *fBLashDist* parameter is set to zero, backlash compensation is disabled.

Typically, a client application would modify this parameter depending on the length of travel and the velocities set for a particular

move.

Note: On dual channel units, the system cannot set the backlash distance of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To obtain the backlash distances for a particular channel, see the GetBLashDist method.

Motor Method SetBowIndex

See Also Example Code

Visual Basic Syntax

Function **SetBowIndex**(lChanID As Long, lBowIndex As Long) As Long

Parameters

lChanID - the channel identifier

lBowIndex – the move profile to be used

Returns

MG Return Code

Details.

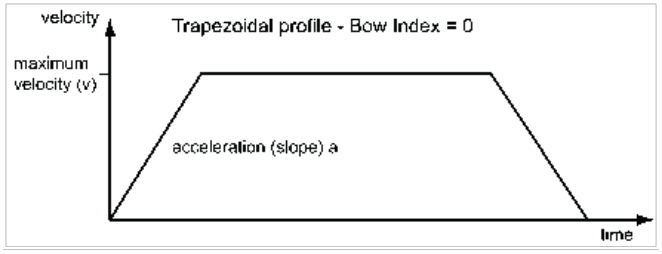
The specific move profile created by the system trajectory generator depends the profile mode and profile parameters presently selected. This method is used to set the profile mode to that specified by the *lBowIndex* parameter, which accepts values as follows:

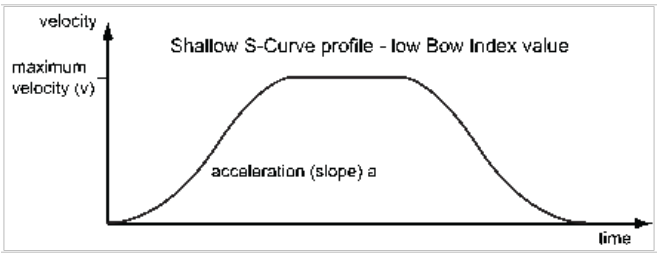
Trapezoidal Point to Point Profile	0
S-curve Point to Point Profile	1 to 18

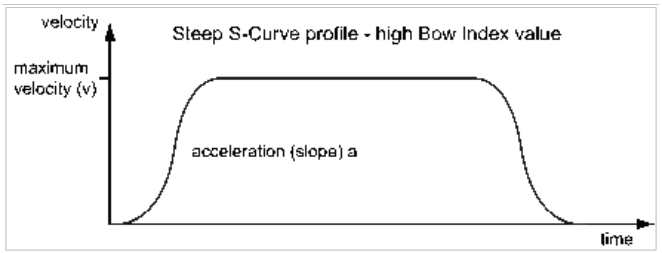
In either case, the velocity, acceleration and end position of the profile are specified using the SetVelParamsSetVelParams_Mot and the MoveRelativeMoveRelative_Mot or MoveAbsoluteMoveAbsolute_Mot methods.

The *Trapezoidal* profile is a standard, symmetrical acceleration/deceleration motion curve, in which the start velocity is always zero. To select a trapezoidal profile, set the BowIndex parameter to '0'.

The *S-curve* profile is a trapezoidal curve with an additional '*BowIndex'* value (1 to 18), which limits the rate of change of acceleration and smooths out the contours of the motion profile. The higher the Bow Index value, then the steeper the acceleration and deceleration phases as shown below. **Note**. High values of Bow Index may cause a move to overshoot:







To obtain the present settings for the Bow Index, see the GetBowIndexGetBowIndex_Mot method.

Motor Method *SetBrakeState*

See Also Example Code

Note. This method is applicable only to the BDC series of benchtop DC Servo Controllers

Visual Basic Syntax

Function **SetBrakeState** (lChanID As Long, lState As Long) As Long

Parameters

lChanID - the channel identifier

lState – the state of the brake (on or off)

Returns

MG Return Code

Details

The 'CONTROL IO' connector on the rear panel exposes a TTL logic output that can be used to control a stage equipped with an electronic logic controlled brake mechanism. This method is used to turn the brake ON or OFF.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL

enumeration. The BDC series features a separate control instance for each channel and therefore the *lChanID* parameter should always be set to *Chan1_ID*.

The brake state is set in the *lState* parameter and takes values from the BRAKE_STATE enumeration as follows:

1. BRAKE_ON
2. BRAKE_OFF

Note. Careful use of the brake/enable servo must be considered when using this function. It is important to note that until the motor is disabled, the controller will continue to servo the motor to its current position and therefore possible brake/servo error conflict can occur if the motor shaft is moved when applying the brake. Furthermore, if the motor is mounted in a vertical position with the brake applied and the motor disabled, the motor will fall under gravity to its lower end stop when the brake is removed unless the motor is enables prior to removing the brake.

The current setting for the brake state can be obtained by calling the GetBrakeState method.

Motor Method *SetButtonParams*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube Driver products ODC001, OST001, TDC001 and TST001

Visual Basic Syntax

Function **SetButtonParams**((lChanID As Long, lButMode As Long, fLeftButPos As Single, fRightButPos As Single) As Long

Parameters

lChanID – the channel identifier

lButMode – the operating mode of the front panel buttons

fLeftButPos – the absolute position associated with the left hand button

fRightButPos – the absolute position associated with the right hand button

Returns

MG Return Code

Details

This method specifies the operation mode of the front panel buttons on the motor controller unit.

Note. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the OptoSTDriver and OptoDCDriver, *lChanID* should always be set to 'CHAN1_ID'.

The buttons on the front of the unit can be used either to jog the motor, or to perform moves to absolute positions.

The mode of operation is specified in the *lButMode* parameter which in turn takes values from the BUTTONMODE enumeration as follows:

BUTTON_MODEJOG: In this mode, the front panel buttons are used to jog the motor. Once set to this mode, the move parameters for the buttons are taken from the 'Jog' parameters set via the 'Move/Jogs' settings tab or the SetJogStepSize and SetJogVelParams methods.

BUTTON_MODEPOSITION: In this mode, each button can be programmed with a different position value, such that the controller will move the motor to that position when the specific button is pressed.

Note. The following parameters are applicable only if 'Go to Position is selected in the 'Button Mode' field.

fLeftButPos: The position to which the motor will move when the left hand button is pressed.

fRightButPos: The position to which the motor will move when the right hand button is pressed.

Note. A 'Home' move can be performed by pressing and holding both buttons for 2 seconds. This function is irrespective of the

'Button Mode' setting.

Motor Method *SetChannelSwitch*

See Also Example Code

Note. This method is applicable only to BSC002 units.

Visual Basic Syntax

Function **SetChannelSwitch**(lChanID As Long) As Long

Parameters

lChanID – The channel identifier

Returns

MG Return Code

Details

Some methods (such as the ShowSettingsDlg method) have no channel selection (ChanID) parameter, and apply to whichever channel is currently selected. This method is for use with older 2-channel units (BSC002) and allows the channel to be selected, in the same way as clicking the channel selector switch on the GUI panel.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Brushless DC Motor Method *SetDCCurrentLoopParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **SetDCCurrentLoopParams**(lChanID As Long, lProp As Long, lInt As Long, lIntLim As Long, lIntDeadBand As Long, lFFwd As Long) As Long

Parameters

lChanID - the channel identifier

lProp - the value of the proportional constant

lInt - the value of the integration constant

lIntLim – the max value for the integration constant

lIntDeadBand – the value of the Integral Dead Band

lFFwd – the Integral Feed Forward value

Returns

MG Return Code

Details.

The motion processors within the BBD series controllers use digital current control as a technique to control the current through each phase winding of the motors. In this way, response times are improved and motor efficiency is increased. This is achieved by comparing the required (demanded) current with the actual current to create a current error, which is then passed through a digital PI-type filter. The filtered current value is used to develop an output voltage for each motor coil.

This method sets the proportional and integration feedback loop constants for the current feedback loop to the values specified

in the *IProp* and *IInt* parameters respectively. These parameters determine the system response characteristics and accept values in the range 0 to 32767.

The *IIntLim* parameter is used to cap the output value of the integrator and prevent runaway of the integral sum at the output. It accepts values in the range 0 to 32767.

Note. The following two parameters assist in fine tuning the motor drive current and help reduce audible noise and/or oscillation when the stage is in motion. A certain amount of trial and error may be experienced in order to obtain the optimum settings.

The *IIntDeadBand* parameter allows an integral dead band to be set, such that when the error is within this dead band, the integral action stops, and the move is completed using the proportional term only. It accepts values in the range 0 to 32767.

The *IFFwd* parameter is a feed-forward term that is added to the output of the PI filter. It accepts values in the range 0 to 32767.

The IChanID parameter is not valid for the BBD series brushless DC motor controller and is preset at the factory to '1'.

To obtain the present settings for the Current Loop PI constants, see the GetDCCurrentLoopParams method.

Brushless DC Motor Method *SetDCJoystickParams*

See Also Example Code

Visual Basic Syntax

Function **SetDCJoystickParams**(lChanID As Long, fMaxVelLO As Float, fMaxVelHI As Float, fAccnLO As Float, fAccnHI As Float, lDirSense As Long) As Long

Parameters

lChanID - the channel identifier

fMaxVelLO – the max velocity of a move when low gear mode is selected

fMaxVelHI – the max velocity of a move when high gear mode is selected

fAccnlLO – the acceleration of a move when low gear mode is selected

fAccnHI – the acceleration of a move when high gear mode is selected

lDirSense – the direction sense of the move

Returns

MG Return Code

Details.

The MJC001 joystick console has been designed for use by microscopists to provide intuitive, tactile, manual positioning of the stage. The console consists of a two axis joystick for XY control which features both low and high gear modes. This method is used to set max velocity and acceleration values for these modes.

The *fMaxVelLO* and *fAccnlLO* parameters are used to specify the max velocity and acceleration of a move when low gear mode is selected.

Similarly, the *fMaxVelHI* and *fAccnlHI* parameters are used to specify the max velocity and acceleration of a move when high gear mode is selected.

The actual direction sense of any joystick initiated move is dependent upon the application. The *lDirSense* parameter can be used to reverse the sense of direction for a particular application and is useful when matching joystick direction sense to actual stage direction sense.

Brushless DC Motor Method *SetDCMotorOutputParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech

Support for more information.

Visual Basic Syntax

Function **SetDCMotorOutputParams**(lChanID As Long, fContCurrLim As Float, fEnergyLim As Float, fMotorLim As Float, fMotorBias As Float) As Long

Parameters

lChanID - the channel identifier

fContCurrLim - the continuous current limit, as a percentage of max current.

fEnergyLim - the accumulated energy limit, as a percentage of the default maximum.

fMotorLim – the limit for the motor drive signal

fMotorBias – the value of the motor bias

Returns

MG Return Code

Details.

This method contains parameters which define the limits that can be applied to the motor drive signal.

The system incorporates a current 'foldback' facility, whereby the continuous current level can be capped. The continuous current limit is set in the *fContCurrLim* parameter, which accepts values as a percentage of maximum peak current, in the range 0% to 100%, which is the default maximum level set at the factory (this maximum value cannot be altered).

When the current output of the drive exceeds the limit set in the *fContCurrLim* parameter, accumulation of the excess current energy begins. The *fEnergyLim* parameter specifies a limit for this accumulated energy, as a percentage of the factory set default maximum, in the range 0% to 100%. When the accumulated energy exceeds the value specified in the *fEnergyLim* parameter, a 'current foldback' condition is said to exist, and the commanded current is limited to the value specified in the *fContCurrLim* parameter. When this occurs, the Current Foldback status bit (bit 25) is set in the Status Register. When the accumulated energy above the *fContCurrLim* value falls to 0, the limit is removed and the status bit is cleared.

The *fMotorLim* parameter sets a limit for the motor drive signal and accepts values in the range 0 to 32767 (100%). If the system produces a value greater than the limit set, the motor command takes the limiting value. For example, if *fMotorLim* is set to 30000 (91.6%), then signals greater than 30000 will be output as 30000 and values less than -30000 will be output as -30000.

When an axis is subject to a constant external force in one direction (such as a vertical axis pulled downwards by gravity) the servo filter can compensate by adding a constant DC bias to the output. This bias is set in the *fMotorBias* parameter, which accepts values in the range -32767 to 32768. The default value is 0. Once set, the motor bias is applied while the position loop is enabled.

To obtain the present settings for the motor parameters and limits, see the GetDCMotorOutputParams method.

Brushless DC Motor Method *SetDCPositionLoopParams*

See Also Example Code
Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax
Function **SetDCPositionLoopParams**(lChanID As Long, lProp As Long, lInt As Long, lIntLim As Long, lDeriv As Long, lDerivTime As Long, lLoopGain As Long, lVelFFwd As Long, lAccFFwd As Long, lPosErrLim As Long) As Long
Parameters
lChanID - the channel identifier
lProp - the value of the proportional constant
lInt - the value of the integration constant
lIntLim – the max value for the integration sum
lDeriv – the value of the derivative constant
lDerivTime – the sampling rate for calculating the derivative term, in cycles
lLoopGain – a scaling factor applied to the output of the PID loop

lVelFFwd – A velocity term, added to the output of the PID filter to assist tuning
lAccFFwd – An acceleration term, added to the output of the PID filter to assist tuning
lPosErrLim – the value of the position error limit

Returns
MG Return Code

Details
The motion processors within the BBD series controllers use a position control loop to determine the motor command output. The purpose of the position loop is to match the actual motor position and the demanded position. This is achieved by comparing the demanded position with the actual encoder position to create a position error, which is then passed through a digital PID-type filter. The filtered value is the motor command output.
This method sets the operating parameters of the PID position loop to the values specified in the *lProp, lInt* and *lDeriv* parameters respectively. These parameters determine the system response characteristics and accept values in the range 0 to 32767.
The *lIntLim* parameter is used to cap the value of the *Integrator* to prevent runaway of the integral sum at the output. It accepts values in the range 0 to 32767. If set to 0 then the integration term in the PID loop is ignored.
Under normal circumstances, the derivative term of the PID loop is recalculated at every servo cycle. However, it may be desirable to reduce the sampling rate to a lower value, in order to increase stability or simplify tuning. The *lDerivTime* parameter is used to set the sampling rate. For example, if set to 10, the derivative term is calculated every 10 servo cycles. The value is set in cycles, in the range 1 to 32767.
The *lLoopGain* parameter is a scaling factor applied to the output of the PID loop. It accepts values in the range 0 to 65535, where 0 is 0% and 65535 is 100%.
The *lVelFFwd* and *lAccFFwd* parameters are velocity and acceleration feed-forward terms that are added to the output of the PID filter to assist in tuning the motor drive signal. They accept values in the range 0 to 32767.
Under certain circumstances, the actual encoder position may differ from the demanded position by an excessive amount. Such a large position error is often indicative of a potentially dangerous condition such as motor failure, encoder failure or excessive mechanical friction. To warn of, and guard against this condition, a maximum position error can be set in the *lPosErrLim* parameter, in the range 0 to 65535. The actual position error is continuously compared against the limit entered, and if exceeded, the Motion Error bit (bit 15) of the Status Register is set and the associated axis is stopped.
The lChanID parameter is not valid for the BBD series brushless DC motor controller and is preset at the factory to '1'.
To obtain the current settings for the Position Loop PID constants, see the GetDCPositionLoopParams method. **Brushless DC Motor Method*SetDCProfileModeParams***

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **SetDCProfileModeParams**(lChanID As Long, lProfMode As Long, fJerk As Float) As Long

Parameters

lChanID - the channel identifier

lProfMode – the move profile to be used

fJerk – the jerk value to be used if S-curve mode is selected

Returns

MG Return Code

Details.

The system incorporates a trajectory generator, which performs calculations to determine the instantaneous position, velocity and acceleration of each axis at any given moment. During a motion profile, these values will change continuously. Once the move is complete, these parameters will then remain unchanged until the next move begins.

The specific move profile created by the system depends on several factors, such as the profile mode and profile parameters presently selected, and other conditions such as whether a motion stop has been requested. This method is used to set the profile mode to that specified by the *lProfMode* parameter, which accepts values as follows:

Trapezoidal Point to Point Profile 0

S-curve Point to Point Profile 2

In either case, the velocity, acceleration and end position of the profile are specified using the SetVelParams and the MoveRelative or MoveAbsolute methods.

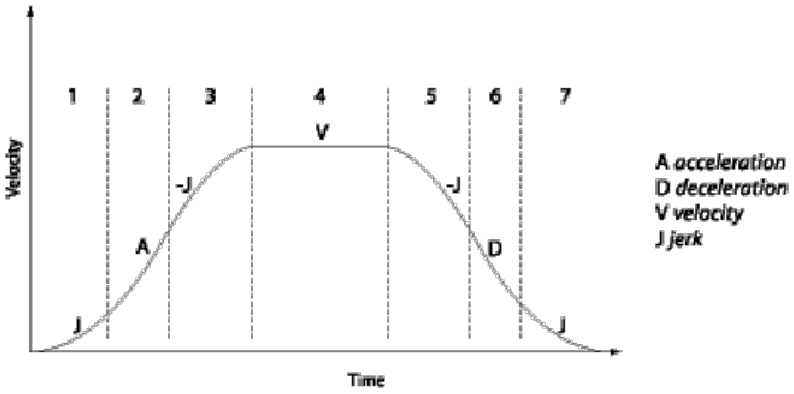
The *Trapezoidal* profile is a standard, symmetrical acceleration/deceleration motion curve, in which the start velocity is always zero.

The *S-curve* profile is a trapezoidal curve with an additional '*Jerk'* parameter, which limits the rate of change of acceleration and smooths out the contours of the motion profile.

The Jerk value is specified in mm/s3 in the *fJerk* parameter, and accepts values in the range 0 to 46566139. It is used to specify the maximum rate of change in acceleration in a single cycle of the basic trapezoidal curve.

In this profile mode, the acceleration increases gradually from 0 to the specified acceleration value, then decreases at the same rate until it reaches 0 again at the specified velocity. The same sequence in reverse brings the axis to a stop at the programmed destination position.

Example



The figure above shows a typical S-curve profile. In segment (1), the S-curve profile drives the axis at the specified jerk (J) until the maximum acceleration (A) is reached. The axis continues to accelerate linearly (jerk = 0) through segment (2). The profile then applies the negative value of jerk to reduce the acceleration to 0 during segment (3). The axis is now at the maximum velocity (V), at which it continues through segment (4). The profile then decelerates in a similar manner to the acceleration phase, using the jerk value to reach the maximum deceleration (D) and then bring the axis to a stop at the destination.

To obtain the present settings for the DC Profile Mode parameters, see the GetDCProfileModeParams method.

Brushless DC Motor Method *SetDCSettledCurrentLoopParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **SetDCSettledCurrentLoopParams**(lChanID As Long, lSettledProp As Long, lSettledInt As Long, lSettledIntLim As Long, lSettledIntDeadBand As Long, lSettledFFwd As Long) As Long

Parameters

lChanID - the channel identifier

lSettledProp - the value of the proportional constant

lSettledInt - the value of the integration constant

lSettledIntLim – the max value for the integration constant

lSettledIntDeadBand – the value of the Integral Dead Band

lSettledFFwd – the Integral Feed Forward value

Returns

MG Return Code

Details.

This method assists in maintaining stable operation and reducing noise at the demanded position. It allows the system to be tuned such that errors caused by external vibration and manual handling (e.g. loading of samples) are minimized, and is applicable only when the stage is settled, i.e. the Axis Settled status bit (bit 14) is set.

It sets the proportional and integration feedback loop constants for the current feedback loop to the values specified in the *lSettledProp* and *lSettledInt* parameters respectively. These parameters determine the system response characteristics and accept values in the range 0 to 32767.

The *lSettledIntLim* parameter is used to cap the output value of the integrator and prevent runaway of the integral sum at the output. It accepts values in the range 0 to 32767.

Note. The following two parameters assist in fine tuning the motor drive current and help reduce audible noise and/or oscillation when the stage is near the target position. A certain amount of trial and error may be experienced in order to obtain the optimum settings.

The *lSettledIntDeadBand* parameter allows an integral dead band to be set, such that when the error is within this dead band, the integral action stops, and the move is completed using the proportional term only. It accepts values in the range 0 to 32767.

The *SettledlFFwd* parameter is a feed-forward term that is added to the output of the PI filter. It accepts values in the range 0 to 32767.

The lChanID parameter is not valid for the BBD series brushless DC motor controller and is preset at the factory to '1'.

To obtain the present settings for the Current Loop PI constants, see the GetDCSettledCurrentLoopParams method.

Brushless DC Motor Method*SetDCTrackSettleParams*

See Also Example Code

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

Visual Basic Syntax

Function **SetDCTrackSettleParams**(lChanID As Long, lSettleTime As Long, lSettleWnd As Long, lTrackWnd As Long) As Long

Parameters

lChanID - the channel identifier

lSettleTime - the No of cycles that the axis must be settled before the 'Settled' status bit is set

lSettleWnd - the No of encoder counts that the position error must be less than or equal to, before the axis is considered 'settled'

lTrackWnd - the maximum allowable position error

Returns

MG Return Code

Details.

Moves are generated by an internal profile generator, and are based on either a trapezoidal or s-curve trajectory. A move is considered complete when the profile generator has completed the calculated move and the axis has 'settled' at the demanded position.

The system incorporates a monitoring function, which continuously indicates whether or not the axis has 'settled'. The 'Settled'

indicator is bit 14 in the Status Register and is set when the associated axis is settled. Note that the status bit is controlled by the processor, and cannot be set or cleared manually.

The axis is considered to be ‘settled’ when the following conditions are met:

the axis is at rest (i.e. not performing a move),

the error between the demanded position and the actual motor position is less than or equal to a specified number of encoder counts (0 to 65535) set in the *lSettleWnd* parameter (Settle Window),

the above two conditions have been met for a specified number of cycles (settle time, 1 cycle = 102.4 µs), set in the *lSettleTime* parameter (range 0 to 32767).

The above settings are particularly important when performing a sequence of moves. If the PID parameters are set such that the settle window cannot be reached, the first move in the sequence will never complete, and the sequence will stall. The settle window and settle time values should be specified carefully, based on the required positional accuracy of the application. If positional accuracy is not a major concern, the settle time should be set to '0'. In this case, a move will complete when the motion calculated by the profile generator is completed, irrespective of the actual position attained, and the settle parameters described above will be ignored.

The processor also provides a ‘tracking window’, which is used to monitor servo performance outside the context of motion error. The tracking window is a programmable position error limit within which the axis must remain, but unlike the position error limit set in the SetDCPositionLoopParams method, the axis is not stopped if it moves outside the specified tracking window. This function is useful for processes that rely on the motor’s correct tracking of a set trajectory within a specific range. The tracking window may also be used as an early warning for performance problems that do not yet qualify as motion error.

The size of the tracking window (i.e. the maximum allowable position error while remaining within the tracking window) is specified in the *lTrackWnd* parameter, in the range 0 to 65535. If the position error of the axis exceeds this value, the Tracking Indicator status bit (bit 13) is set to 0 in the Status Register. When the position error returns to within the window boundary, the status bit is set to 1.

To obtain the present settings for the Track and Settle parameters, see the GetDCTrackSettleParams method.

Brushless DC Motor Method SetDCTriggerParams

See Also Example Code

Note. This method is applicable only to brushless DC driver units BBDxxx and TBD001.

Visual Basic Syntax

Function **SetDCTriggerParams**(lChanID As Long, lTrigInMode As Long, lTrigOutMode As Long) As Long

Parameters

lChanID - the channel identifier

lTrigInMode – sets the input trigger mode for the specified channel

lTrigOutMode – sets the output trigger mode for the specified channel

Returns

MG Return Code

Details

This method is used to configure the brushless DC motor controller for triggered move operation. It is possible to configure a particular controller to respond to trigger inputs, generate trigger outputs or both respond to and generate a trigger output. When a trigger input is received, the unit can be set to initiate a move (relative, absolute or home). Similarly the unit can be set to generate a trigger output signal when a specified event (e.g move initiated) occurs. For those units configured for both input and output triggering, a move can be initiated via a trigger input while at the same time, a trigger output can be generated to initiate a move on another unit.

The trigger settings can be used to configure multiple units in a master – slave set up, thereby allowing multiple channels of motion to be synchronized. Multiple moves can then be initiated via a single software or hardware trigger command.

The channel to be set is specified by the *IChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration. For single channel units such as the TDB001, this parameter is locked at '0' (Channel 1) and cannot be changed.

The Trigger In input can be configured to initiate a relative, absolute or homing home, either on the rising or falling edge of the signal driving it. As the trigger input is edge sensitive, it needs to see a logic LOW to HIGH transition ("rising edge") or a logic HIGH to LOW transition ("falling edge") for the move to be started. Additionally, the move parameters must be downloaded to the unit prior to the move using the relevant relative move or absolute move methods as described below. A move already in progress will not be interrupted; therefore external triggering will not work until the previous move has been completed.

The *ITrigInMode* parameter sets the input trigger mode for the specified channel. The various trigger modes are specified by the DCTRIGINMODE enumeration as follows:

DCTRIGIN_DISABLED – triggering operation is disabled

DCTRIGINRISE_RELMOVE – a relative move (specified using the latest *MoveRelative* or MoveRelativeEx settings) is initiated on the specified channel when a rising edge input signal is received on the TRIG IN connector.

DCTRIGINFALL_RELMOVE – as above, but the relative move is initiated on receipt of a falling edge signal.

DCTRIGINRISE_ABSMOVE – an absolute move (specified using the latest MoveAbsolute or MoveAbsoluteEx settings) is initiated on the specified channel when a rising edge input signal is received on the TRIG IN connector.

DCTRIGINFALL_ABSMOVE – as above, but the absolute move is initiated on receipt of a falling edge signal.

DCTRIGINRISE_HOMEMOVE – a home move (specified using the latest *MoveHome* settings) is initiated on the specified channel when a rising edge input signal is received on the TRIG IN connector.

DCTRIGINFALL_HOMEMOVE – as above, but the home move is initiated on receipt of a falling edge signal.

The Trigger Out output can be configured to be asserted to either logic HIGH or LOW as a function of certain motion-related conditions, such as when a move is in progress (In Motion), complete (Move Complete) or reaches the constant velocity phase on its trajectory (Max Vel). The logic state of the output will remain the same for as long as the chosen condition is true. The logic state associated with the condition can be selected to be either LOW or HIGH.

The *ITrigOutMode* parameter sets the output trigger mode for the specified channel. The various trigger modes are specified by the DCTRIGOUTMODE enumeration as follows:

DCTRIGOUT_DISABLED – triggering operation is disabled.

DCTRIGOUTHIGH_INMOTION – The output trigger goes high (5V) when the stage is in motion.

DCTRIGOUTLOW_INMOTION – The output trigger goes low (0V) when the stage is in motion.

DCTRIGOUTHIGH_MOTIONCOMPLETE – The output trigger goes high (5V) when the current move is completed.

DCTRIGOUTLOW_MOTIONCOMPLETE – The output trigger goes low (0V) when the current move is completed.

DCTRIGOUTHIGH_MAXVELOCITY – The output trigger goes high (5V) when the stage reaches max velocity (as set using the SetVelParams method).

DCTRIGOUTLOW_MAXVELOCITY – The output trigger goes low (0V) when the stage reaches max velocity (as set using the SetVelParams method).

In addition, the trigger out signal can be controlled from software by calling the LLSetGetDigOPs method.

The trigger parameters are obtained using the GetDCTriggerParams method.

Please refer to the handbook supplied with your unit for further information on identifying the trigger connections.

Motor Method *SetDispMode*

See Also Example Code

Visual Basic Syntax

Function **SetDispMode**(lDispMode As Long) As Long

Parameters

lDispMode – The display mode selected

Returns

MG Return Code

Details

This method sets the display mode for the associated GUI panel. The *lDispMode* parameter takes values from the DISPLAYMODE enumeration as follows:

1. DISPMODE_PANEL
2. DISPMODE_GRAPH
3. DISPMODE_POSITION

When set to DISPMODE_PANEL, the display shows the standard GUI panel.

If DISPMODE_GRAPH is seleceted, the display is changed to a graphical display, showing the position of the motor channel(s). Moves to absolute positions can then be initiated by positioning the mouse within the display and clicking, see Graphical Control of Motor Positions for more information.

When set to DISPMODE_POSITION, only the 'Position' display is shown.

The current display setting can be returned by calling the GetDispMode method.

Motor Method *SetEncCalibTableParams*

See Also Example Code

Visual Basic Syntax

Function **SetEncCalibTableParams**(lChanID As Long, lEncCalib As Long, fCalibStep As Float, lCalibDwell As Long, lQEPSense As Long) As Long

Parameters

lChanID – the channel identifier

lEncCalib – the Encoder Calibration factor (counts/mm or counts/degree)

fCalibStep – the stepping stepping distance to use when a calibration table is acquired

lCalibDwell – specifies the dwell time (in ms) to wait before reading the encoder count

lQEPSense – specifies the sense of the encoder signals being read by the controller electronics, (positive or negative)

Returns

MG Return Code

Details

When 'Encoder Positioning' is selected, all position displays and motor moves are based on positions derived from the encoder system fitted to the stage/actuator.

Example. The encoders currently fitted to Our stages are set to 10000 counts per mm (i.e. 0.1micron per count). Therefore, to move 1 mm, the controller will drive out the appropriate number of microsteps to result in an encoder count change of 10000. To display position values, the software converts the encoder count into a 'real world' position by dividing by 10000.

For a perfect 1mm pitch lead screw, a microstep count of 25600 would equate exactly to an encoder count of 10000. However, due to lead screw pitch, non-linearity and other cyclic errors, this is not achievable in the real world. It is the purpose of the encoder feedback handling within the APT software to accommodate for this and achieve the required encoder position (a more accurate position reading).

One way to accommodate for this lead screw non-linearity is for the system to acquire a look up table of microstep count verses encoder count readings. Using this 'calibration' table the system is then able to adjust the microstep count required (to drive the motor) to achieve the required encoder count.

When the 'Calibrate' button in the encoder Settings panel is clicked, (or the CalibrateEnc method is called), the system begins the calibration sequence by first homing the channel and then moving through a series of microstep position points taking the encoder reading at each point.

This method sets the positions at which the encoder count ,associated with the channel specified by the *lChanID* parameter, is read. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The *lEncCalib* parameter sets the overall encoder calibration factor in counts/mm or counts/degree. On encoder equipped stages currently available, this factor is factory set to 10000 (i.e. 10000 counts/mm or 0.1micron per count).

The *fCalibStep* parameter specifies the stepping distance to use between positions, e.g. a stepping distance of 0.2mm instructs the system to stop and read the encoder count every 0.2mm.

In many cases it is good practice to allow the stage mechanics to settle for a short period (allowing vibrations and other transients to die down) before sampling the encoder reading. The *lCalibDwell* parameter specifies the dwell time (in ms) to wait at each position point visited, before reading the encoder count.

The *lQEPSense* parameter takes values specified by the ENCQEPSENSE Enumeration, and specifies the sense of the encoder signals being read by the controller electronics, i.e. positive or negative. There is only one setting applicable for a particular stage/encoder system and when set should not be altered.

Current settings for the Calibration Table Parameters can be obtained by calling the GetEncCalibTableParams method.

Note. Before it can be used, the calibration table must be enabled by calling the SetEncPosControlParams method (UseCal parameter) or via the Encoder Control Panel.

Motor Method *SetEncPosControlParams*

See Also Example Code

Visual Basic Syntax

Function **SetEncPosControlParams** (lChanID As Long, lPosSrcMode As Long, lPosCorrMode As Long, bUseCalib As variant_Bool) As Long

Parameters

lChanID – the channel identifier

lPosSrcMode – the source of positional information, microstep count or encoder count.

lPosCorrMode – the position correction mode.

bUseCalib – use calibration table if True

Returns

MG Return Code

Details

This method sets the position source and the correction mode for a move initiated on the channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The positioning mode is set in the *lPosSrcMode* parameter, which in turn takes values from the ENCPOSSOURCEMODE enumeration as follows:

ENC_POSSRC_MICROSTEP - By default the software is set to 'Microstep Positioning' mode. In this mode, all position displays and motor moves are based on positions derived from the internal microstep counters within the controller (i.e. are not based on the encoder count).

For example, the APT system can control to a resolution of 25600 microsteps per revolution. For a stage or actuator with a 1 mm pitch lead screw, the controller will generate 25600 microsteps to move 1 mm. To display position values, the software will

convert the microstep count into a ‘real world’ position by dividing by 25600.

ENC_POSSRC_ENCODER - In this mode the position displays and all motor moves are based on positions derived from the encoder system fitted to the stage/actuator. So for example, the encoders currently fitted to Our stages are set to 10000 counts per mm (i.e. 0.1micron per count). To move 1 mm, the controller will drive out the appropriate number of microsteps to result in an encoder count change of 10000. To display position values, the software will convert the encoder count into a ‘real world’ position by dividing by 10000.

Note that the positions values returned by the GetPosition and GetPositionEx methods are derived from the microstep count in ‘Microstep Positioning’ mode and from the encoder count in ‘Encoder Positioning’ mode.

The position correction mode to be used is set in the *lPosCorrMode* parameter, which in turn, takes values from the ENCPOSCONTROLMODE Enumeration‘ as follows:

ENC_POSCTRL_POSITION – This setting selects ‘Position (& Correct)’ mode, whereby the system first attempts to move to the required encoder based position (with or without calibration table support), and then adjusts the motor position, using a series of very small correction steps, until the required encoder position is achieved.

The various correction stepping parameters that affect this operation, are set using the SetEncPosCorrectParams method or via the ‘Encoder’ tab of the settings tabbed dialog accessed using the ‘Settings’ button on the main graphical panel.

Note that depending on the nature of the lead screw, the stage/actuator may end up either side of the required position. The correction moves will be applied in both forward or reverse directions as required in order to achieve the required encoder position.

ENC_POSCTRL_POSITIONSTOPSHORT - In some applications it is desirable to always ‘reach’ the required final position from the same direction (conventionally using positive moves on Our stages). To support this, the ‘Position (Stop Short & Correct)’ correction mode is available.

When this mode is selected the system stops short of the intended position by a user specified distance and then issues small position correction steps to achieve the required final encoder position. Again the user adjustable parameters associated with this operation are set using the SetEncPosCorrectParams method or via the ‘Encoder’ tab of the settings tabbed dialog accessed using the ‘Settings’ button on the main graphical panel.

ENC_POSCTRL_DISABLED – Position correction is switched off.

Note. Before using an encoded position correction mode, the backlash correction distance should be set to zero – see the SetBLashDist method.

If a calibration table has been acquired using the SetEncCalibTableParams method, it must be enabled by setting the *UseCalib* parameter to True.

Motor Method *SetEncPosCorrectParams*

See Also Example Code

Visual Basic Syntax

Function **SetEncPosCorrectParams** (lChanID As Long, lPosSetPtWnd As Long, lSnguStepWnd As Long, lStopShortDist As Long, lCorrMoveStep As Long) As Long

Parameters

lChanID – the channel identifier

lPosSetPtWnd – the tolerance (in encoder counts) within which the system must achieve the required position setpoint when an encoded move is corrected.

lSnguStepWnd – the window within which the system attempts to reach the required position setpoint by single microstep stepping.

lStopShortDist –the distance (in microsteps) to stop short of an encoded move (when the ‘Position (Stop Short & Correct)’ position correction mode is enabled

lCorrMoveStep –the size of correction step (in microsteps) to be used when the system issues correction moves outside of the distance set in the *lSnguStepWnd* parameter

Returns

MG Return Code

Details

In addition to the calibration table described in the SetEncCalibTableParams method, the APT software can be set to invoke further positioning correction at the end of an encoded move. This can sometimes be required when there has been thermal drift in the mechanics since the time the calibration table was acquired. There are two position correction modes available, 'Position (& Correct)' and 'Position (Stop Short & Correct)' selected using the SetEncPosControlParams method (or by using the 'Position Correct Mode' drop down list in the 'Encoder Control Panel').

This method applies settings to an encoded move initiated on the channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The *lPosSetPtWnd* parameter specifies a window of acceptability or tolerance (in encoder counts) that the system must achieve.

For example, when set to a value of 1, the system must get to within 1 encoder count either side of the required position. This setting is useful for noisy systems (i.e. mechanical noise causing jitter on the encoder count) where achieving a precise encoder count may prove difficult. For an encoder calibration factor of 10000/mm, a value of 1 for this setting means the system will position to within +-0.1micron of the required position.

When 'Position (& Correct)' mode is selected (see the SetEncPosControlParams method), the system first attempts to move to the required encoder based position (with or without calibration table support), and then adjusts the motor position, using a series of very small correction steps, until the required encoder position is achieved. The *lSnguStepWnd* specifies the window (or tolerance), within which the system attempts to reach the required position setpoint by single micro stepping.

For example, if this parameter is set to a value of 10, and if the encoded move initially ends up within 10 counts (1 micron for a calibration factor of 10000) either side of the required position, the system will single microstep to reach the required position

If, at the end of the initial encoded move, the system detects that the encoder position is outside of this window, then the system steps at whatever multiple of microsteps is specified by the *lCorrMoveStep* parameter. When the encoder position falls within the window specified by the *lSnguStepWnd* parameter, single microstepping can take place.

Assuming 25600 microsteps per revolution of the stepper motor and a 1mm pitch lead screw, a value of 5 equates to approx. 0.2 micron step size.

In some applications it is desirable to always 'reach' the required final position from the same direction (conventionally using positive moves on Our stages). When the 'Position (Stop Short & Correct) position correction mode is selected (see the SetEncPosControlParams method), the *lStopShortDist* parameter specifies the distance (in microsteps) to stop short of an encoded move.

Assuming 25600 microsteps per revolution of the stepper motor and a 1mm pitch lead screw, a value of 250 equates to approx. 10 microns stop short distance.

The current settings for these parameters can be obtained by calling the GetEncPosCorrectParams method.

Motor Method *SetHomeParams*

See Also Example Code

Visual Basic Syntax

Function **SetHomeParams**(lChanID As Long, lDirection As Long, lLimSwitch As Long, fHomeVel As Single, fZeroOffset As Single) As Long

Parameters

lChanID - the channel identifier

lDirection - the direction sense to move when homing

lLimSwitch - The limit switch associated with the home position

fHomeVel - the velocity at which the motors should move when Homing

fZeroOffset - the distance (in mm or degrees) of the limit switch from the Home position

Returns

MG Return Code

Details

This method notifies the system to the homing parameters for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

These parameters are specific for the design of the stage and are required to enable the system to accommodate the homing and moving characteristics of a specific stage and axis. For parameter settings on a particular stage, refer to Stage Information

The direction sense for a move to Home (either forward or reverse) is entered in the *lDirection* parameter, which in turn, takes values from the HOMEDIR enumeration,

The limit switch associated with the home position (hardware forward, or hardware reverse) is entered in the *lLimSwitch* parameter, which in turn, takes values from the HOMELIMSWITCH enumeration.

The maximum velocity of a homing move is entered in the *fHomeVel* parameter. The minimum velocity and acceleration are as set in the SetVelParams method.

The distance of the Home position from the Home Limit Switch is entered in the *fZeroOffset* parameter.

They can be returned by calling the GetHomeParams method.

Motor Method *SetHWLimSwitches*

See Also Example Code

Visual Basic Syntax

Function **SetHWLimSwitches**(lChanID As Long, lRevLimSwitch As Long, lFwdLimSwitch As Long) As Long

Parameters

lChanID - the channel identifier

lRevLimSwitch - the action of the reverse limit switch when contact is made

lFwdLimSwitch - the action of the forward limit switch when contact is made

Returns

MG Return Code

Details

This method notifies the system to the action of the forward and reverse hardware limit switches associated with the channel specified by the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Although the action that the forward and reverse hardware limit switches make on contact is inherent in the stage design, it must be entered in the *lFwdLimSwitch* and *lRevLimSwitch* parameters respectively, which in turn, take values from the HWLIMSWITCH enumeration.

If set to HWLIMSW_IGNORE, the limit switch is ignored.

If set to HWLIMSW_MAKES, the switch closes on contact.

If set to HWLIMSW_BREAKS, the switch opens on contact

For details on the limit switch action of a particular stage, refer to *Stage Information*

The limit switch settings can be returned by calling the GetHWLimSwitches method.

Motor Method *SetIndicatorLEDMode*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube Driver products ODC001, OST001, TDC001 and TST001

Visual Basic Syntax

Function **SetIndicatorLEDMode**(lChanID As Long, lLEDMode As Long) As Long

Parameters

lChanID – the channel identifier

lLEDMode – the operating mode of the front panel LED

Returns

MG Return Code

Details

This method specifies the operation mode of the front panel LED on the motor controller unit.

Note. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the OptoSTDriver and OptoDCDriver, *lChanID* should always be set to 'CHAN1_ID'.

A single LED is fitted to the front panel of the unit. This LED can be configured to indicate certain driver states by setting the *lLEDMode* parameter, which takes values from the INDICATORLEDMODE enumeration as follows:

1. LEDMODE_IDENT: The LED will flash when the 'Ident' button is clicked on the APT Software GUI panel.
2. LEDMODE_LIMITSWITCH: The LED will flash when the motor reaches a forward or reverse limit switch.

8 LEDMODE_MOVING: The LED is lit when the motor is moving.

All modes are enabled by default. However, it is recognised that in a light sensitive environment, stray light from the LED could be undesirable. Therefore it is possible to enable selectively, one or all of the LED indicator modes described above by setting the appropriate value in the *lLEDMode* parameter.

For example, if the *lLEDMode* parameter is set to '9' (1 + 8), only the LEDMODE_IDENT and LEDMODE_MOVING modes are enabled. Similarly, if the *lLEDMode* parameter is set to '11' (1 + 2 + 8) all modes will be enabled.

Motor Method *SetJogMode*

See Also Example Code

Visual Basic Syntax

Function **SetJogMode**(lChanID As Long, lMode As Long, lStopMode) As Long

Parameters

lChanID - the channel identifier

lMode - the jogging mode

lStopMode - the way in which the motor stops when the jog button is released

Returns

MG Return Code

Details

This method sets the jogging mode for the channel(s) specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The jogging mode is specified the *lMode* parameter, which takes values specified by the JOGMODE enumeration.

Jog commands can be issued via the Motor Control GUI panel or by using a remote handset. When a jog command is received,

if the jog mode is set to 'Step' the motor moves by the step size specified in the SetJogStepSize method (or the GUI Settings panel). If the jog mode is set to 'Continuous', the motor continues to move until the jog signal is removed (i.e. the jog button is released). Note. Continuous Jog mode is not supported from the GUI controls.

The way in which the motor stops when the jog signal is removed, is set in the *lStopMode* parameter which, in turn, takes values from the STOPMODE enumeration.

If STOP_IMMEDIATE is selected, moves are stopped abruptly in a non-profiled manner, (i.e. the parameter *fAccn*, set using the SetVelParams method, is ignored) and there is a risk that steps, and therefore positional integrity, could be lost.

If STOP_PROFILED is selected, moves are stopped in a profiled manner using the velocity profile parameters set using the SetVelParams method.

To retrieve the jogging mode for a particular channel, see the GetJogMode method.

Motor Method *SetJogStepSize*

See Also Example Code

Visual Basic Syntax

Function **SetJogStepSize_Mot**(lChanID As Long, fStepSize As Single) As Long

Parameters

lChanID - the channel identifier

fStepSize - the size of step taken when the jog signal is initiated

Returns

MG Return Code

Details

Note: This method is applicable only if the Jog Mode is set to 'Step' in the SetJogMode method.

This method sets the distance to move when a jog command is initiated. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The step size is specified in real world units (mm or degrees dependent upon the stage) in the *fStepSize* parameter:

To retrieve the jog step size for a particular channel, see the GetJogStepSize method.

Example

```
Private Sub cmdSetJogParams_Click()
' NB: this is for use by the handheld control unit!!!
' Set jog parameters-velocities and step size
Dim sngJogMinVel As Single, sngJogAccn As Single, sngJogMaxVel As Single
Dim sngJogStepSize As Single
' Get user specified parameters
sngJogMinVel = CSng(Val(txtJogMinVel))
sngJogMaxVel = CSng(Val(txtJogMaxVel))
sngJogAccn = CSng(Val(txtJogAccn))
sngJogStepSize = CSng(Val(txtJogStepSize))
MG17Motor1.SetJogStepSize CHAN1_ID, sngJogStepSize
MG17Motor1.SetJogVelParams CHAN1_ID, sngJogMinVel, sngJogAccn, sngJogMaxVel
End Sub
```

Motor Method *SetJogVelParams*

See Also Example Code

Visual Basic Syntax

Function **SetJogVelParams**(lChanID As Long, fMinVel As Single, fAccn As Single, fMaxVel As Single) As Long

Parameters

lChanID - the channel identifier

fMinVel - the minimum velocity at which to start jog moves

fAccn - the rate at which the velocity climbs to, and slows from, maximum velocity

fMaxVel - the maximum velocity at which to perform jog moves

Returns

MG Return Code

Details

This method allows the trapezoidal velocity profile parameters to be set for all jog moves. Parameters are set in real world units (mm or degrees) dependent upon stage type.

The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Notes.

The *fMinVel* parameter value is locked at zero and cannot be adjusted.

If the Jog Mode is set to "Step' and the step size is small, then the maximum velocity may not be reached.

On dual channel units, the system cannot set the jog velocity parameters of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

The real world units associated with a particular channel can be obtained by calling the GetStageAxisInfo method.

The velocity parameters associated with a particular channel can be obtained by calling the GetJogVelParams method.

Example

```
Private Sub cmdSetJogParams_Click()

' NB: this is for use by the handheld control unit!!!

' Set jog parameters-velocities and step size

Dim sngJogMinVel As Single, sngJogAccn As Single, sngJogMaxVel As Single

Dim sngJogStepSize As Single

' Get user specified parameters

sngJogMinVel = CSng(Val(txtJogMinVel))

sngJogMaxVel = CSng(Val(txtJogMaxVel))

sngJogAccn = CSng(Val(txtJogAccn))

sngJogStepSize = CSng(Val(txtJogStepSize))

MG17Motor1.SetJogStepSize CHAN1_ID, sngJogStepSize

MG17Motor1.SetJogVelParams CHAN1_ID, sngJogMinVel, sngJogAccn, sngJogMaxVel

End Sub
```

Motor Method SetKCubePanelParams

This method is applicable only to K-Cube Series motor controllers

See Also Example Code

Visual Basic Syntax
Function **SetKCubePanelParams** (lChanID As Long, lWheelMode As Long, fWheelVel As Float, fWheelAccn As Float, lWheelDirSense As Long, fPresetPos1 As Float, fPresetPos2 As Long, lDispBrightness As Long, lDispTimeout As Long, lDispDimLevel As Long) As Long

Parameters
lChanID - the channel identifier
lWheelMode - The operating mode of the top panel velocity wheel
fWheelVel - The max velocity of a move initiated by the top panel velocity wheel
fWheelAccn - The max acceleration of a move initiated by the top panel velocity wheel
lWheelDirSense - The direction sense of a move initiated by the top panel velocity wheel
fPresetPos1 - The preset position 1 when operating in go to position mode
fPresetPos2 - The preset position 2 when operating in go to position mode

lDispBrightness - The display brightness when the unit is active
lDispTimeout - The display time out (in minutes). A static display will be dimmed after this time period has elapsed.
lDispDimLevel - The display brightness after time out.

Returns
MG Return Code

Details
This method sets the operating parameters of the velocity wheel on the top panel of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode is specified in the lWheelMode parameter, which in turn takes values from the KMOTWHEELMODE enumeration as follows:

1 Velocity Control Mode - Deflecting the wheel starts a move with the velocity proportional to the deflection. The maximum velocity (i.e. velocity corresponding to the full deflection of the joystick wheel) and acceleration are specified in the fWheelVel and fWheelAccn parameters.

2 Jog Mode - Deflecting the wheel initiates a jog move, using the parameters specified by the SetJogStepSize and SetJogVelParams methods. Keeping the wheel deflected repeats the move automatically after the current move has completed.

3 Go To Position Mode - Deflecting the wheel starts a move from the current position to one of the two predefined “teach” positions. The teach positions are specified in number of steps from the home position in the fPresetPos1 and fPresetPos2 parameters.

The direction of a move initiated by the velocity wheel is specified in the lWheelDirSense parameter, which takes values from the KMOTWHEELDIRSENSE enumeration as follows:

0 The wheel is disabled

1 Upwards rotation of the wheel results in a positive motion (i.e. increased position count).

The following option applies only when the Wheelmode is set to Velocity Control Mode (1). If set to Jog Mode (2) or Go to Position Mode (3), the following option is ignored.

2 Upwards rotation of the wheel results in a negative motion (i.e. decreased position count).

In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit. The brightness is set in the lDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Furthermore, 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after the time interval specified in the lDispTimeout parameter has elapsed. The time interval is specified in minutes in the range 0 (never) to 480. The dim level is set in the lDispDimLevel parameter, as a value from 0 (Off) to 10 (brightest) but is also limited by the DispBrightness parameter.

The current setting for panel parameters may be obtained by calling the GetKCubePanelParams method.

KCube Motor Method SetKCubePosTriggerParams

This method is applicable only to K-Cube Series motor controllers

See Also Example Code

Visual Basic Syntax
Function SetKCubePosTriggerParams (lChanID As Long, fStartPosFwd As Float, fPosIntervalFwd As Float, lNumPulsesFwd As Long, fStartPosRev As Float, fPosIntervalRev As Float, lNumPulsesRev As Long, fPulseWidth As Float, lNumCycles As Long) As Long

Parameters
lChanID - the channel identifier
fStartPosFwd - Trigger pulse output position start forward [in position counts - encoder counts or microsteps].
fPosIntervalFwd - Trigger pulse output position interval forward [in position counts - encoder counts or microsteps].
lNumPulsesFwd - Number of forward output pulses during a move.
fStartPosRev - Trigger pulse output position start backward [in position counts - encoder counts or microsteps].
fPosIntervalRev - Trigger pulse output position interval backward [in position counts - encoder counts or microsteps].
lNumPulsesRev - Number of backward output pulses during a move.
fPulseWidth - Trigger output pulse width (from 1 µs to 1000000 µs).
lNumCycles - Number of forward/reverse move cycles.

Returns
MG Return Code

Details

The K-Cube motor controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be set to be used as input or output triggers. This method sets operating parameters used when the triggering mode is set to a trigger out position steps mode by calling the SetKCubeTriggerParams method.

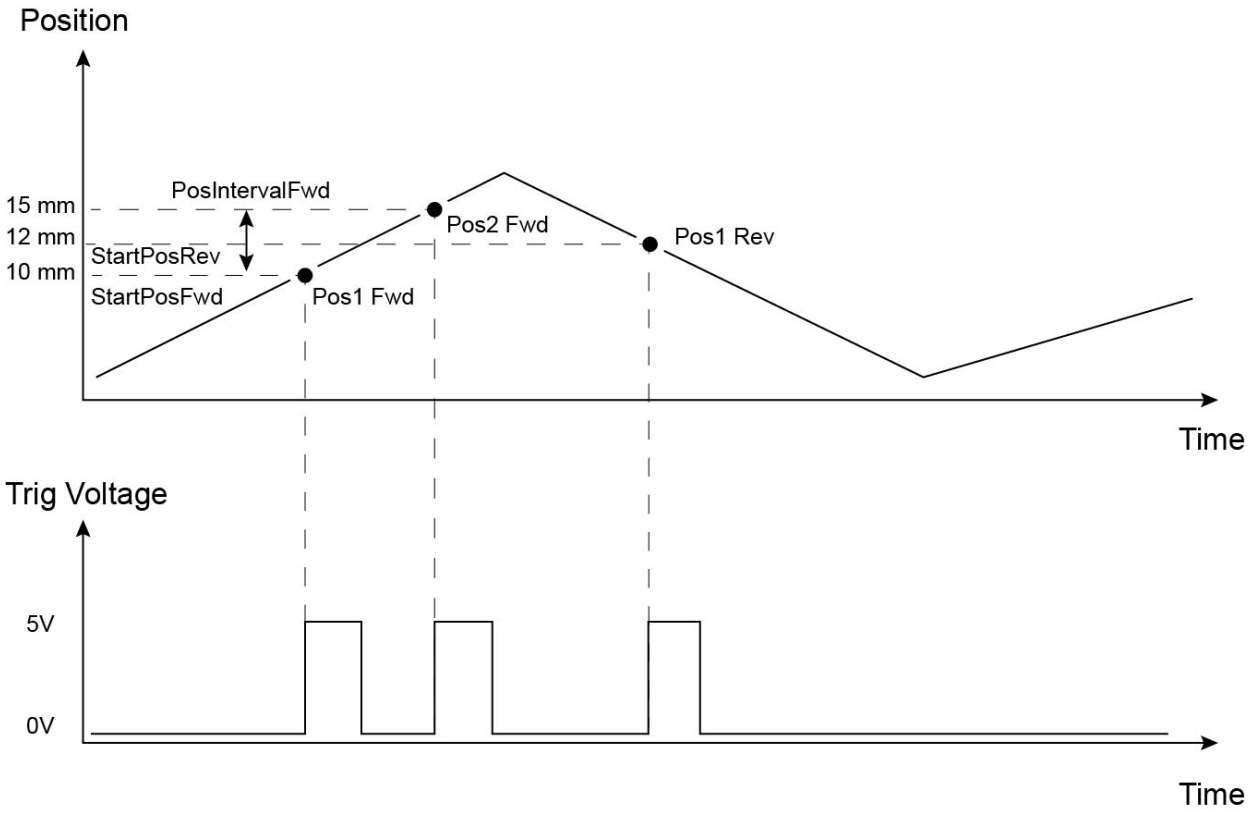
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

As soon as position triggering is selected on either of the TRIG ports, the port will assert the inactive logic state, set in the SetKCubeTriggerParams method. As the stage moves in its travel range and the actual position matches the position set in the fStartPosFwd parameter, the TRIG port will output its active logic state. The active state will be output for the length of time specified by the fPulseWidth parameter, then return to its inactive state and schedule the next position trigger point at the "fStartPosFwd value plus the value set in the fPosIntervalFwd parameter. Thus when this second position is reached, the TRIG output will be asserted to its active state again. The sequence is repeated the number of times set in the lNumPulsesFwd parameter.

When the number of pulses set in the lNumPulsesFwd parameter has been generated, the trigger engine will schedule the next position to occur at the position specified in the fStartPosRev parameter. The same sequence as the forward direction is now repeated in reverse, except that the fPosIntervalRev and lNumPulsesRev parameters apply. When the number of pulses has been output, the entire forward-reverse sequence will repeat the number of times specified by lNumCycles parameter. This means that the total number of pulses output will be lNumCycles x *(lNumPulsesFwd + lNumPulsesRev).*

Once the total number of output pulses have been generated, the trigger output will remain inactive.

When a unidirectional sequence is selected, only the forward or reverse part of the sequence will be activated.



Example for a move from 0 to 20 mm and back.

In forward direction: The first trigger pulse occurs at 10 mm (StartPosFwd), the next trigger pulse occurs after another 5 mm (PosIntervalFwd), the stage then moves to 20 mm.

In reverse direction: The next trigger occurs when the stage gets to 12 mm.

Note that the position triggering scheme works on the principle of always triggering at the next scheduled position only, regardless of the actual direction of movement. If, for example, a position trigger sequence is set up with the forward start position at 10 mm, but initially the stage is at 15 mm, the first forward position trigger will occur when the stage is moving in the reverse direction. Likewise, if the stage does not complete all the forward position trigger points, the reverse triggering will not activate at all. For normal operation it is assumed that all trigger points will be reached during the course of the movement.

The position triggering sequence can be stopped at any time by changing the triggering function of the port to one of the other options in the SetKCubeTriggerParams method (for example, general purpose output). If the function of the port is then changed back to position triggering, the sequence will re-start from the beginning.

To obtain the current position trigger parameter settings, see the GetKCubePosTriggerParams method.

KCube Motor Method SetKCubeTriggerParams

This method is applicable only to K-Cube Series motor controllers

See Also Example Code

Visual Basic Syntax
Function **SetKCubeTriggerParams** (lChanID As Long, lTrig1Mode As Long, lTrig1Polarity As Long, lTrig2Mode As Long, lTrig2Polarity As Long) As Long

Parameters
lChanID - the channel identifier
lTrig1Mode - TRIG1 operating mode
lTrig1Polarity - The active state of TRIG1 (i.e. logic high or logic low)

Trig2Mode - TRIG2 operating mode
lTrig2Polarity - The active state of TRIG2 (i.e. logic high or logic low)

Returns
MG Return Code

Details

The K-Cube motor controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be used to read an external logic signal or output a logic level to control external equipment. Either of them can be independently configured as an input or an output and the active logic state can be selected High or Low to suit the requirements of the application. Electrically the ports output 5 Volt logic signals and are designed to be driven from a 5 Volt logic.

When the port is used in the input mode, the logic levels are TTL compatible, i.e. a voltage level less than 0.8 Volt will be recognised as a logic LOW and a level greater than 2.4 Volt as a logic HIGH. The input contains a weak pull-up, so the state of the input with nothing connected will default to a logic HIGH. The weak pull-up feature allows a passive device, such as a mechanical switch to be connected directly to the input.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method sets the operating parameters of the TRIG1 and TRIG2 connectors on the front panel of the unit.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode of the trigger is specified in the lTrig1Mode and lTrig2Mode parameters, which takes values from the KMOTTRIGMODE enumeration as follows:

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a relative, absolute or home move as follows:

0x00 The trigger IO is disabled

0x01 General purpose logic input (read through status bits using the LLGetStatusBits method).

0x02 Input trigger for relative move.

0x03 Input trigger for absolute move.

0x04 Input trigger for home move.

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger. The relative distance or absolute position to move, is that set in the SetRelMoveDist and SetAbsMovePos methods.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output, or to indicate motion status or to produce a trigger pulse at configurable positions as follows:

0x0A General purpose logic output (set using the LLSetGetDigOPs method).

0x0B Trigger output active (level) when motor 'in motion'. The output trigger goes high (5V) or low (0V) (as set in the lTrig1Polarity and lTrig2Polarity parameters) when the stage is in motion.

0x0C Trigger output active (level) when motor at 'max velocity'.

0x0D Trigger output active (pulsed) at pre-defined positions moving forward (set using wStartPosFwd, lIntervalFwd, wNumPulsesFwd and lPulseWidth parameters in the SetKCubePosTriggerParams method). Only one Trigger port at a time can be set to this mode.

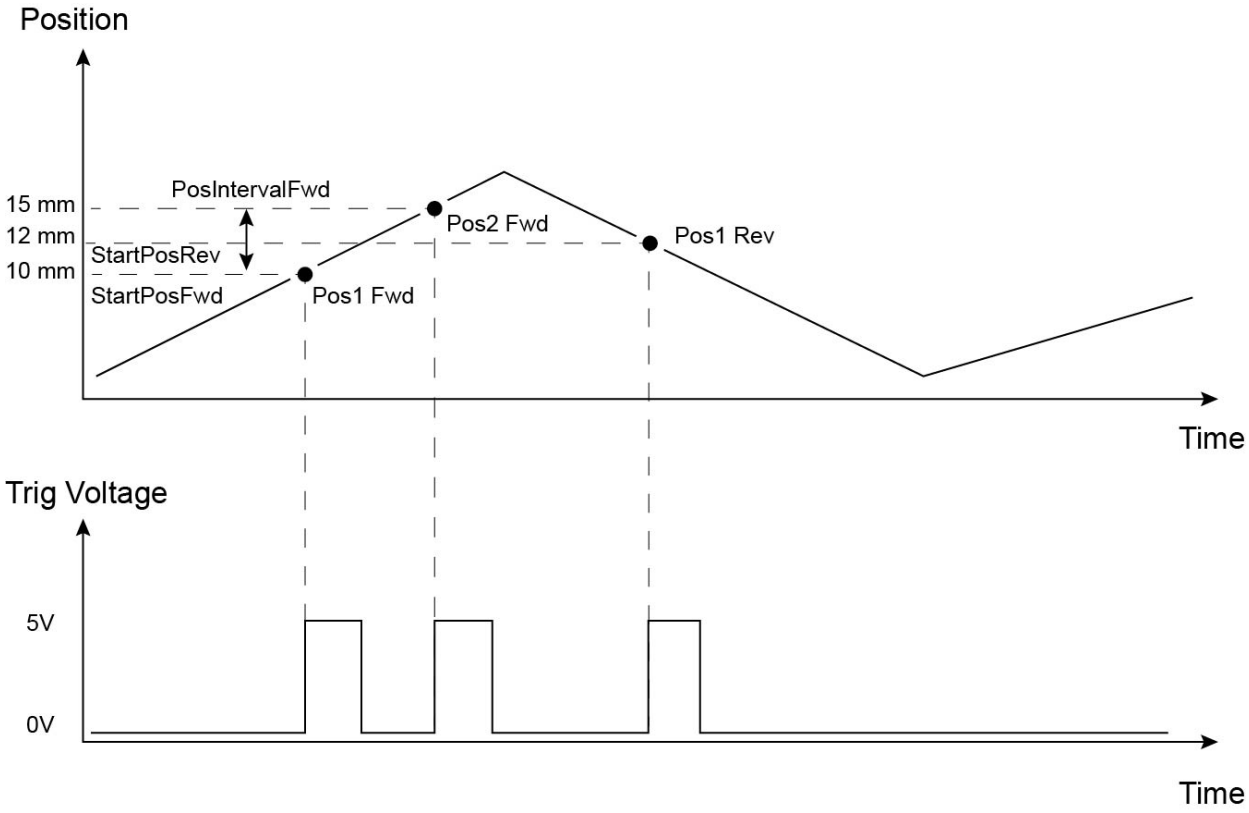
0x0E Trigger output active (pulsed) at pre-defined positions moving backwards (set using wStartPosRev, lIntervalRev, wNumPulsesRev and lPulseWidth parameters in the SetKCubePosTriggerParams method). Only one Trigger port at a time can be set to this mode.

0x0F Trigger output active (pulsed) at pre-defined positions moving forwards and backward. Only one Trigger port at a time can be set to this mode.

Trigger Out Position Steps

In the last three modes described above, the controller outputs a configurable number of pulses, of configurable width, when the actual position of the stage matches the position values configured as the Start Position and Position Interval - see SetKCubePosTriggerParams method. These modes allow external equipment to be triggered at exact position values. The position pulses are generated by dedicated hardware, allowing a very low latency of less than 1 usec. The low latency of this triggering mode provides a very precise indication of a position match (assuming a stage velocity of 10 mm/sec, the less than 1 usec latency would in itself only result in a 10 nm position uncertainty, which is normally well below the accuracy limitations of the mechanics.)

Using the last three modes above, position triggering can be configured to be unidirectional (forward or revese only) or bidirectional (both). In bidirectional mode the forward and reverse pulse sequences can be configured separately. A cycle count setting (set in the SetKCubePosTriggerParams method, *INumCycles* parameter) allows the uni- or bidirectional position triggering sequence to be repeated a number of times.



Example for a move from 0 to 20 mm and back.

In forward direction: The first trigger pulse occurs at 10 mm (StartPosFwd), the next trigger pulse occurs after another 5 mm (PosIntervalFwd), the stage then moves to 20 mm.

In reverse direction: The next trigger occurs when the stage gets to 12 mm.

Please note that position triggering can only be used on one TRIG port at a time, as there is only one set of position trigger parameters.

The operation of the position triggering mode is described in more detail in the SetKCubePosTriggerParams method.

The polarity of the trigger pulse is specified in the lTrig1Polarity and lTrig2Polarity parameters, which takes values from the KMOTTRIGPOLARITY enumeration as follows: :

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

For details on obtain the current trigger parameter settings, see the GetKCubeTriggerParams method.

Motor Method *SetMotorParams*

See Also Example Code

Visual Basic Syntax

Function **SetMotorParams**(lChanID As Long, lStepsPerRev As Long, lGearBoxRatio As Long) As Long

Parameters

lChanID - the channel identifier

lStepsPerRev - sets the number of full steps per resolution of the stepper motor

lGearBoxRatio - sets the ratio of the motor's gearbox (if fitted)

Returns

MG Return Code

Details

This method is used to obtain the 'resolution' characteristics of the stepper motor connected to the channel specified by the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. The resolution of the motor, combined with other characteristics (such as lead screw pitch) of the associated actuator or stage, determines the overall resolution.

The number of full steps per revolution of the stepper motor is specified in the *lStepsPerRev* parameter, which has minimum and maximum limits of '1' and '10000' respectively.

If the motor has a gearbox, the ratio of the gearbox is specified in the *lGearboxRatio* parameter. For example, if the gearbox has a reduction ratio of X:1 (i.e. every 1 turn at the output of the gearbox requires X turns of the motor shaft) then the *lStepsPerRev* value is set to X. This parameter has minimum and maximum limits of '1' and '1000' respectively.

Note. The stepper motors used on the majority of Our stages/actuators have 200 full steps per rev and no gearbox fitted. For these motors the *lStepsPerRev* and *lGearboxRatio* parameters have values of 200 and 1 respectively. As an exception to this, the ZST family of actuators use 24 steps per rev stepper motors fitted with a 76:1 reduction gearbox. In this case, the *lStepsPerRev* and *lGearboxRatio* should be set to '24' and '76' respectively.

Note. The correct default values for *lStepsPerRev* and *lGearboxRatio* are applied automatically when the APTConfig.exe utility is used to associate a specific stage or actuator type with a motor channel. See the APTConfig helpfile for more details.

The resolution characteristics of the motor are obtained using the GetMotorParams method.

Motor Method *SetPhaseCurrents*

See Also Example Code
Visual Basic Syntax
Function **SetPhaseCurrents**(lChanID As Long, lRestVal As Long, lMoveVal As Long) As Long
Parameters
lChanID - the channel identifier
lRestVal - the phase current value when the motor is at rest
lMoveVal - the phase current value when the motor is moving
Returns
MG Return Code
Details
The current needed to hold a motor in a fixed position is much smaller than that required for a move. It is good practice to decrease the current in a stationary motor in order to reduce heating, and thereby minimize thermal movements caused by expansion.
This method sets the phase current for the motor associated with the channel specified by the *lChanID* parameter, which in turn

takes values specified by the HWCHANNEL enumeration.
The rest current and the move current values are set as a percentage of full current, in the *lRestVal* and *lMoveVal* parameters respectively. Typically, move current should be set to 100% and rest current to a value significantly less than this.

Note. For BSC20x series users.
The moving phase current is set automatically by the unit and cannot be adjusted. However, the *lRestVal* parameter can still be used to set the resting phase current value. ,

Motor Method *SetPositionOffset*

See Also Example Code

Visual Basic Syntax

Function **SetPositionOffset**(lChanID As Long, fPosOffset As Float) As Long

Parameters

lChanID – The channel identifier

fPosOffset –The zero offset value

Returns

MG Return Code

Details

This method allows a position offset to be entered for the channel specified by the *lChanID* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the OptoSTDriver and OptoDCDriver, *lChanID* should always be set to 'CHAN1_ID'.

For example, in an application using a 50mm travel linear stage, it may be advantageous to display zero at mid travel, with movement either side of the mid point being displayed as positive or negative. If the *fPosOffset* parameter is set to –25, the GUI position display will show zero at 25mm. A movement of 2mm in a positive direction will be displayed as 2.00 and similarly, a movement of 3mm in a negative direction will be displayed as –3.00.

The position offset value entered can be returned by calling the GetPositionOffset method.

Motor Method *SetPotParams*

See Also Example Code

Note. This method is applicable only to the OptoDriver and T-Cube Driver products ODC001, OST001, TDC001 and TST001

Visual Basic Syntax

Function **SetPotParams**((lChanID As Long, lVel1PotVal As Long, fVel1 As Single, lVel2PotVal As Long, fVel2 As Single, lVel3PotVal As Long, fVel3 As Single, lVel4PotVal As Long, fVel4 As Single,) As Long

Parameters

lChanID – the channel identifier

lVel1PotVal – the Velocity 1 Pot Deflection Value

fVel1 – the velocity associated with deflection value 1

lVel2PotVal – the Velocity 2 Pot Deflection Value

fVel2 – the velocity associated with deflection value 2

lVel3PotVal – the Velocity 3 Pot Deflection Value

fVel3 – the velocity associated with deflection value 3

lVel4PotVal – the Velocity 4 Pot Deflection Value

fVel4 – the velocity associated with deflection value 4

Returns

MG Return Code

Details

This method specifies a number of potentiometer deflection values and associated velocites pertaining to the operation of the motor controller unit.

Note. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration. For single channel units such as the OptoSTDriver and OptoDCDriver, *lChanID* should always be set to 'CHAN1_ID'.

The potentiometer slider is sprung such that when released it returns to it's central position. In this central position the motor is stationary. As the slider is moved away from the center, the motor begins to move; the speed of this movement increases as the slider deflection is increased. Bidirectional control of motor moves is possible by moving the slider in both directions.

Operation with the OptoDCDriver

The speed of the motor increases by discrete amounts rather than continuously, as a function of slider deflection. These speed settings can be altered via the 'Potentiometer Control Settings' parameters.

There are 4 pairs of parameters, each pair specifies a pot deflection value (in the range 0 to 127) together with an associated velocity (set in real world units, mm or degrees) to be applied at or beyond that deflection. As each successive deflection is reached by moving the pot slider, the next velocity value is applied. These settings are applicable in either direction of pot deflection, i.e. 4 possible velocity settings in the forward or reverse motion directions.

For example, consider the settings below:

lVel1PotVal – 20

fVel1 – 0.1

lVel2PotVal – 50

fVel2 – 0.2

lVel3PotVal – 80

fVel3 – 0 3

lVel4PotVal – 100

fVel4 – 0.5

The parameters above indicate that when the pot has been deflected to 20 (approx 1/6 full scale deflection) the motor will start to move at 0.1mm/sec. At a deflection of 50 (approx 2/5 full scale deflection) the motor velocity will increase to 0.2m/sec, and so on.

Note. It is acceptable to set velocities equal to each other to reduce the number of speeds, however this is not allowed for the deflection settings, whereby the Velocity 4 Pot Deflection value must be greater than Velocity 3 Pot Deflection value.

Operation with the OptoSTDriver

The velocity of the move is entered in real world units (mm or degrees), in the *fVel1* parameter. All other parameters are ignored. The velocity profile is derived from the 'Velocity Profile' settings in the 'Move/Jogs' settings tab.

The current settings for these parameters can be obtained by calling the GetPotParams method.

Motor Method *SetRelMoveDist*

See Also Example Code

Visual Basic Syntax

Function **SetRelMoveDist**(lChanID As Long, fRelDist As Single) As Long

Parameters

lChanID - the channel identifier

fRelDist - the relative distance associated with the specified channel

Returns

MG Return Code

Details

This method sets the distance, relative to the present position, which the channel specified by the *lChanID* parameter will move, the next time the *MoveRelative* method is called. The distance is set to the value specified in the *fRelDist* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the relative distance of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To get the relative distances for a particular channel, see the GetRelMoveDist method.

Motor Method *SetRotStageModes*

See Also Example Code

Note. This method is applicable only to Rotational Stages

Visual Basic Syntax

Function **SetRotStageModes** (lMoveMode As Long, lPosReportMode As Long) As Long

Parameters

lMoveMode - the move direction, positive, negative or quickest

lPosReportMode – the position reporting mode, 360 or total.

Returns

MG Return Code

Details

This method is applicable only to rotational stages. It specifies the direction in which a move is performed, and the way in which positional information is reported.

The *lMoveMode* parameter specifies the move direction and takes values from the ROTMOVEMODE enumeration as follows:

1. ROT_MOVE_POS Rotate Positive
2. ROT_MOVE_NEG Rotate Negative
3. ROT_MOVE_SHORT Rotate Quickest

Note. The *lMoveMode* parameter is applicable only if the Absolute Position Reporting Mode is set to ‘Equivalent Angle 0 to 360 degrees’ in the Rotation Stages settings panel – see the Operation section, Rotation Stages tab, OR the *lPosReportMode* parameter (see below) is set to ROT_POSDISP_360.

The *lPosReportMode* parameter specifies the position reporting mode. It takes values from the ROTPOSDISPMODE enumeration as follows:

1. ROT_POSDISP_360 Position reporting is 0 to 360°
2. ROT_POSDISP_TOTAL Position reporting is the total degree count

Motor Method *SetStageAxisInfo*

See Also Example Code

Visual Basic Syntax

Function **SetStageAxisInfo**(lChanID As Long, fMinPos As Single, fMaxPos As Single, lUnits As Long, fPitch As Single, lDirSense As Long) As Long

Parameters

lChanID - the channel identifier

fMinPos - the minimum limit position

fMaxPos - the maximum limit position

lUnits - the units (mm or degrees)

fPitch - the pitch (in mm) of the motor lead screw (i.e. the distance to travel per revolution of the motor)

lDirSense - reserved for future use

Returns

MG Return Code

Details

Note. Stage information is set automatically when the stage is assigned using the APT Config utility. This method should be called only if the stage type has not been set using APT Config.

This method sets the characteristics of the stage and axis associated with the channel specified by the *lChanID.* These properties are inherent in the design of the stage and are required to enable the system to drive correctly the particular type of stage and axis. For details on a particular stage, refer to *Stage Information*

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

The *lUnits* parameter takes values specified by the STAGEUNITS enumeration and contains the units of measure for stage motion, either millimeters or degrees, whichever is relevant.

The positions of the minimum and maximum travel limits are set in the *fMinPos* and *fMaxPos* parameters respectively, and are determined by their displacement from the Home position.

The *fZeroOffset* parameter sets the distance between the Home position and the minimum limit - see Motor Controller Operators Guide for further information.

The *fPitch* parameter sets the pitch of the motor leadscrew (in mm), which is used to calculate the distance moved for each revolution of the motor.

The stage axis information for a particular channel can be obtained using the GetStageAxisInfo method.

Motor Method *SetTriggerParams*

See Also Example Code

Note for BSC001 or BSC002 Users. On units to modification state B or earlier, triggering functionality is enabled by fitting a trigger enable plug. These plugs are available from the company on request.

A label indicating the build and modification state of the hardware is located on the underside of the unit. Modification states are ‘scored out’ as they become current.

Note. This method is not applicable to OptoSTDriver (OST001) and OptoDCDriver (ODC001) units.

Visual Basic Syntax

Function **SetTriggerParams**(lChanID As Long, lTrigMode As Long, lTrigMove As Long) As Long

Parameters

lChanID - the channel identifier

lTrigMode – sets the trigger mode for the specified channel

lTrigMove – sets the type of move to be initiated on receipt of a trigger signal

Returns

MG Return Code

Details

This method is used to configure the APT stepper controller for triggered move operation. It is possible to configure a particular controller to respond to trigger inputs, generate trigger outputs or both respond to and generate a trigger output. When a trigger input is received, the stepper unit can be set to initiate a move (relative, absolute or home). Similarly the unit can be set to generate a trigger output signal when instructed to move via a software (USB) command. For those units configured for both input and output triggering, a move can be initiated via a trigger input while at the same time, a trigger output can be generated to initiate a move on another unit.

The trigger settings can be used to configure multiple units in a master – slave set up, thereby allowing multiple channels of motion to be synchronized. Multiple moves can then be initiated via a single software or hardware trigger command.

The channel to be set is specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration.

The *lTrigMode* parameter sets the trigger mode for the specified channel. The various trigger modes are specified by the TRIGMODE enumeration as follows:

TRIGMODE_DISABLED – triggering operation is disabled

TRIGMODE_IN – a move (specified using the *lTrigMove* parameter below) is initiated on the specified channel when a trigger input signal (i.e. rising edge) is received on the Trig In connector (either BNC or D-type pin depending on the unit).

TRIGMODE_INOUT – As for TRIGMODE_IN, but now a trigger output signal will also be generated on the Trig Output connector when the move begins.

TRIGMODE_OUT – A trigger output signal is generated on the Trig Out connector when a move (relative, absolute or home) is initiated via software (either through the GUI panel or ActiveX method call).

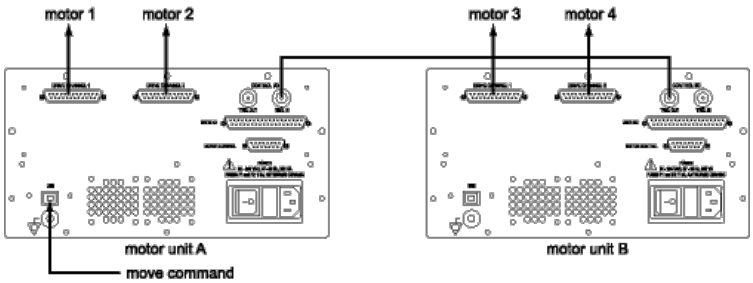
The *lTrigMove* parameter determines the type of move to be initiated on the specified channel when a trigger input signal is detected on the Trig In connector. The move types are specified by the TRIGMOVE enumeration as follows:

TRIGMOVE_REL – triggered relative move. Relative move distance and velocity profile parameters can be set prior to triggering using the SetRelMoveDist and SetVelParams methods respectively.

TRIGMOVE_ABS – triggered absolute move. Absolute move position and velocity profile parameters can be set prior to triggering using the SetAbsMovePos and SetVelParams methods respectively.

TRIGMOVE_HOME – triggered home sequence. Homing parameters can be set prior to triggering using the SetHomeParams method

For example, consider the following diagram for a BSC002 controller:



Software setup

Motor Unit A (Master)
SetTriggerParams (CHAN1_ID, TRIGMODE_OUT, TRIGMOVE_REL)

Motor Unit B (Slave)
SetTriggerParams (CHAN1_ID, TRIGMODE_IN, TRIGMOVE_REL)
SetTriggerParams (CHAN2_ID, TRIGMODE_IN, TRIGMOVE_REL)

Initiate move on Motor Unit A via software,
MoveRelative (CHANBOTH_ID, 1.0, 1.0, False)

When the move command is received via the USB link, unit A channels 1 and 2 start to move simultaneously by the specified relative distances. Unit A also sends a trigger signal to Unit B. Furthermore, channels 1 and 2 on unit B have been set to 'TrigMode_IN', so on receipt of the trigger from unit A, these channels also start to move.

The trigger parameters are obtained using the GetTriggerParams method.

Please refer to the handbook supplied with your unit for further information on identifying the trigger connections.

Motor Method *SetVelParams*

See Also **Example Code**

Visual Basic Syntax

Function **SetVelParams**(lChanID As Long, fMinVel As Single, fAccn As Single, fMaxVel As Single) As Long

Parameters

lChanID - the channel identifier

fMinVel - the minimum velocity at which to start and end a move

fAccn - the rate at which the velocity climbs from minimum to maximum, and slows from maximum to minimum

fMaxVel - the maximum velocity at which to perform a move

Returns

MG Return Code

Details

This method is applicable to moves initiated from software by calling the MoveRelative, MoveAbs or MoveVelocity commands and allows the trapezoidal velocity profile parameters to be set for all moves.

Parameters are set in real world units (mm or degrees) dependent upon stage type.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Notes.

The *fMinVel* parameter value is locked at zero and cannot be adjusted.

On dual channel units, the system cannot set the velocity parameters of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

The real world units associated with a particular channel can be obtained by calling the GetStageAxisInfo method.

The velocity parameters associated with a particular channel can be obtained by calling the GetVelParams method.

Example

```
Private Sub cmdSetVelParams_Click()

' Set user-specified velocity parameters

Dim sngMinVel As Single, sngAccn As Single, sngMaxVel As Single

' Get user specified parameters

sngMinVel = CSng(Val(txtMinVel))

sngMaxVel = CSng(Val(txtMaxVel))

sngAccn = CSng(Val(txtAccn))

' Set velocity parameters as above

MG17Motor1.SetVelParams CHANBOTH_ID, sngMinVel, sngAccn, sngMaxVel

End Sub
```

Motor Method *ShowSettingsDlg*

See Also Example Code

Visual Basic Syntax

Function **ShowSettingsDlg**() As Long

Returns

MG Return Code

Details

This method displays the settings dialog window (in a similar way as clicking the ‘Settings’ button on the GUI panel).

Note. For older 2-channel units (BSC002) the channel can be selected by using the ‘Channel’ switch on the GUI panel, or by calling the SetChannelSwitch method.

Method *StartCtrl*

See Also Example Code

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns

MG Return Code

Details

This method is used in conjunction with the HWSerialNum property, to activate an ActiveX control instance at runtime.

After the HWSerialNum property has been set to the required serial number (at design time or run time), this method can be called at run time to initiate the hardware enumeration sequence and activate the control.

Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

MG17Motor1.HWSerialNum = 20000001

' Start motor control

MG17Motor1.StartCtrl

End Sub
```

Method *StopCtrl*

See Also **Example Code**

Visual Basic Syntax

Function **StopCtrl**() As Long

Returns

MG Return Code

Details

This method is called to deactivate the operation of an ActiveX control instance.

When this method is called, communication to the associated hardware unit is stopped. It is efficient programming practice to call StopCtrl on any ActiveX control instances that are no longer required to be active. For example, this may be the case in a multiple window client application where multiple instances of an ActiveX control need to communicate at different times with the same hardware unit

Example

```
Private Sub Form_Unload(Cancel As Integer)

' Stop system control and unload forms.

frmSystem.MG17System1.StopCtrl

Unload frmSystem

MG17Motor1.StopCtrl

Unload Me

End Sub
```

Motor Method *StopImmediate*

See Also Example Code

Visual Basic Syntax

Function **StopImmediate**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This method is called to stop motor moves on the channel specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration.

Moves are stopped abruptly in a non-profiled manner, (i.e. the parameter *fAccn*, set using the SetVelParams method, is ignored) and there is a risk that steps, and therefore positional integrity, could be lost.

This method can be called to stop prematurely, absolute or relative moves and, more importantly, is called to stop MoveAtVelocity moves.

Motor Method *StopProfiled*

See Also Example Code

Visual Basic Syntax

Function **StopProfiled**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This method is called to stop motor moves on the channel specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration.

Moves are stopped in a profiled manner using the velocity profile parameters set using the SetVelParams method.

This method can be called to stop prematurely, absolute or relative moves and, more importantly, is called to stop MoveAtVelocity moves.

MG Stage Information

Stage and Axis Type	APT 600 - linear axes	APT 600 - angular axes	NanoMax 300	NanoMax 600 X and Y	NanoMax 600 Z	NanoMax 600 Roll, Pitch,Yaw
Min Pos	0	0	0	0	0	0
Max Pos	12	6	4	4	4	6
Pitch	1.	1.	0.5	0.5	0.5	0.5
Units	mm	degrees	mm	mm	mm	degrees
Homing						
Direction Fwd/Rev	Rev	Rev	Rev	Rev	Rev	Rev
Limit Switch Rev/Fwd, HW/SW	Rev HW	Rev HW	Rev HW	Rev HW	Rev HW	Rev HW
Zero Offset	3	6	3	5.5	3	2.
Velocity	1.	1.	1.	1.	1.	1.
Hardware Limit Switches (Makes, Breaks, Ignore/Absent)						
Rev Switch	Breaks	Breaks	Breaks	Breaks	Breaks	Breaks
Fwd Switch	Absent	Absent	Absent	Absent	Absent	Absent

The following list of constants can be copied into a Visual Basic module and used instead of the 'magic numbers' detailed in the table above.

```
' STAGE AXIS CONSTANTS

' APT 600 linear axes

Public Const MIN_POS_APT600_LIN As Single = 0

Public Const MAX_POS_APT600_LIN As Single = 12

Public Const UNITS_APT600_LIN As Long = UNITS_MM

Public Const PITCH_APT600_LIN As Single = 1

Public Const DIR_SENSE_APT600_LIN As Long = 0 '???***

Public Const REV_SW_APT600_LIN As Long = HWLIMSW_BREAKS

Public Const FWD_SW_APT600_LIN As Long = HWLIMSW_IGNORE

' APT 600 angular axes

Public Const MIN_POS_APT600_ANG As Single = 0

Public Const MAX_POS_APT600_ANG As Single = 6

Public Const UNITS_APT600_ANG As Long = UNITS_DEG

Public Const PITCH_APT600_ANG As Single = 1

Public Const DIR_SENSE_APT600_ANG As Long = 0

Public Const REV_SW_APT600_ANG As Long = HWLIMSW_BREAKS

Public Const FWD_SW_APT600_ANG As Long = HWLIMSW_IGNORE

' NanoMax 300

Public Const MIN_POS_NANO300 As Single = 0

Public Const MAX_POS_NANO300 As Single = 4

Public Const UNITS_NANO300 As Long = UNITS_MM

Public Const PITCH_NANO300 As Single = 0.5

Public Const DIR_SENSE_NANO300 As Long = 0

Public Const REV_SW_NANO300 As Long = HWLIMSW_BREAKS

Public Const FWD_SW_NANO300 As Long = HWLIMSW_IGNORE

' NanoMax 600 linear

Public Const MIN_POS_NANO600_LIN As Single = 0

Public Const MAX_POS_NANO600_LIN As Single = 4

Public Const UNITS_NANO600_LIN As Long = UNITS_MM

Public Const PITCH_NANO600_LIN As Single = 0.5

Public Const DIR_SENSE_NANO600_LIN As Long = 0
```

```
Public Const REV_SW_NANO600_LIN As Long = HWLIMSW_BREAKS

Public Const FWD_SW_NANO600_LIN As Long = HWLIMSW_IGNORE

' NanoMax 600 angular

Public Const MIN_POS_NANO600_ANG As Single = 0

Public Const MAX_POS_NANO600_ANG As Single = 4

Public Const UNITS_NANO600_ANG As Long = UNITS_MM

Public Const PITCH_NANO600_ANG As Single = 0.5

Public Const DIR_SENSE_NANO600_ANG As Long = 0

Public Const REV_SW_NANO600_ANG As Long = HWLIMSW_BREAKS

Public Const FWD_SW_NANO600_ANG As Long = HWLIMSW_IGNORE
```

Example

The following code sample shows how these constants can be used to configure a stepper motor controller. In this example, channel 1 is configured to control a linear axis, and channel 2 an angular axis, of an APT600 stage.

```
Private Sub cmdSetStage_Click()

' Set up stage parameters

' Set up limit switches for linear APT 600 axis on channel 1 of motor control

MG17Motor1.SetHWLimSwitches CHAN1_ID, REV_SW_APT600_LIN, FWD_SW_APT600_LIN

' Set up limit switches for angular APT 600 axis on channel 2 of motor control

MG17Motor1.SetHWLimSwitches CHAN2_ID, REV_SW_APT600_ANG, FWD_SW_APT600_ANG

' Set up stage axis info for linear APT 600 axis on channel 1 of motor control

MG17Motor1.SetStageAxisInfo CHAN1_ID, MIN_POS_APT600_LIN, MAX_POS_APT600_LIN, _

UNITS_APT600_LIN, PITCH_APT600_LIN, DIR_SENSE_APT600_LIN

' Set up stage axis info for angular APT 600 axis on channel 2 of motor control

MG17Motor1.SetStageAxisInfo CHAN2_ID, MIN_POS_APT600_ANG, MAX_POS_APT600_ANG, _

UNITS_APT600_ANG, PITCH_APT600_ANG, DIR_SENSE_APT600_ANG

End Sub
```

Motor Property *APTHelp*

See Also Example Code

Visual Basic Syntax

Property APTHelp As Boolean

Returns

MG Return Code

Details

This property specifies the help file that will be accessed when the user presses the F1 key. If APTHelp is set to 'True', the main server helpfile MG17Base will be launched. If APTHelp is set to 'False', the helpfile is the responsibility of the application programmer. **Motor Property *DisplayMode***

See Also Example Code

Visual Basic Syntax

Property DisplayMode As Long

Returns

MG Return Code

Details

This property allows the display mode of the virtual display panel to be set/read.

It accepts values from the DISPLAYMODE Enumeration as follows:

DISPMODE_PANEL 1 Display set to panel view

DISPMODE_GRAPH 2 Display set to graphical view

Property *HWSerialNum*

See Also Example Code

Visual Basic Syntax

Property HWSerialNum As Long

Returns

MG Return Code

Details

This property specifies the serial number of the hardware unit to be associated with an ActiveX control instance.

The serial number must be set in the *HWSerialNum* property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.

Every APT control unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.

For example, if two or more stepper hardware units are connected to the PC, different instances of the stepper ActiveX Control can be included in a client application. If each of these Control instances is programmed with a unique hardware serial number, then they will communicate with their associated hardware units. In this way, multiple graphical control panels communicating with multiple hardware units can easily be incorporated into a custom client application.

After a serial number has been allocated, an ActiveX control instance must be activated at run time by calling the StartCtrl method.

Note. The factory programmed serial number also has a model type identifier incorporated. This ensures for example that the serial number for an APT stepper motor controller cannot be assigned to an APT NanoTrak ActiveX control. If a serial number is set that is either illegal or for which no hardware unit exists, then the ActiveX control will remain inactive, i.e. the control's GUI will appear 'dead'.

Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

MG17Motor1.HWSerialNum = 20000001

' Start motor control
```

```
MG17Motor1.StartCtrl

End Sub
```

Introduction to NanoTrak Control Programming

The 'NanoTrak' ActiveX Control provides the functionality required for a client application to control one or more NanoTrak auto-alignment controller products. The NanoTrak system comes in both benchtop (BNT001), 19" rack modular (MNA601), and Miniature (TNA001) formats, all of which are covered by the NanoTrak ActiveX Control.
To specify the particular controller being addressed, every unit is factory programmed with a unique 8-digit serial number. This serial number is key to the operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus. The serial number must be allocated using the HWSerialNum property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.
The methods of the NanoTraks object can then be used to perform activities such as latching/unlatching, reading power levels, obtaining/setting circle size and position and determining if 'NanoTracking' is currently taking place.
For details on the use of the NanoTrak controller, refer to the manual for the unit, which is available from www.thorlabs.com.

NanoTrak Enumeration AMPFEEDBACKTYPE

Note. This enumeration is not applicable to issue 1 bench top units.

This enumeration contains constants that specify the type of feedback applied to the HV amplifier circuit when operating in closed loop mode.

Constant	Name	Purpose
1.	FEEDBACK_TYPE_DC	Feedback supplied by a.c. signals from the strain gauge in the piezo actuator
2.	FEEDBACK_TYPE_AC	Feedback supplied by a differential 0-10V d.c. signal

NanoTrak Enumeration APT_PARENT_HWTYPES

This enumeration contains constants that specify the hardware type of the enclosure associated with the ActiveX control.

Constant	Name	Purpose
29	ETHNET_MMR601	6 Bay Modular Rack, Ethernet
30	USB_MMR601	6 Bay Modular Rack, USB

NanoTrak Enumeration AUTORANGINGMODE

This enumeration contains constants that specify the way in which the Nanotrak changes range when autoranging.

Constant	Name	Purpose
1.	AUTORANGE_ALL	All ranges are visited
2.	AUTORANGE _ODD	Odd numbered ranges only are visited
3.	AUTORANGE _EVEN	Even numbered ranges only are visited

NanoTrak Enumeration BNCUNITSMODE

This enumeration contains constants that specify the display mode when using the BNC input.

Constant	Name	Purpose
1.	BNCUNITS_VOLTS	Readings displayed in Volts
2.	BNCUNITS_PERCENT	Readings displayed as a percentage of full input range
3.	BNCUNITS_USER	Readings displayed as a calibrated value according to user specified settings

NanoTrak Enumeration CIRCDIAMODE

The diameter of the scanning circle can be adjusted in two ways, each exclusive of the other. This enumeration contains

constants that specify the way in which the circle diameter is adjusted.

Constant	Name	Purpose
1.	CIRCDIA_USER	Diameter adjusted to the size specified by the user
3	CIRCDIA_LUT	Diameter adjusted via Look Up Table

NanoTrak Enumeration HWCHANNEL

.

This enumeration contains constants that specify individual channels on an APT hardware unit.

Constant	Name	Purpose
0	CHAN1_ID	Selects channel 1
1.	CHAN2_ID	Selects channel 2
10	CHANBOTH_ID	Selects both channels

Note. Some methods that contain an lChanID parameter do not accept CHANBOTH_ID as a valid value. In this case, an error value is returned.

NanoTrak Enumeration HWMODE

This enumeration contains constants that specify the required operating mode of the NanoTrak unit.

Constant	Name	Purpose
1.	HWMODE_PIEZO	Operate as Piezo Controller
2.	HWMODE_NANOTRAK	Operate as NanoTrak

NanoTrak Enumeration INPUTSOURCE

This enumeration contains constants that specify the input signal source.

Constant	Name	Purpose
1.	INPUT_TIA	Sets the I/P source to the 'OPTICAL IN' connector
2.	INPUT_BNC1V	Sets the I/P source to the 'SIG IN' connector 1V max
3.	INPUT_BNC2V	Sets the I/P source to the 'SIG IN' connector 2V max
4.	INPUT_BNC5V	Sets the I/P source to the 'SIG IN' connector 5V max
5.	INPUT_BNC10V	Sets the I/P source to the 'SIG IN' connector 10V max

NanoTrak Enumeration LOWPASSFILTER

This enumeration contains constants that specify the low pass filter cut off frequency applied to the output of the TIA circuit. It can be used to improve the signal to noise of readings on the graphical display.

Constant	Name	Purpose
0	LP_NONE	Low pass filter inactive
1.	LP_1HZ	Cut off all signals above 1Hz
2.	LP_3HZ	Cut off all signals above 3Hz
3.	LP_10HZ	Cut off all signals above 10Hz
4.	LP_30HZ	Cut off all signals above 30Hz
5.	LP_100HZ	Cut off all signals above 100Hz

NanoTrak Enumeration RANGINGMODE

This enumeration contains constants that specify the ranging mode.

Constant	Name	Purpose
1.	RANGING_AUTO	Automatic ranging

2.	RANGING_USER	User specified (manual) ranging
3.	RANGING_USER_SET	User specified (manual) ranging with the range set to the value specified in the SetRange method
4.	RANGING_AUTO_SET	Automatic ranging with the range set to the value specified in the SetRange method

NanoTrak Enumeration READRANGE

This enumeration contains constants that set the range of the internal amplifier.

Constant	Name	Purpose
3.	RANGE_3NANO	3 nA full scale
4.	RANGE_10NANO	10 nA full scale
5.	RANGE_30NANO	30 nA full scale
6.	RANGE_100NANO	100 nA full scale
7.	RANGE_300NANO	300 nA full scale
8.	RANGE_1MICRO	1. µA full scale
9	RANGE_3MICRO	3 µA full scale
10	RANGE_10MICRO	10 µA full scale
11	RANGE_30MICRO	30 µA full scale
12	RANGE_100MICRO	100 µA full scale
13	RANGE_300MICRO	300 µA full scale
14	RANGE_1MILLI	1. mA full scale
15	RANGE_3MILLI	3 mA full scale
16	RANGE_10MILLI	10 mA full scale

NanoTrak Enumeration OPCONTROLMODE

This enumeration contains constants that specify the control loop feedback mode.

Constant	Name	Purpose
1.	OPEN_LOOP	No feedback
2.	CLOSED_LOOP	Position feedback
3.	OPEN_LOOP_SMOOTH	No feedback. Transition from closed to open loop is achieved over a longer period in order to minimize voltage transients (spikes).
4.	CLOSED_LOOP_SMOOTH	Position feedback. Transition from closed to open loop is achieved over a longer period in order to minimize voltage transients (spikes).

NanoTrak Enumeration OUTPUTCURRENTLIMIT

Note. This enumeration is not applicable to issue 1 bench top units.

This enumeration contains constants that specify the maximum current output for the HV amplifier.

Constant	Name	Purpose
1.	CURRENTLIMIT_100MA	HV amplifier output limited to 100mA
2.	CURRENTLIMIT_250MA	HV amplifier output limited to 250mA
3.	CURRENTLIMIT_500MA	HV amplifier output limited to 500mA

NanoTrak Enumeration OUTPUTLPFILTER

Note. This enumeration is not applicable to issue 1 bench top units.

This enumeration contains constants that specify the cut off frequency of the Low Pass filter applied to the high voltage piezo drive output of the HV amplifier circuit.

Note. This setting is applied to both channels simultaneously.

Constant	Name	Purpose
1.	OUTPUTFILTER_10HZ	Cut off all signals above 10Hz
2.	OUTPUTFILTER_100HZ	Cut off all signals above 100Hz
3.	OUTPUTFILTER_5KHZ	Cut off all signals above 5KHz
4.	OUTPUTFILTER_NONE	Low pass filter inactive

NanoTrak Enumeration SIGADJTYPE

This enumeration contains constants that specify the way in which the circle diameter is adjusted.

Constant	Name	Purpose
1.	SIGADJ_LIN	Linear
2.	SIGADJ_LOG	Logarithmic
3.	SIGADJ_X2	X2
4.	SIGADJ_X3	X3

NanoTrak Enumeration TIAUNITSMODE

This enumeration contains constants that specify the display mode when using the PIN input.

Constant	Name	Purpose
1.	TIAUNITS_AMPS	Readings displayed in Amps
2.	TIAUNITS_Watts	Readings displayed in Watts
3.	TIAUNITS_DB	Readings displayed in dBs

NanoTrak Enumeration TNA_LVOUTPUTRANGE

The output signals from the NanoTrak T-Cube are routed to the piezo drivers to position the piezo actuators. Earlier piezo t-cubes accept a 5V input while later cubes accept a 10V input.

This enumeration contains constants that control the NanoTrak output signal range.

Constant	Name	Purpose
1.	TNA_LVOUTRANGE_5V	Signal range 0 to 5V
2.	TNA_LVOUTRANGE_10V	Signal range 0 to 10V

NanoTrak Enumeration TNA_LVOUTPUTROUTE

This enumeration contains constants that control way in which the NanoTrak output signals are routed to the associated piezo drivers.

Constant	Name	Purpose
1.	LVOUTROUTE_SMAONLY	Signals routed via the external SMA terminals
2.	LVOUTROUTE_SMAHUB	Signals can be routed via the external SMA terminals or via the Hub internal lines

NanoTrak Enumeration TRAKMODE

This enumeration contains constants that specify the tracking status.

Constant	Name	Purpose
1.	TRAK_LATCHED	Latched mode (tracking off)
2.	TRAK_TRACKING_SIG	Tracking on, threshold signal exceeded
3.	TRAK_TRACKING_NOSIG	Tracking on, threshold signal not achieved

NanoTrak Enumeration TRACKMODESET

This enumeration contains constants that specify the tracking mode.

Constant	Name	Purpose
1.	TRAK_CIRC	Sets a circular scan pattern, using the horizontal and vertical axes.
2.	TRAK_HORZ	Sets a horizontal line scan pattern.
3.	TRAK_VERT	Sets a vertical line scan pattern.

NanoTrak Enumeration UNDEROVERREAD

This enumeration contains constants that specify whether the reading is within range.

Constant	Name	Purpose
1.	WITHINRANGE	The reading is within range
2.	UNDERREAD	The reading is under range
3.	OVERREAD	The reading is over range

Piezo Events HWResponse

See Also Example Code

Visual Basic Syntax

Event **HWResponse**(lHWCode As Long, lMsgIdent As Long)

Parameters

lHWCode - Value identifying the hardware condition encountered

lMsgIdent - The internal identifier of the command that caused the problem condition to occur

Returns

MG Return Code

Details

This event is fired if the hardware unit encounters a fault, failure or warning condition. In normal operation the HWResponse event will not be fired. It is good programming practice to handle this event in case hardware problems occur.

NanoTrak Event SettingsChanged

See Also Example Code

Visual Basic Syntax

Event **SettingsChanged**(lFlags As Long)

Parameters

lFlags - for future use,

Returns

MG Return Code

Details

This event is fired by the system whenever the user has altered settings via the 'Settings' button on the user GUI interface. This is useful for informing the client application of settings changes.

The event is also fired when the following GUI panel controls are adjusted:

- Circle diameter pot
- Auto and Manual ranging mode buttons
- Track and Latch mode buttons.

Note. The *lFlags* parameter is for future use and currently resides only as a placeholder.

NanoTrak Method *DoEvents*

See Also Example Code

Visual Basic Syntax

Function **DoEvents**() As Long

Returns

MG Return Code

Details

This method enables the server to allow message processing to take place within a client application. For example, if the client application is processing a lengthy loop and is making multiple calls to the server, this can often cause the client application to become non-responsive because other user interface message handling, such as button clicks, cannot be processed. Calling the DoEvents method allows such client application message handling to take place.

Note. The DoEvents method works in the same way as the DoEvents keyword found in Visual Basic.

NanoTrak Method *EnableHWChannel*

See Also Example Code

Visual Basic Syntax

Function **EnableHWChannel**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This method enables the channel(s) specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

When the method is called, the piezo drive voltage, stored when the channel was disabled using the DisableHWChannel method, is reapplied and the front panel channel LED is lit.

To disable a particular channel, see the DisableHWChannel method.

NanoTrak Method *DeleteParamSet*

See Also Example Code

Visual Basic Syntax

Function **DeleteParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be deleted

Returns

MG Return Code

Details

This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed.

NanoTrak Method *GetAmpControlMode*

See Also Example Code

Visual Basic Syntax

Function **GetAmpControlMode**(lChanID As Long, plMode As Long) As Long

Parameters

lChanID - the channel identifier

plMode - returns the control loop feedback status

Returns

MG Return Code

Details

This method retrieves the control loop feedback status for the amplifier channel specified by the *lChanID* parameter, and returns a value in the *plMode* parameter, which in turn takes values from the OPCONTROLMODE enumeration.

If set to OPEN_LOOP, no feedback is employed.

If set to CLOSED_LOOP, position feedback is provided (usually by a strain gauge).

If set to OPEN_LOOP_SMOOTH or CLOSED_LOOP_SMOOTH the feedback status is the same as above however the transition from open to closed loop (or vise versa) is achieved over a longer period in order to minimize voltage transients (spikes).

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: The system cannot obtain the control mode of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

The Control Loop mode may be set using the SetAmpControlMode method

Note: It is advisable to always operate the control loops for both channels in open loop mode (the system default) when NanoTracking. Open loop mode provides the larger operating bandwidths required for circle operation.

NanoTrak Method *GetAmpFeedbackSig*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **GetAmpFeedbackSig** (lChanID as Long, plType As Long) As Long

Parameters

lChanID - the channel identifier

plType - the type of feedback signal selected.

Returns

MG Return Code

Details.

This method obtains the type of feedback signal used when operating the HV amplifier channels in closed loop operation. The feedback type is set using the SetAmpFeedbackSig method.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

All rack-mounted units (and bench top units at issue 2 or later) support two types of feedback signal, AC mode and DC mode. The feedback type is specified in the *lType* parameter which, in turn, takes values from the AMPFEEDBACKTYPE enumeration as follows:

FEEDBACK_TYPE_DC

FEEDBACK_TYPE_AC

The AC mode refers to the use of AC excited strain gauge feedback signals as generated by the complete range of our piezo actuators and piezo equipped multi axis stages. All versions of the APT NanoTrak electronics support this feedback mode. On later versions of the APT hardware it is possible to set the feedback signal type to DC. This configures the feedback circuit to accept a differential 0-10V DC feedback signal in order to control position output. This latter feedback mode is available for use with third party piezo positioning systems capable of generating a standard voltage position feedback signal.

Note. On later generation NanoTrak electronics (module and bench top), the function of the pin connections on the 9-way D-type feedback connector is dependent on the feedback mode selected. Refer to the pin-out tables in the associated handbook for details.

NanoTrak Method *GetAmpOutputParams*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **GetAmpOutputParams** (lChanID as Long, plCurrentLim As Long, plLPFilter As Long) As Long

Parameters

lChanID - the channel identifier

plCurrentLim - the maximum current output limit

plLPFilter –the value of the hardware low pass filter

Returns

MG Return Code

Details.

This method returns the output characteristics of the HV amplifier channels fitted to the NanoTrak unit. The characteristics are set using the SetAmpOutputParams method.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

The *lCurrentLim* parameter defines the maximum current output. It takes values from the the OUTPUTCURRENTLIMIT enumeration as follows:

CURRENTLIMIT_100MA

CURRENTLIMIT_250MA

CURRENTLIMIT_500MA

The *ILPFilter* parameter is the value of the hardware low pass filter applied to both of the HV amplifier output channels, and can be used to improve stability and reduce noise on the HV outputs. It is not channel specific and the *IChanID* parameter is ignored for this particular setting. It takes values from the OUTPUTLPFILTER enumeration as follows:

OUTPUTLPFILTER_10HZ

OUTPUTLPFILTER_100HZ

OUTPUTLPFILTER_5KHZ

OUTPUTLPFILTER_NONE

Note. This low pass filter setting is different to the low pass filter specified using the SetLPFilter method, which applies to the output signals from the internal TIA (PIN current amplifier) circuit and is used to improve the signal to noise of associated readings shown on the graphical panel.

The HV Amplifier parameter settings can be set by calling the SetAmpOutputParams method.

NanoTrak Method *GetCircDia*

See Also Example Code

Visual Basic Syntax

Function **GetCircDia**(pfDia As Single) As Long

Parameters

pfDia - the returned circle diameter value

Returns

MG Return Code

Details

This method obtains the user circle diameter setting and returns a value in the *pfDia* parameter.

The circle diameter is measured on a scale (NanoTrak units) such that 10 units is the width and height of the screen. If, for example, the NanoTrak is driving a 20 micron actuator, a circle diameter of 1 unit will result in a real diameter of 2 microns. The scanning circle diameter can have values in the range 0 to 5 with a value of 5 corresponding to one half the range of piezo motion (i.e. 10 microns).

Note. The scanning circle diameter can be set using the SetCircDia method.

NanoTrak Method *GetCircDiaLUTVal*

See Also Example Code

Visual Basic Syntax

Function **GetCircDiaLUTVal**(lRangeIndex As Long, pfDia As Single) As Long

Parameters

lRangeIndex - the NanoTrak input range

pfDia - the associated circle diameter value

Returns

MG Return Code

Details

When automatic look up table (LUT) diameter adjustment mode is enabled (using the SetCircDiaMode method), the system uses values in a LUT to modify circle diameter in relation to the input range currently selected.

This method obtains the LUT value in the *pfDia* parameter for the NanoTrak range specified in the *lRangeIndex* parameter.

To set a LUT value for a particular range, see the SetCircDiaLUTVal method.

NanoTrak Method *GetCircDiaMode*

See Also Example Code

Visual Basic Syntax

Function **GetCircDiaMode**(plMode As Long) As Long

Parameters

plMode - The returned circle diameter adjustment mode.

Returns

MG Return Code

Details

This method obtains the mode of circle diameter adjustment currently enabled and returns a value in the *plMode* parameter, which, in turn, takes values from the CIRCDIAMODE enumeration.

If CIRCDIA_USER is specified, the circle diameter remains at the value set using the SetCircDia method.

If CIRCDIA_LUT is specified, the circle diameter is adjusted automatically, using a table of range dependent diameter values (set using the SetCircDiaLUTVal method.

To set the mode of circle diameter adjustment, see the SetCircDiaMode method.

NanoTrak Method *GetCircFreq*

See Also Example Code

Visual Basic Syntax

Function **GetCircFreq**(pfFreq As Single) As Long

Parameters

pfFreq - The returned circle frequency (15 to 200 Hz).

Returns

MG Return Code

Details

This method obtains the circle scanning frequency and returns a value in the *pfFreq* parameter.

The circle scanning frequency lies in the range 15 to 200Hz. The factory default setting for the scanning frequency is 43.75Hz. This means that a stage driven by the NanoTrak makes 43.75 circular movements per second. Different frequency settings allow more than one NanoTrak to be used in the same alignment scenario. Refer to the NanoTrak Operating Guide for more information.

To obtain the current setting for the circle scanning frequency, see the SetCircFreq method.

NanoTrak Method *GetCircHomePos*

See Also Example Code

Visual Basic Syntax

Function **GetCircHomePos**(pfHorzHomePos As Single, pfVertHomePos As Single) As Long

Parameters

pfHorzHomePos - The returned horizontal coordinate of the circle home position (0 to 10 NT units).

pfVertHomePos - The returned vertical coordinate of the circle home position (0 to 10 NT units).

Returns

MG Return Code

Details

This method returns the circle home position horizontal and vertical coordinates in the *pfHorzHomePos* and *PfVertHomePos* parameters respectively.

Both position outputs can have values in the range 0.0 – 10.0 (NanoTrak units), with (5.0,5.0) representing the display's origin (screen center).

The home position is used when the MoveCircHome method is called.

To set the home position, see the SetCircHomePos method.

NanoTrak Method *GetCircPosReading*

See Also Example Code

Visual Basic Syntax

Function **GetCircPosReading**(pfHorzPos As Single, pfVertPos As Single, pfAbsReading As Single, plRange As Long, pfRelReading As Single, plUnderOverRead As Long) As Long

Parameters

pfHorzPos - Returns the horizontal position of the circle (0 to 10 NT units).

pfVertPos - Returns the vertical position of the circle (0 to 10 NT units).

pfAbsReading - Returns the absolute signal value at the current position, in units as displayed on the GUI panel.

plRange - Returns the input signal range currently selected (1 to 16).

pfRelReading - Returns the relative signal strength (0 to 100%) for the range currently selected.

plUnderOverRead - Identifies if the unit is under reading or over reading the input signal.

Returns

MG Return Code

Details

This method obtains the current horizontal and vertical position of the circle and returns values in the *pfHorzPos* and *PfVertPos* parameters respectively.

Both position outputs can have values in the range 0.0 – 10.0 (NanoTrak units), with (5.0,5.0) representing the display's origin (screen center).

The absolute signal value at the current position is returned in the *pfAbsReading* parameter, in units as displayed on the GUI panel.

The input signal range currently selected (1 to 16), is returned in the *plRange* parameter, which takes values from the READRANGE enumeration.

The relative signal strength at the current position is returned in the *pfRelReading* parameter, 0 to 100% of the range currently selected. This value matches the length of the input signal bargraph on the GUI panel. (e.g. if the 3MICRO range is currently

selected, then a pfRelReading value of 50% equates to 1.5 MICRO).

A value is returned in the *plUnderOverRead* parameter to identify whether the unit is under reading or over reading the input signal. This parameter takes values from the UNDEROVERREAD enumeration. (e.g. if a user specified range of 3MICRO is currently applied, the *plUnderOverRead* parameter returns 'OVERREAD' for input signals greater than 3MICROS).

NanoTrak Method *GetInputSrc*

See Also Example Code

Visual Basic Syntax

Function **GetInputSrc**(plSource As Long) As Long

Parameters

lSource - The returned input source.

Returns

MG Return Code

Details

This method obtains the input source currently selected and returns a value in the *lSource* parameter, which in turn takes values from the INPUTSOURCE enumeration.

The INPUT_BNC settings are used when NanoTraking to optimise a voltage feedback signal. Typically, these inputs are selected when using an external power meter which generates a voltage output.

Note. In this case the internal amplifier circuit is bypassed and autoranging functionality is not required. Furthermore, although tracking occurs as normal, the tracking indicator on the GUI panel is inoperative.

The INPUT_TIA setting is used when NanoTraking to optimise a PIN current feedback signal. This input source should be selected when using the rear panel Optical I/P connector or when an external detector head is connected to the optional SMB connector. This option uses the internal amplifier circuit and associated functionality (e.g. autoranging).

The input source can be set using the SetInputSrc method.

NanoTrak Method *GetLoopGain*

See Also Example Code

Visual Basic Syntax

Function **GetLoopGain**(plGain As Long) As Long

Parameters

plGain - The returned gain setting (100 to 10,000).

Returns

MG Return Code

Details

Note. This method is applicable only when operating in 'User' Gain mode.

The gain setting is returned in the *plGain* parameter and lies in the range 100 to 10,000.

The loop gain can be set using the SetLoopGain method.

NanoTrak Method *GetLPFilter*

See Also Example Code

Visual Basic Syntax

Function **GetLPFilter**(plFilter As Long) As Long

Parameters

plFilter - Returns the current setting of the LP filter.

Returns

MG Return Code

Details

This method obtains the cut off frequency of the digital low pass (LP) filter applied to output readings of the internal amplifier (TIA) circuitry. If the readings displayed or returned are unstable, the filter can be used to remove any unwanted high frequency components and improve input signal stability.

The setting is returned in *plFilter* parameter, which in turn, takes values from the LOWPASSFILTER enumeration.

The cut off frquency of the LP filter can be set by using the SetLPFilter method.

NanoTrak Method *GetMaxTravel*

See Also Example Code

Visual Basic Syntax

Function **GetMaxTravel**(lChanID As Long, plMaxTravel As Long) As Long

Parameters

lChanID - the channel identifier

plMaxTravel - returns the maximum piezo travel (in microns)

Returns

MG Return Code

Details

In the case of actuators with built in position sensing, the NanoTrak Control Unit can detect the range of travel of the actuator since this information is programmed in the electronic circuit inside the actuator.

This method retrieves the maximum travel for the piezo actuator associated with the channel specified by the *lChanID* parameter, and returns a value (in microns) in the *plMaxTravel* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: The system cannot obtain the maximum travel of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

Example

```
Private Sub cmdGetTravel_Click()

Dim sngMaxTravel As Single

MG17NanoTrak1.GetMaxTravel CHAN1_ID, sngMaxTravel

' inform user of piezo max travel

MsgBox "Maximum travel of piezo = " & sngMaxTravel

End Sub
```

NanoTrak Method *GetParentHWInfo*

See Also Example Code

Visual Basic Syntax

Function **GetParentHWInfo** (plHWSerialNum As Long, plHWType As Long) As Long

Parameters

plHWSerialNum– the serial number of the host enclosure

plHWType – the hardware type of the host enclosure

Returns

MG Return Code

Details

This method returns the serial number and hardware type identifier of the parent (or host) controller enclosure.

In certain APT products, individual daughter cards or modules are fitted to a common enclosure (e.g. MMR601 modular rack) which is programmed at the factory with a unique serial number. The *plHWSerialNum* parameter is used to retrieve this serial number and is useful when trying to establish which daughter cards are located within a specific enclosure.

The *plHWType* parameter returns values specified by the APT_PARENT_HWTYPES enumeration as follows:

29 ETHNET_MMR601 6 Bay Modular Rack, Ethernet

30 USB_MMR601 6 Bay Modular Rack, USB

NanoTrak Method *GetPhaseComp*

See Also Example Code

Visual Basic Syntax

Function **GetPhaseComp**(plHorzAngle As Long, plVertAngle as Long) As Long

Parameters

plHorzAngle - The phase compensation setting (-180° to +180°) for the horizontal component of the circle path

plVertAngle - The phase compensation setting (-180° to +180°) for the vertical component of the circle path

Returns

MG Return Code

Details

The feedback loop scenario in a typical NanoTrak application can involve the operation of various electronic and electromechanical components (e.g. power meters and piezo actuators) that could introduce phase shifts around the loop and thereby effect tracking efficiency and stability.. These phase shifts can be cancelled by setting the 'Phase Compensation' factors.

This method obtains the phase compensation setting for the horizontal and vertical components of the circle path. The value is returned in the *plHorzAngle* and *plVertAngle* parameters respectively and lies in the range -180.0 to 180.0 degrees.

The phase compensation can be set using the SetPhaseComp method.

Typically in both phase offsets will be set the same, although some electromechanical systems may exhibit different phase lags in the different components of travel and so require different values.

NanoTrak Method *GetRange*

See Also Example Code

Visual Basic Syntax

Function **GetRange**(plRange As Long) As Long

Parameters

plRange - The returned range setting.

Returns

MG Return Code

Details

The NanoTrak unit is equipped with an internal trans-impedance amplifier (TIS) circuit (and associated range/power level displays and control buttons in the GUI). This amplifier operates when an external input signal is connected to the Optical/PIN connector on the rear panel. There are 14 range settings (1 - 14) that can be used to select the best range to measure the input signal (displayed on the GUI panel relative input signal bar and display).

This method returns the current range setting of the internal amplifier in the *plRange* parameter (1 - 14), which takes values from the READRANGE enumeration.

The range can be set using the SetRange method.

The range mode can be set using the SetRangingMode method.

NanoTrak Method *GetRangingMode*

See Also Example Code

Visual Basic Syntax

Function **GetRangingMode**(plMode As Long, plAutoRangeMode As Long) As Long

Parameters

plMode - The returned ranging mode (Auto or User).

plAutoRangingMode - The returned setting for the way in which the range changes are performed.

Returns

MG Return Code

Details

This method obtains the current ranging mode in the *lMode* parameter, which in turn, takes values from the RANGINGMODE enumeration.

When the auto-ranging mode is selected, range changes occur whenever the relative input power signal reaches the upper or lower end of the currently set range.

The *lAutoRangingMode* parameter returns values from the AUTORANGINGMODE enumeration, and specifies how these range changes are implemented by the system.

If ALL_RANGES is selected, the unit visits all ranges when ranging between two input signal levels.

If ODD_RANGES (or EVEN_RANGES) is selected only those odd (or even) numbered ranges between the two input signals levels will be visited. These latter two modes are useful when large rapid input signal fluctuations are anticipated. This is because the number of ranges visited is halved to give a more rapid response.

The ranging mode can be set using the SetRangingMode method.

NanoTrak Method *GetRangingParams*

See Also Example Code

Visual Basic Syntax

Function **GetRangingParams** (plRangeUpLim As Long, plRangeDownLim As Long, plSettleSamples As Long) As Long

Parameters

plRangeUpLim - the signal threshold at which the NanoTrak increments to the next higher range

plRangeDownLim - the signal threshold at which the system decrements to the next lower range

plSettleSamples –the amount of averaging applied to the readings before autoranging

Returns

MG Return Code

Details.

This method returns the autoranging characteristics for the NanoTrak input amplifier circuit (Trans-impedance amplifier (TIA) circuit). The characteristics are set using the SetRangingParams method

When autoranging, the NanoTrak unit adjusts continually the TIA range as appropriate for the input signal level. When the relative signal on a particular range drifts below the limit set in the *lRangeDownLim* parameter, the NanoTrak unit decrements the range to the next lower setting. Similarly, when the relative signal rises above the limit specified in the *lRangeUpLim* parameter, the unit increments the range to the next higher setting. The relative signal is displayed on the NanoTrak GUI panel by a green horizontal bar, see NanoTrak GUI Panel.

The *lSettleSamples* parameter determines the amount of averaging applied to the signal before autoranging takes place. Higher SettleSamples values improve the signal to noise ratio when dealing with noisy feedback signals. However, higher SettleSamples values also slow down the autoranging response. In a particular application, the SettleSamples value should be adjusted to obtain the best autoranging response combined with a noise free signal.

NanoTrak Method *GetReading*

See Also Example Code

Visual Basic Syntax

Function **GetReading**(pfAbsReading As Single, plRange As Long, pfRelReading As Single, plUnderOverRead As Long) As Long

Parameters

pfAbsReading - Returns the absolute signal value at the current position, in units as displayed on the GUI panel.

plRange - Returns the input signal range currently selected (1 to 16).

pfRelReading - Returns the relative signal strength (0 to 100%) for the range currently selected.

plUnderOverRead - Identifies if the unit is under reading or over reading the input signal.

Returns

MG Return Code

Details

This method obtains the absolute signal value at the current and returns a value in the *pfAbsReading* parameter, in units as displayed on the GUI panel.

The input signal range currently selected (1 to 16), is returned in the *plRange* parameter, which takes values from the READRANGE enumeration.

The relative signal strength at the current position is returned in the *pfRelReading* parameter, 0 to 100% of the range currently selected. This value matches the length of the input signal bargraph on the GUI panel. (e.g. if the 3MICRO range is currently

selected, then a pfRelReading value of 50% equates to 1.5 MICRO).

A value is returned in the *plUnderOverRead* parameter to identify whether the unit is under reading or over reading the input signal. This parameter takes values from the UNDEROVERREAD enumeration. (e.g. if a user specified range of 3MICRO is currently applied, the *plUnderOverRead* parameter returns 'OVERREAD' for input signals greater than 3MICROS).

NanoTrak Method *GetTrackmode*

See Also Example Code

Visual Basic Syntax

Function **GetTrackMode**(plMode As Long) As Long

Parameters

plMode - The returned tracking status

Returns

MG Return Code

Details

This method obtains the tracking mode settings in the *plMode* parameter, which takes values from the TRAKMODE enumeration.

NanoTrak Method *GetTrackThreshold*

See Also Example Code

Visual Basic Syntax

Function **GetTrackThreshold** (pfThreshold As Single) As Long

Parameters

pfThreshold - the signal threshold above which tracking is likely

Returns

MG Return Code

Details

Note. This method is applicable only when the input source is set to TIA - see SetInputSrc method.

This method obtains the tracking threshold of the NanoTrak and returns a value in the *pfThreshold* parameter. The value is returned in Amps, and is dependent upon the application. Typically, the value is set to lie above the ‘noise floor’ of the particular physical arrangement. When the input signal level exceeds this value, the tracking LED is lit on the GUI panel. Note there is no guarantee that tracking is taking place if this threshold value is set inappropriately. E.g. if the tracking threshold is set to below the noise floor, then the GUI will show a lit tracking LED even though no tracking is taking place.

The threshold can be set by calling the SetTrackThreshold method.

NanoTrak Method *GetUnitsMode*

See Also Example Code

Visual Basic Syntax

Function **GetUnitsMode**(plTIAMode As Long, pfAmpPerWatt As Single, plBNCMode As Long, pfVoltCalib As Single) As Long

Parameters

plTIAMode - the display mode for a TIA input

pfAmpPerWatt - the conversion factor for Amps to Watts

plBNCMode - the display mode for a BNC input

pfVoltCalib - the voltage calibration factor.

Returns

MG Return Code

Details

This method obtains the setting for units relating to the input signal measured by the NanoTrak. The units displayed can be specified for both the TIA and BNC input sources (specified using the SetInputSrc method).

When using the TIA (PIN) input, the *lTIAMode* and *fAmpPerWatt* parameters are used to determine the input signal readings displayed on the GUI and returned using the GetReading and GetCircPosReading methods.

When using the BNC voltage inputs, the *lBNCMode* and *fVoltCalib* parameters are used in a similar way.

The *plTIAMode* parameter takes values from the TIAUNITSMODE enumeration, and is applicable only if the input source is set to TIA mode (see SetInputSrc method).

If TIAUNITS_AMPS is selected, readings are displayed or returned in mAmps, i.e. the raw PIN current input signal to the internal amplifier.If TIAUNITS_WATTS is selected, readings are displayed or returned in mWatts, using a Amp:Watt calibration factor specified in the *pfAmpPerWatt* parameter.

Note. This is not intended to be a calibrated power reading, but rather a convenient mechanism for converting current to power. The calibration factor will usually be a wavelength dependent property.

If TIAUNITS_DB is selected readings are displayed or returned in dBs. The signal is first converted to watts according to the calibration factor specified in the the *pfAmpPerWatt* parameter, and then into dBs. Consequently this conversion is also wavelength dependent.

The *plBNCMode* parameter specifies the units relating to the input signal measured by the NanoTrak, and is applicable only if a BNC input source has been specified (see SetInputSrc method). It takes values from the BNCUNITSMODE enumeration as follows:

If BNCUNITS_VOLTS is selected, readings are displayed or returned in Volts, i.e. the raw input signal to the BNC connector.

If BNCUNITS_PERCENT is selected, readings are displayed or returned as a percentage of full scale input, e.g. a 1V signal into a 2V input range BNC will display a value of 50%.

If BNCUNITS_USER is selected, readings are displayed or returned as a calibrated value using the *Voltage Calibration* parameter, e.g. if *Voltage Calibration* has been set to '20', a 1V signal into a 2V input range BNC will display a value of 10.

The units mode can be set using the SetUnitsMode method.

NanoTrak Method *Identify*

See Also Example Code

Visual Basic Syntax

Function **Identify**() As Long

Returns

MG Return Code

Details

This method allows the hardware unit associated with the ActiveX control instance to be identified. The associated unit is specified by the HWSerialNum property

When the method is called, the front panel LEDs on the relevant hardware unit will flash for a short period.

NanoTrak Method *Latch*

See Also Example Code

Visual Basic Syntax

Function **Latch**() As Long

Returns

MG Return Code

Details

This method sets the tracking mode to 'Latched'. Scanning is then disabled and the piezo drives are held at the present position.

Example

```
Private Sub cmdLatch_Click()

MG17NanoTrak1.Latch

End Sub
```

NanoTrak Method *NT_LLGetStatusBits*

See Also Example Code

Visual Basic Syntax
Function **LLGetStatusBits**((lChanID As Long, plStatusBits As Long) As Long

Parameters
lChanID – the channel identifier
plStatusBits – the 32-bit integer containing the status flags.

Returns
MG Return Code

Details
This low level method returns a number of status flags pertaining to the operation of the NanoTrak controller channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.
These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description
0x00000001	1	Tracking Status (1 - Tracking, 0 - Latched).
0x00000004	2	Tracking (1 - with signal, 0 - without signal
0x00000004	3	Tracking channel A only (1 - chan A only, 0 - both channels)
0x00000008	4	Tracking channel Bonly (1 - chan B only, 0 - both channels)
0x00000010	5	TIA ranging mode (1 - auto ranging, 0 - manual ranging
0x00000020	6	TIA under read (1 - under read, 0 - reading within current range)
0x00000040	7	TIA over read (1 - over read, 0 - reading within current range)
0x00010000	17	Channel A Connected (1 - connected, 0 - not connected)
0x00020000	18	Channel B Connected (1 - connected, 0 - not connected)
0x00040000	19	Channel A Enabled (1 - Enabled, 0 - not Enabled)
0x00080000	20	Channel B Enabled (1 - Enabled, 0 - not Enabled)

0x00100000 21	Position control mode - CH A (1 - Closed Loop, 2 - Open Loop)
0x00200000 22	Position control mode - CH B (1 - Closed Loop, 2 - Open Loop)

Note. Bits 21 to 28 (Digital Input States) are only applicable if the associated digital input is fitted to your controller – see the relevant handbook for more details

0x00100000	21	Digital input 1 state (1 - logic high, 0 - logic low).
0x00200000	22	Digital input 2 state (1 - logic high, 0 - logic low).
0x00400000	23	Digital input 3 state (1 - logic high, 0 - logic low).
0x00800000	24	Digital input 4 state (1 - logic high, 0 - logic low).
0x01000000	25	Digital input 5 state (1 - logic high, 0 - logic low).
0x02000000	26	Digital input 6 state (1 - logic high, 0 - logic low).
0x04000000	27	Digital input 7 state (1 - logic high, 0 - logic low).
0x08000000	28	Digital input 8 state (1 - logic high, 0 - logic low).
0x10000000	29	Power OK, (Mirrors POWER LED for Rack modules)
0x20000000	30	Active, (Mirrors ACTIVE LED for Rack modules)
0x40000000	31	Error, (Mirrors ERROR LED for Rack modules)

NanoTrak Method *LLSaveHWDefaults*

See Also Example Code

Visual Basic Syntax

Function **LLSaveHWDefaults**() As Long

Returns

MG Return Code

Details

This method allows the current settings of the operation parameters to be saved into the onboard 'Flash' memory of the hardware unit. These parameters then become the default settings when the unit is next powered up.

Note. These defaults are overridden when the APT server is booted and begins communication with the hardware unit.

There may be occasions when the hardware unit is operated in the absence of the APT server, in which case the parameter defaults saved using this method then apply.

Typically however, the APT units will be operated using the PC server program.

NanoTrak Method *LoadParamSet*

See Also Example Code

Visual Basic Syntax

Function **LoadParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be loaded

Returns

MG Return Code

Details

This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName

parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

NanoTrak Method *MoveCircHome*

See Also Example Code

Visual Basic Syntax

Function **MoveCircHome**() As Long

Returns

MG Return Code

Details

This method moves the circle to the 'Home' position as set by the *SetCircHomePos* method.

NanoTrak Method *SaveParamSet*

See Also Example Code

Visual Basic Syntax

Function **SaveParamSet**(bstrName As String) As Long

Parameters

bstrName – the name under which the parameter set is to be saved

Returns

MG Return Code

Details

This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved. Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

NanoTrak Method *SetAmpControlMode*

See Also Example Code

Visual Basic Syntax

Function **SetAmpControlMode**(lChanID As Long, lMode As Long) As Long

Parameters

lChanID - the channel identifier

lMode - sets the control loop status

Returns

MG Return Code

Details

When in closed-loop mode, position is maintained by a feedback signal from the piezo actuator. This is only possible when using actuators equipped with position sensing.

This method sets the control loop status for the amplifier channel specified by the *lChanID* parameter, to the value specified in the *lMode* parameter, which in turn takes values from the OPCONTROLMODE enumeration.

If set to OPEN_LOOP, no feedback is employed.

If set to CLOSED_LOOP, position feedback is provided (usually by a strain gauge).

If set to OPEN_LOOP_SMOOTH or CLOSED_LOOP_SMOOTH the feedback status is the same as above however the transition from open to closed loop (or vise versa) is achieved over a longer period in order to minimize voltage transients (spikes).

In closed loop mode the output channels are relying on a feedback signal (i.e. from a piezo actuator based strain gauge) in order to operate to a setpoint determined by the NanoTrak algorithm. This is not an advised mode of operation when Nanotraking, where it is more usual to operate the HV output channels in open loop mode.

The current Control Loop mode may be obtained using the GetAmpControlMode method.

NanoTrak Method *SetAmpFeedbackSig*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **SetAmpFeedbackSig** (lChanID as Long, lType As Long) As Long

Parameters

lChanID - the channel identifier

lType - the type of feedback signal selected.

Returns

MG Return Code

Details.

This method sets the type of feedback signal used when operating the HV amplifier channels in closed loop operation.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

All rack-mounted units (and bench top units at issue 2 or later) support two types of feedback signal, AC mode and DC mode. The feedback type is specified in the *lType* parameter which, in turn, takes values from the AMPFEEDBACKTYPE enumeration as follows:

FEEDBACK_TYPE_DC

FEEDBACK_TYPE_AC

The AC mode refers to the use of AC excited strain gauge feedback signals as generated by the complete range of our piezo actuators and piezo equipped multi axis stages. All versions of the APT NanoTrak electronics support this feedback mode. On later versions of the APT hardware it is possible to set the feedback signal type to DC. This configures the feedback circuit to accept a differential 0-10V DC feedback signal in order to control position output. This latter feedback mode is available for use with third party piezo positioning systems capable of generating a standard voltage position feedback signal.

Note. On later generation NanoTrak electronics (module and bench top), the function of the pin connections on the 9-way D-type feedback connector is dependent on the feedback mode selected. Refer to the pin-out tables in the associated handbook for details.

The feedback type can be obtained by calling the GetAmpFeedbackSig method.

NanoTrak Method *SetAmpOutputParams*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **SetAmpOutputParams** (lChanID as Long, lCurrentLim As Long, lLPFilter As Long) As Long

Parameters

lChanID - the channel identifier

lCurrentLim - the maximum current output limit

lLPFilter –the value of the hardware low pass filter

Returns

MG Return Code

Details.

This method sets the output characteristics of the HV amplifier channels fitted to the NanoTrak unit. Both low pass filtering and current control settings can be made to tailor the piezo output drive to a particular application.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

The *lCurrentLim* parameter sets the maximum current output and takes values from the OUTPUTCURRENTLIMIT enumeration as follows:

CURRENTLIMIT_100MA

CURRENTLIMIT_250MA

CURRENTLIMIT_500MA

The *lLPFilter* parameter sets the cut off frequency of the hardware low pass filter applied to both of the HV amplifier output channels, and can be used to improve stability and reduce noise on the HV outputs. It is not channel specific and the *lChanID* parameter is ignored for this particular setting. It takes values from the OUTPUTLPFILTER enumeration as follows:

OUTPUTLPFILTER_10HZ

OUTPUTLPFILTER_100HZ

OUTPUTLPFILTER_5KHZ

OUTPUTLPFILTER_NONE

Note. This low pass filter setting is different to the low pass filter specified using the SetLPFilter method, which applies to the output signals from the internal TIA (PIN current amplifier) circuit and is used to improve the signal to noise of associated readings shown on the graphical panel.

The existing HV Amplifier parameter settings can be obtained by calling the GetAmpOutputParams method.

NanoTrak Method *SetCircDia*

See Also Example Code

Visual Basic Syntax

Function **SetCircDia**(fDia As Single) As Long

Parameters

fDia - the circle diameter value

Returns

MG Return Code

Details

This method sets the circle to the value specified in the *fDia* parameter.

The circle diameter is measured on a scale (NanoTrak units) such that 10 units is the width and height of the screen. If, for example, the NanoTrak is driving a 20 micron actuator, a circle diameter of 1 unit will result in a real diameter of 2 microns. The scanning circle diameter can have values in the range 0 to 5 with a value of 5 corresponding to one half the range of piezo motion.

Note. If the NanoTrak is latched when this method is called, the circle diameter will not change. When the NanoTrak is set to Track mode, the circle diameter will change to the value set in this method.

The scanning circle diameter can be obtained using the GetCircDia method.

Example

```
Private Sub cmdSetCircleSize_Click()

Dim sngDia As Single

sngDia = CSng(Val(txtDia))

MG17NanoTrak1.SetCircDiaMode CIRCDIA_USER

MG17NanoTrak1.SetCircDia sngDia

End Sub
```

NanoTrak Method *SetCircDiaLUTVal*

See Also Example Code

Visual Basic Syntax

Function **SetCircDiaLUTVal**(lRangeIndex As Long, fDia As Single) As Long

Parameters

lRangeIndex - the NanoTrak input range

fDia - the associated circle diameter value (0 to 5 NT units)

Returns

MG Return Code

Details

This method enables a look up table (LUT) of circle diameter values (in NanoTrak units) to be specified as a function of input range.

When automatic LUT diameter adjustment mode is enabled (using the SetCircDiaMode method), the system uses values in this LUT to modify circle diameter in relation to the input range currently selected.

This LUT diameter adjustment mode allows appropriate circle diameters to be applied on an application specific basis.

The *lRangeIndex* parameter contains the input range of the NanoTrak (1 to 14) and the *fDia* parameter contains the associated circle diameter (0 to 5 NT units).

To obtain a LUT value for a particular range, see the GetCircDiaLUTVal method.

NanoTrak Method *SetCircDiaMode*

See Also Example Code

Visual Basic Syntax

Function **SetCircDiaMode**(lMode As Long) As Long

Parameters

lMode - The required mode of adjustment for the circle diameter.

Returns

MG Return Code

Details

This method allows the different modes of circle diameter adjustment to be enabled and disabled. The desired mode of adjustment is specified in the *lMode* parameter, which, in turn, takes values from the CIRCDIAMODE enumeration.

If CIRCDIA_USER is specified, the circle diameter remains at the value set using the SetCircDia method.

If CIRCDIA_LUT is specified, the circle diameter is adjusted automatically, using a table of range dependent diameter values (set using the SetCircDiaLUTVal method.

To obtain the mode of circle diameter adjustment, see the GetCircDiaMode method.

Example

```
Private Sub cmdSetCircleSize_Click()

Dim sngDia As Single

sngDia = CSng(Val(txtDia))

MG17NanoTrak1.SetCircDiaMode CIRCDIA_USER

MG17NanoTrak1.SetCircDia sngDia

End Sub
```

NanoTrak Method *SetCircFreq*

See Also Example Code

Visual Basic Syntax

Function **SetCircFreq**(fFreq As Single) As Long

Parameters

fFreq - The circle frequency (15 to 200 Hz).

Returns

MG Return Code

Details

This method sets the circle scanning frequency to the value specified in the *fFreq* parameter.

The circle scanning frequency lies in the range 15 to 200Hz. The factory default setting for the scanning frequency is 43.75Hz. This means that a stage driven by the NanoTrak makes 43.75 circular movements per second. Different frequency settings allow more than one NanoTrak to be used in the same alignment scenario. Refer to the NanoTrak Operating Guide for more information.

To obtain the current setting for the circle scanning frequency, see the GetCircFreq method.

NanoTrak Method *SetCircHomePos*

See Also Example Code

Visual Basic Syntax

Function **SetCircHomePos**(fHorzHomePos As Single, fVertHomePos As Single) As Long

Parameters

fHorzHomePos - The horizontal coordinate of the circle home position (0 to 10 NT units).

fVertHomePos - The vertical coordinate of the circle home position (0 to 10 NT units).

Returns

MG Return Code

Details

This method sets the circle home position to the horizontal and vertical coordinates specified in the *fHorzHomePos* and *fVertHomePos* parameters respectively.

Both position outputs can have values in the range 0.0 – 10.0 (NanoTrak units), with (5.0,5.0) representing the display's origin (screen center).

The home position is used when the MoveCircHome method is called.

To obtain the current setting for the home position, see the GetCircHomePos method.

Example

```
Private Sub cmdSetPosition_Click()

'***THIS DOESN'T WORK IN SIMULATOR MODE!!!***

Dim sngHorizPos As Single, sngVertPos As Single

sngHorizPos = CSng(Val(txtHorizPos))

sngVertPos = CSng(Val(txtVertPos))

MG17NanoTrak1.SetCircHomePos sngHorizPos, sngVertPos

MG17NanoTrak1.MoveCircHome

End Sub
```

NanoTrak Method *SetHWMode*

See Also Example Code

Visual Basic Syntax

Function **SetHWMode**(lMode As Long) As Long

Parameters

lMode - The hardware operating mode (Piezo controller or NanoTrak).

Returns

MG Return Code

Details

This method sets the operating mode of the NanoTrak unit.

The default mode of operation is NanoTrak, however the unit can be configured as a Piezo controller by setting the *lMode* parameter, which in turn accepts values from the HWMODE enumeration.

Note. The hardware unit must be rebooted before changes to operating mode can take effect.

Note. When the HW operating mode of a NanoTrak unit has been changed to Piezo operation, then the Piezo ActiveX control must be used to communicate with the unit. Use the same serial number as used on the NanoTrak control in order to establish communication with the unit.

Example

```
Private Sub cmdNTToPZMode_Click()

MG17NanoTrak1.SetHWMode HWMODE_PIEZO

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

MG17Piezo1.HWSerialNum = "21" & Right(CStr(MG17NanoTrak1.HWSerialNum), 6)

' Start Piezo control

MG17Piezo1.StartCtrl

End Sub
```

NanoTrak Method *SetInputSrc*

See Also Example Code

Visual Basic Syntax

Function **SetInputSrc**(lSource As Long) As Long

Parameters

lSource - The input source.

Returns

MG Return Code

Details

This method sets the input source of the NanoTrak to the option specified in the *lSource* parameter, which in turn takes values from the INPUTSOURCE enumeration.

The INPUT_BNC settings are used when NanoTraking to optimise a voltage feedback signal. Typically, these inputs are selected when an external power meter which generates a voltage output, is connected to the rear panel SIG IN connector.

Note. In this case the internal amplifier circuit is bypassed and the 'Range' bar on the GUI panel is switched off (autoranging functionality is not required). Furthermore, although tracking occurs as normal, the tracking indicator on the GUI panel is inoperative.

The INPUT_TIA setting is used when NanoTraking to optimise a PIN current feedback signal. The TIA (trans impedence amplifier) input source should be selected when using the rear panel OPTICAL/PIN I/P connector with either an integral detector, or an external detector head connected to the optional SMB adapter. This option uses the internal amplifier circuit and associated functionality (e.g. autoranging).

The current input source can be obtained using the GetInputSrc method.

NanoTrak Method *SetLoopGain*

See Also **Example Code**

Visual Basic Syntax

Function **SetLoopGain**(lGain As Long) As Long

Parameters

lGain - The gain setting (100 to 10,000).

Returns

MG Return Code

Details

Note. This method is applicable only when operating in 'User' Gain mode.

The gain is set to the value specified in the *lGain* parameter and lies in the range 100 to 10,000.

The loop gain can be obtained using the GetLoopGain method.

Example

```
Private Sub cmdSetGain_Click()

Dim lngGain As Long

lngGain = CLng(Val(txtGain))

MG17NanoTrak1.SetLoopGain lngGain

End Sub
```

NanoTrak Method *SetLPFilter*

See Also Example Code

Visual Basic Syntax

Function **SetLPFilter**(lFilter As Long) As Long

Parameters

lFilter - The upper threshold setting of the low pass filter.

Returns

MG Return Code

Details

This method specifies the cut off frequency of the digital low pass (LP) filter applied to output readings of the internal amplifier (TIA) circuitry. If the readings displayed or returned are unstable, this setting can be used to remove any unwanted high frequency components and improve input signal stability.

The setting is specified in the *lFilter* parameter, which in turn, takes values from the LOWPASSFILTER enumeration.

The current setting of the LP filter can be obtained by using the GetLPFilter method.

NanoTrak Method *SetPhaseComp*

See Also **Example Code**

Visual Basic Syntax

Function **SetPhaseComp**(lHorzAngle As Long, lVertAngle as Long) As Long

Parameters

lHorzAngle - The phase compensation setting (-180° to +180°) for the horizontal component of the circle path

lVertAngle - The phase compensation setting (-180° to +180°) for the vertical component of the circle path

Returns

MG Return Code

Details

The feedback loop scenario in a typical NanoTrak application can involve the operation of various electronic and electromechanical components (e.g. power meters and piezo actuators) that could introduce phase shifts around the loop and thereby effect tracking efficiency and stability. These phase shifts can be cancelled by setting the 'Phase Compensation' factors.

This method sets the phase compensation for the horizontal and vertical components of the circle path, to the value specified in the *lHorzAngle* and *lVertAngle* parameters respectively. The value is specified in degrees and lies in the range -180.0 to 180.0 degrees.

Typically in both phase offsets will be set the same, although some electromechanical systems may exhibit different phase lags in the different components of travel and so require different values.

The phase compensation can be obtained using the GetPhaseComp method.

Example

```
Private Sub cmdSetPhase_Click()

Dim lngHorizAng As Long, lngVertAng As Long

lngHorizAng = CSng(Val(txtHorizAng))

lngVertAng = CSng(Val(txtVertAng))

MG17NanoTrak1.SetPhaseComp lngHorizAng, lngVertAng

End Sub
```

NanoTrak Method *SetPowerCaptureParams*

See Also Example Code

Visual Basic Syntax

Function **SetPowerCaptureParams** (bstrFileName As String, lNumSamples As Long, lSampleInterval As Long) As Long

Parameters

bstrFileName - the name and location of the file in which to save the power samples

lNumSamples - the number of samples to be taken

lSampleInterval –for future use

Returns

MG Return Code

Details.

This method is used to store the power readings measured by the NanoTrak. The number of samples to be taken is specified in the *lNumSamples* parameter. The name and location of the file in which to store the data is specified in the *bstrFilename*

parameter, (e.g.C:\LogFiles\Log1.txt).

The sample rate is set in the *ISampleInterval* parameter. Currently, this is locked at 100ms and cannot be adjusted.

The data is stored data in units as selected in the GUI panel (see NanoTrak GUI Panel. for more details) or via the SetUnitsMode method.

Consider the example power log shown below.

Count The sample number.

Time The elapsed time (in seconds).

Abs Pwr The absolute signal strength, as shown on the NanoTrak GUI display.

Range The NanoTrak range associated with the Rel Pwr column.

Rel Pwr The relative signal strength displayed on the NanoTrak GUI (32767 full scale).

NanoTrak power log generated 13:55:21 Tuesday, January 03, 2006

Count Time Abs Pwr Range Rel Pwr

1. 0.1 0.000000 3 0
2. 0.2 0.000000 3 812
3. 0.3 0.000000 3 0
4. 0.4 0.000000 3 29
5. 0.5 0.000000 3 420
6. 0.6 0.000010 4 31235
7. 0.7 0.000033 5 32767
8. 0.8 0.000110 6 32767
9. 0.9 0.000330 7 32767
10. 1.0 0.001100 8 32767
11. 1.1 0.000280 9 2785
12. 1.2 0.000190 9 1889
13. 1.3 0.000103 9 1020
14. 1.4 0.000070 9 697
15. 1.5 0.000014 8 409
16. 1.6 0.000330 7 32767
17. 1.7 0.001100 8 32767
18. 1.8 0.003300 9 32767
19. 1.9 0.000854 10 2545
20. 2.0 0.000579 10 1724

NanoTrak Method *SetRange*

See Also Example Code

Visual Basic Syntax

Function **SetRange**(lRange As Long) As Long

Parameters

lRange - The range setting.

Returns

MG Return Code

Details

The NanoTrak unit is equipped with an internal trans-impedance amplifier (TIS) circuit (and associated range/power level displays and control buttons in the GUI). This amplifier operates when an external input signal is connected to the Optical/PIN connector on the rear panel. There are 14 range settings (1 - 14) that can be used to select the best range to measure the input signal (displayed on the GUI panel relative input signal bar and display).

This method sets the range of the internal amplifier to the value specified in the *lRange* parameter (1 - 14), which takes values from the READRANGE enumeration.

The range can be obtained using the GetRange method.

The range mode can be obtained using the GetRangingMode method.

NanoTrak Method *SetRangingMode*

See Also **Example Code**

Visual Basic Syntax

Function **SetRangingMode**(lMode As Long, lAutoRangeMode As Long) As Long

Parameters

lMode - The ranging mode (Auto or User).

lAutoRangingMode - The way in which the range changes are performed.

Returns

MG Return Code

Details

This method sets the ranging mode to the option set in the *lMode* parameter, which in turn, takes values from the RANGINGMODE enumeration as follows:

RANGING_AUTO changes to Auto ranging at the range currently selected.

RANGING_USER changes to manual ranging at the range currently selected.

RANGING_USER_SET changes to manual ranging at the range set in the SetRange method (or the 'Settings' panel).

RANGING_AUTO_SET changes to Auto ranging at the range set in the SetRange method (or the 'Settings' panel).

When auto-ranging mode is selected, range changes occur whenever the relative input power signal reaches the upper or lower end of the currently set range.

The *lAutoRangingMode* parameter accepts values from the AUTORANGINGMODE enumeration, and specifies how these range changes are implemented by the system.

If ALL_RANGES is selected, the unit visits all ranges when ranging between two input signal levels.

If ODD_RANGES (or EVEN_RANGES) is selected, only those odd (or even) numbered ranges between the two input signals levels will be visited. These latter two modes are useful when large rapid input signal fluctuations are anticipated. This is

because the number of ranges visited is halved to give a more rapid response.

The current ranging mode setting can be obtained using the GetRangingMode method.

Example

```
Private Sub cmdSetRangeMode_Click()

' Autorange through even ranges only

MG17NanoTrak1.SetRangingMode RANGING_AUTO, AUTORANGE_EVEN

End Sub

Private Sub cmdTrack_Click()

MG17NanoTrak1.Track

End Sub
```

NanoTrak Method *SetRangingParams*

See Also Example Code

Visual Basic Syntax

Function **SetRangingParams** (lRangeUpLim As Long, lRangeDownLim As Long, lSettleSamples As Long) As Long

Parameters

lRangeUpLim - the signal threshold at which the NanoTrak increments to the next higher range

lRangeDownLim - the signal threshold at which the system decrements to the next lower range

lSettleSamples –the amount of averaging applied to the readings before autoranging

Returns

MG Return Code

Details.

This method is used to modify the autoranging characteristics of the NanoTrak input amplifier circuit (Trans-impedance amplifier (TIA) circuit).

When autoranging, the NanoTrak unit adjusts continually the TIA range as appropriate for the input signal level. When the relative signal on a particular range drifts below the limit set in the *lRangeDownLim* parameter, the NanoTrak unit decrements the range to the next lower setting. Similarly, when the relative signal rises above the limit specified in the *lRangeUpLim* parameter, the unit increments the range to the next higher setting. The relative signal is displayed on the NanoTrak GUI panel by a green horizontal bar, see NanoTrak GUI Panel.

The *lSettleSamples* parameter determines the amount of averaging applied to the signal before autoranging takes place. Higher SettleSamples values improve the signal to noise ratio when dealing with noisy feedback signals. However, higher SettleSamples values also slow down the autoranging response. In a particular application, the SettleSamples value should be adjusted to obtain the best autoranging response combined with a noise free signal.

Values can be set from ‘2’ to ‘32’, with a default setting value of ‘4’.

Current Ranging Parameter settings can be obtained by calling the GetRangingParams method.

NanoTrak Method *SetTrackThreshold*

See Also Example Code

Visual Basic Syntax

Function **SetTrackThreshold** (fThreshold As Single) As Long

Parameters

fThreshold - the signal threshold above which tracking is likely

Returns

MG Return Code

Details

This method sets the tracking threshold of the NanoTrak to the value set in the *fThreshold* parameter. The value is set in Amps, and is dependent upon the application. Typically, the value is set to lie above the 'noise floor' of the particular physical arrangement. When the input signal level exceeds this value, the tracking LED is lit on the GUI panel. Note there is no guarantee that tracking is taking place if this threshold value is set inappropriately. E.g. if the tracking threshold is set to below the noise floor, then the GUI will show a lit tracking LED even though no tracking is taking place.

NanoTrak Method *SetUnitsMode*

See Also Example Code

Visual Basic Syntax

Function **SetUnitsMode**(lTIAMode As Long, fAmpPerWatt As Single, lBNCMode As Long, fVoltCalib As Single) As Long

Parameters

lTIAMode - the display mode for a TIA input

fAmpPerWatt - the conversion factor for Amps to Watts

lBNCMode - the display mode for a BNC input

fVoltCalib - the conversion factor for Volts to relative scale

Returns

MG Return Code

Details

This method specifies the units relating to the input signal measured by the NanoTrak. The units displayed can be specified for both the TIA and BNC input sources (specified using the SetInputSrc method).

When using the TIA (PIN) input, the *lTIAMode* and *fAmpPerWatt* parameters are used to determine the input signal readings displayed and returned using the GetReading and GetCircPosReading methods.

When using the BNC voltage inputs, the *lBNCMode* and *fVoltCalib* parameters are used in a similar way.

The *lTIAMode* parameter is specified using the TIAUNITSMODE enumeration, and is applicable only if the input source is set to TIA mode (see SetInputSrc method).

If TIAUNITS_AMPS is selected, readings are displayed or returned in mAmps, i.e. the raw PIN current input signal to the internal amplifier.If TIAUNITS_WATTS is selected, readings are displayed or returned in mWatts, using a Amp:Watt calibration factor specified in the *pfAmpPerWatt* parameter.

Note. This is not intended to be a calibrated power reading, but rather a convenient mechanism for converting current to power. The calibration factor will usually be a wavelength dependent property.

If TIAUNITS_DB is selected readings are displayed or returned in dBs. The signal is first converted to watts according to the calibration factor specified in the the *pfAmpPerWatt* parameter, and then into dBs. Consequently this conversion is also wavelength dependent.

The *lBNCMode* parameter is specified using the BNCUNITSMODE enumeration and is applicable only if the input source is set to a BNC mode (see SetInputSrc method).

If BNCUNITS_VOLTS is selected, readings are displayed or returned in Volts, i.e. the raw input signal to the BNC connector.

If BNCUNITS_PERCENT is selected, readings are displayed or returned as a percentage of full scale input, e.g. a 1V signal into a 2V input range BNC will display a value of 50%.

If BNCUNITS_USER is selected, readings are displayed or returned as a calibrated value using the *Voltage Calibration* parameter, e.g. if *Voltage Calibration* has been set to '20', a 1V signal into a 2V input range BNC will display a value of 10.

The current units mode setting can be obtained using the GetUnitsMode method.

NanoTrak Method *StartCtrl*

See Also Example Code

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns

MG Return Code

Details

This method is used in conjunction with the HWSerialNum property, to activate an ActiveX control instance at runtime.

After the HWSerialNum property has been set to the required serial number (at design time or run time), this method can be called at run time to initiate the hardware enumeration sequence and activate the control.

Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

MG17NanoTrak1.HWSerialNum = 22000001

' Start NanoTrak control

MG17NanoTrak1.StartCtrl

End Sub
```

NanoTrak Method *StartPowerCapture*

See Also Example Code

Visual Basic Syntax

Function **StartPowerCapture** As Long

Returns

MG Return Code

Details.

This method is used to start the power capture process specified by the SetPowerCaptureParams method.

To stop the capture process prematurely (i.e. before the number of samples entered in the *INumSamples* parameter have been taken), see the StopPowerCapture method.

NanoTrak Method *StopCtrl*

See Also Example Code

Visual Basic Syntax

Function **StopCtrl**() As Long

Returns

MG Return Code

Details

This method is called to deactivate the operation of an ActiveX control instance.

When this method is called, communication to the associated hardware unit is stopped. It is efficient programming practice to call StopCtrl on any ActiveX control instances that are no longer required to be active. For example, this may be the case in a multiple window client application where multiple instances of an ActiveX control need to communicate at different times with the same hardware unit

Example

```
Private Sub Form_Unload(Cancel As Integer)

' Stop system control and unload forms.

frmSystem.MG17System1.StopCtrl

Unload frmSystem

MG17NanoTrak1.StopCtrl

Unload Me

End Sub
```

NanoTrak Method *StopPowerCapture*

See Also Example Code

Visual Basic Syntax

Function **StopPowerCapture** As Long

Returns

MG Return Code

Details.

This method is used to stop the power capture process specified by the SetPowerCaptureParams method.

To start the capture process, see the StartPowerCapture method.

NanoTrak Method *TNA_GetDispIntensity*

See Also Example Code

Note. This method is applicable only to T-Cube NanoTrak controllers.

Visual Basic Syntax

Function **GetDispIntensity**(lChanID As Long, plIntensity As Single) As Long

Parameters

lChanID - the channel identifier

plIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method obtains the setting for the intensity of the LED display on the front of the unit. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The intensity is returned in the *plIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The display intensity can be set by calling the TNA_SetDispIntensity method

NanoTrak Method *TNA_GetSigOutputParams*

See Also Example Code

Visual Basic Syntax

Function **TNA_GetSigOutputParams** (plVoltRange As Long, plSigRouting As Long) As Long

Parameters

plVoltRange - the voltage range of the NanoTrak output signals (piezo drive signals) routed to the rear panel SMA connectors or via the T-Cube controller hub (TCH001)

plSigRouting - the way in which the NanoTrak output signals (piezo drive signals) are routed to the piezo drivers

Returns

MG Return Code

Details.

The output signals from the NanoTrak T-Cube are routed to the piezo drivers to position the piezo actuators. Earlier piezo T-cubes accept a 5V input while later cubes accept a 10V input. Other piezo amplifiers with 5V or 10V input ranges may be driven from the NanoTrak T-Cube.

This method returns parameter values which control the NanoTrak output signal ranges and the way in which these signals are routed to the associated piezo drivers.

The voltage range of these signals is returned in the '*plVoltRange'* parameter, which in turn, takes values the TNA_LVOUTRANGE enumeration as follows:

1. TNA_LVOUTRANGE_5V Signal range 0 to 5V
2. TNA_LVOUTRANGE_10V Signal range 0 to 10V

The signal routing is returned in the *plSigRouting* parameter which in turn, takes values from the TNA_LVOUTPUTROUTE parameter as follows:

1. LVOUTROUTE_SMAONLY Signals routed via the external

SMA terminals

2. LVOUTROUTE_SMAHUB Signals can be routed via the

external SMA terminals and

the Hub internal lines

The parameter values can be set by calling the SetSigOutputParams method.

NanoTrak Method *TNA_SetDispIntensity*

See Also Example Code

Note. This method is applicable only to T-Cube NanoTrak controllers.

Visual Basic Syntax

Function **SetDispIntensity**(lChanID As Long, lIntensity As Single) As Long

Parameters

lChanID - the channel identifier

lIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method sets the intensity of the LED display on the front of the unit. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The current setting for display intensity may be obtained by calling the TNA_GetDispIntensity method

NanoTrak Method *TNA_SetSigOutputParams*

See Also Example Code

Visual Basic Syntax

Function **SetSigOutputParams** (lVoltRange As Long, lSigRouting As Long) As Long

Parameters

lVoltRange - the voltage range of the NanoTrak output signals (piezo drive signals) routed to the rear panel SMA connectors or via the T-Cube controller hub (TCH001)

lSigRouting - the way in which the NanoTrak output signals (piezo drive signals) are routed to the piezo drivers

Returns

MG Return Code

Details.

The output signals from the NanoTrak T-Cube are routed to the piezo drivers to position the piezo actuators. Earlier piezo T-cubes accept a 5V input while later cubes accept a 10V input. Other piezo amplifiers with 5V or 10V input ranges may be driven from the NanoTrak T-Cube.

This method is used to set parameters which control the NanoTrak output signal ranges and the way in which these signals are routed to the associated piezo drivers.

The voltage range of these signals is set in the '*lVoltRange'* parameter, which in turn, takes values from the TNA_LVOUTRANGE enumeration as follows:

1. TNA_LVOUTRANGE_5V Signal range 0 to 5V
2. TNA_LVOUTRANGE_10V Signal range 0 to 10V

The signal routing is set in the *lSigRouting* parameter which in turn, takes values from the TNA_LVOUTPUTROUTE parameter as follows:

1. LVOUTROUTE_SMAONLY Signals routed via the external

SMA terminals

2. LVOUTROUTE_SMAHUB Signals can be routed via the

external SMA terminals and

the Hub internal lines

Current Parameter settings can be obtained by calling the GetSigOutputParams method.

NanoTrak Method *Track*

See Also Example Code

Visual Basic Syntax

Function **Track**() As Long

Returns

MG Return Code

Details

This method sets the tracking mode to 'Track'. During Track mode, the NanoTrak detects any drop in signal strength resulting from misalignment of the input and output devices, and makes positional adjustments to maintain the maximum.

NanoTrak Method *TrackEx*

See Also Example Code

Visual Basic Syntax

Function **TrackEx**(lMode As Long) As Long

Parameters

lMode - The tracking mode (circular, horizontal or vertical).

Returns

MG Return Code

Details

This method sets the tracking mode to the option set in the *lMode* parameter, which in turn, takes values from the TRACKMODESET enumeration as follows:

TRAK_CIRC sets a circular scan pattern, using both horizontal and vertical axes.

TRAK_HORZ sets a horizontal line scan pattern.

TRAK_VERT sets a vertical line scan pattern.

NanoTrak Property APTHelp

See Also Example Code

Visual Basic Syntax

Property APTHelp As Boolean

Returns

MG Return Code

Details

This property specifies the help file that will be accessed when the user presses the F1 key. If APTHelp is set to 'True', the main server helpfile will be launched. If APTHelp is set to 'False', the helpfile is the responsibility of the application programmer.

NanoTrak Property HWSerialNum

See Also Example Code

Visual Basic Syntax

Property HWSerialNum As Long

Returns

MG Return Code

Details

This property specifies the serial number of the hardware unit to be associated with an ActiveX control instance.

The serial number must be set in the *HWSerialNum* property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.

Every APT control unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.

For example, if two or more stepper hardware units are connected to the PC, different instances of the stepper ActiveX Control can be included in a client application . If each of these Control instances is programmed with a unique hardware serial number, then they will communicate with their associated hardware units . In this way, multiple graphical control panels communicating with multiple hardware units can easily be incorporated into a custom client application.

After a serial number has been allocated, an ActiveX control instance must be activated at run time by calling the StartCtrl method.

Note. The factory programmed serial number also has a model type identifier incorporated. This ensures for example that the serial number for an APT stepper motor controller cannot be assigned to an APT NanoTrak ActiveX control. If a serial number is set that is either illegal or for which no hardware unit exists, then the ActiveX control will remain inactive, i.e. the control's GUI will appear 'dead'.

Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)
```

```
MG17NanoTrak1.HWSerialNum = 22000001

' Start NanoTrak control

MG17NanoTrak1.StartCtrl

End Sub
```

Introduction to Piezo Control Programming

The 'Piezo' ActiveX Control provides the functionality required for a client application to control one or more of the APT series of piezo controller units. This range of controllers covers both open and closed loop piezo control in a variety of formats including compact Cube type controllers, benchtop units and 19" rack based modular drivers.

The Strain Gauge reader units (KSG101 and TSG001) are also accessed using the Piezo ActiveX Control.

A single Piezo ActiveX Control instance can be used to control any of the aforementioned products. To specify the particular controller being addressed, every unit is factory programmed with a unique 8-digit serial number. This serial number is key to the operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus. The serial number must be specified using the HWSerialNum property before an ActiveX control instance can communicate with the hardware unit. This can be done at design time or at run time. Note that the appearance of the ActiveX Control GUI (graphical user interface) will change to the required format when the serial number has been entered.
The Methods and Properties of the Piezo ActiveX Control can be used to perform activities such as selecting output voltages, reading the strain gauge position feedback, operating open and closed loop modes and enabling force sensing mode. With a few exceptions, these methods are generic and apply equally to both single and dual channel units.
Where applicable, the target channel is identified in the *IChanID* parameter and on single channel units, this must be set to CHAN1_ID. On dual channel units, this can be set to CHAN1_ID, CHAN2_ID or CHANBOTH_ID as required. See the HWCHANNEL enumeration for more details.
For details on the operation of the piezo controller, refer to the manual for the unit, which is available from www.thorlabs.com.

Piezo Control Method Summary

Method Name	Description	Applicable APT Controllers (model part number 3 letter prefix)
DeleteParamSet	Deletes stored settings for specific controller.	BPC, MPZ, TPZ
DisableHWChannel	Disables the drive output.	BPC, MPZ, TPZ
DoEvents	Allows client application to process other activity.	BPC, MPZ, TPZ, TSG
EnableHWChannel	Enables the drive output.	BPC, MPZ, TPZ
GetAmpFeedbackSig	Gets the feedback signal type (AC or DC).	BPC(001/2 Series, Rev 2), MPZ
GetAmpOutputParams	Gets the HV amplifier output parameters.	BPC(001/2 Series), MPZ
GetControlMode	Gets the loop operating mode (open/closed).	BPC, MPZ, TPZ
GetDispIntensity	Gets the front panel LED display intensity.	TPZ, TSG
GetForceSenseParams	Gets the force sensing mode parameters.	BPC, MPZ
GetHubAnalogueChanIn	Gets the hub analogue input channel used for close loop operation.	TPZ
GetIPSource	Gets the HV amplifier input source.	BPC, MPZ, TPZ
GetJogStepSize	Gets the jogging step size.	BPC, MPZ, TPZ
GetMaxTravel	Gets the maximum travel of a strain gauge equipped piezo actuator	BPC, MPZ
GetOutputLUTParams	Gets the output voltage waveform (LUT) operating parameters.	BPC, MPZ, TPZ
GetOutputLUTTrigParams	Gets the output voltage waveform (LUT) triggering parameters.	BPC, MPZ, TPZ
GetOutputLUTValue	Gets a specific voltage output value in the voltage waveform (LUT) table.	BPC, MPZ, TPZ
GetParentHWInfo	Gets the identification information of the host controller.	BPC(101/2/3 Series), MPZ
GetPIConsts	Gets the closed loop operating (proportional, integration) parameters.	BPC(001/2 Series), MPZ
GetPosOutput	Gets the piezo actuator extension in closed loop mode.	BPC, MPZ, TPZ, TSG
GetVoltageOutputLimit	Gets the HV output voltage maximum limit.	TPZ
GetVoltOutput	Gets the HV output voltage.	BPC, MPZ, TPZ
GetVoltPosDispMode	Gets the GUI display mode (voltage or position).	BPC, MPZ
Identify	Identifies the controller by flashing unit LEDs.	BPC, MPZ, TPZ, TSG
LLGetDigIPs	Gets digital input states encoded in 32 bit integer.	BPC, MPZ
LLGetStatusBits	Gets the controller status bits encoded in 32 bit integer.	BPC, MPZ, TPZ
LLSetGetDigOPs	Sets or Gets the user digital output bits encoded in 32 bit integer.	BPC, MPZ

LoadParamSet	Loads stored settings for specific controller.	BPC, MPZ, TPZ
SaveParamSet	Saves settings for a specific controller.	BPC, MPZ, TPZ
SetAmpFeedbackSig	Sets the feedback signal type (AC or DC).	BPC, MPZ
SetAmpOutputParams	Sets the HV amplifier output parameters.	BPC(001/2 Series), MPZ
SetControlMode	Sets the loop operating mode (open/closed).	BPC, MPZ, TPZ
SetDispIntensity	Sets the front panel LED display intensity.	TPZ, TSG
SetForceSenseParams	Sets the force sensing mode parameters.	BPC, MPZ
SetHubAnalogueChanIn	Sets the hub analogue input channel used for close loop operation.	TPZ
SetHWMode	Sets Piezo Amp or NanoTrak operating mode.	BPC, MPZ
SetIPSource	Sets the HV amplifier input source.	BPC, MPZ, TPZ
SetJogStepSize	Sets the jogging step size.	BPC, MPZ, TPZ
SetLVTrigBNCMode	Sets the operating mode of the rear panel triggering/monitor BNC's.	BPC(001/2 Series, Rev 2)
SetOutputLUTParams	Sets the output voltage waveform (LUT) operating parameters.	BPC, MPZ, TPZ
SetOutputLUTTrigParams	Sets the output voltage waveform (LUT) triggering parameters.	BPC, MPZ, TPZ
SetOutputLUTValue	Sets a specific voltage output value in the voltage waveform (LUT) table.	BPC, MPZ, TPZ
SetPIConsts	Sets the closed loop operating (proportional, integration) parameters.	BPC(001/2 Series), MPZ
SetPosOutput	Sets the piezo actuator extension in closed loop mode.	BPC, MPZ, TPZ, TSG
SetVoltageOutputLimit	Sets the HV output voltage maximum limit.	TPZ
SetVoltOutput	Sets the HV output voltage.	BPC, MPZ, TPZ
SetVoltPosDispMode	Sets the GUI display mode (voltage or position).	BPC, MPZ
SG_DeleteParamSet	Deletes stored settings for specific controller.	TSG
SG_GetDispIntensity	Gets the front panel LED display intensity.	TSG
SG_GetDispMode	Gets the front panel reading display mode.	TSG
SG_GetForceCalib	Gets the force calibration factor.	TSG
SG_GetHubAnalogueChanOut	Gets the hub analogue output channel used for close loop operation.	TSG
SG_GetMaxTravel	Gets the maximum travel of a strain gauge equipped device	TSG
SG_GetReading	Gets the strain gauge reading.	TSG
SG_LoadParamSet	Loads stored settings for specific controller.	TSG
SG_SaveParamSet	Saves settings for a specific controller.	TSG
SG_SetDispIntensity	Sets the front panel LED display intensity.	TSG
SG_SetDispMode	Sets the front panel reading display mode.	TSG
SG_SetForceCalib	Sets the force calibration factor.	TSG
SG_SetHubAnalogueChanOut	Sets the hub analogue output channel used for close loop operation.	TSG
SG_ZeroPosition	Nulls the strain gauge reading to take out offset errors.	TSG
StartCtrl	Starts the ActiveX Control (starts communication with controller)	BPC, MPZ, TPZ, TSG
StartOutputLUT	Starts outputting the voltage waveform (LUT).	BPC, MPZ, TPZ
StopCtrl	Stops the ActiveX Control (stops communication with controller)	BPC, MPZ, TPZ, TSG
StopOutputLUT	Stops outputting the voltage waveform (LUT).	BPC, MPZ, TPZ
ZeroPosition	Nulls the strain gauge reading to take out offset errors.	BPC, MPZ

Piezo Enumeration AMPFEEDBACKTYPE

Note. This enumeration is not applicable to issue 1 bench top units.

This enumeration contains constants that specify the type of feedback applied to the HV amplifier circuit when operating in closed loop mode.

Constant	Name	Purpose
1.	FEEDBACK_TYPE_DC	Feedback supplied by a.c. signals from the strain gauge in the piezo actuator
2.	FEEDBACK_TYPE_AC	Feedback supplied by a differential 0-10V d.c. signal

Strain Gauge Enumeration DISPUNITSMODE

This enumeration contains constants that specify the mode of the front panel display.

Constant Name	Purpose

1. *DISPUNITS_POSITION* The display shows the strain gauge signal as a position in microns.
2. *DISPUNITS_VOLTAGE* The display shows the strain gauge signal as a voltage.
3. *DISPUNITS_FORCE* The display shows the strain gauge signal as a force in Newtons.

Piezo Driver Enumeration HUBANALOGUEINPUT

Note. This enumeration is applicable only to T-Cube Piezo Drivers

When the T-Cube Strain Gauge Reader is used in conjunction with the T-Cube Piezo Driver unit (TPZ001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit

This enumeration contains constants that specify the feedback channel to be used on the T-Cube Controller hub.

Constant	Name	Purpose
1.	HUB_ANALOGUEIN_1	Feedback signals run through all T-Cube bays
2.	HUB_ANALOGUEIN_2	Feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6)

Strain Gauge Enumeration HUBANALOGUEOUTPUT

When the T-Cube Strain Gauge Reader is used in conjunction with the T-Cube Piezo Driver unit (TPZ001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit

This enumeration contains constants that specify the feedback channel to be used on the T-Cube Controller hub.

Constant	Name	Purpose
1.	HUB_ANALOGUEOUT_1	Feedback signals run through all T-Cube bays.
2.	HUB_ANALOGUEOUT_2	Feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6)

`Piezo Enumeration HWCHANNEL

Note. This enumeration is applicable only to APT units that are channel based.

This enumeration contains constants that specify individual channels on an APT hardware unit.

Constant	Name	Purpose
0	CHAN1_ID	Selects channel 1
1.	CHAN2_ID	Selects channel 2 (dual channel units only)
10	CHANBOTH_ID	Selects both channels (dual channel units only)

Notes.

The methods are generic and apply equally to both single and dual channel units. On single channel units, the *lChanID* parameter must be set to CHAN1_ID. On dual channel units, this can be set to CHAN1_ID, CHAN2_ID or CHANBOTH_ID as required.

Some methods that contain an lChanID parameter do not accept CHANBOTH_ID as a valid value. In this case, an error value is returned.

Piezo Enumeration HWMODE

This enumeration contains constants that specify the operating mode of the NanoTrak unit.

Constant	Name	Purpose
1.	HWMODE_PIEZO	Operate as Piezo Controller
2.	HWMODE_NANOTRAK	Operate as NanoTrak

Piezo Enumeration INPUTSOURCE

This enumeration contains constants that specify the input source(s) which controls the output from the HV amplifier circuit (i.e. the drive to the piezo actuators).

Note. The valid enumeration constants are dependent upon the issue of associated hardware unit as explained below.

On build 1 benchtop APT units (click here to identify the build of your unit), the low voltage (0 – 10V) drive inputs (rear panel BNC connectors) are single ended. For each channel a single BNC connector is used to supply a single voltage for amplification and enumeration accepts values as follows:

Constant	Name	Purpose
1.	INPUT_SWONLY	Unit responds only to software inputs and the HV amp output is that set using the SetVoltOutput method.
2.	INPUT_POSEXTBNC	Unit sums the positive analog signal on the rear panel BNC connector with the voltage set using the SetVoltOutput method
3.	INPUT_NEGEXTBNC	Unit sums the negative analog signal on the rear panel BNC connector with the voltage set using the SetVoltOutput method

On rack mounted Piezo modules, and build 2 bench top Piezo and NanoTrak units, the low voltage drive inputs accept differential signals, and a pair of BNC connectors are used to supply the drive input voltage. In this case, the INPUTSOURCE enumeration has valid constants as follows:

Constant	Name	Purpose
1.	INPUT_SWONLY	Unit responds only to software inputs and the HV amp output is that set using the SetVoltOutput method.
4	INPUT_EXTBNC	Unit sums the differential signal on the rear panel EXT IN (+) and EXT IN (-) BNC connectors with the voltage set using the SetVoltOutput method
5	INPUT_POT	The HV amp output is controlled by a potentiometer input (connected to the rear panel User I/O D-type connector) summed with the voltage set using the SetVoltOutput method.
6	INPUT_POTEXTBNC	The HV amp output is controlled by the sum of all three input sources (potentiometer, BNC′s and software).

If you are in doubt about the build number of your hardware unit, click here.

Piezo Enumeration *KPZGEARBOX*

This enumeration contains constants that specify the rate of change of voltage when the operating mode of the wheel on the top panel of the unit is set to voltage in the SetKCubePanelParams method.

Constant	Name	Purpose
1	KPZ_VOLTGEARBOX_HIGH	Voltage adjusts at a high rate, i.e. 10 steps per click
2	KPZ_VOLTGEARBOX_MED	Voltage adjusts at a medium rate, i.e. 5 steps per click
3	KPZ_VOLTGEARBOX_LOW	Voltage adjusts at a low rate, i.e. 1 step per click

Piezo Enumeration *KPZTRIGMODE*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating mode of the trigger.

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a drive voltage change as follows:

Constant	Name	Purpose
0X00	KPZ_TRIG_DISABLE	The trigger IO is disabled
0X01	KPZ_TRIG_IN_GPI	General purpose logic input (read through status bits using the LLGetStatusBits method).
0X02	KPZ_TRIG_IN_VOLTSTEP_UP	On receipt of the trigger, the drive voltage increases by the value set in the SetKCubePanelParams method.
0X03	PZ_TRIG_IN_VOLTSTEP_DOWN	Input trigger for voltage step down. On receipt of the trigger, the drive voltage decreases by the value set in the SetKCubePanelParams method.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output.

Constant	Name	Purpose
0X0A	KPZ_TRIG_OUT_GPO	General purpose logic output (set using the LLSetGetDigOPs method).

Piezo Enumeration *KPZTRIGPOLARITY*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating polarity of the trigger.

Constant	Name	Purpose
0X01	KPZ_TRIGPOL_HIGH	The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).
0X02	KPZ_TRIGPOL_LOW	The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

Piezo Enumeration *KPZWHEELDIRSENSE*

This enumeration contains constants used by the SetKCubePanelParams method, that specify the direction sense of a move initiated by the velocity wheel on the top panel of the unit.

Constant	Name	Purpose
0x01	KPZ_WHEEL_DIRSENSE_POS	Upwards rotation of the wheel results in an increased voltage.
0x02	KPZ_WHEEL_DIRSENSE_NEG	Upwards rotation of the wheel results in a decreased voltage.

Piezo Enumeration *KPZWHEELMODE*

This enumeration contains constants used by the SetKCubePanelParams method, that specify the operating mode of the wheel on the top panel of the unit.

Constant	Name	Purpose
1	KPZ_WHEEL_MODE_VOLTAGE	Deflecting the wheel changes the drive voltage. The change is proportional to the deflection. The rate of change is set in the KPZGEARBOX enumeration:
2	KPZ_WHEEL_MODE_JOG	Deflecting the wheel initiates a jog move, using the parameters specified by the *SetKCubePanelParams method*. One jog step per click of the wheel.
3	KPZ_WHEEL_MODE_GOTOVOLTAGE	Deflecting the wheel starts a move from the current position to one of the two predefined “teach” positions.

KCube Strain Gauge Reader Enumeration *KSGTRIGMODE*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKSGTriggerParams method, that specify the operating mode of the trigger.

Constant	Name	Purpose
0x00	TRIG_DISABLE	The trigger IO is disabled
0x01	TRIG_IN_GPI	General purpose logic input (read through status bits using the LLGetStatusBits method).
0x0A	TRIG_OUT_GPO	General purpose logic output (set using the LLSetGetDigOPs method).
0x0B	TRIG_OUT_LESSTHANLOWERLIMIT	The trigger is active when the strain gauge input is less than the lower limit.
0x0C	TRIG_OUT_MORETHANLOWERLIMIT	The trigger is active when the strain gauge input is greater than the lower limit.
0x0D	TRIG_OUT_LESSTHANUPPERLIMIT	The trigger is active when the strain gauge input is less than the upper limit.
0x0E	TRIG_OUT_MORETHANUPPERLIMIT	The trigger is active when the strain gauge input is greater than the upper limit.
0x0F	TRIG_OUT_BETWEENLIMITS	The trigger is active when the strain gauge input is between the two limits.
0x10	TRIG_OUT_OUTSIDELIMITS	The trigger is active when the strain gauge input is outside either of the two limits.

KCube Strain Gauge Reader Enumeration *KSGTRIGPOLARITY*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the

SetKSGTriggerParams method, that specify the operating polarity of the trigger.

Constant	Name	Purpose
0X01	KPZ_TRIGPOL_HIGH	The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).
0X02	KPZ_TRIGPOL_LOW	The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

Piezo Enumeration LLPZREQIDS

Constant	Name	Purpose
0	DEV_PARAMS	
1.	DIGOUTPUTS	
2.	CHANENABLEDSTATE	
3.	POSCONTROLMODE	
4.	OUTPUTVOLTS	
5.	OUTPUTPOS	
6.	MAXTRAVEL	
7.	IPVOLTSOURCE	
8.	PICONSTS	
9.	ZEROOFFSETS	

Piezo Enumeration LUTVALUETYPE

This enumeration contains constants that specify the value type (voltage or position) for the samples downloaded from the LUT.

Constant	Name	Purpose
1.	LUTVALTYPE_VOLTS	Values interpreted as voltage
2.	LUTVALTYPE_POS	Values interpreted as position

Piezo Enumeration LVOUTTRIGBNCMODE

This enumeration contains constants that the operating mode of the rear panel BNC connectors.

Constant	Name	Purpose
1.	BNCMODE_LVOUT	Low Voltage Output
2.	BNCMODE_TRIG	Trigger Output

Piezo Enumeration OPCONTROLMODE

This enumeration contains constants that specify the control loop feedback mode.

Constant	Name	Purpose
1.	OPEN_LOOP	No feedback
2.	CLOSED_LOOP	Position feedback
3.	OPEN_LOOP_SMOOTH	No feedback. Transition from closed to open loop is achieved over a longer period in order to minimize voltage transients (spikes).
4.	CLOSED_LOOP_SMOOTH	Position feedback. Transition from closed to open loop is achieved over a longer period in order to minimize voltage transients (spikes).

Piezo Enumeration OPDISPMODE

This enumeration contains constants that specify the GUI display mode.

Constant	Name	Purpose
1.	DISP_VOLTS	Display shows Volts
2.	DISP_POS	Display shows microns

Piezo Enumeration OUTPUTCURRENTLIMIT

Note. This enumeration is not applicable to issue 1 bench top units.

This enumeration contains constants that specify the maximum current output for the HV amplifier.

Constant	Name	Purpose
1.	CURRENTLIMIT_100MA	HV amplifier output limited to 100mA
2.	CURRENTLIMIT_250MA	HV amplifier output limited to 250mA
3.	CURRENTLIMIT_500MA	HV amplifier output limited to 500mA

Piezo Enumeration OUTPUTLPFILTER

Note. This enumeration is not applicable to issue 1 bench top units.

This enumeration contains constants that specify the cut off frequency of the Low Pass filter applied to the high voltage piezo drive output of the HV amplifier circuit.

Note. This setting is applied to both channels simultaneously.

Constant	Name	Purpose
1.	OUTPUTFILTER_10HZ	Cut off all signals above 10Hz
2.	OUTPUTFILTER_100HZ	Cut off all signals above 100Hz
3.	OUTPUTFILTER_5KHZ	Cut off all signals above 5KHz
4.	OUTPUTFILTER_NONE	Low pass filter inactive

Piezo Enumeration OUTPUTLUTMODE

This enumeration contains constants that specify the output mode of the trigger waveform.

Constant	Name	Purpose
1.	OUTPUTLUT_CONTINUOUS	The waveform is output continuously
2.	OUTPUTLUT_FIXED	A fixed number of waveform cycles are output, as specified in the SetOutputLUTParams method

Piezo Driver Enumeration OUTPUTVOLTAGELIMIT

Note. This enumeration is applicable only to T-Cube Piezo Drivers

The piezo actuator connected to the T-Cube has a specific maximum operating voltage range. This enumeration contains constants that specify the maximum drive voltage.

Constant	Name	Purpose
1.	VOLTAGELIMIT_75V	75V limit
2.	VOLTAGELIMIT_100V	100V limit
3.	VOLTAGELIMIT_150V	150V limit

Piezo Enumeration TRIGGERSENSE

This enumeration contains constants that specify voltage sense and edge of input and output triggers.

Constant	Name	Purpose
1.	TRIGSENSE_HI	The trigger goes from low (0V) to high (5V) i.e. a rising edge
2.	TRIGSENSE_LO	The trigger goes from high (5V) to low (0V) i.e. a falling edge

Piezo Method *DeleteParamSet*

Visual Basic Syntax
Function **DeleteParamSet**(bstrName As String) As Long

Parameters
bstrName – the name of the parameter set to be deleted

Returns
MG Return Code

Details
This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed. **Piezo Method *DisableHWChannel***

See Also Example Code

Visual Basic Syntax

Function **DisableHWChannel** (lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This method disables the channel(s) specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

When the method is called, the piezo drive voltage is set to zero and the front panel channel LED is extinguished. The drive voltage is stored and reapplied when the EnableHWChannel method is called.

Piezo Method *DoEvents*

See Also Example Code

Visual Basic Syntax

Function **DoEvents**() As Long

Returns

MG Return Code

Details

This method enables the server to allow message processing to take place within a client application. For example, if the client application is processing a lengthy loop and is making multiple calls to the server, this can often cause the client application to become non-responsive because other user interface message handling, such as button clicks, cannot be processed. Calling the DoEvents method allows such client application message handling to take place.

Note. The DoEvents method works in the same way as the DoEvents keyword found in Visual Basic.

Piezo Method *GetAmpFeedbackSig*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **GetAmpFeedbackSig** (lChanID as Long, plType As Long) As Long

Parameters

lChanID - the channel identifier

plType - the type of feedback signal selected.

Returns

MG Return Code

Details.

This method obtains the type of feedback signal used when operating the HV amplifier channels in closed loop operation. The feedback type is set using the SetAmpFeedbackSig method.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

All rack-mounted units (and bench top units at issue 2 or later) support two types of feedback signal, AC mode and DC mode. The feedback type is specified in the *lType* parameter which, in turn, takes values from the AMPFEEDBACKTYPE enumeration as follows:

FEEDBACK_TYPE_DC

FEEDBACK_TYPE_AC

The AC mode refers to the use of AC excited strain gauge feedback signals as generated by the complete range of our piezo actuators and piezo equipped multi axis stages. All versions of the APT NanoTrak electronics support this feedback mode. On later versions of the APT hardware it is possible to set the feedback signal type to DC. This configures the feedback circuit to accept a differential 0-10V DC feedback signal in order to control position output. This latter feedback mode is available for use with third party piezo positioning systems capable of generating a standard voltage position feedback signal.

Note. On later generation Piezo electronics (module and bench top), the function of the pin connections on the 9-way D-type feedback connector is dependent on the feedback mode selected. Refer to the pin-out tables in the associated handbook for details.

Piezo Method *GetAmpOutputParams*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **GetAmpOutputParams** (lChanID as Long, plCurrentLim As Long, plLPFilter As Long) As Long

Parameters

lChanID - the channel identifier

plCurrentLim - the maximum current output limit

plLPFilter –the value of the hardware low pass filter

Returns

MG Return Code

Details.

This method returns the output characteristics of the HV amplifier channels fitted to the NanoTrak unit. The characteristics are set using the SetAmpOutputParams method.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

The *lCurrentLim* parameter defines the maximum current output. It takes values from OUTPUTCURRENTLIMIT enumeration as

follows:

CURRENTLIMIT_100MA

CURRENTLIMIT_250MA

CURRENTLIMIT_500MA

The *lLPFilter* parameter is the value of the hardware low pass filter applied to the HV amplifier output channels, and can be used to improve stability and reduce noise on the HV outputs. It is not channel specific and the *lChanID* parameter is ignored for this particular setting. It takes values from the OUTPUTLPFILTER enumeration as follows:

OUTPUTLPFILTER_10HZ

OUTPUTLPFILTER_100HZ

OUTPUTLPFILTER_5KHZ

OUTPUTLPFILTER_NONE

The existing HVAmplifier parameter settings can be set by calling the SetAmpOutputParams method.

Piezo Method *GetControlMode*

See Also Example Code

Visual Basic Syntax

Function **GetControlMode**(lChanID As Long, plMode As Long) As Long

Parameters

lChanID - the channel identifier

plMode - returns the control loop feedback status

Returns

MG Return Code

Details

This method retrieves the control loop feedback status for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the control mode of both channels simultaneously, and the use of CHANBOTH_ID is not allowed. On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The current setting for the Control Mode is returned in the *plMode* parameter, which in turn takes values from the OPCONTROLMODE enumeration.

If set to OPEN_LOOP, no feedback is employed.

If set to CLOSED_LOOP, position feedback is provided (usually by a strain gauge).

Note. The following options are not applicable to the T-Cube Piezo Driver (TPZ001).

If set to OPEN_LOOP_SMOOTH or CLOSED_LOOP_SMOOTH the feedback status is the same as above however the transition from open to closed loop (or vise versa) is achieved over a longer period in order to minimize voltage transients (spikes).

The Control Loop mode may be set using the SetControlMode method.

T-Cube Piezo Driver Method GetDispIntensity

See Also Example Code

Note. This method is applicable only to T-Cube Piezo Drivers

Visual Basic Syntax

Function **GetDispIntensity**(lChanID As Long, plIntensity As Single) As Long

Parameters

lChanID - the channel identifier

plIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method returns the intensity of the LED display on the front of the unit. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The current setting for intensity is returned in the *plIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The display intensity may be set by calling the SetDispIntensity method

Piezo Method *GetForceSenseParams*

See Also Example Code

Visual Basic Syntax

Function **GetForceSenseParams** (lChanID As Long, pbForceMode as Boolean, pfCalib as Float, pfOffset As Float) As Long

Parameters

lChanID - the channel identifier

pbForceMode – flag used to enable force sensing mode

pfCalib– the calibration factor for the associated force sensor

pfOffset – the zero offset value

Returns

MG Return Code

Details

The controller unit can also be used to control a force sensor. This method is used to obtain various characteristics when operating in Force Sensing mode. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified in the HWCHANNEL enumeration.

Force Sensing mode is selected if the *pbForceMode* parameter returns ‘True’. When this mode is selected, an ‘F’ is displayed next to the digital display on the GUI panel. The system defaults to ‘Position’ display mode and the digital display shows the force detected by the sensor – see Display Modes for further information.

The *bCalib* parameter returns the calibration factor for the type of force sensor being used. For example, if set to 1, the GUI digital display shows a detected force of 0 to 1; if set to 10, the display shows 0 to 10. The units of force are dependent on the type of force sensor used. The default setting for this parameter is 30, to be compatible with our FSC102 force sensor, which is specified to read forces up to 30N.

The force sensor may display an offset when no contact is apparent. The *fOffset* parameter returns the value used to remove any latent offset, such that the sensor only detects a force when contact is experienced.

T-Cube Piezo Driver Method.*GetHubAnalogueChanIn*

See Also Example Code

Note. This method is applicable only to T-Cube Piezo Drivers

Visual Basic Syntax

Function **GetHubAnalogueChanIn**(lChanID As Long, plHubChan As Long) As Long

Parameters

lChanID – the channel identifier

plHubChan – the hub channel to be used

Returns

MG Return Code

Details

When the T-Cube Piezo Driver unit is used in conjunction with the T-Cube Strain Gauge Reader (TSG001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit. High precision closed loop operation is then possible using our complete range of feedback-equipped piezo actuators.

This method returns a value which represents the way in which the feedback signal is routed to the Piezo unit.

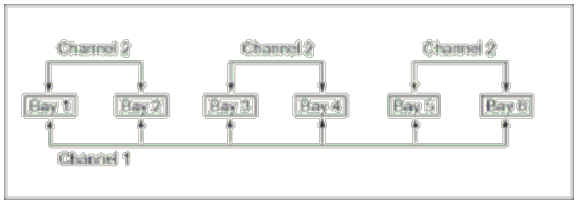
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID

The feedback channel being used is returned in the *plHubChan* parameter, which in turn, takes values from the HUBANALOGUEINPUT enumeration as follows:

If *plHubChan* returns HUB_ANALOGUEIN_1, the feedback signals run through all T-Cube bays.

If *plHubChan* returns HUB_ANALOGUEIN_2, the feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6). This setting is useful when several pairs of Strain Gauge/Piezo Driver cubes are being used on the same hub.



The feedback channel can be set using the SetHubAnalogueChanIN method.

Piezo Method *GetIPSource*

See Also Example Code

Visual Basic Syntax

Function **GetIPSource**(lChanID As Long, plSource As Long) As Long

Parameters

lChanID - the channel identifier

plSource - returns the input source

Returns

MG Return Code

Details

This method retrieves the analog input settings for the channel(s) specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot obtain the input source of both channels simultaneously, and the use of CHANBOTH_ID is not allowed. On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The method returns a value in the *plSource* parameter, which in turn takes values from the INPUTSOURCE enumeration.

The *plSource* parameter determines the input source(s) which controls the output from the HV amplifier circuit (i.e. the drive to the piezo actuators).

On issue 1 benchtop APT units (click here to identify the issue of your unit), the low voltage (0 – 10V) drive inputs (rear panel BNC connectors) are single ended. For each channel a single BNC connector is used to supply a single voltage for amplification and the p*lSource* parameter accepts values as follows:

If INPUT_SWONLY is selected, the unit responds only to software inputs and the output to the piezo actuator is that set using the SetVoltOutput method.

If INPUT_POSEXTBNC is selected, the unit sums the positive analog signal on the rear panel EXT IN BNC connector, with the voltage set using the SetVoltOutput method.

If INPUT_NEGEXTBNC is selected, the unit sums the negative analog signal on the rear panel EXT IN BNC connector, with the voltage set using the SetVoltOutput method.

On rack mounted units, and bench top units at issue 2 or later, the low voltage drive inputs accept differential signals, and a pair of BNC connectors are used to supply the drive input voltage. In this case, the INPUTSOURCE enumeration has values as follows:

If INPUT_SWONLY is selected, the unit responds only to software inputs and the HV amp output is that set using the SetVoltOutput method.

If INPUT_EXTBNC is selected, the unit sums the differential signal on the rear panel EXT IN (+) and EXT IN (-) BNC connectors with the voltage set using the SetVoltOutput method.

If INPUT_POT is selected, the HV amp output is controlled by a potentiometer input (connected to the rear panel User I/O D-type connector) summed with the voltage set using the SetVoltOutput method.

If INPUT_POTEXTBNC is selected, the HV amp output is controlled by the sum of all three input sources (potentiometer, BNC's and software).**Note**: On dual channel units, the system cannot obtain the analog input source of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

The input source may be set using the SetIPSource method.

Piezo Method *GetJogStepSize*

See Also Example Code

Visual Basic Syntax

Function **GetJogStepSize**(lChanID As Long, pfStepSize As Single) As Long

Parameters

lChanID - the channel identifier

pfStepSize - the size of step taken when the jog signal is initiated

Returns

MG Return Code

Details

This method returns the distance to move when a jog command is initiated. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The step size is returned in real world units (mm or degrees dependent upon the stage) in the *pfStepSize* parameter:

To set the jogging step size for a particular channel, see the SetJogStepSize method.

Piezo Method GetKCubePanelParams

This method is applicable only to K-Cube Series piezo controllers

See Also Example Code

Visual Basic Syntax
Function **GetKCubePanelParams** (lChanID As Long, lWheelMode As Long, pfWheelGear As Float, pfWheelJogStep As Float, plWheelDirSense As Long, pfPresetVolt1 As Float, pfPresetVolt2 As Long, plDispBrightness As Long, plDispTimeout As Long, plDispDimLevel As Long) As Long

Parameters
lChanID - the channel identifier
plWheelMode - The operating mode of the top panel wheel
pfWheelGear - The rate at which the voltage is adjusted by the top panel wheel when in voltage mode
pfWheelJogStep - The jog step size when a jog is initiated by the top panel wheel
plWheelDirSense - The direction sense of a change initiated by the top panel wheel
pfPresetVolt1 - The preset drive voltage 1 when operating in voltage mode
pfPresetVolt2 - The preset drive voltage 2 when operating in voltage mode
plDispBrightness - The display brightness when the unit is active
plDispTimeout - The display time out (in minutes). A static display will be dimmed after this time period has elapsed.
plDispDimLevel - The display brightness after time out.

Returns
MG Return Code

Details
This method sets the operating parameters of the velocity wheel on the top panel of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode is returned in the plWheelMode parameter, which in turn takes values from the KPZWHEELMODE enumeration as follows:

1 Voltage Mode - Deflecting the wheel changes the drive voltage. The change is proportional to the deflection. The rate of change is returned in the pfWheelGear parameter, which in turn takes values from the KPZGEARBOX enumeration as follows:

0x01 - Voltage adjusts at a high rate, i.e. 10 steps per click

0x02 - Voltage adjusts at a medium rate, i.e. 5 steps per click

0x03 - Voltage adjusts at a low rate, i.e. 1 step per click

2 Jog Mode - Deflecting the wheel initiates a jog move, using the parameters returned by the pfWheelJogStep parameter. One jog step per click of the wheel.

3 Go To Voltage Mode - Deflecting the wheel starts a move from the current position to one of the two predefined “teach” positions. The teach positions are specified as a drive voltage in the pfPresetVolt1 and pfPresetVolt2 parameters.

The direction of a move initiated by the velocity wheel is specified in the plWheelDirSense parameter, which in turn takes values from the KPZWHEELDIRSENSE enumeration as follows:

1 Upwards rotation of the wheel results in an increased voltage.

2 Upwards rotation of the wheel results in a decreased voltage.

In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit. The brightness is returned in the plDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely

by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Furthermore, 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after the time interval returned in the plDispTimeout parameter has elapsed. The time interval is returned in minutes in the range 0 (never) to 480. The dim level is returned in the plDispDimLevel parameter, as a value from 0 (Off) to 10 (brightest) but is also limited by the plDispBrightness parameter.

The panel parameters may be set by calling the SetKCubePanelParams method.

Piezo Method GetKCubeTriggerParams

This method is applicable only to K-Cube Series piezo controllers

See Also Example Code

Visual Basic Syntax
Function **GetKCubeTriggerParams** (lChanID As Long, plTrig1Mode As Long, plTrig1Polarity As Long, plTrig2Mode As Long, plTrig2Polarity As Long) As Long

Parameters
lChanID - the channel identifier
plTrig1Mode - TRIG1 operating mode
plTrig1Polarity - The active state of TRIG1 (i.e. logic high or logic low)
plTrig2Mode - TRIG2 operating mode
plTrig2Polarity - The active state of TRIG2 (i.e. logic high or logic low)

Returns
MG Return Code

Details
The K-Cube piezo controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be used to read an external logic signal or output a logic level to control external equipment. Either of them can be independently configured as an input or an output and the active logic state can be selected High or Low to suit the requirements of the application. Electrically the ports output 5 Volt logic signals and are designed to be driven from a 5 Volt logic.

When the port is used in the input mode, the logic levels are TTL compatible, i.e. a voltage level less than 0.8 Volt will be recognised as a logic LOW and a level greater than 2.4 Volt as a logic HIGH. The input contains a weak pull-up, so the state of the input with nothing connected will default to a logic HIGH. The weak pull-up feature allows a passive device, such as a mechanical switch to be connected directly to the input.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method sets the operating parameters of the TRIG1 and TRIG2 connectors on the front panel of the unit.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode of the trigger is returned in the lTrig1Mode and lTrig2Mode parameters, which in turn accept values from the KPZTRIGMODE enumeration as follows:

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a drive

voltage change as follows:

0x00 The trigger IO is disabled

0x01 General purpose logic input (read through status bits using the LLGetStatusBits method).

0x02 Input trigger for voltage step up. On receipt of the trigger, the drive voltage increases by the value set in the SetKCubePanelParams method, fWheelJogStep parameter.

0x03 Input trigger for voltage step down. On receipt of the trigger, the drive voltage decreases by the value set in the SetKCubePanelParams method, fWheelJogStep parameter.

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output.

0x0A General purpose logic output (set using the LLSetGetDigOPs method).

The polarity of the trigger pulse is returned in the lTrig1Polarity and lTrig2Polarity parameters, which in turn accept values from the KPZTRIGPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

For details on setting the trigger parameters, see the SetKCubeTriggerParams method.

K-Cube Strain Gauge Method GetKSGPanelParams

This method is applicable only to K-Cube Series Strain Gauge Readers

See Also Example Code

Visual Basic Syntax
Function **GetKSGPanelParams** (lChanID As Long, plDispBrightness As Long, plDispTimeout As Long, plDispDimLevel As Long) As Long

Parameters
lChanID - the channel identifier
plDispBrightness - The display brightness when the unit is active
plDispTimeout - The display time out (in minutes). A static display will be dimmed after this time period has elapsed.
plDispDimLevel - The display brightness after time out.

Returns
MG Return Code

Details
In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit.
This method returnes the present setting for the display brightness of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit. The brightness is returned in the plDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely

by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Furthermore, 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after the time interval returned in the plDispTimeout parameter has elapsed. The time interval is returned in minutes in the range 0 (never) to 480. The dim level is returned in the plDispDimLevel parameter, as a value from 0 (Off) to 10 (brightest) but is also limited by the plDispBrightness parameter.

The current setting for panel parameters may be obtained by calling the SetKSGPanelParams method.

K-Cube Strain Gauge Method GetKSGTriggerParams

This method is applicable only to K-Cube Series strain gauge readers

See Also Example Code

Visual Basic Syntax
Function SetKSGTriggerParams (lChanID As Long, plTrig1Mode As Long, plTrig1Polarity As Long, plTrig2Mode As Long, plTrig2Polarity As Long, pfLowerLim As Float, pfUpperLim As Float, plSmoothingSamples As Long) As Long

Parameters
lChanID - the channel identifier
plTrig1Mode - TRIG1 operating mode
plTrig1Polarity - The active state of TRIG1 (i.e. logic high or logic low)
plTrig2Mode - TRIG2 operating mode
plTrig2Polarity - The active state of TRIG2 (i.e. logic high or logic low)
pfLowerLim - The lower limit as a percentage of full scale deflection.
pfUpperLim - The upper limit as a percentage of full scale deflection.
plSmoothingSamples - The number of samples over which to average the displayed reading

Returns
MG Return Code

Details

The K-Cube strain gauge reader has two bidirectional trigger ports (TRIG1 and TRIG2) that can be used as a general purpose digital input/output, or can be configured to output a logic level to control external equipment.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity. The active logic state can be selected High or Low to suit the requirements of the application.

Warning. Do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

The Trigger can be used to monitor a specific area, and output a signal when the devices moves away from this region of interest. This signal can then be used to give a warning by sounding a bell or turning on an LED. The triggers are set using a combination of the lTrig1Mode and lTrig2Mode parameters, and the fLowerLim and fUpperLim parameters.

This method obtains the present settings for the operating parameters of the TRIG1 and TRIG2 connectors on the front panel of the unit.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode of the trigger is returned in the plTrig1Mode and plTrig2Mode parameters, which in turn accept values from the KSGTRIGMODE enumeration as follows:

0x00 TRIG_DISABLED - The trigger IO is disabled

0x01 TRIG_IN_GPI - General purpose logic input (read through status bits using the LLGetStatusBits method).

0x0A TRIG_OUT_GPO - General purpose logic output (set using the LLSetGetDigOPs method).

0x0B TRIG_OUT_LESSTHANLOWERLIMIT - The trigger is active when the strain gauge input is less than the lower limit.

0x0C TRIG_OUT_MORETHANLOWERLIMIT - The trigger is active when the strain gauge input is greater than the lower limit.

0x0D TRIG_OUT_LESSTHANUPPERLIMIT - The trigger is active when the strain gauge input is less than the upper limit.

0x0E TRIG_OUT_MORETHANUPPERLIMIT - The trigger is active when the strain gauge input is greater than the upper limit.

0x0F TRIG_OUT_BETWEENLIMITS - The trigger is active when the strain gauge input is between the two limits.

0x10 TRIG_OUT_OUTSIDELIMITS - The trigger is active when the strain gauge input is outside either of the two limits.

The polarity of the trigger pulse is returned in the plTrig1Polarity and plTrig2Polarity parameters, which in turn accept values from the KSGTRIGPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

The pfLowerLim and pfUpperLim parameters return the lower and upper limits described in the trigger mode details above. They are set in the range -100 to 100.

The reading shown on the display is an average of the number of samples returned in the plSmoothingSamples parameter, between 0 and 1000. As a new sample is taken, the earliest sample is discarded.

To set the trigger parameters, see the SetKSGTriggerParams method.

Piezo Method *GetMaxOPVoltage*

See Also Example Code

Visual Basic Syntax

Function **GetMaxOPVoltage**(lChanID As Long, pfMaxVoltage As Float) As Long

Parameters

lChanID - the channel identifier

pfMaxVoltage - the maximum drive voltage of the piezo actuator being driven

Returns

MG Return Code

Details.

The piezo actuator connected to the controller has a specific maximum operating voltage range. This method returns a value for the maximum output in the *pfMaxVoltage* parameter, in the range 0 to 150 V.

Note. Legacy piezo stages (i.e. those fitted with an ident resistor) are driven by 75 V. The server will detect automatically if such a stage is fitted and in this case will limit the max voltage to 75 V.

For multichannel units, the applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units, the *lChanID* parameter should be set to CHAN1_ID.

The current setting for maximum voltage may be obtained using the SetMaxOPVoltage method.

Piezo Method *GetMaxTravel*

See Also Example Code

Visual Basic Syntax

Function **GetMaxTravel**(lChanID As Long, plMaxTravel As Long) As Long

Parameters

lChanID - the channel identifier

plMaxTravel - returns the maximum travel (in microns)

Returns

MG Return Code

Details

In the case of actuators with built in position sensing, the Piezoelectric Control Unit can detect the range of travel of the actuator since this information is programmed in the electronic circuit inside the actuator.

This method retrieves the maximum travel for the piezo actuator associated with the channel specified by the *lChanID* parameter, and returns a value (in microns) in the *plMaxTravel* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot obtain the maximum travel of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

Example

```
Private Sub cmdGetTravel_Click()

Dim sngMaxTravel As Single

MG17Piezo1.GetMaxTravel CHAN1_ID, sngMaxTravel

' inform user of piezo max travel

MsgBox "Maximum travel of piezo = " & sngMaxTravel

End Sub
```

Piezo Method *GetOutputLUTParams*

See Also Example Code

Visual Basic Syntax

Function **GetOutputLUTParams** (lChanID As Long, plMode as Long, plCycleLength As Single, plNumCycles as Long, pfLUTValDelay As Single, pfPreCycleDelay As Single, pfPostCycleDelay As Single) As Long

Parameters

lChanID - the channel identifier

plMode – specifies the LUT (waveform) output mode (continuous or fixed)

plCycleLength – specifies the number of LUT values to output for a single waveform cycle (0 to 7999)

plNumCycles – specifies the number of waveform cycles to output

pfLUTValDelay – specifies the delay in milliseconds that the system waits after sending each LUT output value

pfPreCycleDelay – the delay time before the system clocks out the first LUT value

pfPostCycleDelay – the delay time after an LUT table has been output before the cycle is repeated

Returns

MG Return Code

Details

It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples. In addition, a trigger in and trigger out signal is also available.

This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.

This method is returns parameters which control the output of the LUT array. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The *plMode* parameter specifies whether the waveform will be output continuously (i.e. until a StopOPLUT command is received) or a specified number of times. It takes values specified by the OUTPUTLUTMODE enumeration as follows:

1. - OUTPUTLUT_CONTINUOUS – The waveform is output continuously.
2. - OUTPUTLUT_FIXED – A fixed number of waveform cycles are output (as specified in the *lNumCycles* parameter).

The *plNumCycles* parameter specifies the number of cycles (1 to 2147483648) to be output when the *plMode* parameter is set to fixed. If *lMode* is set to *Continuous*, the *plNumCycles* parameter is ignored. In both cases, the waveform is not output until a StartOPLUT command is received.

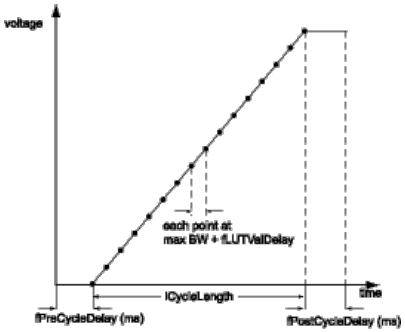
The *plCycleLength* parameter specifies how many samples will be output in each cycle of the waveform. It can be set in the range 1 to 8000, and must be less than or equal to the total number of samples that were loaded. (To set the LUT array values for a particular channel, see the SetOutputLUTValue method).

By default, the system outputs LUT values at the maximum bandwidth possible, i.e. 7KHz (or 0.14ms per LUT value) for BPC00x models or 1KHz (or 01.0ms per LUT value) for BPC10x models. The *pfLUTValDelay* parameter specifies the time interval between neighbouring samples, i.e. for how long the sample will remain at its present value. To increase the time between samples, set the *pfLUTValDelay* parameter to the required additional delay in milliseconds (resolution steps of 0.14ms for BPC00x models or 1.0ms for BPC10x models). In this way, the user can stretch or shrink the waveform without affecting its overall shape.

In some applications, during waveform generation the first and the last samples may need to be handled differently from the rest of the waveform. For example, in a positioning system it may be necessary to start the movement by staying at a certain position for a specified length of time, then perform a movement, then remain at the last position for another specified length of time. This is the purpose of *pfPreCycleDelay* and *pfPostCycleDelay* parameters, i.e. they specify the length of time that the first and last samples are output for, independently of the *pfLUTValDelay* parameter.

The *pfPreCycleDelay* parameter allows a delay time to be set before the system starts to clock out the LUT values. The delay can be set between 0 and 10 seconds in 0.14ms increments for BPC00x models or 1.0ms increments for BPC10x models. The system then outputs the first value in the LUT until the PreCycleDelay time has expired.

The *pfPostCycleDelay* parameter specifies the delay imposed by the system after a LUT table has been output. The delay can be set between 0 and 10 seconds in 0.14ms increments for BPC00x models or 1.0ms increments for BPC10x models. The system then outputs the last value in the cycle until the PostCycleDelay time has expired.



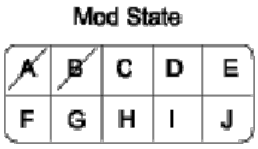
To set the parameters which control the output of the LUT array for a particular channel, see the SetOutputLUTParams method.

Piezo Method *GetOutputLUTTrigParams*

See Also Example Code

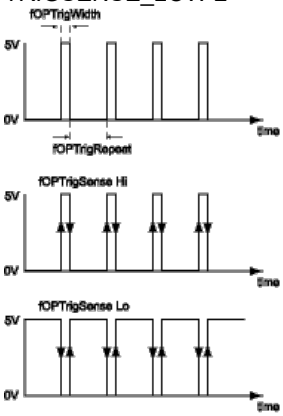
This method is not applicable to TPZ001 T-Cube Piezo Drivers

Notes for BPC001 or BPC002 Users.
The Input Trigger functionality is not available on BPC001 and BPC002 units.
On units to modification state B or earlier, triggering functionality is enabled by fitting a trigger enable plug. These plugs are available from the company on request.
A label indicating the build and modification state of the hardware is located on the underside of the unit. Modification states are 'scored out' as they become current.



Visual Basic Syntax
Function **GetOutputLUTTrigParams** (lChanID As Long, pbOPTrigEnable as Boolean, pbIPTrigEnable as Boolean, plOPTrigStart As Single, pfOPTrigWidth As Single, plOPTrigSense As Single, plIPTrigSense As Single, pbOPTrigRepeat As Boolean) As Long
Parameters
lChanID - the channel identifier
pbOPTrigEnable – flag used to enable O/P triggering
pbIPTrigEnable – flag used to enable the initiation of waveform outputs from a hardware input trigger
plOPTrigStart – specifies the LUT value (position in the array) at which to initiate an output trigger
pfOPTrigWidth – sets the width of the output trigger (in milliseconds)
plOPTrigSense – determines the voltage sense and edge of output triggers
plIPTrigSense – determines the voltage sense and edge of input triggers
pbOPTrigRepeat – flag which determines if repeated output triggering is enabled
plOPTrigRepeat – specifies the repeat interval between output triggers
Returns
MG Return Code
Details
It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples. In addition, a trigger in and trigger out signal is also available. The "trigger out" function enables the user to set the trigger output high or low at the specified time, for a specified length of time, once or a predefined number of times for each output cycle. The "trigger in" function allows the user to start waveform generation (either on a single channel or multiple channels) synchronized to an external event.
This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.
This method is used to return various parameters associated with triggering (both input and output) and LUT (waveform) outputs. The applicable channel is specified by the *lChanID* parameter, which takes values specified in the HWCHANNEL enumeration. Note that OP triggering is possible only on one channel at a time, and the use of CHANBOTH_ID is not allowed, however IP triggering can be performed on multiple channels.
Output triggering is enabled if the *pbOPTrigEnable* parameter returns 'True' (default 'False'). With *OPTrigEnable* set to True, the system can be configured to generate one or more hardware trigger pulses during a LUT (waveform) cycle output. The *plOPTrigStart* parameter returns the LUT value (position in the LUT array) at which to initiate an output trigger. In this way, it is possible to synchronize an output trigger with the output of a particular voltage value. Values are set in the range 1 to 8000 but must also be less than the *lCycleLength* parameter set in the SetOutputLUTParams method.
Input triggering is enabled if the *bIPTrigEnable* parameter returns 'True' (default 'False'). With *bIPTrigEnable* set to '*False*', the waveform generator will start as soon as it receives a "start waveform" software command. If however, *bIPTrigEnable* is set to '*True*, waveform generation will only start if a software command is received AND the trigger input is in its active state. In most cases, the trigger input will be used to synchronize waveform generation to an external event. In this case, the software "start waveform" command can be viewed as a command to "arm" the waveform generator and the waveform will start as soon as the input becomes active.
The trigger input can be used to trigger a single channel or multiple channels. In this latter case ensure that input triggering is enabled on all the desired channels. Using the trigger input for multiple channels is particularly useful to synchronize all channels to the same event.
The *pfOPTrigWidth* parameter returns the width of the trigger pulse in milliseconds. Values are entered in 0.14ms increments for BPC00x models or 1ms increments for BPC10x models.
The *plOPTrigSense* parameter determines the voltage sense and edge of the O/P triggers and takes values from the TRIGGERSENSE enumeration as follows:

TRIGSENSE_HI 1
TRIGSENSE_LOW 2



If set to TRIGSENSE_HI, the trigger outputs go from low (0V) to high (5V), i.e. a rising edge.
The *plIPTrigSense* parameter determines the voltage sense and edge of the I/P triggers and takes values from the TRIGGERSENSE enumeration as follows:
TRIGSENSE_HI 1
TRIGSENSE_LOW 2
If set to TRIGSENSE_LOW, the inputs will respond to a trigger which goes from high (5V) to low (0V) i.e. a falling edge
Note. If triggering functionality is required on both channels, it is important that the *plOPTrigSense* and *plIPTrigSense* parameters are set the same on each channel. Care must be taken if triggering is enabled on both channels, as it is possible that superimposed or merged triggering could occur.
The *pbOPTrigRepeat* parameter is a flag which determines if repeated O/P triggering is enabled for the specified channel. If *pbOPTrigRepeat* is set to True, repeated O/P triggering takes place with an interval between triggers as specified by the *plOPTrigRepeat* parameter. This is useful for multiple triggering during a single voltage O/P sweep.
The *plOPTrigRepeat* parameter specifies the repeat interval between O/P triggers when *pbOPTrigRepeat* is set to True. This parameter is specified in the number of LUT values between triggers (0 to 7999). If this value is greater than the lCycleLength parameter (set in the *SetOPLUTParams* method) then by definition, a repeated trigger will not occur during a single waveform cycle output.
To set the various parameters associated with triggering (both input and output) and LUT (waveform) outputs, see the SetOutputLUTTrigParams method. **Piezo Method *GetOutputLUTValue***

See Also Example Code

Function **GetOutputLUTValue**(lChanID As Long, lIndex as Long, pfValue As Single (float)) As Long
Parameters
lChanID - the channel identifier
lIndex - the position in the array of the value to be set (0 to 7999)
pfValue - the voltage value to be set (0 to 75V)
Returns
MG Return Code
Details
It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples.
This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.
The waveform is stored as values in an array. BPC20x and BPC30x controllers can output a maximum of 8000 samples per channel. The TPZ001 T-Cube driver can output 512 samples (due to memory limitations).
The samples can have the meaning of voltage or position; if open loop operation is specified when the samples are output, then their meaning is voltage and vice versa, if the channel is set to closed loop operation, the samples are interpreted as position values. If the waveform to be output requires less than 8000 samples, it is sufficient to download the desired number of samples.

This method is used to return values from the LUT array which makes up the required output waveform. The applicable channel (s) is specified by the *IChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
If only a sub set of the array is being used (as specified by the *lcyclelength* parameter of the SetOutputLUTParams method), then only the first lcyclelength values need to be set. In this manner, any arbitrary voltage waveform can be programmed into the LUT.
Note. The LUT values are output by the system at a maximum bandwidth of 7KHz, e.g.500 LUT values will take approximately 71 ms to be clocked out and the full 8000 LUT values will take approximately 1.14 secs.
To set the LUT array values for a particular channel, see the SetOutputLUTValue method. **Piezo Method *GetParentHWInfo***

See Also Example Code

Visual Basic Syntax

Function **GetParentHWInfo** (plHWSerialNum As Long, plHWType As Long) As Long

Parameters

plHWSerialNum– the serial number of the host enclosure

plHWType – the hardware type of the host enclosure

Returns

MG Return Code

Details

This method returns the serial number and hardware type identifier of the parent (or host) controller enclosure.

In certain APT products, individual daughter cards or modules are fitted to a common enclosure (e.g. BPC103 three channel benchtop piezo controller) which is programmed at the factory with a unique serial number. The *plHWSerialNum* parameter is used to retrieve this serial number and is useful when trying to establish which daughter cards are located within a specific enclosure.

The *plHWType* parameter returns values specified by the APT_PARENT_HWTYPES enumeration as follows:

28 USB_BPC003 3 Channel, 75V Benchtop Piezo Controller

29 ETHNET_MMR601 6 Bay Modular Rack, Ethernet

30 USB_MMR601 6 Bay Modular Rack, USB

Piezo Method *GetPIConsts*

See Also Example Code

Visual Basic Syntax

Function **GetPIConsts**(lChanID As Long, plProp As Long, plInt As Long) As Long

Parameters

lChanID - the channel identifier

plProp - the proportional constant

plInt - the integration constant

Returns

MG Return Code

Details.

This method retrieves the current setting of the proportional and integration feedback loop constants for the channel specified by the *lChanID* parameter, and returns a value in the value specified in the *plProp* and *plInt* parameters respectively.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot obtain the PI constants of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To set the PI constants, see the SetPIConsts method.

Piezo Method *GetPosOutput*

See Also Example Code

Visual Basic Syntax

Function **GetPosOutput**(lChanID As Long, pfPosition As Single) As Long

Parameters

lChanID - the channel identifier

pfPosition - returns the current position of the piezo actuator

Returns

MG Return Code

Details.

Note. The SetPosOutput method is applicable only to actuators equipped with position sensing and operating in closed loop mode.

This method sets the position of the piezo actuator associated with the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the position of both piezo actuators simultaneously, and the use of CHANBOTH_ID is not allowed.

The position of the actuator is relative to the datum set for the arrangement using the *ZeroPosition* method. Once set, the extension of the actuator will be read and scaled automatically to a position in microns, for whichever actuator is connected to the control module.

The position value is returned in the *pfPosition* parameter.

For the T-Cube Piezo Driver (TPZ001), this position setpoint is returned as a percentage (0-100%) reflecting min-max piezo extension, e.g. for a 30 micron piezo stack, a position setpoint of 50% equates to 15 micron.

For all other Piezo controller models, the position setpoint is returned in microns.

Actuators fitted with feedback circuitry can be interrogated by the Piezo module for maximum travel in microns - see the GetMaxTravel method.

If the controller is in 'Force Sensing Mode' (see the SetForceSenseParams method) then the *pfPosition* parameter returns a force value.

The position of the actuator may be set using the SetPosOutput method.

Piezo Method GetPPCIOSettings

See Also Example Code

Visual Basic Syntax
Function G**etPPCIOSettings** (lChanID As Long, plControlSrc As Long, plMonitorOPSig As Long, plMonitorOPBandwidth As Long, plFeedbackSrc As Long) As Long

Parameters
lChanID - the channel identifier
plControlSrc - the analog input source
plMonitorOPSig - the Output Monitoring Signal Type
plMonitorOPBandwidth - the Output Monitoring Signal filter option
plFeedbackSrc - The Closed Loop feedback signal type

Returns
MG Return Code

Details
This method is used to set various input and output parameter values associated with the rear panel BNC IO connectors. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. In the case of the PPC001 controller this should always be set to '1'.

The *plControlSrc* parameter determines the input source(s) which controls the output from the HV amplifier circuit (i.e. the drive to the piezo actuators) as follows:

Software Only = 0
EXT BNC + Software = 1
Joystick + Software = 2
EXT BNC + Joystick + Software = 3

If Software Only (0) is selected, the unit responds only to software inputs and the output to the piezo actuator is that set using the SetVoltOutput method, or the Output knob on the GUI panel.
If EXT BNC + Software (1) is selected, the unit sums the analog signal on the rear panel EXT IN BNC connector, with the voltage set using the SetVoltOutput method or the Output knob on the GUI panel.
If Joystick + Software (2) is selected, the unit sums the analog signal the external joystick, with the voltage set using the SetVoltOutput method or the Output knob on the GUI panel.
If EXT BNC + Joystick + Software (3) is selected, the unit sums all three signals.

The signal on the rear panel EXT OUT BNC can be used to monitor the piezo actuator on an oscilloscope or other device. The type of signal can be returned in the *plMonitorOPSig* parameter as follows:

Drive Voltage = 1
Raw Position = 2
Linearized Position = 3

If Drive Voltage (1) is selected, the signal driving the EXT OUT (Monitor) BNC is a scaled down version of the piezo output voltage, with 150 V piezo voltage corresponding to 10V.

If Raw Position (2) is selected, the signal driving the EXT OUT (Monitor) BNC is the output voltage of the position demodulator. This signal shows a slight nonlinearity as a function of position and a small offset voltage. As a result it is not as accurate as the linearized position. However, having not undergone any digital processing it is free of any potential digital signal processing effects and can be more advantageous for loop tuning and transient response measurement.

If Linearized Position (3) is selected, the signal driving EXT OUT is linearized and scaled so that the 0 to full range corresponds to 0 to 10 Volts.

The signal on the rear panel EXT OUT BNC can also be filtered to limit the output bandwidth to the range of interest in most closed loop applications, i.e. 200Hz.

The filter is returned in the plMonitorOPBandwidth parameter as follows:

No Filter = 1
200 Hz Low Pass Filter = 2

When operating in closed loop mode, the feedback can be supplied by either a Capacitive or a Strain Gauge sensor. The plFeedbackSrc parameter is used to return the feedback type as follows:

Strain Gauge = 1
Capacitive = 2

The input and output settings may be set using the SetPPCIOSettings method.

Piezo Method *GetPPCNotchParams*

Note. This method is applicable only to the PPC001 Precision Piezo Controller

See Also Example Code

Visual Basic Syntax
Function **GetPPCNotchParams**(plChanID As Long, plFilterNo As Long, pfNotch1CentreFreq As Float, pfNotch1Q As Float, plNotch1FilterOn As Long, pfNotch2CentreFreq As Float, pfNotch2Q As Float, plNotch2FilterOn As Long) As Long

Parameters
pl*ChanID* - the channel identifier
plFilterNo -
pfNotch1CentreFreq - the centre frequency of notch filter 1
pfNotch1Q - the Q factor of notch filter 1
plNotch1FilterOn - enables and disables notch filter 1

pfNotch2CentreFreq - the centre frequency of notch filter 2
pfNotch2Q - the Q factor of notch filter 2
plNotch2FilterOn - enables and disables notch filter 2

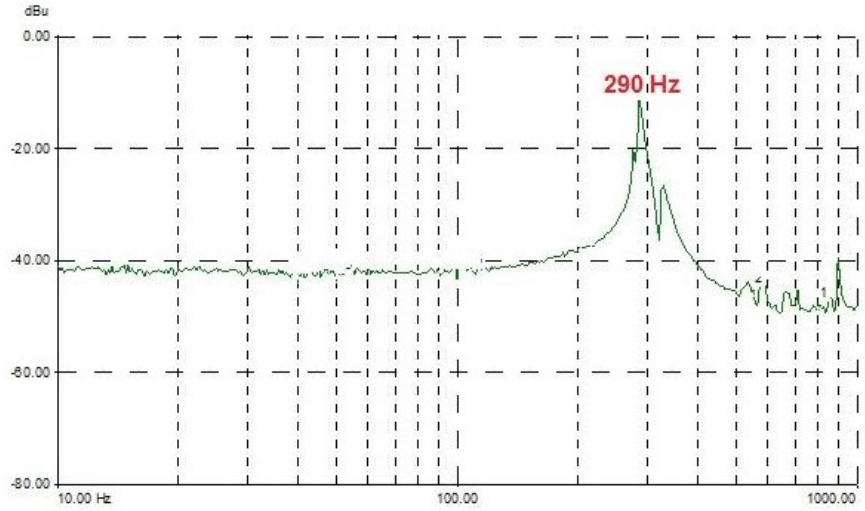
Returns
MG Return Code

Details
This method is used to obtain the current parameter settings associated with the notch filters for the PID loop associated with a particular ActiveX Control instance. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. In the case of the PPC001 controller this should always be set to '1'.

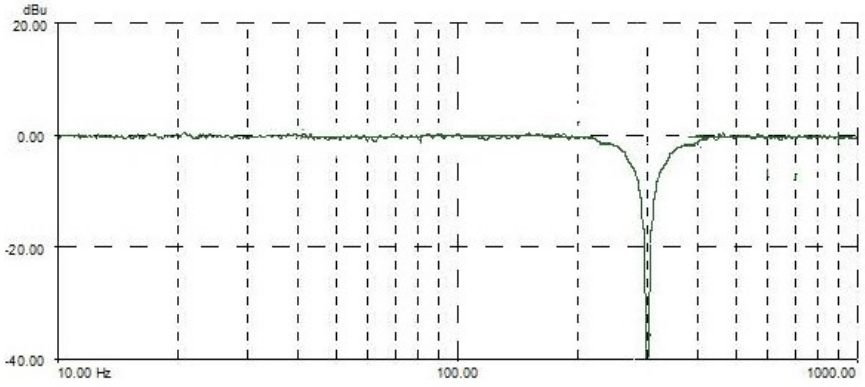
The filter being addressed by the method is specified in the *plFilterNo* parameter as follows:

Notch Filter 1 = 1
Notch Filter 2 = 2
Both Filters = 3

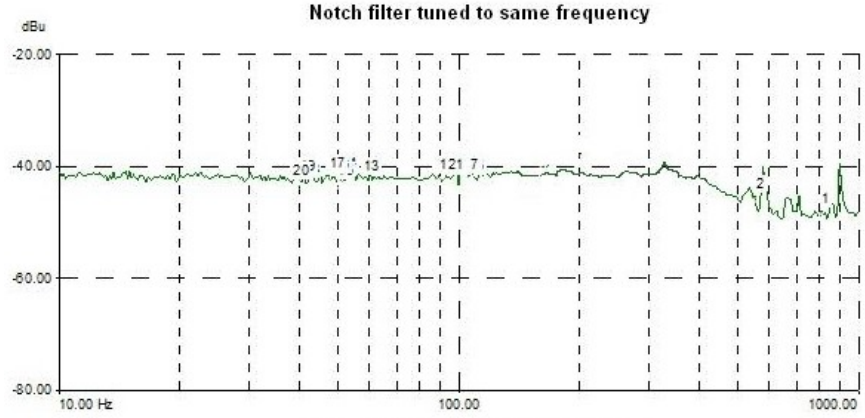
Notch Filter Settings - Due to their construction, most actuators are prone to mechanical resonance at well-defined frequencies. The underlying reason is that all spring-mass systems are natural harmonic oscillators. This proneness to resonance can be a problem in closed loop systems because, coupled with the effect of the feedback, it can result in oscillations. With some actuators, the resonance peak is either weak enough or at a high enough frequency for the resonance not to be troublesome. With other actuators the resonance peak is very significant and needs to be eliminated for operation in a stable closed loop system. The notch filter is an adjustable electronic anti-resonance that can be used to counteract the natural resonance of the mechanical system.



Frequency response of actuator showing resonance at 290 Hz



Notch filter tuned to same frequency



The resonance is largely eliminated

As the resonant frequency of actuators varies with load in addition to the minor variations from product to product, the notch filter is tuneable so that its characteristics can be adjusted to match those of the actuator. In addition to its centre frequency, the bandwidth of the notch (or the equivalent quality factor, often referred to as the Q-factor) can also be adjusted. In simple terms, the Q factor is the centre frequency/bandwidth, and defines how wide the notch is, a higher Q factor defining a narrower ("higher quality") notch. Optimizing the Q factor requires some experimentation but in general a value of 5 to 10 is in most cases a good starting point.

Notch filter 1 is enabled/disabled in the plNotch1FilterOn parameter as follows:

Notch Filter ON = 1
Notch Filter OFF = 2

The centre frequency of notch filter 1 is set in the pfNotch1CentreFreq parameter, in the range 20 to 500. The Q Factor is set in the pf*Notch1Q parameter, in the range 0.2 to 100.*

Notch filter 2 is set similarly in the Notch Filter 2 parameters.

The current Notch Filter settings can be obtained by calling the SetPPCNotchParams.

Piezo Method GetPPCPIDParams

Note. This method is applicable only to the PPC001 Precision Piezo Controller

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

See Also Example Code

Visual Basic Syntax
Function **GetPPCPIDParams** (plChanID As Long, pfProp As Float, pfInt As Float, pfDeriv As Float, pfDerivLPCutOffFreq As Float, plDerivFilterOn As Long) As Long

Parameters
plChanID - the channel identifier
pfProp - the value of the proportional constant
pfInt - the value of the integration constant
pfDeriv - the value of the differential constant
pfDerivLPCutOffFreq - the cut off frequency of the derivative low pass filter
plDerivFilterOn - enables and disables the derivative low pass filter

Returns
MG Return Code

Details.

When operating in Closed Loop mode, the proportional, integral and differential (PID) constants can be used to fine tune the behaviour of the feedback loop to changes in the output voltage or position. While closed loop operation allows more precise control of the position, feedback loops need to be adjusted to suit the different types of focus mount assemblies that can be connected to the system. Due to the wide range of objectives that can be used with the PFM450 and their different masses, some loop tuning may be necessary to optimize the response of the system and to avoid instability.

This method obtains the current values for these PID parameters.The default values have been optimized to work with the actuator shipped with the controller and any changes should be made with caution.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. In the case of the PPC001 controller this should always be set to '1'.

pfProp- The Proportional parameter makes a change to the output which is proportional to the current error value. A high proportional gain results in a large change in the output for a given error. If set too high, the system can become unstable. It accepts values in the range 0 to 10000 (default 10)

pfInt - The Integral parameter accelerates the process towards the demanded set point value, ensuring that the positional error is eventually reduced to zero. If set too high, the output can overshoot the demand value. It accepts values in the range 0 to 10000. (default 5).

pfDeriv - The Derivative parameter damps the rate of change of the output, thereby decreasing the overshoot which may be caused by the integral term. However, the differential term also slows down system response. If set too high, it could lead to instability due to signal noise amplification. It accepts values in the range 0 to 10000 (default 0).

Note. The default values have been optimised to work with the sensor shipped with the controller.
Due to the type of load the PFM450 presents to the PPC001 controller, the system is most sensitive to the integral (Int) tuning parameter and the notch filter settings. In most applications the deriviative parameters can be left at their default values.

pfDerivLPCutOffFreq - The output of the derivative (differential) part of the PID controller can be passed through a tuneable low pass filter. Whilst the derivative component of the PID loop often improves stability (as it acts as a retaining force against abrupt changes in the system), it is prone to amplifying noise present in the system, as the derivative component is sensitive to changes between adjacent samples. To reduce this effect, a low pass filter can be applied to the samples. As noise often tends to contain predominantly high frequency components, the low pass filter can significantly decrease their contribution, often without diminishing the beneficial, stabilizing effect of the derivative action. In some applications enabling this filter can improve the overall closed loop performance. The Cut Off Frequency is specified in Hz in the range 0 to 10000.

lDerivFilterOn - enables and disables the low pass filter as follows:
Filter On = 1
Filter Off = 2

The parameters can be set by calling the SetPPCPIDParams method.

Piezo Method *GetVoltOutput*

See Also Example Code

Visual Basic Syntax

Function **GetVoltOutput**(lChanID As Long, pfVoltage As Single) As Long

Parameters

lChanID - the channel identifier

pfVoltage - returns the voltage currently applied to the piezo actuator

Returns

MG Return Code

Details.

This method returns the output voltage applied to the piezo actuator, as a value specified in the *pfVoltage* parameter.

Note. On the T-Cube Piezo Drivers (TPZ001), the output voltage can be set up to 150V, on all other units, the maximum voltage is 75V.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the output voltage of both channels simultaneously, and the use of CHANBOTH_ID is not allowed. On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The voltage applied to the actuator may be set using the SetVoltOutput method.

T-Cube Piezo Driver Method *GetVoltageOutputLimit*

See Also Example Code

Note. This method is applicable only to T-Cube Piezo Drivers

Visual Basic Syntax

Function **GetVoltageOutputLimit**(lChanID As Long, plLimit As Single) As Long

Parameters

lChanID - the channel identifier

plLimit - the maximum drive voltage of the piezo actuator

Returns

MG Return Code

Details.

The piezo actuator connected to the T-Cube has a specific maximum operating voltage range. This method returns the maximum output voltage in the *lLimit* parameter. This parameter takes values specified by the OUTPUTVOLTAGELIMIT enumeration as follows.

1. VOLTAGELIMIT_75V 75V limit
2. VOLTAGELIMIT_100V 100V limit
3. VOLTAGELIMIT_150V 150V limit

The applicable channel is specified by the *IChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *IChanID* parameter should be set to CHAN1_ID.

The maximum voltage limit may be set by calling the SetVoltageOutputLimit method.

Piezo Method *GetVoltPosDispMode*

See Also Example Code

Visual Basic Syntax

Function **GetVoltPosDispMode**(lChanID As Long, plMode As Long) As Long

Parameters

lChanID - the channel identifier

plMode - returns the display setting (Volts or Microns)

Returns

MG Return Code

Details.

This method retrieves the current GUI display mode for the channel specified by the *IChanID* parameter, and returns a value in the *plMode* parameter, which in turn takes values from the OPDISPMODE enumeration.

The *IChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot obtain the display mode of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

The GUI display mode may be set by calling the SetVoltPosDispMode method

T-Cube Quad Detector Method Identify

See Also Example Code

Visual Basic Syntax

Function **Identify**() As Long

Returns

MG Return Code

Details

This method allows the hardware unit associated with the ActiveX control instance to be identified. The associated unit is specified by the HWSerialNum property.

When the method is called, the front panel LEDs on the relevant hardware unit will flash for a short period.

Piezo Method *LLGetStatusBits*

See Also Example Code

Visual Basic Syntax

Function **LLGetStatusBits**((lChanID As Long, plStatusBits As Long) As Long

Parameters

lChanID – the channel identifier

plStatusBits – the 32-bit integer containing the status flags.

Returns

MG Return Code

Details

This low level method returns a number of status flags pertaining to the operation of the piezo controller channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description
1. x00000001	1.	Piezo actuator connected (1 - connected, 0 - not connected).
	2. to 4	For Future Use
0x00000010	5	Piezo channel has been zero'd (1 - zero'd, 0 not zero'd).
0x00000020	6	Piezo channel is zeroing (1 - zeroing, 0 - not zeroing).
0x00000040	7 to 8	For Future Use
0x00000100	9	Strain gauge feedback connected (1 - connected, 0 - not connected).
	10	For Future Use
0x00000400	11	Position control mode (1 - closed loop, 0 - open loop).
	12 to 20	For Future Use

Note. Bits 21 to 28 (Digital Input States) are only applicable if the associated digital input is fitted to your controller – see the relevant handbook for more details

0x00100000	21	Digital input 1 state (1 - logic high, 0 - logic low).
0x00200000	22	Digital input 2 state (1 - logic high, 0 - logic low).
0x00400000	23	Digital input 3 state (1 - logic high, 0 - logic low).
0x00800000	24	Digital input 4 state (1 - logic high, 0 - logic low).
0x01000000	25	Digital input 5 state (1 - logic high, 0 - logic low).
0x02000000	26	Digital input 6 state (1 - logic high, 0 - logic low).
0x04000000	27	Digital input 7 state (1 - logic high, 0 - logic low).
0x08000000	28	Digital input 8 state (1 - logic high, 0 - logic low).
	29	For Future Use
0x20000000	30	Active (1 – indicates unit is active, 0 – not active)
0x40000000	31	For Future Use
0x80000000	32	Channel enabled (1 – enabled, 0- disabled)

Piezo Method *LoadParamSet*

See Also Example Code

Visual Basic Syntax

Function **LoadParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be loaded

Returns

MG Return Code

Details

This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

Piezo Method *SaveParamSet*

See Also Example Code

Visual Basic Syntax

Function **SaveParamSet**(bstrName As String) As Long

Parameters

bstrName – the name under which the parameter set is to be saved

Returns

MG Return Code

Details

This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved. Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

Piezo Method *SetAmpFeedbackSig*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **SetAmpFeedbackSig** (lChanID as Long, lType As Long) As Long

Parameters

lChanID - the channel identifier

lType - the type of feedback signal selected.

Returns

MG Return Code

Details.

This method sets the type of feedback signal used when operating the HV amplifier channels in closed loop operation.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

All rack-mounted units (and bench top units at issue 2 or later) support two types of feedback signal, AC mode and DC mode. The feedback type is specified in the *lType* parameter which, in turn, takes values from the AMPFEEDBACKTYPE enumeration as follows:

FEEDBACK_TYPE_DC

FEEDBACK_TYPE_AC

The AC mode refers to the use of AC excited strain gauge feedback signals as generated by the complete range of our piezo actuators and piezo equipped multi axis stages. All versions of the APT Piezo electronics support this feedback mode. On later versions of the APT hardware it is possible to set the feedback signal type to DC. This configures the feedback circuit to accept a differential 0-10V DC feedback signal in order to control position output. This latter feedback mode is available for use with

third party piezo positioning systems capable of generating a standard voltage position feedback signal.

Note. On later generation Piezo electronics (module and bench top), the function of the pin connections on the 9-way D-type feedback connector is dependent on the feedback mode selected. Refer to the pin-out tables in the associated handbook for details.

The feedback type can be obtained by calling the GetAmpFeedbackSig method.

Piezo Method *SetAmpOutputParams*

See Also Example Code

Note. This method is not applicable to issue 1 bench top units.

Visual Basic Syntax

Function **SetAmpOutputParams** (lChanID as Long, lCurrentLim As Long, lLPFilter As Long) As Long

Parameters

lChanID - the channel identifier

lCurrentLim - the maximum current output limit

lLPFilter –the value of the hardware low pass filter

Returns

MG Return Code

Details.

This method sets the output characteristics of the HV amplifier channels fitted to the Piezo unit. Both low pass filtering and current control settings can be made to tailor the piezo output drive to a particular application.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration and identifies the channel to which the settings are applied.

The *lCurrentLim* parameter sets the maximum current output and takes values from the OUTPUTCURRENTLIMIT enumeration as follows:

CURRENTLIMIT_100MA

CURRENTLIMIT_250MA

CURRENTLIMIT_500MA

The *lLPFilter* parameter is the value of the hardware low pass filter applied to the HV amplifier output channels, and can be used to improve stability and reduce noise on the HV outputs. It is not channel specific and the *lChanID* parameter is ignored for this particular setting. It takes values from the OUTPUTLPFILTER enumeration as follows:

OUTPUTLPFILTER_10HZ

OUTPUTLPFILTER_100HZ

OUTPUTLPFILTER_5KHZ

OUTPUTLPFILTER_NONE

The existing HVAmplifier parameter settings can be obtained by calling the GetAmpOutputParams method.

Piezo Method *SetControlMode*

See Also Example Code

Visual Basic Syntax

Function **SetControlMode**(lChanID As Long, lMode As Long) As Long

Parameters

lChanID - the channel identifier

lMode - the control loop status

Returns

MG Return Code

Details

When in closed-loop mode, position is maintained by a feedback signal from the piezo actuator. This is only possible when using actuators equipped with position sensing.

This method sets the control loop status for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the control mode of both channels simultaneously, and the use of CHANBOTH_ID is not allowed. On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The Control Mode is specified in the *lMode* parameter, which in turn takes values from the OPCONTROLMODE enumeration.

If set to OPEN_LOOP, no feedback is employed.

If set to CLOSED_LOOP, position feedback is provided (usually by a strain gauge).

Note. The following options are not applicable to the T-Cube Piezo Driver (TPZ001).

If set to OPEN_LOOP_SMOOTH or CLOSED_LOOP_SMOOTH the feedback status is the same as above however the transition from open to closed loop (or vise versa) is achieved over a longer period in order to minimize voltage transients (spikes).

The Control Loop mode may be obtained using the GetControlMode method.

Example

```
Private Sub cmdSetClosedLoop_Click()

' Set closed loop, display mode to position, and zero sensor

MG17Piezo1.SetControlMode CHAN1_ID, CLOSED_LOOP

MG17Piezo1.SetVoltPosDispMode CHAN1_ID, DISP_POS

MG17Piezo1.ZeroPosition CHAN1_ID

End Sub
```

T-Cube Piezo Driver Method *SetDispIntensity*

See Also Example Code

Note. This method is applicable only to T-Cube Piezo Drivers

Visual Basic Syntax

Function **SetDispIntensity**(lChanID As Long, lIntensity As Single) As Long

Parameters

lChanID - the channel identifier

lIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method sets the intensity of the LED display on the front of the unit. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The current setting for display intensity may be obtained by calling the GetDispIntensity method

Piezo Method *SetForceSenseParams*

See Also Example Code

Visual Basic Syntax

Function **SetForceSenseParams** (lChanID As Long, bForceMode as Boolean, fCalib as Float, fOffset As Float) As Long

Parameters

lChanID - the channel identifier

bForceMode – flag used to enable force sensing mode

fCalib– the calibration factor for the associated force sensor

fOffset – the zero offset value

Returns

MG Return Code

Details

The controller unit can also be used to control a force sensor. This method is used to set various characteristics when operating in Force Sensing mode. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified in the HWCHANNEL enumeration.

Force Sensing mode is specified by setting the *bForceMode* parameter to 'True'. When this mode is selected, an 'F' is displayed next to the digital display on the GUI panel. The system defaults to 'Position' display mode and the digital display shows the force detected by the sensor – see Display Modes for further information.

The *bCalib* parameter specifies the calibration factor for the type of force sensor being used. For example, if set to 1, the GUI digital display shows a detected force of 0 to 1; if set to 10, the display shows 0 to 10. The units of force are dependent on the type of force sensor used. The default setting for this parameter is 30, to be compatible with our FSC102 force sensor, which is specified to read forces up to 30N.

The force sensor may display an offset when no contact is apparent. The *fOffset* parameter is used to remove any latent offset, such that the sensor only detects a force when contact is experienced.

T-Cube Piezo Driver Method *SetHubAnalogueChanIn*

See Also Example Code

Note. This method is applicable only to T-Cube Piezo Drivers

Visual Basic Syntax

Function **SetHubAnalogueChanIn**(lChanID As Long, lHubChan As Long) As Long

Parameters

lChanID – the channel identifier

lHubChan – the hub channel to be used

Returns

MG Return Code

Details

When the T-Cube Piezo Driver unit is used in conjunction with the T-Cube Strain Gauge Reader (TSG001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit. High precision closed loop operation is then possible using our complete range of feedback-equipped piezo actuators.

This method is used to select the way in which the feedback signal is routed to the Piezo unit.

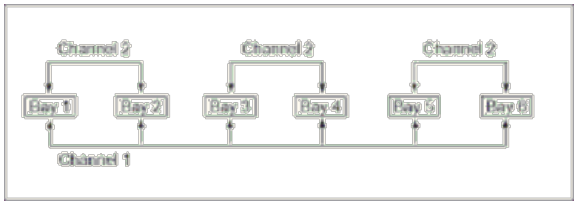
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID

The feedback channel to be used is set in the *lHubChan* parameter, which in turn, takes values from the HUBANALOGUEINPUT enumeration as follows:

If *lHubChan* is set to HUB_ANALOGUEIN_1, the feedback signals run through all T-Cube bays.

If *lHubChan* is set to HUB_ANALOGUEIN_2, the feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6). This setting is useful when several pairs of Strain Gauge/Piezo Driver cubes are being used on the same hub.



The current setting for the feedback channel can be obtained using the GetHubAnalogueChanIN method.

Piezo Method *SetHWMode*

See Also **Example Code**

Visual Basic Syntax

Function **SetHWMode**(lMode As Long) As Long

Parameters

lMode - The hardware operating mode (Piezo controller or NanoTrak).

Returns

MG Return Code

Details

This method sets the operating mode of the NanoTrak controller and is not applicable to Piezo controllers without NanoTrak functionality.

The default mode of operation is NanoTrak, however the unit can be configured as a Piezo controller by setting the *lMode* parameter, which in turn accepts values from the HWMODE enumeration.

Note. The hardware unit must be rebooted before changes to operating mode can take effect.

Note. When the HW operating mode of a NanoTrak unit has been changed to Piezo operation, then the Piezo ActiveX control must be used to communicate with the unit. Use the same serial number as used on the NanoTrak control in order to establish

communication with the unit.

Example

```
Private Sub cmdPZToNTMode_Click()

MG17Piezo1.SetHWMode HWMODE_NANOTRAK

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

MG17NanoTrak1.HWSerialNum = "22" & Right(CStr(MG17Piezo1.HWSerialNum), 6)

' Start NanoTrak control

MG17NanoTrak1.StartCtrl

End Sub
```

Piezo Method *SetJogStepSize*

See Also Example Code

Visual Basic Syntax

Function **SetJogStepSize**(lChanID As Long, fStepSize As Single) As Long

Parameters

lChanID - the channel identifier

fStepSize - the size of step taken when the jog signal is initiated

Returns

MG Return Code

Details

This method sets the distance to move when a jog command is initiated. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The step size is specified in real world units (mm or degrees dependent upon the stage) in the *fStepSize* parameter:

To retrieve the jogging mode for a particular channel, see the GetJogStepSize method.

Piezo Method *SetIPSource*

See Also Example Code

Visual Basic Syntax

Function **SetIPSource**(lChanID As Long, lSource As Long) As Long

Parameters

lChanID - the channel identifier

lSource - the analog input source

Returns

MG Return Code

Details

This method sets the analog input source for the channel(s) specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot obtain the input source of both channels simultaneously, and the use of CHANBOTH_ID is not allowed. On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The *lSource* parameter determines the input source(s) which controls the output from the HV amplifier circuit (i.e. the drive to the piezo actuators). It takes values from the INPUTSOURCE enumeration.

On issue 1 benchtop APT units (click here to identify the issue of your unit), the low voltage (0 – 10V) drive inputs (rear panel BNC connectors) are single ended. For each channel a single BNC connector is used to supply a single voltage for amplification and the *lSource* parameter accepts values as follows:

If INPUT_SWONLY is selected, the unit responds only to software inputs and the output to the piezo actuator is that set using the SetVoltOutput method.

If INPUT_POSEXTBNC is selected, the unit sums the positive analog signal on the rear panel EXT IN BNC connector, with the voltage set using the SetVoltOutput method.

If INPUT_NEGEXTBNC is selected, the unit sums the negative analog signal on the rear panel EXT IN BNC connector, with the voltage set using the SetVoltOutput method.

On rack mounted units, and bench top units at issue 2 or later, the low voltage drive inputs accept differential signals, and a pair of BNC connectors are used to supply the drive input voltage. In this case, the INPUTSOURCE enumeration has values as follows:

If INPUT_SWONLY is selected, the unit responds only to software inputs and the HV amp output is that set using the SetVoltOutput method.

If INPUT_EXTBNC is selected, the unit sums the differential signal on the rear panel EXT IN (+) and EXT IN (-) BNC connectors with the voltage set using the SetVoltOutput method.

If INPUT_POT is selected, the HV amp output is controlled by a potentiometer input (connected to the rear panel User I/O D-type connector) summed with the voltage set using the SetVoltOutput method.

If INPUT_POTEXTBNC is selected, the HV amp output is controlled by the sum of all three input sources (potentiometer, BNC's and software).

On the latest APT units (BPC10X), the low voltage drive inputs accept differential signals, and a pair of BNC connectors are used to supply the drive input voltage. In this case, the INPUTSOURCE enumeration has values as follows:

If INPUT_SWONLY is selected, the unit responds only to software inputs and the HV amp output is that set using the SetVoltOutput method.

If INPUT_EXTBNC is selected, the unit sums the differential signal on the rear panel EXT IN (+) and EXT IN (-) BNC connectors with the voltage set using the SetVoltOutput method.

The current input source setting may be obtained using the GetIPSource method.

Piezo Method SetKCubePanelParams

This method is applicable only to K-Cube Series piezo controllers

See Also Example Code

Visual Basic Syntax
Function **SetKCubePanelParams** (lChanID As Long, lWheelMode As Long, fWheelGear As Float, fWheelJogStep As Float, lWheelDirSense As Long, fPresetVolt1 As Float, fPresetVolt2 As Long, lDispBrightness As Long, lDispTimeout As Long, lDispDimLevel As Long) As Long

Parameters
lChanID - the channel identifier
lWheelMode - The operating mode of the top panel wheel
fWheelGear - The rate at which the voltage is adjusted by the top panel wheel when in voltage mode

fWheelJogStep - The jog step size when a jog is initiated by the top panel wheel
lWheelDirSense - The direction sense of a change initiated by the top panel wheel
fPresetVolt1 - The preset drive voltage 1 when operating in voltage mode
fPresetVolt2 - The preset drive voltage 2 when operating in voltage mode
lDispBrightness - The display brightness when the unit is active
lDispTimeout - The display time out (in minutes). A static display will be dimmed after this time period has elapsed.
lDispDimLevel - The display brightness after time out.

Returns
MG Return Code

Details
This method sets the operating parameters of the velocity wheel on the top panel of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode is specified in the lWheelMode parameter, which in turn takes values from the KPZWHEELMODE enumeration as follows:

0x01 Voltage Mode - Deflecting the wheel changes the drive voltage. The change is proportional to the deflection. The rate of change is set in the fWheelGear parameter, which in turn takes values from the KPZGEARBOX enumeration as follows:

0x01 - Voltage adjusts at a high rate, i.e. 10 steps per click

0x02 - Voltage adjusts at a medium rate, i.e. 5 steps per click

0x03 - Voltage adjusts at a low rate, i.e. 1 step per click

0x02 Jog Mode - Deflecting the wheel initiates a jog move, using the parameters specified by the fWheelJogStep parameter. One jog step per click of the wheel.

0x03 Go To Voltage Mode - Deflecting the wheel starts a move from the current position to one of the two predefined “teach” positions. The teach positions are specified as a drive voltage in the fPresetVolt1 and fPresetVolt2 parameters.

The direction of a move initiated by the velocity wheel is specified in the lWheelDirSense parameter, which in turn takes values from the KPZWHEELDIRSENSE enumeration as follows:

0x01 Upwards rotation of the wheel results in an increased voltage.

0x02 Upwards rotation of the wheel results in a decreased voltage.

In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit. The brightness is set in the lDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Furthermore, 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after the time interval specified in the lDispTimeout parameter has elapsed. The time interval is set in minutes in the range 0 (never) to 480. The dim level is set in the lDispDimLevel parameter, as a value from 0 (Off) to 10 (brightest) but is also limited by the DispBrightness parameter.

The current setting for panel parameters may be obtained by calling the GetKCubePanelParams method.

Piezo Method SetKCubeTriggerParams

This method is applicable only to K-Cube Series piezo controllers

See Also Example Code

Visual Basic Syntax
Function **SetKCubeTriggerParams** (lChanID As Long, lTrig1Mode As Long, lTrig1Polarity As Long, lTrig2Mode As Long, lTrig2Polarity As Long) As Long

Parameters
lChanID - the channel identifier
lTrig1Mode - TRIG1 operating mode
lTrig1Polarity - The active state of TRIG1 (i.e. logic high or logic low)
lTrig2Mode - TRIG2 operating mode
lTrig2Polarity - The active state of TRIG2 (i.e. logic high or logic low)

Returns
MG Return Code

Details

The K-Cube piezo controllers have two bidirectional trigger ports (TRIG1 and TRIG2) that can be used to read an external logic signal or output a logic level to control external equipment. Either of them can be independently configured as an input or an output and the active logic state can be selected High or Low to suit the requirements of the application. Electrically the ports output 5 Volt logic signals and are designed to be driven from a 5 Volt logic.

When the port is used in the input mode, the logic levels are TTL compatible, i.e. a voltage level less than 0.8 Volt will be recognised as a logic LOW and a level greater than 2.4 Volt as a logic HIGH. The input contains a weak pull-up, so the state of the input with nothing connected will default to a logic HIGH. The weak pull-up feature allows a passive device, such as a mechanical switch to be connected directly to the input.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method sets the operating parameters of the TRIG1 and TRIG2 connectors on the front panel of the unit.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode of the trigger is specified in the lTrig1Mode and lTrig2Mode parameters, which in turn accept values from the KPZTRIGMODE enumeration as follows:

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a drive voltage change as follows:

0x00 The trigger IO is disabled

0x01 General purpose logic input (read through status bits using the LLGetStatusBits method).

0x02 Input trigger for voltage step up. On receipt of the trigger, the drive voltage increases by the value set in the SetKCubePanelParams method, fWheelJogStep parameter.

0x03 Input trigger for voltage step down. On receipt of the trigger, the drive voltage decreases by the value set in the SetKCubePanelParams method, fWheelJogStep parameter.

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output.

0x0A General purpose logic output (set using the LLSetGetDigOPs method).

The polarity of the trigger pulse is specified in the lTrig1Polarity and lTrig2Polarity parameters, which in turn accept values from the KPZTRIGPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

For details on obtain the current trigger parameter settings, see the GetKCubeTriggerParams method.

K-Cube Strain Gauge Method SetKSGPanelParams

This method is applicable only to K-Cube Series Strain Gauge Readers

See Also Example Code

Visual Basic Syntax
Function **SetKSGPanelParams** (lChanID As Long, lDispBrightness As Long, lDispTimeout As Long, lDispDimLevel As Long) As Long

Parameters
lChanID - the channel identifier
lDispBrightness - The display brightness when the unit is active
lDispTimeout - The display time out (in minutes). A static display will be dimmed after this time period has elapsed.
lDispDimLevel - The display brightness after time out.

Returns
MG Return Code

Details
In certain applications, it may be necessary to adjust the brightness of the LED display on the top of the unit.
This method sets the display brightness of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
For single channel units like the K-Cubes, this should be set to '1'.

The brightness is set in the lDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Furthermore, 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after the time interval specified in the lDispTimeout parameter has elapsed. The time interval is specified in minutes in the range 0 (never) to 480. The dim level is set in the lDispDimLevel parameter, as a value from 0 (Off) to 10 (brightest) but is also limited by the DispBrightness parameter.

The current setting for panel parameters may be obtained by calling the GetKSGPanelParams method.

K-Cube Strain Gauge Method SetKSGTriggerParams

This method is applicable only to K-Cube Series strain gauge readers

See Also Example Code

Visual Basic Syntax
Function SetKSGTriggerParams (lChanID As Long, lTrig1Mode As Long, lTrig1Polarity As Long, lTrig2Mode As Long, lTrig2Polarity As Long, fLowerLim As Float, fUpperLim As Float, lSmoothingSamples As Long) As Long

Parameters
lChanID - the channel identifier
lTrig1Mode - TRIG1 operating mode
lTrig1Polarity - The active state of TRIG1 (i.e. logic high or logic low)
lTrig2Mode - TRIG2 operating mode
lTrig2Polarity - The active state of TRIG2 (i.e. logic high or logic low)
fLowerLim - The lower limit as a percentage of full scale deflection.
fUpperLim - The upper limit as a percentage of full scale deflection.
lSmoothingSamples - The number of samples over which to average the displayed reading

Returns
MG Return Code

Details

The K-Cube strain gauge reader has two bidirectional trigger ports (TRIG1 and TRIG2) that can be used as a general purpose digital input/output, or can be configured to output a logic level to control external equipment.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity. The active logic state can be selected High or Low to suit the requirements of the application.

Warning. Do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

The Trigger can be used to monitor a specific area, and output a signal when the device moves away from this region of interest. This signal can then be used to give a warning by sounding a bell or turning on an LED. The triggers are set using a combination of the lTrig1Mode and lTrig2Mode parameters, and the fLowerLim and fUpperLim parameters.

This method sets the operating parameters of the TRIG1 and TRIG2 connectors on the front panel of the unit.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. For single channel units like the K-Cubes, this should be set to '1'.

The operating mode of the trigger is specified in the lTrig1Mode and lTrig2Mode parameters, which in turn accept values from the KSGTRIGMODE enumeration as follows:

0x00 TRIG_DISABLED - The trigger IO is disabled

0x01 TRIG_IN_GPI - General purpose logic input (read through status bits using the LLGetStatusBits method).

0x0A TRIG_OUT_GPO - General purpose logic output (set using the LLSetGetDigOPs method).

0x0B TRIG_OUT_LESSTHANLOWERLIMIT - The trigger is active when the strain gauge input is less than the lower limit.

0x0C TRIG_OUT_MORETHANLOWERLIMIT - The trigger is active when the strain gauge input is greater than the lower limit.

0x0D TRIG_OUT_LESSTHANUPPERLIMIT - The trigger is active when the strain gauge input is less than the upper limit.

0x0E TRIG_OUT_MORETHANUPPERLIMIT - The trigger is active when the strain gauge input is greater than the upper limit.

0x0F TRIG_OUT_BETWEENLIMITS - The trigger is active when the strain gauge input is between the two limits.

0x10 TRIG_OUT_OUTSIDELIMITS - The trigger is active when the strain gauge input is outside either of the two limits.

The polarity of the trigger pulse is specified in the lTrig1Polarity and lTrig2Polarity parameters, which in turn accept values from the KSGTRIGPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

The fLowerLim and fUpperLim parameters specify the lower and upper limits described in the trigger mode details above. They are set in the range -100 to 100.

The reading shown on the display is an average of the number of samples set in the lSmoothingSamples parameter, between 0 and 1000. As a new sample is taken, the earliest sample is discarded.

To obtain the current trigger parameter settings, see the GetKSGTriggerParams method.

Piezo Method *SetLVTrigBNCMode*

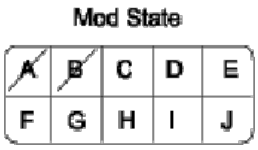
See Also Example Code

Notes for BPC001 or BPC002 Users.

Triggering functionality is only available when operating in open loop (Voltage) mode.

On units to modification state B or earlier, triggering functionality is enabled by fitting a trigger enable plug. These plugs are available from the company on request.

A label indicating the build and modification state of the hardware is located on the underside of the unit. Modification states are 'scored out' as they become current.



Visual Basic Syntax

Function **SetLVTrigBNCMode** (lMode As Long) As Long

Parameters

lMode - the selected mode of the rear panel BNC connectors

Returns

MG Return Code

Details

The Control IO BNC connectors on the rear panel are dual function. When set to Low Voltage (LV) outputs they mirror the voltage on the Piezo drive HV connectors and can be connected to an oscilloscope for monitoring purposes. When set to Trigger mode they provide the trigger input and output connections.

This method is used to set the mode of the rear panel BNC connectors and. takes values specified by the LVOUTTRIGBNCMODE

BNCMODE_LVOUT 1 LV Output

BNCMODE_TRIG 2 Trigger Output

Piezo Method *SetMaxOPVoltage*

See Also Example Code

Visual Basic Syntax

Function **SetMaxOPVoltage**(lChanID As Long, fMaxVoltage As Float) As Long

Parameters

lChanID - the channel identifier

fMaxVoltage - the maximum drive voltage of the piezo actuator being driven

Returns

MG Return Code

Details.

The piezo actuator connected to the controller has a specific maximum operating voltage range. This method sets the maximum output to the value specified in the *fMaxVoltage* parameter, in the range 0 to 150 V.

Note. Legacy piezo stages (i.e. those fitted with an ident resistor) are driven by 75 V. The server will detect automatically if such a stage is fitted and in this case will limit the max voltage to 75 V.

For multichannel units, the applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units, the *lChanID* parameter should be set to CHAN1_ID.

The current setting for maximum voltage may be obtained using the GetMaxOPVoltage method.

Piezo Method *SetOutputLUTParams*

See Also Example Code

Visual Basic Syntax

Function **SetOutputLUTParams** (lChanID As Long, lMode as Long, lCycleLength As Single, lNumCycles as Long, fLUTValDelay As Single, fPreCycleDelay As Single, fPostCycleDelay As Single) As Long

Parameters

lChanID - the channel identifier

lMode – specifies the LUT (waveform) output mode (continuous or fixed)

fCycleLength – specifies the number of LUT values to output for a single waveform cycle (0 to 7999)

lNumCycles – specifies the number of waveform cycles to output

fLUTValDelay – specifies the delay in milliseconds that the system waits after setting each LUT output value

fPreCycleDelay – the delay time before the system clocks out the LUT values

fPostCycleDelay – the delay time after an LUT table has been output before the cycle is repeated

Returns

MG Return Code

Details

It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples. In addition, a trigger in and trigger out signal is also available.

This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.

This method is used to set parameters which control the output of the LUT array. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

The *lMode* parameter specifies whether the waveform will be output continuously (i.e. until a StopOPLUT command is received) or a specified number of times. It takes values specified by the OUTPUTLUTMODE enumeration as follows:

1. - OUTPUTLUT_CONTINUOUS – The waveform is output continuously.
2. - OUTPUTLUT_FIXED – A fixed number of waveform cycles are output (as specified in the *lNumCycles* parameter).

The *lNumCycles* parameter specifies the number of cycles (1 to 2147483648) to be output when the *lMode* parameter is set to fixed. If *lMode* is set to *Continuous*, the *lNumCycles* parameter is ignored. In both cases, the waveform is not output until a StartOPLUT command is received.

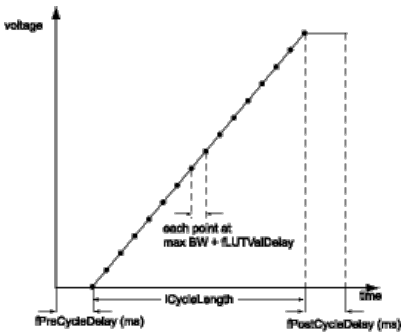
The *lCycleLength* parameter specifies how many samples will be output in each cycle of the waveform. It can be set in the range 1 to 8000, and must be less than or equal to the total number of samples that were loaded. (To set the LUT array values for a particular channel, see the SetOutputLUTValue method).

By default, the system outputs LUT values at the maximum bandwidth possible, i.e. 7KHz (or 0.14ms per LUT value) for BPC00x models or 1KHz (or 01.0ms per LUT value) for BPC10x models. The *fLUTValDelay* parameter specifies the time interval between neighbouring samples, i.e. for how long the sample will remain at its present value. To increase the time between samples, set the *fLUTValDelay* parameter to the required additional delay in milliseconds (resolution steps of 0.14ms for BPC00x models or 1.0ms for BPC10x models). In this way, the user can stretch or shrink the waveform without affecting its overall shape.

In some applications, during waveform generation the first and the last samples may need to be handled differently from the rest of the waveform. For example, in a positioning system it may be necessary to start the movement by staying at a certain position for a specified length of time, then perform a movement, then remain at the last position for another specified length of time. This is the purpose of *fPreCycleDelay* and *fPostCycleDelay* parameters, i.e. they specify the length of time that the first and last samples are output for, independently of the *fLUTValDelay* parameter.

The *fPreCycleDelay* parameter allows a delay time to be set before the system starts to clock out the LUT values. The delay can be set between 0 and 10 seconds in 0.14ms increments for BPC00x models or 1.0ms increments for BPC10x models. The system then outputs the first value in the LUT until the PreCycleDelay time has expired.

The *fPostCycleDelay* parameter specifies the delay imposed by the system after a LUT table has been output. The delay can be set between 0 and 10 seconds in 0.14ms increments for BPC00x models or 1.0ms increments for BPC10x models. The system then outputs the last value in the cycle until the PostCycleDelay time has expired.



To retrieve the parameters which control the output of the LUT array for a particular channel, see the GetOutputLUTParams method.

Piezo Method*SetOutputLUTTrigParams*

See Also Example Code

This method is not applicable to TPZ001 T-Cube Piezo Drivers

Notes for BPC001 or BPC002 Users.
The Input Trigger functionality is not available on BPC001 and BPC002 units.
On units to modification state B or earlier, triggering functionality is enabled by fitting a trigger enable plug. These plugs are available from the company on request.
A label indicating the build and modification state of the hardware is located on the underside of the unit. Modification states are 'scored out' as they become current.

Mod State

~~A~~	~~B~~	C	D	E
F	G	H	I	J

Visual Basic Syntax
Function **SetOutputLUTTrigParams** (lChanID As Long, bOPTrigEnable as Boolean, bIPTrigEnable as Boolean, lOPTrigStart As Single, fOPTrigWidth As Single, lOPTrigSense As Single, lIPTrigSense As Single, bOPTrigRepeat As Boolean) As Long
Parameters

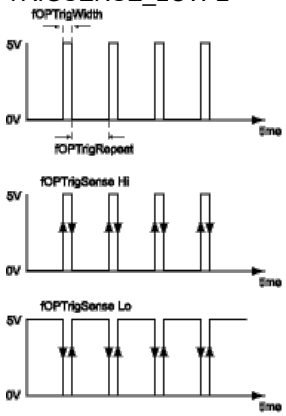
lChanID - the channel identifier
bOPTrigEnable – flag used to enable O/P triggering
bIPTrigEnable –flag used to enable the initiation of waveform outputs from a hardware input trigger
lOPTrigStart – specifies the LUT value (position in the array) at which to initiate an output trigger
fOPTrigWidth – sets the width of the output trigger (in milliseconds)
lOPTrigSense – determines the voltage sense and edge of output triggers
lIPTrigSense – determines the voltage sense and edge of input triggers
bOPTrigRepeat – flag which determines if repeated output triggering is enabled
lOPTrigRepeat – specifies the repeat interval between output triggers

Returns

MG Return Code

Details

It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples. In addition, a trigger in and trigger out signal is also available. The "trigger out" function enables the user to set the trigger output high or low at the specified time, for a specified length of time, once or a predefined number of times for each output cycle. The "trigger in" function allows the user to start waveform generation (either on a single channel or multiple channels) synchronized to an external event.
This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.
This method is used to set various parameters associated with triggering (both input and output) and LUT (waveform) outputs. The applicable channel is specified by the *lChanID* parameter, which takes values specified in the HWCHANNEL enumeration. Note that OP triggering is possible only on one channel at a time, and the use of CHANBOTH_ID is not allowed, however IP triggering can be performed on multiple channels.
Output triggering is enabled by setting the *bOPTrigEnable* parameter to 'True' (default 'False'). With *OPTrigEnable* set to True, the system can be configured to generate one or more hardware trigger pulses during a LUT (waveform) cycle output. The *lOPTrigStart* parameter specifies the LUT value (position in the LUT array) at which to initiate an output trigger. In this way, it is possible to synchronize an output trigger with the output of a particular voltage value. Values are set in the range 1 to 8000 but must also be less than the *lCycleLength* parameter set in the SetOutputLUTParams method.
Input triggering is enabled by setting the *bIPTrigEnable* parameter to 'True' (default 'False'). With *bIPTrigEnable* set to '*False'*, the waveform generator will start as soon as it receives a "start waveform" software command. If however, *bIPTrigEnable* is set to '*True*, waveform generation will only start if a software command is received AND the trigger input is in its active state. In most cases, the trigger input will be used to synchronize waveform generation to an external event. In this case, the software "start waveform" command can be viewed as a command to "arm" the waveform generator and the waveform will start as soon as the input becomes active.
The trigger input can be used to trigger a single channel or multiple channels. In this latter case ensure that input triggering is enabled on all the desired channels. Using the trigger input for multiple channels is particularly useful to synchronize all channels to the same event.
The *fOPTrigWidth* parameter sets the width of the trigger pulse in milliseconds. Values are entered in 0.14ms increments for BPC00x models or 1ms increments for BPC10x models.
The *lOPTrigSense* parameter determines the voltage sense and edge of the O/P triggers and takes values from the TRIGGERSENSE enumeration as follows:
TRIGSENSE_HI 1
TRIGSENSE_LOW 2



If set to TRIGSENSE_HI, the trigger outputs go from low (0V) to high (5V), i.e. a rising edge.
The *lIPTrigSense* parameter determines the voltage sense and edge of the I/P triggers and takes values from the TRIGGERSENSE enumeration as follows:
TRIGSENSE_HI 1
TRIGSENSE_LOW 2
If set to TRIGSENSE_LOW, the inputs will respond to a trigger which goes from high (5V) to low (0V) i.e. a falling edge
Note. If triggering functionality is required on both channels, it is important that the *lOPTrigSense* and *lIPTrigSense* parameters are set the same on each channel. Care must be taken if triggering is enabled on both channels, as it is possible that superimposed or merged triggering could occur.

The *bOPTrigRepeat* parameter is a flag which determines if repeated O/P triggering is enabled for the specified channel. If *bOPTrigRepeat* is set to True, repeated O/P triggering takes place with an interval between triggers as specified by the *lOPTrigRepeat* parameter. This is useful for multiple triggering during a single voltage O/P sweep.
The *lOPTrigRepeat* parameter specifies the repeat interval between O/P triggers when *bOPTrigRepeat* is set to True. This parameter is specified in the number of LUT values between triggers (0 to 7999). If this value is greater than the lCycleLength parameter (set in the *SetOPLUTParams* method) then by definition, a repeated trigger will not occur during a single waveform cycle output.
Note. The total time for the *TrigRepeat*, together with any time delays must be larger than the time set in the *TrigWidth* parameter.
To retrieve the various parameters associated with triggering (both input and output) and LUT (waveform) outputs, see the GetOutputLUTTrigParams method. **Piezo Method *SetOutputLUTValue***
See Also Example Code
Visual Basic Syntax
Function **SetOutputLUTValue**(lChanID As Long, lIndex as Long, fValue As Single (float)) As Long
Parameters
lChanID - the channel identifier
lIndex - the position in the array of the value to be set (0 to 7999)
fValue - the voltage value to be set (0 to 75V)
Returns
MG Return Code
Details
It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples.
This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.
The waveform is stored as values in an array. BPC20x and BPC30x controllers can output a maximum of 8000 samples per channel. The TPZ001 T-Cube driver can output 512 samples (due to memory limitations).
The samples can have the meaning of voltage or position; if open loop operation is specified when the samples are output, then their meaning is voltage and vice versa, if the channel is set to closed loop operation, the samples are interpreted as position values. If the waveform to be output requires less than 8000 samples, it is sufficient to download the desired number of samples.

This method is used to load the LUT array with the required output waveform. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
If only a sub set of the array is being used (as specified by the *lcyclelength* parameter of the SetOutputLUTParams method), then only the first lcyclelength values need to be set. In this manner, any arbitrary voltage waveform can be programmed into the LUT.
Note. The LUT values are output by the system at a maximum bandwidth of 7KHz, e.g. for a BPCxxx unit, 500 LUT values will take approximately 71 ms to be clocked out and the full 8000 LUT values will take approximately 1.14 secs.
To retrieve the LUT array for a particular channel, see the GetOutputLUTValue method. **Piezo Method *SetOutputLUTValueTypes***

See Also Example Code

THIS METHOD IS APPLICABLE ONLY TO USERS OF BPC001, BPC002 AND MPZ601 PRODUCTS AND MUST BE CALLED BEFORE THE LUT VALUES ARE DOWNLOADED.

Visual Basic Syntax

Function **SetOutputLUTValueTypes** (lType as Long) As Long

Parameters

lType - the type of value (voltage or position) read in from the look up table.

Returns

MG Return Code

Details

It is possible to use the controller in an arbitrary Waveform Generator Mode (WGM). Rather than the unit outputting an adjustable but static voltage or position, the WGM allows the user to define a voltage or position sequence to be output, either periodically or a fixed number of times, with a selectable interval between adjacent samples.

This waveform generation function is particularly useful for operations such as scanning over a particular area, or in any other application that requires a predefined movement sequence.

The waveform is stored as values in an array, with a maximum of 8000 samples per channel. The samples can have the meaning of voltage or position; if open loop operation is specified when the samples are output, then their meaning is voltage and vice versa, if the channel is set to closed loop operation, the samples are interpreted as position values. If the waveform to be output requires less than 8000 samples, it is sufficient to download the desired number of samples.

This method specifies whether the samples output from the LUT are voltage or position values. The value type is specified in the *lType* parameter, which in turn takes values from the LUTVALUETYPE enumeration as follows:

1. – LUTVALTYPE_VOLTS – The LUT samples are interpreted as voltage values.
2. - LUTVALTYPE_POS – The LUT samples are interpreted as position values.

Notes on using this method.

This method must be called BEFORE the LUT values are downloaded.

The LUT values are scaled to either voltage or position while the LUT is being downloaded. If the value type needs to be changed during operation (e.g. the system was in open loop with volts type selected, but now needs to change to closed loop with position type) the method must be called again, and the LUT values downloaded again.

Piezo Method *SetPIConsts*

See Also Example Code

Visual Basic Syntax

Function **SetPIConsts**(lChanID As Long, lProp As Long, lInt As Long) As Long

Parameters

lChanID - the channel identifier

lProp - the value of the proportional constant

lInt - the value of the integration constant

Returns

MG Return Code

Details.

This method sets the proportional and integration feedback loop constants for the channel specified by the *lChanID* parameter, to the value specified in the *lProp* and *lInt* parameters respectively. These parameters determine the response characteristics when operating in closed loop mode.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the PI constants of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

To obtain the current settings for the PI constants, see the GetPIConsts method.

Piezo Method *SetPosOutput*

See Also Example Code

Visual Basic Syntax

Function **SetPosOutput**(lChanID As Long, fPosition As Single) As Long

Parameters

lChanID - the channel identifier

fPosition - the position of the piezo actuator

Returns

MG Return Code

Details.

Note. The SetPosOutput method is applicable only to actuators equipped with position sensing and operating in closed loop mode.

This method sets the position of the piezo actuator associated with the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the position of both piezo actuators simultaneously, and the use of CHANBOTH_ID is not allowed.

The position of the actuator is relative to the datum set for the arrangement using the *ZeroPosition* method. Once set, the extension of the actuator will be read and scaled automatically to a position in microns, for whichever actuator is connected to the control module.

The position value is specified in the *fPosition* parameter.

For the T-Cube Piezo Driver (TPZ001), this position setpoint is specified as a percentage (0-100%) reflecting min-max piezo extension, e.g. for a 30 micron piezo stack, a position setpoint of 50% equates to 15 micron.

For all other Piezo controller models, the position setpoint is specified in microns.

Actuators fitted with feedback circuitry can be interrogated by the Piezo module for maximum travel in microns - see the GetMaxTravel method.

The current position of the actuator may be obtained using the GetPosOutput method.

Example

```
Private Sub cmdSetPosition_Click()

' NB: position can only be set in closed loop mode

Dim sngPosition As Single

sngPosition = CSng(Val(txtPosition))

MG17Piezo1.SetPosOutput CHAN1_ID, sngPosition

End Sub
```

Piezo Method SetPPCIOSettings

See Also Example Code

Visual Basic Syntax
Function **SetPPCIOSettings** (lChanID As Long, lControlSrc As Long, lMonitorOPSig As Long, lMonitorOPBandwidth As Long, lFeedbackSrc As Long) As Long

Parameters
lChanID - the channel identifier
lControlSrc - the analog input source
lMonitorOPSig - the Output Monitoring Signal Type
lMonitorOPBandwidth - the Output Monitoring Signal filter option
lFeedbackSrc - The Closed Loop feedback signal type

Returns
MG Return Code

Details
This method is used to set various input and output parameter values associated with the rear panel BNC IO connectors. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. In

the case of the PPC001 controller this should always be set to '1'.

The *lControlSrc* parameter determines the input source(s) which controls the output from the HV amplifier circuit (i.e. the drive to the piezo actuators) as follows:

Software Only = 0

EXT BNC + Software = 1

Joystick + Software = 2

EXT BNC + Joystick + Software = 3

If Software Only (0) is selected, the unit responds only to software inputs and the output to the piezo actuator is that set using the SetVoltOutput method, or the Output knob on the GUI panel.
If EXT BNC + Software (1) is selected, the unit sums the analog signal on the rear panel EXT IN BNC connector, with the voltage set using the SetVoltOutput method or the Output knob on the GUI panel.
If Joystick + Software (2) is selected, the unit sums the analog signal the external joystick, with the voltage set using the SetVoltOutput method or the Output knob on the GUI panel.
If EXT BNC + Joystick + Software (3) is selected, the unit sums all three signals.

The signal on the rear panel EXT OUT BNC can be used to monitor the piezo actuator on an oscilloscope or other device. The type of signal can be set in the *lMonitorOPSig* parameter as follows:

Drive Voltage = 1
Raw Position = 2
Linearized Position = 3

If Drive Voltage (1) is selected, the signal driving the EXT OUT (Monitor) BNC is a scaled down version of the piezo output voltage, with 150 V piezo voltage corresponding to 10V.

If Raw Position (2) is selected, the signal driving the EXT OUT (Monitor) BNC is the output voltage of the position demodulator. This signal shows a slight nonlinearity as a function of position and a small offset voltage. As a result it is not as accurate as the linearized position. However, having not undergone any digital processing it is free of any potential digital signal processing effects and can be more advantageous for loop tuning and transient response measurement.

If Linearized Position (3) is selected, the signal driving EXT OUT is linearized and scaled so that the 0 to full range corresponds to 0 to 10 Volts.

The signal on the rear panel EXT OUT BNC can also be filtered to limit the output bandwidth to the range of interest in most closed loop applications, i.e. 200Hz.

The filter is set in the lMonitorOPBandwidth parameter as follows:

No Filter = 1
200 Hz Low Pass Filter = 2

When operating in closed loop mode, the feedback can be supplied by either a Capacitive or a Strain Gauge sensor. The lFeedbackSrc parameter is used to specify the feedback type as follows:

Strain Gauge = 1
Capacitive = 2

The current input and output settings may be obtained using the GetPPCIOSettings method.

Piezo Method *SetPPCNotchParams*

Note. This method is applicable only to the PPC001 Precision Piezo Controller

See Also Example Code

Visual Basic Syntax
Function **SetPPCNotchParams**(lChanID As Long, lFilterNo As Long, fNotch1CentreFreq As Float, fNotch1Q As Float, lNotch1FilterOn As Long, fNotch2CentreFreq As Float, fNotch2Q As Float, lNotch2FilterOn As Long) As Long

Parameters
l*ChanID* - the channel identifier

lFilterNo - the filter number being addressed, either 1, 2 or both
fNotch1CentreFreq - the centre frequency of notch filter 1
fNotch1Q - the Q factor of notch filter 1
lNotch1FilterOn - enables and disables notch filter 1
fNotch2CentreFreq - the centre frequency of notch filter 2
fNotch2Q - the Q factor of notch filter 2
lNotch2FilterOn - enables and disables notch filter 2

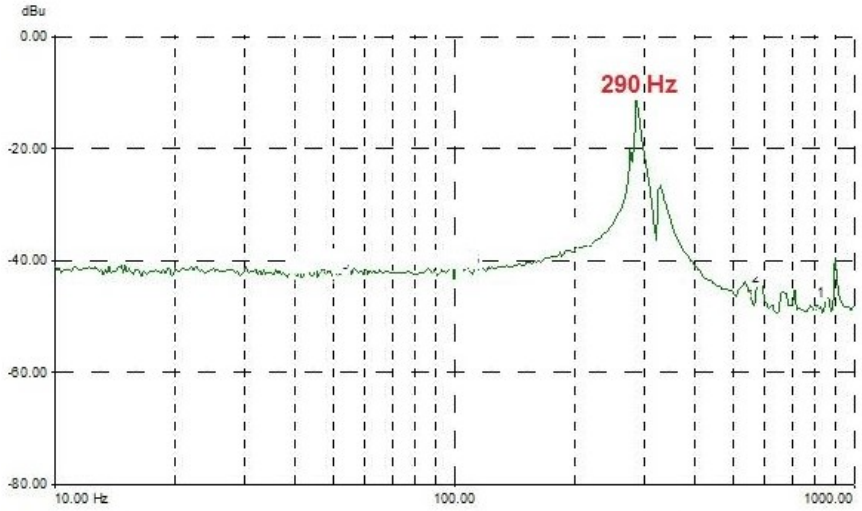
Returns
MG Return Code

Details
This method is used to set parameters associated with the notch filters for the PID loop associated with a particular ActiveX Control instance. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. In the case of the PPC001 controller this should always be set to '1'.

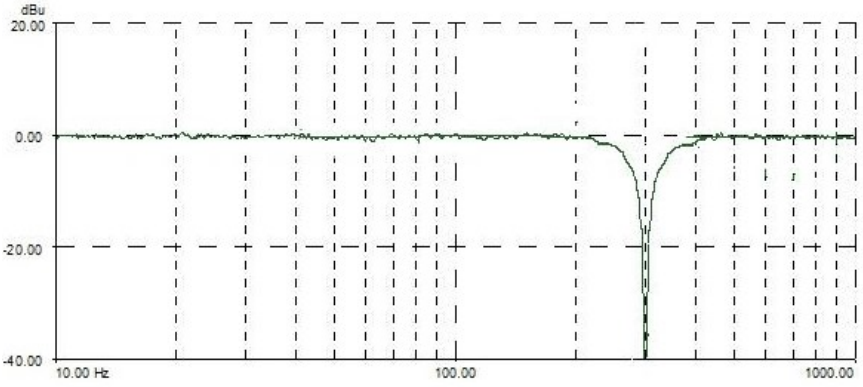
The filter being addressed by the method is specified in the *lFilterNo* parameter as follows:

Notch Filter 1 = 1
Notch Filter 2 = 2
Both Filters = 3

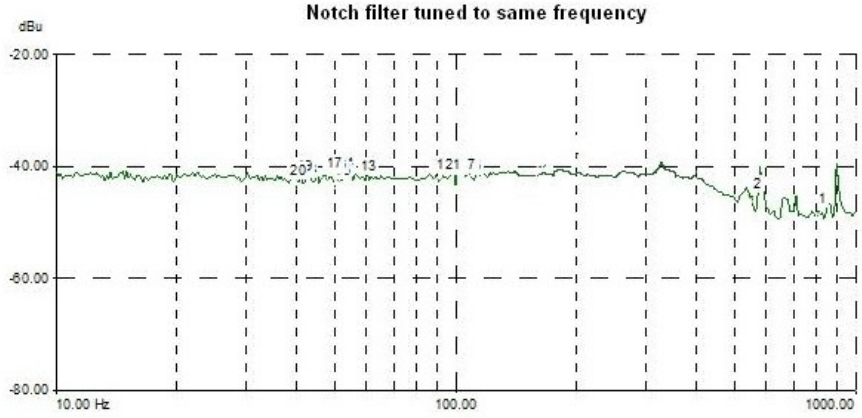
Notch Filter Settings - Due to their construction, most actuators are prone to mechanical resonance at well-defined frequencies. The underlying reason is that all spring-mass systems are natural harmonic oscillators. This proneness to resonance can be a problem in closed loop systems because, coupled with the effect of the feedback, it can result in oscillations. With some actuators, the resonance peak is either weak enough or at a high enough frequency for the resonance not to be troublesome. With other actuators the resonance peak is very significant and needs to be eliminated for operation in a stable closed loop system. The notch filter is an adjustable electronic anti-resonance that can be used to counteract the natural resonance of the mechanical system.



Frequency response of actuator showing resonance at 290 Hz



Notch filter tuned to same frequency



The resonance is largely eliminated

As the resonant frequency of actuators varies with load in addition to the minor variations from product to product, the notch filter is tuneable so that its characteristics can be adjusted to match those of the actuator. In addition to its centre frequency, the bandwidth of the notch (or the equivalent quality factor, often referred to as the Q-factor) can also be adjusted. In simple terms, the Q factor is the centre frequency/bandwidth, and defines how wide the notch is, a higher Q factor defining a narrower ("higher quality") notch. Optimizing the Q factor requires some experimentation but in general a value of 5 to 10 is in most cases a good starting point.

Notch filter 1 is enabled/disabled in the lNotch1FilterOn parameter as follows:

Notch Filter ON = 1
Notch Filter OFF = 2

The centre frequency of notch filter 1 is set in the fNotch1CentreFreq parameter, in the range 20 to 500. The Q Factor is set in the fNotch1Q parameter, in the range 0.2 to 100.

Notch filter 2 is set similarly in the Notch Filter 2 parameters.

The current Notch Filter settings can be obtained by calling the GetPPCNotchParams.

Piezo Method SetPPCPIDParams

Note. This method is applicable only to the PPC001 Precision Piezo Controller

Caution. The parameters contained in this method have been preset to their optimum values and should not require adjustment under normal operation. Altering these values can adversely affect performance of the system. Please contact Thorlabs Tech Support for more information.

See Also Example Code

Visual Basic Syntax
Function **SetPPCPIDParams** (lChanID As Long, fProp As Float, fInt As Float, fDeriv As Float, fDerivLPCutOffFreq As Float, lDerivFilterOn As Long) As Long

Parameters
lChanID - the channel identifier
fProp - the value of the proportional constant
fInt - the value of the integration constant
fDeriv - the value of the differential constant
fDerivLPCutOffFreq - the cut off frequency of the derivative low pass filter
lDerivFilterOn - enables and disables the derivative low pass filter

Returns
MG Return Code

Details.

When operating in Closed Loop mode, the proportional, integral and differential (PID) constants can be used to fine tune the behaviour of the feedback loop to changes in the output voltage or position. While closed loop operation allows more precise control of the position, feedback loops need to be adjusted to suit the different types of focus mount assemblies that can be connected to the system. Due to the wide range of objectives that can be used with the PFM450 and their different masses, some loop tuning may be necessary to optimize the response of the system and to avoid instability.

This method sets values for these PID parameters.The default values have been optimized to work with the actuator shipped with the controller and any changes should be made with caution.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. In the case of the PPC001 controller this should always be set to '1'.

fProp- The Proportional parameter makes a change to the output which is proportional to the current error value. A high proportional gain results in a large change in the output for a given error. If set too high, the system can become unstable. It accepts values in the range 0 to 10000 (default 10)

fInt - The Integral parameter accelerates the process towards the demanded set point value, ensuring that the positional error is eventually reduced to zero. If set too high, the output can overshoot the demand value. It accepts values in the range 0 to 10000. (default 5).

fDeriv - The Derivative parameter damps the rate of change of the output, thereby decreasing the overshoot which may be caused by the integral term. However, the differential term also slows down system response. If set too high, it could lead to instability due to signal noise amplification. It accepts values in the range 0 to 10000 (default 0).

Note. The default values have been optimised to work with the sensor shipped with the controller.
Due to the type of load the PFM450 presents to the PPC001 controller, the system is most sensitive to the integral (Int) tuning parameter and the notch filter settings. In most applications the deriviative parameters can be left at their default values.

fDerivLPCutOffFreq - The output of the derivative (differential) part of the PID controller can be passed through a tuneable low pass filter. Whilst the derivative component of the PID loop often improves stability (as it acts as a retaining force against abrupt changes in the system), it is prone to amplifying noise present in the system, as the derivative component is sensitive to changes between adjacent samples. To reduce this effect, a low pass filter can be applied to the samples. As noise often tends to contain predominantly high frequency components, the low pass filter can significantly decrease their contribution, often without diminishing the beneficial, stabilizing effect of the derivative action. In some applications enabling this filter can improve the overall closed loop performance. The Cut Off Frequency is specified in Hz in the range 0 to 10000.

lDerivFilterOn - enables and disables the low pass filter as follows:
Filter On = 1
Filter Off = 2

The present settings for these parameters can be obtained by calling the GetPPCPIDParams method.

Piezo Method *SetVoltOutput*

See Also Example Code

Visual Basic Syntax

Function **SetVoltOutput**(lChanID As Long, fVoltage As Single) As Long

Parameters

lChanID - the channel identifier

fVoltage - the voltage to be applied to the piezo actuator

Returns

MG Return Code

Details.

This method sets the output voltage applied to the piezo actuator, to the value specified in the *fVoltage* parameter.

Note. On the T-Cube Piezo Drivers (TPZ001), the output voltage can be set up to 150V, on all other units, the maximum voltage is 75V.

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the output voltage of both channels simultaneously, and the use of CHANBOTH_ID is not allowed. On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The voltage currently applied may be obtained using the GetVoltOutput method.

Example

```
Private Sub cmdSetVoltage_Click()

' NB: voltage can only be set in open loop mode

Dim sngVoltage As Single

sngVoltage = CSng(Val(txtVoltage))

MG17Piezo1.SetVoltOutput CHAN1_ID, sngVoltage

End Sub
```

T-Cube Piezo Driver Method *SetVoltageOutputLimit*

See Also Example Code

Note. This method is applicable only to T-Cube Piezo Drivers

Visual Basic Syntax

Function **SetVoltageOutputLimit**(lChanID As Long, lLimit As Single) As Long

Parameters

lChanID - the channel identifier

lLimit - the maximum drive voltage of the piezo actuator

Returns

MG Return Code

Details.

The piezo actuator connected to the T-Cube has a specific maximum operating voltage range. This method sets the maximum output to the value specified in the *lLimit* parameter. This parameter takes values specified by the OUTPUTVOLTAGELIMIT enumeration as follows.

1. VOLTAGELIMIT_75V 75V limit
2. VOLTAGELIMIT_100V 100V limit
3. VOLTAGELIMIT_150V 150V limit

The applicable channel is specified by the *lChanID* parameter which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The current setting for maximum voltage may be obtained using the GetVoltOutputLimit method.

Piezo Method *SetVoltPosDispMode*

See Also Example Code

Visual Basic Syntax

Function **SetVoltPosDispMode**(lChanID As Long, lMode As Long) As Long

Parameters

lChanID - the channel identifier

lMode - the display setting (Volts or Microns)

Returns

MG Return Code

Details.

This method sets the GUI display mode for the channel specified by the *lChanID* parameter, to the value specified in the *plMode* parameter, which in turn takes values from the OPDISPMODE enumeration.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the display mode of both channels simultaneously, and the use of CHANBOTH_ID is not allowed.

The current setting for the GUI display mode may be obtained by calling the GetVoltPosDispMode method

Example

```
Private Sub cmdSetClosedLoop_Click()

' Set closed loop, display mode to position, and zero sensor

MG17Piezo1.SetControlMode CHAN1_ID, CLOSED_LOOP

MG17Piezo1.SetVoltPosDispMode CHAN1_ID, DISP_POS

MG17Piezo1.ZeroPosition CHAN1_ID

End Sub
```

T-Cube Strain Gauge Method *SG_DeleteParamSet*

See Also Example Code

Visual Basic Syntax
Function **SG_DeleteParamSet**(bstrName As String) As Long

Parameters
bstrName – the name of the parameter set to be deleted

Returns
MG Return Code

Details
This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube Strain Gauge Method SG_GetHubAnalogueChanOut

See Also Example Code

Visual Basic Syntax

Function **SG_GetHubAnalogueChanOut**(lChanID As Long, plHubChan As Long) As Long

Parameters

lChanID – the channel identifier

plHubChan – the hub feedback channel being used

Returns

MG Return Code

Details

When the T-Cube Strain Gauge Reader is used in conjunction with the T-Cube Piezo Driver unit (TPZ001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit. High precision closed loop operation is then possible using our complete range of feedback-equipped piezo actuators.

This method returns the feedback channel being used to route signals back to the Piezo unit.

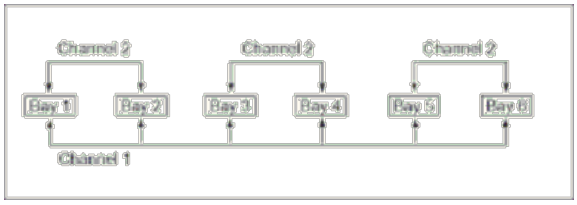
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID

The feedback channel being used is returned in the *plHubChan* parameter, which in turn, takes values from the HUBANALOGUEOUTPUT enumeration as follows:

If *plHubChan* returns HUB_ANALOGUEOUT_1, the feedback signals run through all T-Cube bays.

If *plHubChan* returns HUB_ANALOGUEOUT_2, the feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6). This setting is useful when several pairs of Strain Gauge/Piezo Driver cubes are being used on the same hub.



The feedback channel can be set using the SG_SetHubAnalogueChanOut method.

T-Cube Strain Gauge Method SG_GetDispIntensity

See Also Example Code

Visual Basic Syntax

Function **SG_GetDispIntensity**(lChanID As Long, plIntensity As Single) As Long

Parameters

lChanID - the channel identifier

plIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method returns the current setting for the intensity of the LED display on the front of the unit. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The intensity is returned in the *plIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The display intensity may be set by calling the SG_SetDispIntensity method

T-Cube Strain Gauge Method SG_GetDispMode

See Also Example Code

Visual Basic Syntax

Function **SG_GetDispMode**(lChanID As Long, plMode As Long) As Long

Parameters

lChanID - the channel identifier

plMode - the display setting (Position, Voltage or Force)

Returns

MG Return Code

Details.

The LED display window on the front of the unit (and the display on the GUI panel) can be set to display the strain gauge signal as a position (microns), a voltage (Volts) or as a force (Newtons).

This method returns the display mode for the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The current setting is returned in the *plMode* parameter, which in turn takes values from the DISPUNITSMODE enumeration as follows.

If *plMode* returns *DISPUNITS_POSITION*, the display shows the strain gauge signal as a position in microns.

If *plMode* returns *DISPUNITS_VOLTAGE*, the display shows the strain gauge signal as a voltage.

T-Cube Strain Gauge Method SG_GetForceCalib

See Also Example Code

Visual Basic Syntax

Function **SG_GetForceCalib**(lChanID As Long, plCalib As Long) As Long

Parameters

lChanID - the channel identifier

plCalib – the force calibration factor

Returns

MG Return Code

Details

The Force Sensor connected to the Strain Gauge T-Cube has a specific maximum operating force. This method returns the current setting of the force calibration factor for the channel specified by the *lChanID* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The calibration factor for the type of force sensor being used is returned in the *lCalib* parameter. For example, if set to 1, the GUI digital display shows a detected force of 0 to 1. The default setting for this parameter is 30, to be compatible with our FSC102 force sensor, which is specified to read forces up to 30N.

The Force Calibration factor can be set by calling the SG_SetForceCalib method.

T-Cube Strain Gauge Method SG_GetMaxTravel

See Also Example Code

Visual Basic Syntax

Function **SG_GetMaxTravel**(lChanID As Long, plMaxTravel As Long) As Long

Parameters

lChanID - the channel identifier

plMaxTravel - returns the maximum travel (in microns)

Returns

MG Return Code

Details

In the case of actuators with built in position sensing, the T-Cube Strain Gauge unit can detect the range of travel of the actuator since this information is programmed in the electronic circuit inside the actuator.

This method retrieves the maximum travel for the piezo actuator associated with the channel specified by the *lChanID* parameter, and returns a value (in microns) in the *plMaxTravel* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

T-Cube Strain Gauge Method SG_GetReading

See Also Example Code

Visual Basic Syntax

Function **SG_GetReading**(lChanID As Long, pfReading As Single) As Long

Parameters

lChanID - the channel identifier

pfReading - returns the current position, voltage or force

Returns

MG Return Code

Details.

This method returns the current reading of the strain gauge associated with the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The current reading is returned as a value in the *pfReading* parameter. The units applicable are dependent on the current operating mode (set using the SG_SetDispMode method.

If the unit is operating in Position mode, then *pfReading* returns a position value in microns. If the unit is in Voltage mode, then *pfReading* returns a Voltage. If the controller is in 'Force Sensing Mode' then the *pfReadingn* parameter returns a force value in Newtons.

The returned data values are sampled at 500Hz. This is particularly useful in touch probe or force sensing applications where rapid polling of the force reading is important.

All values are relative to the datum set for the arrangement using the SG_ZeroPosition method.

T-Cube Strain Gauge Method SG_LoadParamSet

See Also Example Code

Visual Basic Syntax

Function **SG_LoadParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be loaded

Returns

MG Return Code

Details

This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube Strain Gauge Method SG_SaveParamSet

See Also Example Code

Visual Basic Syntax

Function **SG_SaveParamSet**(bstrName As String) As Long

Parameters

bstrName – the name under which the parameter set is to be saved

Returns

MG Return Code

Details

This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved. Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

T-Cube Strain Gauge Method SG_SetDispIntensity

See Also Example Code

Visual Basic Syntax

Function **SG_SetDispIntensity**(lChanID As Long, lIntensity As Single) As Long

Parameters

lChanID - the channel identifier

lIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method sets the intensity of the LED display on the front of the unit. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The display intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The current setting for display intensity may be obtained by calling the SG_GetDispIntensity method

T-Cube Strain Gauge Method SG_SetDispMode

See Also Example Code

Visual Basic Syntax

Function **SG_SetDispMode**(lChanID As Long, lMode As Long) As Long

Parameters

lChanID - the channel identifier

lMode - the display setting (Position, Voltage or Force)

Returns

MG Return Code

Details.

The LED display window on the front of the unit (and the display on the GUI panel) can be set to display the strain gauge signal as a position (microns), a voltage (Volts) or as a force (Newtons).

This method sets the display mode for the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The mode is specified in the *lMode* parameter, which in turn takes values from the DISPUNITSMODE enumeration as follows.

If *lMode is set to DISPUNITS_POSITION*, the display shows the strain gauge signal as a position in microns.

If *lMode is set to DISPUNITS_VOLTAGE*, the display shows the strain gauge signal as a voltage.

T-Cube Strain Gauge Method SG_SetForceCalib

See Also Example Code

Visual Basic Syntax

Function **SG_SetForceCalib**(lChanID As Long, lCalib As Long) As Long

Parameters

lChanID - the channel identifier

lCalib – the force calibration factor

Returns

MG Return Code

Details

The Force Sensor connected to the Strain Gauge T-Cube has a specific maximum operating force. This method sets the force calibration factor for the channel specified by the *lChanID* parameter.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The calibration factor for the type of force sensor being used is set in the *lCalib* parameter. For example, if set to 1, the GUI digital display shows a detected force of 0 to 1. The default setting for this parameter is 30, to be compatible with our FSC102 force sensor, which is specified to read forces up to 30N.

The current setting can be obtained by calling the SG_GetForceCalib method.

T-Cube Strain Gauge Method SG_SetHubAnalogueChanOut

See Also Example Code

Visual Basic Syntax

Function **SG_SetHubAnalogueChanOut**(lChanID As Long, lHubChan As Long) As Long

Parameters

lChanID – the channel identifier

lHubChan – the hub channel to be used

Returns

MG Return Code

Details

When the T-Cube Strain Gauge Reader is used in conjunction with the T-Cube Piezo Driver unit (TPZ001) on the T-Cube Controller Hub (TCH001), a feedback signal can be passed from the Strain Gauge Reader to the Piezo unit. High precision closed loop operation is then possible using our complete range of feedback-equipped piezo actuators.

This method is used to select the way in which the feedback signal is routed back to the Piezo unit.

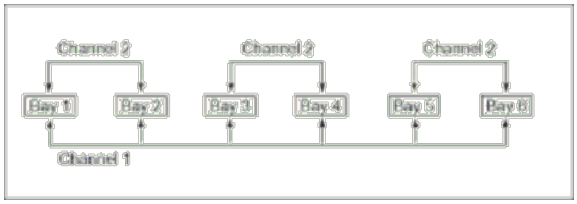
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID

The feedback channel to be used is set in the *lHubChan* parameter, which in turn, takes values from the HUBANALOGUEOUTPUT enumeration as follows:

If *lHubChan* is set to HUB_ANALOGUEOUT_1, the feedback signals run through all T-Cube bays.

If *lHubChan* is set to HUB_ANALOGUEOUT_2, the feedback signals run between adjacent pairs of T-Cube bays (i.e. 1&2, 3&4, 5&6). This setting is useful when several pairs of Strain Gauge/Piezo Driver cubes are being used on the same hub.



The current setting for the feedback channel can be obtained using the SG_GetHubAnalogueChanOut method.

T-Cube Strain Gauge Method SG_ZeroPosition

See Also Example Code

Visual Basic Syntax

Function **SG_ZeroPosition**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details.

This method applies a voltage of zero volts to the strain gauge associated with the channel specified by the *lChanID* parameter, which, in turn takes values specified by the HWCHANNEL enumeration.

Note: On single channel units such as the T-Cubes, the *lChanID* parameter should be set to CHAN1_ID.

The strain gauge signal is then read, and this reading is taken to be the zero reference for all subsequent readings. This routine is typically called during the initialisation or re-initialisation of the piezo/strain gauge arrangement.

Note. Before calling the SG_ZeroPosition method, the OUTPUT control on the associated T-Cube Piezo unit must be turned fully anti-clockwise to zero. If applicable, any signal present on the rear panel EXT IN connector must also be removed.

Piezo Method *StartCtrl*

See Also Example Code

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns

MG Return Code

Details

This method is used in conjunction with the HWSerialNum property, to activate an ActiveX control instance at runtime.

After the HWSerialNum property has been set to the required serial number (at design time or run time), this method can be called at run time to initiate the hardware enumeration sequence and activate the control.

Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

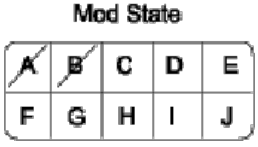
Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

MG17Piezo1.HWSerialNum = 21000001

' Start piezo control

MG17Piezo1.StartCtrl

End Sub
```

Piezo Method *StartOPLUT*

See Also Example Code

Notes for BPC001 or BPC002 Users.
Triggering functionality is only available when operating in open loop (Voltage) mode.
The Input Trigger functionality is not available on BPC001 and BPC002 units.
On units to modification state B or earlier, triggering functionality is enabled by fitting a trigger enable plug. These plugs are available from the company on request.
A label indicating the build and modification state of the hardware is located on the underside of the unit. Modification states are ‘scored out’ as they become current.



Visual Basic Syntax
Function **StartOPLUT** (lChanID As Long) As Long

Parameters
lChanID - the channel identifier

Returns
MG Return Code

Details

This method is used to start the voltage waveform (LUT) outputs. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified in the HWCHANNEL enumeration.

Note for BPC series controller users. If the bIPTrigEnable flag (SetOPLUTTrigParams method) is set to false, this method initiates the waveform immediately. If the bIPTrigEnable flag (SetOPLUTTrigParams method) is set to true, then this method 'arms' the system, in readiness for receipt of an input trigger. **Piezo Method *StopCtrl***

See Also Example Code

Visual Basic Syntax

Function **StopCtrl**() As Long

Returns

MG Return Code

Details

This method is called to deactivate the operation of an ActiveX control instance.

When this method is called, communication to the associated hardware unit is stopped. It is efficient programming practice to call StopCtrl on any ActiveX control instances that are no longer required to be active. For example, this may be the case in a multiple window client application where multiple instances of an ActiveX control need to communicate at different times with the same hardware unit

Example

```
Private Sub Form_Unload(Cancel As Integer)

' Stop system control and unload forms.

frmSystem.MG17System1.StopCtrl

Unload frmSystem

MG17Piezo1.StopCtrl

Unload Me

End Sub
```

Piezo Method *StopOPLUT*

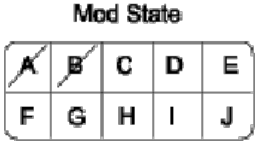
See Also Example Code

Notes for BPC001 or BPC002 Users.

Triggering functionality is only available when operating in open loop (Voltage) mode.

On units to modification state B or earlier, triggering functionality is enabled by fitting a trigger enable plug. These plugs are available from the company on request.

A label indicating the build and modification state of the hardware is located on the underside of the unit. Modification states are 'scored out' as they become current.



Visual Basic Syntax

Function **StopOPLUT** (lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details

This method is used to stop the voltage waveform (LUT) outputs. The applicable channel(s) is specified by the *lChanID* parameter, which takes values specified in the HWCHANNEL enumeration.

Note. This method initiates the waveforms in software when the bIPTrigEnable flag (SetOPLUTTrigParams method) is set to false.

It can be used to stop O/P during a fixed number of waveform cycles (SetOPLUTParams method, lMode parameter set to fixed) before they have completed, or when waveform O/P is set to 'Continuous' mode.

Piezo Method *ZeroPosition*

See Also Example Code

Visual Basic Syntax

Function **ZeroPosition**(lChanID As Long) As Long

Parameters

lChanID - the channel identifier

Returns

MG Return Code

Details.

This method applies a voltage of zero volts to the actuator associated with the channel specified by the *lChanID* parameter, and then reads the position.

This reading is then taken to be the zero reference for all subsequent position readings. This routine is typically called during the initialisation or re-initialisation of the piezo arrangement.

The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.

Note: On dual channel units, the system cannot set the zero position of both piezo actuators simultaneously, and the use of CHANBOTH_ID is not allowed.

Example

```
Private Sub cmdSetClosedLoop_Click()

' Set closed loop, display mode to position, and zero sensor

MG17Piezo1.SetControlMode CHAN1_ID, CLOSED_LOOP

MG17Piezo1.SetVoltPosDispMode CHAN1_ID, DISP_POS

MG17Piezo1.ZeroPosition CHAN1_ID

End Sub
```

Piezo Property APTHelp

See Also Example Code

Visual Basic Syntax

Property APTHelp As Boolean

Returns

MG Return Code

Details

This property specifies the help file that will be accessed when the user presses the F1 key. If APTHelp is set to 'True', the main server helpfile *APTServer* will be launched. If APTHelp is set to 'False', the helpfile is the responsibility of the application programmer.

Piezo Property HWSerialNum

See Also **Example Code**

Visual Basic Syntax

Property HWSerialNum As Long

Returns

MG Return Code

Details

This property specifies the serial number of the hardware unit to be associated with an ActiveX control instance.

The serial number must be set in the *HWSerialNum* property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.

Every APT control unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.

For example, if two or more stepper hardware units are connected to the PC, different instances of the stepper ActiveX Control can be included in a client application . If each of these Control instances is programmed with a unique hardware serial number, then they will communicate with their associated hardware units . In this way, multiple graphical control panels communicating with multiple hardware units can easily be incorporated into a custom client application.

After a serial number has been allocated, an ActiveX control instance must be activated at run time by calling the StartCtrl method.

Note. The factory programmed serial number also has a model type identifier incorporated. This ensures for example that the serial number for an APT stepper motor controller cannot be assigned to an APT NanoTrak ActiveX control. If a serial number is set that is either illegal or for which no hardware unit exists, then the ActiveX control will remain inactive, i.e. the control's GUI will appear 'dead'.

Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

MG17Piezo1.HWSerialNum = 21000001

' Start piezo control
```

```
MG17Piezo1.StartCtrl

End Sub
```

Introduction To Laser Control Object Programming

The ActiveX functionality for the Laser Source and Laser Driver is accessed via the APTLaser Control Object, and provides the functionality required for a client application to control a number of Laser units.
Every hardware unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.
The serial number must be allocated using the HWSerialNum property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.
The methods of the Laser Control Object can then be used to control the Laser Source and Laser Driver units, and activities such as switching between display modes, setting the laser power set point, reading the laser power or current and setting the LED display intensity can be performed.

For details on the operation of the laser source and laser driver units, refer to the manual for the unit, which is available from www.thorlabs.com.

IMPORTANT NOTE FOR LASER DIODE DRIVER USERS.
Before the laser diode controller can be used, it must be configured to operate with the specific laser diode it is intended to drive.

The configuration process consists of the following main steps:
- setting the laser diode polarity,
- setting the maximum laser current,
- adjusting the photodiode range and gain,
- setting the Watts/Amps calibration factor.

These steps only need to be done once for a given laser diode. Once the setup has been completed, the TLD001 stores the corresponding parameters in non-volatile memory and they are loaded automatically at each power-up.
For background information on why these parameters must be set for specific types of laser diode, please see the handbook supplied with the unit.

T-Cube Laser Unit Enumeration DISPUNITSMODE

This enumeration contains constants that specify the units of the digital display.

Constant	Name	Purpose
1.	DISPUNITS_AMPS	The display shows signal as a current in mA.
2.	DISPUNITS_WATTS	The display shows signal as a power in mW.
3.	DISPUNITS_DBM	The display shows signal as a dBm value.

T-Cube Laser Unit Enumeration LDCONTROLMODE

THIS ENUMERATION IS APPLICABLE ONLY TO THE TLD001 T-CUBE LASER DIODE DRIVER

This enumeration contains constants that specify the control mode of the TLD001 unit associated with the ActiveX control instance.

The control mode of the unit is specified by calling the SetControlMode method.

Constant	Name	Purpose
1.	CONTROLMODE_CONSTI	The unit operates in Constant Current mode.
2.	CONTROLMODE_CONSTP	The unit operates in Constant Power mode.

T-Cube Laser Unit Enumeration LDDISPTYPEMODE

THIS ENUMERATION IS APPLICABLE ONLY TO THE TLD001 T-CUBE LASER DIODE DRIVER

For TLD001 Laser Diode Drivers, the LED display window on the front of the unit can be set to display one of 4 operating parameters: current limit (ILIM), laser diode current in mA (ILD), laser diode optical output power in mW (PLD) or photo diode current in mA (IPD).

This enumeration contains constants that specify the units of the digital display. The display mode may be set by calling the SetHWDispUnits method.

Constant	Name	Purpose
1.	DISPTYPE_ILIM	The display shows the present setting for the current limit (ILIM) in mA.
2.	DISPTYPE_ILD	The display shows the present laser diode current (ILD) in mA.
3.	DISPTYPE_PLD	The display shows the present laser diode optical power (PLD) in mW.
4.	DISPTYPE_IPD	The display shows the present photo diode current (IPD) in mA.

T-Cube Laser Unit Enumeration LDPOLARITY

THIS ENUMERATION IS APPLICABLE ONLY TO THE TLD001 T-CUBE LASER DIODE DRIVER

This enumeration contains constants that specify the polarity of the laser diode connected to the TLD001 unit associated with the ActiveX control instance.

The polarity of the laser diode is specified by the SetLaserPolarity method.

Constant	Name	Purpose
1.	POLARITY_CATHODEGROUNDED	Select this value if the laser diode connected to the unit is cathode grounded (CG).
2.	POLARITY_ANODEGROUNDED	Select this value if the laser diode connected to the unit is anode grounded (AG).

T-Cube Laser Unit Enumeration LDPWRCTRLSOURCE

THIS ENUMERATION IS APPLICABLE ONLY TO THE TLD001 T-CUBE LASER DIODE DRIVER

This enumeration contains constants that specify the type of control input.

Constant	Name	Purpose
1.	LDPWR_SWONLY	The unit responds only to software commands and the output to the laser is that set using the SetPowerSetpoint method, or the 'OUTPUT' control on the GUI panel.
2.	LDPWR_EXTONLY	The unit responds only to the analog signal on the rear panel EXT IN SMC connector (0 to 10V).
4	LDPWR_POTONLY	The unit responds only to the front panel potentiometer.

Note. Any combination of the above sources may be added together using a 'Bitwise Or' function. For example, if the *ISource* parameter of the SetControlSrc method is set to '7' then the control source is the sum of the software, external and potentiometer inputs.

T-Cube Laser Unit Enumeration PWRCTRLSOURCE

THIS ENUMERATION IS APPLICABLE ONLY TO THE TLS001 T-CUBE LASER SOURCE

This enumeration contains constants that specify the type of control input.

Constant	Name	Purpose
1.	PWRCTRL_SWONLY	The output is set only by software commands
2.	PWRCTRL_EXT	The output is set by the analog signal on the rear panel EXT IN SMC connector (0 to 10V).
3.	PWRCTRL_SWPOT	The output is set by the sum of the value set in software and that set using the front panel potentiometer.

T-Cube Laser Unit Method DeleteParamSet

See Also Example Code

Visual Basic Syntax

Function **DeleteParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be deleted

Returns

MG Return Code

Details

This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube Laser Unit Method DisableOutput

See Also Example Code

Visual Basic Syntax

Function **DisableOutput**() As Long

Returns

MG Return Code

Details

This method disables the optical output of the laser unit associated with the ActiveX control instance.

Note. The key switch must also be ON and the interlock must be present.

The associated unit is specified by the HWSerialNum property.

The output can be enabled by calling the EnableOutput method.

T-Cube Laser Unit Method DoEvents

See Also Example Code

Visual Basic Syntax

Function **DoEvents**() As Long

Returns

MG Return Code

Details

This method enables the APT server to allow message processing to take place within a client application. For example, if the client application is processing a lengthy loop and is making multiple calls to the server, this can often cause the client application to become non-responsive because other user interface message handling, such as button clicks, cannot be processed. Calling the DoEvents method allows such client application message handling to take place.

Note. The DoEvents method works in the same way as the DoEvents keyword found in Visual Basic.

T-Cube Laser Unit Method EnableOutput

See Also Example Code

Visual Basic Syntax

Function **EnableOutput**() As Long

Returns

MG Return Code

Details

This method enables the optical output of the laser unit associated with the ActiveX control instance.

Note. The key switch must also be ON and the interlock must be present.

The associated unit is specified by the HWSerialNum property.

The output can be disabled by calling the DisableOutput method.

T-Cube Laser Unit Method GetWACalibFactor

See Also Example Code

This method is applicable only to TLD001 Laser Diode Driver T-Cubes

Visual Basic Syntax

Function **GetWACalibFactor**(pfWACalib As Float) As Long

Parameters

pfWACalib – The Amps to Watts conversion factor

Returns

MG Return Code

Details.

Each laser diode has specific relationship between the output power and the photodiode current. This method returns the calibration factor for converting between these two parameters.

The present calibration factor for the type of laser diode being used is returned in the *pfWACalib* parameter. For example, if set to 10, a photodiode current of 1mA produces an output power of 10mW.

The calibration factor may be set by calling the SetWACalibFactor method.

T-Cube Laser Unit Method GetControlMode

See Also Example Code

This method is applicable only to TLD001 Laser Diode Driver T-Cubes

Visual Basic Syntax

Function **GetControlMode** (plMode As Long) As Long

Parameters

plMode – The control mode of the associated laser diode driver unit, CONST I or CONST P.

Returns

MG Return Code

Details

The TLD001 laser diode driver can be operated in either Constant Current or Constant Power mode.

In *Constant Current Mode* (CONST I), a constant drive current is applied to the laser diode. However, due to temperature

fluctuations this does not result in a constant optical power output. As the diode warms up, the optical power will increase noticably from the level at initial switch on. Ambient temperature changes will also effect the output.

This mode is used when the lowest noise and highest response speed is required. Most applications in this mode will also require the temperature to be stabilized by an additional temperature controller. We offer the TTC001 TEC Controller T-Cube for such applications, see www.thorlabs.com for more details.

Constant Power Mode (CONST P) is used to minimize the output power fluctuations described above. This involves a signal from the internal photodiode, integrated into most laser diode packages, being fed back to the TLD001 unit in order to monitor and correct the power output.

An adjustment of the full scale photodiode current in CONST P mode is provided on the unit, in order to compensate for the differences in the photodiode currents between different laser diodes - see the manual supplied with the unit for more information on setting the photodiode current range.

This method is used to obtain the present control mode setting for the laser unit associated with the ActiveX control instance. The mode is returned in the *plMode* parameter, which in turn, takes values from the LDCONTROLMODE enumeration as follows:

If *lMode* is set to CONTROLMODE_CONSTI (1), the unit is operating in constant current mode.

If *lMode* is set to CONTROLMODE_CONSTP (2), the unit is operating in constant power mode.

The control mode may be set by calling the SetControlMode method.

T-Cube Laser Unit Method GetControlSrc

See Also Example Code

Visual Basic Syntax

Function **GetControlSrc** (plSource As Long) As Long

Parameters

plSource - the control source

Returns

MG Return Code

Details

This method returns the present setting for the control source of the laser unit associated with the ActiveX control instance.

The method operates differently depending on whether the laser control object is associated with the T-Cube Laser Source or the T-Cube Laser Driver. The associated unit is specified by the HWSerialNum property.

With either unit, the control source can be specified by calling the SetControlSrc method.

The source(s) which control the output from the laser unit are returned in the *plSource* Parameter.

If the laser control object is associated to a TLS001 Laser Source, the *plSource* parameter takes values from the PWRCTRLSOURCE enumeration as follows:

If PWRCTRL_SWONLY (1) is selected, the unit responds only to software commands and the output to the laser is that set using the SetPowerSetpoint method, or the ‘OUTPUT’ control on the GUI panel.

If PWRCTRL_EXT (2) is selected, the unit responds to the analog signal on the rear panel EXT IN SMC connector (0 to 10V).

If PWRCTRL_SWPOT (3) is selected, the output is the sum of the value set in software and that set using the front panel potentiometer.

If the laser control object is associated to a TLD001 Laser Diode Driver, the *plSource* parameter takes values from the LDPWRCTRLSOURCE enumeration as follows:

If LDPWR_SWONLY (1) is selected, the unit responds only to software commands and the output to the laser is that set using the SetPowerSetpoint method, or the 'OUTPUT' control on the GUI panel.

If LDPWR_EXTONLY (2) is selected, the unit responds only to the analog signal on the rear panel EXT IN SMC connector (0 to 10V).

If LDPWR_POTONLY (4) is selected, the unit responds only to the front panel potentiometer.

Any combination of the above sources may be added together using a 'Bitwise Or' function. For example, if the ISource parameter returns '7' then the control source is the sum of the software, external and potentiometer inputs.

T-Cube Laser Unit Method GetGUIDispUnits

See Also Example Code

Visual Basic Syntax

Function **GetGUIDispUnits**(plUnits As Long) As Long

Parameters

plUnits - the display units

Returns

MG Return Code

Details.

The digital display window on the GUI panel can be set to display the laser output in mW, or dBm.

This method returns the present setting of the display mode of the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The present setting is returned in the *plUnits* parameter, which in turn takes values from the DISPUNITSMODE enumeration as follows.

If *plUnits* returns *DISPUNITS_DBM (1)* the display shows the signal as a dBm value.

If *plUnits* returns *DISPUNITS_WATTS (2)*, the display shows signal as a power in mW.

Note. The GUI panel has a separate display for laser current. The enumeration parameter DISPUNITS_AMPS is only for use when setting the Hardware display units - see SetHWDispUnits .

The display mode may be set by calling the SetGUIDispUnits method.

T-Cube Laser Unit Method GetHWDispUnits

See Also Example Code

Visual Basic Syntax

Function **GetHWDispUnits**(plUnits As Long) As Long

Parameters

plUnits - the display units

Returns

MG Return Code

Details.

The method returns the current setting for the display mode on the associated hardware unit. It operates differently depending

on whether the laser control object is associated with the T-Cube Laser Source or the T-Cube Laser Driver. The associated unit is specified by the HWSerialNum property.

For TLS001 Laser Source units, the LED display window on the front of the unit can be set to display the laser output in mW, mA or dBm.

The display mode is returned in the *plUnits* parameter, which in turn takes values from the DISPUNITSMODE enumeration as follows.

If *plUnits* returns *DISPUNITS_AMPS (1)*, the display shows signal as a current in mA.

If *plUnits* returns *DISPUNITS_WATTS (2)*, the display shows signal as a power in mW.

If *plUnits* returns *DISPUNITS_DBM (3)* the display shows the signal as a dBm value.

For TLD001 Laser Diode Drivers, the LED display window on the front of the unit can be set to display one of 4 operating parameters: current limit (ILIM) in mA, laser diode current in mA (ILD), laser diode optical output power in mW (PLD) or photo diode current in mA (IPD).

The display mode is returned in the *plUnits*, which in turn takes values from the LDDISPTYPEMODE enumeration as follows.

If *plUnits* returns *DISPTYPE_ILIM (1)*, the display shows the present setting for the current limit (ILIM) in mA.

If *plUnits* returns *DISPTYPE_ILD (2)*, the display shows the present laser diode current (ILD) in mA.

If *plUnits* returns *DISPTYPE_PLD (3)*, the display shows the present laser diode optical power (PLD) in mW.

If *plUnits* returns *DISPTYPE_IPD (4)* the display shows the present photo diode current (IPD) in mA.

For both units, The display mode may be set by calling the SetHWDispUnits method.

T-Cube Laser Unit Method GetLaserPolarity

See Also Example Code

This method is applicable only to TLD001 Laser Diode Driver T-Cubes

Visual Basic Syntax

Function **SetLaserPolarity**(plPolarity As Long) As Long

Parameters

plPolarity – The polarity of the associated laser diode (CG or AG)

Returns

MG Return Code

Details

Laser diodes are manufactured in a variety of packages and pin configurations, with or without an internal photodiode. In addition, normally one terminal of the laser diode is connected to the metal case and commoned with either the anode or cathode of the photodiode.

The TLD001 has been designed to drive all the possible configurations but care is needed to ensure that the laser diode is connected to the controller correctly. Although the TLD001 contains reasonable protection against some common wiring errors, such as connecting the laser diode with reverse polarity, some faults, such as accidentally swapping the laser diode and the photodiode can still cause damage to the laser diode module.

For the lowest noise and highest level of protection, the case of the laser diode package should always be connected to the electrical system ground. Since normally the case will also be electrically connected to either the anode or the cathode of the laser diode, this results in two common configurations: anode grounded (AG) and cathode grounded (CG). This can be established from the laser diode data sheet and the device should be connected to the D-type connector on the TLD001 accordingly.

The method returns the present setting of the polarity of the laser diode connected to the TLD001 unit. The polarity is returned in the *plPolarity* parameter, which in turn, takes values from the LDPOLARITY enumeration as follows:

If *plPolarity* returns POLARITY_CATHODEGROUNDED (1), the laser diode connected to the unit is cathode grounded (CG)

If *plPolarity* returns POLARITY_ANODEGROUNDED (2), the laser diode connected to the unit is anode grounded (AG)

The laser diode polarity can be set by calling the SetLaserPolarity method.

T-Cube Laser Unit Method ReadLaserVoltage

See Also Example Code

This method is applicable only to TLD001 Laser Diode Driver T-Cubes

Visual Basic Syntax

Function **ReadLaserVoltage**(plVoltage As Long) As Long

Parameters

plVoltage - returns the laser diode drive voltage in Volts

Returns

MG Return Code

Details

This method retrieves the drive voltage applied to the Laser Diode. The TLD001 laser diode driver associated with the ActiveX control instance is specified by the HWSerialNum property.

The voltage is returned (in Volts) in the *plVoltage* parameter and is derived from the laser diode drive current (ILD) and the Watts/Amps calibration factor, set using the SetWACalibFactor method.

T-Cube Laser Unit Method GetMaxLimits

See Also Example Code

Visual Basic Syntax

Function **GetMaxLimits**(pfPowerLim As Float, pfCurrentLim As Float) As Long

Parameters

pfPowerLim - returns the maximum power limit (in mW)

pfCurrentLim - returns the maximum current limit (in mA)

Returns

MG Return Code

Details

This method retrieves the maximum power (in mW) and current (in mA) limits of the laser unit associated with the ActiveX control instance.

The method operates differently depending on whether the laser control object is associated with the T-Cube Laser Source or the T-Cube Laser Driver. The associated unit is specified by the HWSerialNum property.

For TLS001 Laser Source units, the maximum limits are dependent upon the laser pigtail and differs with each unit (this is due to losses inherent in the manufacturing process).

For TLD001 Laser Diode Drivers, the maximum current limit is dependent upon the laser diode drive current limit, set via the

GUI Settings panel.

The maximum power limit parameter (pfPowerLim) is redundant and will return zero.

T-Cube Laser Unit Method GetPowerSetpoint

See Also Example Code

Visual Basic Syntax

Function **GetPowerSetpoint** (pfSetpoint As Float) As Long

Parameters

pfSetpoint - the value of the setpoint

Returns

MG Return Code

Details

This method returns the output set point of the laser unit associated with the ActiveX control instance.

The method operates differently depending on whether the laser control object is associated with the T-Cube Laser Source or the T-Cube Laser Driver. The associated unit is specified by the HWSerialNum property.

With either unit, the power setpoint can be specified by calling the SetPowerSetpoint method.

Use with T-Cube Laser Source

When used to control the T-Cube laser source, the set point value is returned in the *pfSetpoint* parameter in milliWatts.

Note. The setpoint is returned in mW even if the display is set to read in mA or dBm.

Note. The maximum setpoint value available is dependent upon the laser pigtail and differs with each unit (this is due to losses inherent in the manufacturing process). The max value can be obtained by calling the GetMaxLimits method.

Use with T-Cube Laser Driver

When used to control the T-Cube laser driver, the set point value is returned in the *pfSetpoint* parameter.

When operating in constant current (CONST I) mode, the set point value refers to the laser diode drive current (I LD) The value is returned in mA and ranges from zero to the maximum limit set using the GUI Settings panel.

When operating in constant power (CONST P) mode, the set point value refers to the laser power as derived from the photodiode current (IPD). The value is returned in mW, and ranges from zero to a value dependant on the max power limit set using the PD RANGE switches on the rear panel and the Watt/Amp calibration factor, set using the SetWACalibFactor method.

T-Cube Laser Unit Method GetWavelength

See Also Example Code

This method is applicable only to TLS001 Laser Source T-Cubes

Visual Basic Syntax

Function **GetWavelength**(plWavelength As Long) As Long

Parameters

plWavelength - returns the wavelength of laser output

Returns

MG Return Code

Details

The TLS001 laser source unit is available in two output variants, a 635nm wavelength and a 1550nm wavelength.

This method retrieves the output wavelength of the laser source associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

T-Cube Laser Unit Method Identify

See Also Example Code

Visual Basic Syntax

Function **Identify**() As Long

Returns

MG Return Code

Details

This method allows the hardware unit associated with the ActiveX control instance to be identified. The associated unit is specified by the HWSerialNum property.

When the method is called, the front panel LEDs on the relevant hardware unit will flash for a short period.

T-Cube Laser Unit Method LLGetStatusBits

See Also Example Code

Visual Basic Syntax

Function **LLGetStatusBits** (plStatusBits As Long) As Long

Parameters
plStatusBits – the 32-bit integer containing the status flags.

Returns
MG Return Code

Details
This low level method returns a number of status flags pertaining to the operation of the laser unit associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.
These flags are returned in a single 32 bit integer parameter and can provide additional useful status information to a client application. The individual bits (flags) of the 32 bit integer value are described in the following table.

TLS001 Status Bits

Hex Value	Bit Number	Description
1. x00000001	1.	Laser output enabled state (1 - enabled, 0 - disabled).
2. x00000002	2	Key switch enabled state (1 - enabled, 0 - disabled).
0x00000004	3.	Laser control mode (1 – Power (closed loop), 0 – Current (open loop))
0x00000008	4.	Safety interlock enabled state (1 - enabled, 0 - disabled).
0x00000010	5.	Units mode (1 – mA, 0 - else).
0x00000020	6.	Units mode (1 – mW, 0 - else).
0x00000040	7.	Units mode (1 – dBm, 0 - else).
	8. to 30	For Future Use
0x40000000	31	Error
0x80000000	32	For Future Use

TLD001 Status Bits

Hex Value	Bit Number	Description

1. x00000001	1.		Laser output enabled state (1 - enabled, 0 - disabled).
2. x00000002	2		Key switch enabled state (1 - enabled, 0 - disabled).
0x00000004	3.		Laser control mode (1 – CONST P (closed loop), 0 – CONST I (open loop))
0x00000008	4.		Safety interlock enabled state (1 - enabled, 0 - disabled).
0x00000010	5.		Transimpedance Amplifier range (1 – 10µA, 0 - else).
0x00000020	6.		Transimpedance Amplifier range (1 – 100µA, 0 - else).
0x00000040	7.		Transimpedance Amplifier range (1 – 1mA, 0 - else).
0x00000080	8.		Transimpedance Amplifier range (1 – 10mA, 0 - else).
0x00000100	9.		Laser Diode Polarity (1 – Cathode Grounded, 0 – Anode Grounded)
0x00000200	10.		External SMA Input enabled (1 - enabled, 0 - disabled).
0x00000400	11.		Laser Diode Current Limit Reached (1 – Reached, 0 – Not Reached).
0x00000800	12.		Laser Diode Open Circuit (1 – O/C, 0 – else).
0x00001000	13.		All PSU Voltages are OK) (1 – OK, 0 – Not OK).
0x00002000	14.		TIA Range Over Limit
0x00004000	15.		TIA Range Under Limit
0x80000000	16.	to 32	For Future Use

T-Cube Laser Unit Method LoadParamSet

See Also Example Code

Visual Basic Syntax

Function **LoadParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be loaded

Returns

MG Return Code

Details

This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube Laser Unit Method ReadLaserPower

See Also Example Code

Visual Basic Syntax

Function **ReadLaserPower** (pfPower As Float) As Long

Parameters

pfPower - the returned value of the laser power

Returns

MG Return Code

Details

This method returns the actual output power of the laser unit associated with the ActiveX control instance.

The value is returned in the *pfPower* parameter in milliWatts.

Note. The power is returned in mW even if the display is set to read in mA or dBm.

Note. The maximum value available is dependent upon whether the laser control object is associated with the TLS001 Laser Source or the TLD001 Laser Driver. The associated unit is specified by the HWSerialNum property.

If the associated unit is a TLS001 Laser Source, the maximum value obtained is dependent upon the laser pigtail and differs with each unit (this is due to losses inherent in the manufacturing process). The max value can be obtained by calling the GetMaxLimits method.

If the associated unit is a TLD001 Laser Diode Driver, the maximum value is dependent upon the photodiode current range (IPD), set by the rear panel micro switches, and the Watts/Amp calibration factor, set using the SetWACalibFactor method.

T-Cube Laser Unit Method ReadLaserCurrent

See Also Example Code

Visual Basic Syntax

Function **ReadLaserCurrent** (pfCurrent As Float) As Long

Parameters

pfCurrent - the returned value of the laser current

Returns

MG Return Code

Details

This method returns the actual output current of the laser unit associated with the ActiveX control instance. The value is returned in the *pfCurrent* parameter in milliAmps.

The associated unit is specified by the HWSerialNum property.

Note. If the associated unit is a TLD001 Laser Diode Driver, the maximum value available is dependent upon the laser diode drive current limit, set using the GUI Settings Panel.

T-Cube Laser Unit Method SaveParamSet

See Also Example Code

Visual Basic Syntax

Function **SaveParamSet**(bstrName As String) As Long

Parameters

bstrName – the name under which the parameter set is to be saved

Returns

MG Return Code

Details

This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved. Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

T-Cube Laser Unit Method SetWACalibFactor

See Also Example Code

This method is applicable only to TLD001 Laser Diode Driver T-Cubes

Visual Basic Syntax

Function **SetWACalibFactor**(fWACalib As Float) As Long

Parameters

fWACalib – The Watts to Amps conversion factor

Returns

MG Return Code

Details.

Each laser diode has specific relationship between the output power and the photodiode current. This method sets the calibration factor for converting between these two parameters.

The calibration factor for the type of laser diode being used is set in the *fWACalib* parameter. For example, if set to 10, a photodiode current of 1mA produces an output power of 10mW.

The calibration factor for the particular laser diode being used should be quoted in the associated data sheet. If this is not available, then a test calibration should be performed, using a power meter to measure the output for a known photodiode current - see the handbook supplied with the unit for more details.

The current setting can be obtained by calling the GetWACalibFactor method.

T-Cube Laser Unit Method SetControlMode

See Also Example Code

This method is applicable only to TLD001 Laser Diode Driver T-Cubes

Visual Basic Syntax

Function **SetControlMode** (lMode As Long) As Long

Parameters

lMode – The control mode of the associated laser diode driver unit, CONST I or CONST P.

Returns

MG Return Code

Details

The TLD001 laser diode driver can be operated in either Constant Current or Constant Power mode.

In *Constant Current Mode* (CONST I), a constant drive current is applied to the laser diode. However, due to temperature fluctuations this does not result in a constant optical power output. As the diode warms up, the optical power will increase noticably from the level at initial switch on. Ambient temperature changes will also effect the output.

This mode is used when the lowest noise and highest response speed is required. Most applications in this mode will also require the temperature to be stabilized by an additional temperature controller. We offer the TTC001 TEC Controller T-Cube for such applications, see www.thorlabs.com for more details.

Constant Power Mode (CONST P) is used to minimize the output power fluctuations described above. This involves a signal from the internal photodiode, integrated into most laser diode packages, being fed back to the TLD001 unit in order to monitor and correct the power output.

An adjustment of the full scale photodiode current in CONST P mode is provided on the unit, in order to compensate for the differences in the photodiode currents between different laser diodes - see the manual supplied with the unit for more information on setting the photodiode current range.

The mode is set in the *IMode* parameter, which in turn, takes values from the LDCONTROLMODE enumeration as follows:

If *IMode* is set to CONTROLMODE_CONSTI (1), the unit operates in constant current mode.

If *IMode* is set to CONTROLMODE_CONSTP (2), the unit operates in constant power mode.

The present setting for the control mode may be obtained by calling the GetControlMode method.

T-Cube Laser Unit Method SetControlSrc

See Also Example Code

Visual Basic Syntax

Function **SetControlSrc** (ISource As Long) As Long

Parameters

ISource - the control source

Returns

MG Return Code

Details

This method sets the control source for the laser unit associated with the ActiveX control instance.

The method operates differently depending on whether the laser control object is associated with the T-Cube Laser Source or the T-Cube Laser Driver. The associated unit is specified by the HWSerialNum property.

With either unit, the present setting for the control source can be obtained by calling the GetControlSrc method.

The source(s) which control the output from the laser unit are specified in the *ISource* Parameter.

If the laser control object is associated to a TLS001 Laser Source, the *ISource* parameter takes values from the PWRCTRLSOURCE enumeration as follows:

If PWRCTRL_SWONLY (1) is selected, the unit responds only to software commands and the output to the laser is that set using the SetPowerSetpoint method, or the ‘OUTPUT’ control on the GUI panel.

If PWRCTRL_EXT (2) is selected, the unit responds to the analog signal on the rear panel EXT IN SMC connector (0 to 10V).

If PWRCTRL_SWPOT (3) is selected, the output is the sum of the value set in software and that set using the front panel potentiometer.

If the laser control object is associated to a TLD001 Laser Diode Driver, the *ISource* parameter takes values from the LDPWRCTRLSOURCE enumeration as follows:

If LDPWR_SWONLY (1) is selected, the unit responds only to software commands and the output to the laser is that set using the SetPowerSetpoint method, or the ‘OUTPUT’ control on the GUI panel.

If LDPWR_EXTONLY (2) is selected, the unit responds only to the analog signal on the rear panel EXT IN SMC connector (0 to 10V).

If LDPWR_POTONLY (4) is selected, the unit responds only to the front panel potentiometer.

Any combination of the above sources may be added together using a ‘Bitwise Or’ function. For example, if the ISource parameter is set to ‘7’ then the control source is the sum of the software, external and potentiometer inputs.

T-Cube Laser Unit Method SetDispIntensity

See Also Example Code

Visual Basic Syntax

Function **SetDispIntensity**(lIntensity As Long) As Long

Parameters

lIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method sets the intensity of the LED display on the front of the laser unit associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The current setting for display intensity may be obtained by calling the GetDispIntensity method.

T-Cube Laser Unit Method SetGUIDispUnits

See Also Example Code

Visual Basic Syntax

Function **SetGUIDispUnits**(lUnits As Long) As Long

Parameters

lUnits - the display units

Returns

MG Return Code

Details.

The digital display window on the GUI panel can be set to display the laser power in mW, or dBm.

This method sets the display mode of the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The setting is specified in the *lUnits* parameter, which in turn takes values from the DISPUNITSMODE enumeration as follows.

If *lUnits* is set to *DISPUNITS_DBM (1)* the display shows the signal as a dBm value.

If *lUnits* is set to *DISPUNITS_WATTS (2)* the display shows the signal as a power in mW.

Note. The GUI panel has a separate display for laser current. The enumeration parameter DISPUNITS_AMPS is only for use when setting the Hardware display units - see SetHWDispUnits .

The display mode may be obtained by calling the GetGUIDispUnits method.

T-Cube Laser Unit Method SetHWDispUnits

See Also Example Code

Visual Basic Syntax

Function **SetHWDispUnits**(lUnits As Long) As Long

Parameters

lUnits - the display units

Returns

MG Return Code

Details.

The method specifies the display mode of the associated hardware unit, and operates differently depending on whether the laser control object is associated with the T-Cube Laser Source or the T-Cube Laser Driver. The associated unit is specified by the HWSerialNum property.

For TLS001 Laser Source units, the LED display window on the front of the unit can be set to display the laser output in mW, mA or dBm.

The display mode is specified in the *lUnits* parameter, which in turn takes values from the DISPUNITSMODE enumeration as follows.

If *lUnits* is set to *DISPUNITS_AMPS*, the display shows a current in mA.

If *lUnits* is set to *DISPUNITS_WATTS*, the display shows a power in mW.

If *lUnits* is set to *DISPUNITS_DBM* the display shows a dBm value.

For TLD001 Laser Diode Drivers, the LED display window on the front of the unit can be set to display one of 4 operating parameters: current limit in mA (ILIM), laser diode current in mA (ILD), laser diode optical output power in mW (PLD) or photo diode current in mA (IPD).

The setting is specified in the *lUnits* parameter, which in turn takes values from the LDDISPTYPEMODE enumeration as follows.

If *lUnits* is set to *DISPTYPE_ILIM (1)*, the display shows the present setting for the current limit (ILIM) in mA.

If *lUnits* is set to *DISPTYPE_ILD (2)*, the display shows the present laser diode current (ILD) in mA.

If *lUnits* is set to *DISPTYPE_PLD (3)*, the display shows the present laser diode optical power (PLD) in mW.

If *lUnits* is set to *DISPTYPE_IPD (4)* the display shows the present photo diode current (IPD) in mA.

For both units, the display mode may be obtained by calling the GetHWDispUnits method.

T-Cube Laser Unit Method SetLaserPolarity

See Also Example Code

This method is applicable only to TLD001 Laser Diode Driver T-Cubes

Visual Basic Syntax

Function **SetLaserPolarity**(lPolarity As Long) As Long

Parameters

lPolarity – The polarity of the associated laser diode (CG or AG)

Returns

MG Return Code

Details

Laser diodes are manufactured in a variety of packages and pin configurations, with or without an internal photodiode. In addition, normally one terminal of the laser diode is connected to the metal case and commoned with either the anode or cathode of the photodiode. This can be established from the laser diode data sheet and the device should be connected to the TLD001 accordingly.

This method configures the unit for either an anode grounded or a cathode grounded diode.

The polarity of the laser diode connected to the TLD001 unit is specified in the *lPolarity* parameter, which in turn, takes values from the LDPOLARITY enumeration as follows:

If the laser diode connected to the unit is cathode grounded (CG), then *lPolarity* should be set to POLARITY_CATHODEGROUNDED (1)

If the laser diode connected to the unit is anode grounded (AG), then *lPolarity* should be set to POLARITY_ANODEGROUNDED (2)

See 'Connecting the laser diode' for more information.

The present setting for the laser diode polarity can be obtained by calling the GetLaserPolarity method.

T-Cube Laser Unit Method SetPowerSetpoint

See Also Example Code

Visual Basic Syntax

Function **SetPowerSetpoint** (fSetpoint As Float) As Long

Parameters

fSetpoint - the value of the setpoint

Returns

MG Return Code

Details

This method specifies the output set point of the laser unit associated with the ActiveX control instance.

The method operates differently depending on whether the laser control object is associated with the T-Cube Laser Source or the T-Cube Laser Driver. The associated unit is specified by the HWSerialNum property.

With either unit, the current value for the power setpoint can be obtained by calling the GetPowerSetpoint method.

Use with T-Cube Laser Source

When used to control the T-Cube laser source, the set point value is specified in the *fSetpoint* parameter in milliWatts.

Note. The setpoint is specified in mW even if the display is set to read in mA or dBm.

Note. The maximum setpoint value available is dependent upon the laser pigtail and differs with each unit (this is due to losses inherent in the manufacturing process). The max value can be obtained by calling the GetMaxLimits method.

Use with T-Cube Laser Driver

When used to control the T-Cube laser driver, the set point value is specified in the *fSetpoint* parameter.

When operating in constant current (CONST I) mode, the set point value refers to the laser diode drive current (I LD) The value is specified in mA and ranges from zero to the maximum limit set using the GUI Settings panel.

When operating in constant power (CONST P) mode, the set point value refers to the laser power as derived from the photodiode current (IPD). The value is specified in mW, and ranges from zero to a value dependant upon the max power limit set using the PD RANGE switches on the rear panel and the Watt/Amp calibration factor, set using the SetWACalibFactor method.

T-Cube Laser Unit Method StartCtrl

See Also Example Code

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns

MG Return Code

Details

This method is used in conjunction with the HWSerialNum property, to activate an ActiveX control instance at runtime.

After the HWSerialNum property has been set to the required serial number (at design time or run time), this method can be called at run time to initiate the hardware enumeration sequence and activate the control.

Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

Example

```
Private Sub Form_Load()

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

APTLaser1.HWSerialNum = 86000001

' Start Laser control

APTLaser1.StartCtrl

End Sub
```

T-Cube Laser Unit Method StopCtrl

See Also **Example Code**

Visual Basic Syntax

Function **StopCtrl**() As Long

Returns

MG Return Code

Details

This method is called to deactivate the operation of an ActiveX control instance.

When this method is called, communication to the associated hardware unit is stopped. It is efficient programming practice to call StopCtrl on any ActiveX control instances that are no longer required to be active. For example, this may be the case in a multiple window client application where multiple instances of an ActiveX control need to communicate at different times with the same hardware unit

Example

```
Private Sub Form_Unload(Cancel As Integer)

' Stop laser control and unload forms.

APTLaser1.StopCtrl

Unload Me
```

```
End Sub
```

T-Cube Laser Unit Property HWSerialNum

See Also Example Code

Visual Basic Syntax

Property **HWSerialNum** As Long

Returns

MG Return Code

Details

The APTLaser Control Object can be associated with both the TLS001 Laser Source and the TLD001 Laser Diode Driver.

This property specifies the serial number of the hardware unit to be associated with the ActiveX control instance.

The serial number must be set in the *HWSerialNum* property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.

Every APT control unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.

For example, if two or more T-Cube Laser units are connected to the PC, different instances of the APTLaser ActiveX Control can be included in a client application. If each of these Control instances is programmed with a unique hardware serial number, then they will communicate with their associated hardware units. In this way, multiple graphical control panels communicating with multiple hardware units can easily be incorporated into a custom client application.

After a serial number has been allocated, an ActiveX control instance must be activated at run time by calling the StartCtrl method.

Note. The factory programmed serial number also has a model type identifier incorporated. This ensures for example that the serial number for an APT laser unit cannot be assigned to an APT NanoTrak ActiveX control. If a serial number is set that is either illegal or for which no hardware unit exists, then the ActiveX control will remain inactive, i.e. the control's GUI will appear 'dead'.

Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

APTLaser1.HWSerialNum = 86000001

' (use serial number 86xxxxxx for the Laser Source unit and 64xxxxxx for the Laser Diode
driver unit.)

' Start APTLaser control

APTLaser1.StartCtrl

End Sub
```

T-Cube Laser Unit Property APTHelp

See Also Example Code

Visual Basic Syntax

Property **APTHelp** As Boolean

Returns

MG Return Code

Details

This property specifies the help file that will be accessed when the user presses the F1 key. If APTHelp is set to 'True', the main server helpfile APTServer will be launched. If APTHelp is set to 'False', the helpfile is the responsibility of the application programmer.

Introduction to APT PiezoMotor Controller Programming

The 'APTPZMOTOR' ActiveX Control provides the functionality required for a client application to control one or more of the Inertial Piezo Motor controller units.
A single Piezo Motor ActiveX Control instance can be used to control all four channels of the controller. To specify the particular controller being addressed, every unit is factory programmed with a unique 8-digit serial number. This serial number is key to the operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus. The serial number must be specified using the HWSerialNum property before an ActiveX control instance can communicate with the hardware unit. This can be done at design time or at run time. Note that the appearance of the ActiveX Control GUI (graphical user interface) will change to the required format when the serial number has been entered.
The Methods and Properties of the APTPZMotor ActiveX Control can be used to perform activities such as zeroing stages, absolute and relative moves, and jogging.
The target channel is identified in the *IChanID* parameter. See the HWCHANNEL enumeration for more details.
For details on the operation of the controller, please see the handbook which is available on www.thorlabs.com.

APTPZMOTOR Method Summary

This topic contains a brief description of the methods contained in the APTPZMOTOR ActiveX control. A detailed description of the methods, properties and associated enumerations is given in subsequent topics (see contents list). Each method or property name and associated parameter list is written using Visual Basic syntax.

Note. Some methods use boolean parameters to specify or return certain settings. Boolean parameters in the ActiveX Drivers have been implemented as long (32bit) Integers. In this way when passing boolean parameters a TRUE setting is defined as a non zero value and a FALSE setting is defined as a zero value. When methods return boolean parameters a TRUE setting will be indicated by the value 1 and a FALSE setting by the value 0
.

Method Name	Description
DeleteParamSet	Deletes stored settings for specific controller.
DisableHWChannel	Disables the drive output.
DoEvents	Allows client application to process other activity.
EnableEventDlg	Enables the Event Dialog Box for all methods
EnableHWChannel	Enables the drive output.
GetButtonParams	Gets the front panel button settings (Cube drivers).
GetDriveOPParams	Gets the output parameters.
GetJogMode	Gets the jogging button operating modes.
GetJogStepSize	Gets the jogging step size.
GetJogOPParams	Gets the jogging output parameters.
GetPositionSteps	Gets the current motor position.
GetPotParams	Gets the velocity control potentiometer parameters (Cube drivers).
Identify	Identifies the controller by flashing unit LEDs.
LLGetStatusBits	Gets the status bits encoded in 32 bit integer for the specified channel
LLReqDevParams	Sets the encoder counts for the associated position
LLSetGetDevParams	Sets or Gets the channel card digital output bits encoded in 32 bit integer.
LLSetGetDevParamsEx	Sets or Gets the channel card digital output bits encoded in 32 bit integer.
LoadParamSet	Loads stored settings for specific controller.

MoveAbsoluteStepsEx	Initiates an absolute move with specified number of steps.
MoveJog	Initiates a jog move.
MoveRelativeStepsEx	Initiates a relative move with specified number of steps.
MoveVelocity	Initiates a move at constant velocity with no end point.
SaveParamSet	Saves settings for a specific controller.
SetButtonParams	Sets the front panel button settings (Cube drivers).
SetDriveOPParams	Sets the output parameters.
SetJogMode	Sets the jogging button operating modes.
SetJogStepSize	Sets the jogging step size.
SetJogOPParams	Sets the jogging output parameters.
SetPotParams	Sets the velocity control potentiometer parameters (Cube drivers).
SetPositionSteps	Sets the current motor position.
ShowEventDlg	Display the Event Dialog panel.
ShowSettingsDlg	Display the GUI Settings panel.
StartCtrl	Starts the ActiveX Control (starts communication with controller)
StartCtrl1	Starts the ActiveX Control (starts communication with controller)
StopCtrl	Stops the ActiveX Control (stops communication with controller)
Stop	Stops a move immediately.

Piezo Motor Enumeration *BUTTONMODE*

This enumeration contains constants that specify the operating mode of the front panel buttons when the SetButtonParams method is called.

The front panel buttons can be configured to 'jog' the motor or to move to a previously defined position. If set to Jogging, the jogging parameters for the buttons are taken from the 'Jog' parameters set in the SetJogStepSize and SetJogOPParams methods. If set to Position, the motor will move to the predefined position associated with the button pressed.

Constant	Name	Purpose
1.	BUTTON_MODEJOG	Sets the operating mode to 'Jogging'
2.	BUTTON_MODEGOTO	Sets the operating mode to 'Go To Position'

Piezo Motor Enumeration *CHANENABLEMODE*

This enumeration contains constants that specify which channel or pair of channels is enabled on the KIM101 Piezo Motor hardware unit.

Constant	Name	Purpose
0	NONE	All Channel Disabled
1	SINGLE1	Selects channel 1
2	SINGLE2	Selects channel 2
3	SINGLE3	Selects channel 3
4	SINGLE4	Selects channel 4
5	PAIR12	Selects channels 1 and 2
6	PAIR34	Selects channels 3 and 4

Piezo Motor Enumeration *HWCHANNEL*

This enumeration contains constants that specify individual channels on an APT Piezo Motor hardware unit.

Constant	Name	Purpose

0	CHAN1_ID	Selects channel 1
1	CHAN2_ID	Selects channel 2
2	CHAN3_ID	Selects channel 3
3	CHAN4_ID	Selects channel 4
10	CHANBOTH_ID	Selects both channels (dual channel units only)

Notes.
The methods applicable to the TIM001 and KIM101 Piezo Motor Controllers do not accept CHANBOTH_ID as a valid value. In this case, an error value is returned.

Piezo Motor Enumeration *JOGDIR*

This enumeration contains constants that specify the direction moved when a jog command is initiated on the channel, specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	JOG_FWD	Moves in the forward direction
2.	JOG_REV	Moves in the reverse direction

Piezo Motor Enumeration *JOGMODE*

Jog commands can be issued by calling the MoveJog method, via the Motor Control GUI panel or by using the buttons on the unit.

This enumeration contains constants that specify the jog mode applicable to the channel specified in the HWCHANNEL enumeration.

Constant	Name	Purpose
1.	JOG_CONTINUOUS	When a jog command is received, the motor continues to move until the jog signal is removed (i.e the jog button is released).
2.	SINGLE_STEP	When a jog command is received, the motor moves by the step size specified in the SetJogStepSize method (or the GUI).

Piezo Motor Enumeration JOYSTICKDIRSENSE

This enumeration contains constants used by the SetKCubePanelParams method, that specify the direction sense of a move initiated by the joystick on the top panel of the unit.

Constant	Name	Purpose
0	KPZMOT_JS_DIRSENSE_DISABLED	The wheel is disabled.
1	KPZMOT_JS_DIRSENSE_POS	Upwards rotation of the wheel results in a positive motion (i.e. increased position count).
2	KPZMOT_JS_DIRSENSE_NEG	The following option applies only when the JSMode parameter is set to Velocity Control Mode (1) the SetKCubePanelParams method. If set to Jog Mode (2) or Go to Position Mode (3), the following option is ignored. Upwards rotation of the wheel results in a negative motion (i.e. decreased position count).

Piezo Motor Enumeration *JOYSTICKMODE*

This enumeration contains constants used by the SetKCubeJogParams message, that specify the mode of operation of the top panel joystick.

Constant	Name	Purpose
1	Velocity	Deflecting the joystick starts a move with the velocity proportional to the deflection. The maximum velocity (i.e. velocity corresponding to the full deflection of the joystick) is specified in the *plJSMaxStepRate* parameter of the SetKCubeJogParams message.
2	Jog	Deflecting the joystick initiates a jog move, using the parameters specified by the SetKCubeJogParams message. Keeping the joystick deflected repeats the move automatically after the current move has completed.
3	GoToPosition	Deflecting the joystick starts a move from the current position to one of the two predefined “teach” positions. The teach positions are specified in number of steps from the zero position in the PresetPos1 and PresetPos2 parameters. In this mode, move the joystick left (Ch1 and 3) or up (Ch 2

and 4) to go to position 1, and right or down to go to position 2

Piezo Motor Enumeration TRIGMODE

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating mode of the trigger.

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a relative, absolute or home move as follows:

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Constant	Name	Purpose
0X00	KPZMOT_TRIG_DISABLE	The trigger IO is disabled
0X01	KPZMOT_TRIGIN_GPI	General purpose logic input (read through status bits using the LLGetStatusBits method).
0X02	KPZMOT_TRIGIN_RELMOVE	Input trigger for relative move.
0X03	KPZMOT_TRIGIN_ABSMOVE	Input trigger for absolute move.
0X04	KPZMOT_TRIGIN_RESETCOUNT	Input trigger for a count reset.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output, or to indicate motion status or to produce a trigger pulse at configurable positions as follows:

Constant	Name	Purpose
0X0A	KPZMOT_TRIGOUT_GPO	General purpose logic output (set using the LLSetGetDigOPs method).
0X0B	KPZMOT_TRIGOUT_INMOTION	Trigger output active (level) when motor 'in motion'. The output trigger goes high (5V) or low (0V) (as set in the SetKCubeTriggerParams method.
0X0C	KPZMOT_TRIGOUT_MAXVELOCITY	Trigger output active (level) when motor at 'max velocity'.
0X0D	KPZMOT_TRIGOUT_POSSTEPS_FWD	Trigger output active (pulsed) at pre-defined positions moving forward (set using the SetKCubePosTriggerParams method). Only one Trigger port at a time can be set to this mode.
0X0E	KPZMOT_TRIGOUT_POSSTEPS_REV	Trigger output active (pulsed) at pre-defined positions moving backwards (set using the SetKCubePosTriggerParams method). Only one Trigger port at a time can be set to this mode.
0X0F	KPZMOT_TRIGOUT_POSSTEPS_BOTH	Trigger output active (pulsed) at pre-defined positions moving forwards and backward. Only one Trigger port at a time can be set to this mode.
0X10	KPZMOT_TRIGOUT_FWDLIMIT	Trigger output active (level) when the FWD limit switch is activated.
0X11	KPZMOT_TRIGOUT_REVLIMIT	Trigger output active (level) when the REV limit switch is activated.
0X12	KPZMOT_TRIGOUT_EITHERLIMIT	Trigger output active (level) when either the FWD or REV limit switch is activated.

Piezo Motor Enumeration *TRIGPOLARITY*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating polarity of the trigger.

Constant	Name	Purpose
0X01	KPZMOT_TRIGPOL_HIGH	The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).
0X02	KPZMOT_TRIGPOL_LOW	The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

Piezo Motor Method *DeleteParamSet*

Visual Basic Syntax

Function **DeleteParamSet**(bstrName As String) As Long

Parameters
bstrName – the name of the parameter set to be deleted

Returns
MG Return Code

Details
This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed.

Piezo Motor Method *DoEvents*

Visual Basic Syntax
Function **DoEvents**() As Long

Returns
MG Return Code

Details
This method enables the APT server to allow message processing to take place within a client application. For example, if the client application is processing a lengthy loop and is making multiple calls to the server, this can often cause the client application to become non-responsive because other user interface message handling, such as button clicks, cannot be processed. Calling the DoEvents method allows such client application message handling to take place.
Note. The DoEvents method works in the same way as the DoEvents keyword found in Visual Basic.

Example

```
Private Sub cmdDoEvents_Click()
' Set the global looping flag to begin looping.
m_blooping = True
' show hour glass mouse pointer
MousePointer = 11
' Begin lengthy processing loop.
Do while m_blooping
' Initiate a relative move and wait until complete.
APTPZMotor1.MoveRelativeStepsEx CHAN1_ID, 0.1, 0.1, True
' Before looping to initiate relative move,
' call DoEvents to allow other client application
' message handling (e.g. button click handling to
' allow m_blooping to be set to false) to take place.
APTPZMotor1.DoEvents
Loop
' Show standard mouse pointer.
MousePointer = 0
```

Piezo Motor Method *EnableHWChannel*

Visual Basic Syntax
Function **EnableHWChannel**(lChanID As Long) As Long

Parameters
lChanID - the channel identifier

Returns
MG Return Code

Details
This method enables the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.
After a channel is enabled, it is good practice to zero the motor, thereby restoring positional integrity.

It is possible to enable only one channel at a time, therefore a channel is automatically disabled when another channel is enabled.

Piezo Motor Method *GetAbsMovePos*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **GetAbsMovePos**(lChanID As Long, plAbsPos As Long) As Long

Parameters
lChanID - the channel identifier
plAbsPos - the absolute position to move to.

Returns
MG Return Code

Details

This method is used to obtain the relative distance moved on receipt of an input trigger, when the trigger mode is set to ABSMOVE - 03 in the SetKCubeTriggerParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The absolute position to move is returned (in position steps) in the *plAbsPos* parameter.

The relative move distance can be set by calling the SetAbsMovePos method.

Piezo Motor Method GetButtonParams

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **GetButtonParams**((lChanID As Long, plButMode As Long, plButPos As Long, plButPos2 As Long) As Long

Parameters
lChanID – the channel identifier
plButMode – the operating mode of the front panel buttons
plButPos1 – the absolute position associated with the left hand button
plButPos2 – the absolute position associated with the right hand button

Returns
MG Return Code

Details
This method gets the operation mode of the front panel buttons for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The buttons on the front of the unit can be used either to jog the motor, or to perform moves to absolute positions.
The mode of operation is returned in the *plButMode* parameter which in turn takes values from the BUTTONMODE enumeration as follows:

BUTTON_MODEJOG: In this mode, the front panel buttons are used to jog the motor. Once set to this mode, the move parameters for the buttons are taken from the ‘Jog’ parameters set via the ‘Move/Jogs' settings tab or the SetJogStepSize and SetJogOPParams methods.

BUTTON_MODEPOSITION: In this mode, each button can be programmed with a different position value, such that the controller will move the motor to that position when the specific button is pressed.

The following parameters are applicable only if ‘BUTTON_MODEPOSITION is selected in the ‘plButMode’ parameter. In each

case, the position is set in number of steps, measured from the zero position.

plButPos1: The position to which the motor will move when the top button is pressed.

plButPos2: The position to which the motor will move when the bottom button is pressed.

The button parameters can be set by calling the SetButtonParams method.

Piezo Motor Method GetDriveOPParams

Visual Basic Syntax
Function **GetDriveOPParams**(lChanID As Long, plVoltage As Long, plStepRate As Long, plStepAccn As Long) As Long

Parameters
lChanID - the channel identifier
plVoltage - the maximum piezo drive voltage
plStepRate - the velocity to move when a command is initiated
plStepAccn - the acceleration up to the step rate

Returns
MG Return Code

Details
This method returns various drive parameters which define the speed and acceleration of moves initiated in the following ways:

by clicking in the position display
via the panel buttons when 'Go To Position' mode is selected
via software using the MoveVelocity, MoveAbsoluteStepsEx or MoveRelativeStepsEx methods.

Drive parameters for Jog moves are returned in the GetJogOPParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The maximum piezo drive voltage is returned in the plVoltage parameter, in the range 80V to 130V.

The piezo motor moves by ramping up the drive voltage to the value set in the lVoltage parameter and then dropping quickly to zero, then repeating. One cycle is termed a step. The velocity to move when a command is initiated is returned in the plStepRate parameter. The step rate is returned in steps/sec, in the range 1 to 2,000.

The acceleration up to the step rate of the drive command is specified in the plStepAccn parameter in the range 1 to 100,000 cycles/sec/sec.

The drive parameters can be set by calling the SetDriveOPParams method.

Piezo Motor Method *GetKCubeChanEnableMode*

This method is applicable only to K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax
Function **GetKCubeChanEnableMode** (plMode As Long) As Long

Parameters
plMode - the channel mode, either single channels, or channel pairs

Returns
MG Return Code

Details

The K-Cube piezo motor controllers have four drive output channels, and in some applications (e.g. if the actuators are fitted to a 2-axis mirror mount), it may be advantageous to move two axes at the same time by moving the joystick diagonally. The method allows channels to be enabled individually, or in pairs.

The current mode of operation is returned in the *plMode* parameter, which in turn takes values from the CHANENABLEMODE Enumeration as follows:

00 - None, i.e. all channels disabled
01 - Channel 1 enabled
02 - Channel 2 enabled
03 - Channel 3 enabled
04 - Channel 4 enabled
05 - Channels 1 and 2 enabled
06 - Channels 3 and 4 enabled

To specify the channel enable mode, see the SetKCubeChanEnableMode method.

Piezo Motor Method *GetKCubeFeedbackSig*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **GetKCubeFeedbackSig**(lChanID As Long, plFBMode As Long, plEncoderConst As Long) As Long

Parameters
lChanID - the channel identifier
plFBMode - the type of digital input providing the feedback

The following parameter is not implemented at this time and is ignored.
plEncoderConst - the calibration constant for converting encoder counts to real world units

Returns
MG Return Code

Details

The USER IO connector on the rear panel exposes two pairs of four digital inputs. These inputs can be used by a drive channel to receive a signal from the actuator being driven, either a differential QEP encoder feedback signal, or the FWD and REV limit switch signals. This sub message sets up the QEP/Limit switch selection for a specified channel..

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The feedback mode is returned in the *plFBMode* parameter, which takes values specified by the FEEDBACKMODE enumeration as follows:

00 - Disabled - The digital inputs are disabled
01 - LIMSWITCH - The inputs accept a signal when the limit switches are activated.

The following option is for future use and is not implemented at this time.

02 – ENCODER. The inputs accept a feedback signal from the encoder in the actuator

The following parameter is for future use and is not implemented at this time. The description is provided for information only.

If the *FBMode* parameter above is set to Encoder 02, the *plEncoderConst* parameter returns the calibration constant for converting encoder counts to real world units (mm or degrees) for the actuator being driven.

The Feedback signal mode can be set by calling the SetKCubeFeedbackSig method.

Piezo Motor Method *GetKCubeJogParams*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **GetKCubeJogParams**(lChanID As Long, plJogMode As Long,plJogStepSizeFwd As Long, plJogStepSizeRev As Long, plJogStepRate As Long, plJogStepAccn As Long) As Long

Parameters
lChanID - the channel identifier
plJogMode - the jogging mode
plJogStepSizeFwd- the number of pulses which make up a forward jog step
plJogStepSizeRev - the number of pulses which make up a reverse jog step
plJogStepRate - the velocity to move when a command is initiated
plJogStepAccn - the acceleration up to the step rate

Returns
MG Return Code

Details

This method returns various parameters which define the speed and acceleration of moves initiated in the following ways:

by clicking the jog buttons on the GUI panel

by moving the joystick on the unit when 'Jog Mode' mode is selected in the SetKCubePanelParams method.

via software using the MoveJog method.

It differs from the normal motor jog message in that there are two jog step sizes, one for forward and one for reverse. The reason for this is that due to the inherent nature of the PIA actuators going further in one direction as compared with another this will allow the user to potentially make adjustments to get fore and aft movement the same or similar.

Drive parameters for other moves are returned in the GetDriveOPParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The jogging mode is returned in the *plJogMode* parameter, which takes values specified by the JOGMODE enumeration as follows:

01 - Continuous
02 - Single Step

When a jog command is received, if the jog mode is set to 'Single Step' the motor moves by the step size specified in the JogStepSizeFwd and JogStepSizeRev parameters. If the jog mode is set to 'Continuous', the motor continues to move until the jog signal is removed (i.e. the joystick is released) when the motor will stop immediately.

A jog step consists of the number of drive pulses and the size of a jog step is specified by the number of pulses contained in a step. This is returned in the JogStepSizeFwd and JogStepSizeRev parameters in the range 1 to 2,000.

The piezo motor moves by ramping up the drive voltage (to the value set in the SetDriveOPParams method) and then dropping quickly to zero, then repeating. One cycle is termed a step. The velocity to move when a jog command is initiated is returned in the plJogStepRate parameter. The step rate is specified in steps/sec, in the range 1 to 2,000.

The acceleration up to the step rate of the drive command is specified in the plJogStepAccn parameter in the range 1 to 100,000 cycles/sec/sec.

The jog output parameters can be set by calling the SetKCubeJogParams method.

Piezo Motor Method GetKCubePanelParams

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax
Function **GetKCubePanelParams** (lChanID As Long, plJSMode As Long, plJSMaxStepRate As Long, plJSDirSense As Long, plPresetPos1 As Long, plPresetPos2 As Long, plDispBrightness As Long,) As Long

Parameters
lChanID - the channel identifier
plJSMode - The operating mode of the top panel joystick
plJSMaxStepRate - The max velocity of a move initiated by the top panel joystick (i.e. the max step rate for full joystick deflection)
plJSDirSense - The direction sense of a move initiated by the top panel joystick
plPresetPos1 - The preset position 1 when operating in go to position mode
plPresetPos2 - The preset position 2 when operating in go to position mode
plDispBrightness - The display brightness when the unit is active

Returns
MG Return Code

Details
This method returns the operating parameters of the joystick on the top panel of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The operating mode is returned in the plJSMode parameter, which in turn takes values from the JOYSTICKMODE enumeration as follows:

1 Velocity Control Mode - Deflecting the joystick starts a move with the velocity proportional to the deflection. The maximum velocity (i.e. velocity corresponding to the full deflection of the joystick) is specified in the plJSMaxStepRate parameter following.

2 Jog Mode - Deflecting the joystick initiates a jog move, using the parameters specified by the SetKCubeJogParams message. Keeping the joystick deflected repeats the move automatically after the current move has completed.

3 Go To Position Mode - Deflecting the joystick starts a move from the current position to one of the two predefined “teach” positions. The teach positions are specified in number of steps from the zero position in the PresetPos1 and PresetPos2 parameters. In this mode, move the joystick left (Ch1 and 3) or up (Ch 2 and 4) to go to position 1, and right or down to go to position 2.

The plJSMaxStepRate parameter returns the max velocity of a move initiated by the top panel joystick (i.e. the max step rate for full joystick deflection), in the range 1 to 2000 position steps/sec.

The direction of a move initiated by the joystick is specified in the plJSDirSense parameter, which in turn takes values from the JOYSTICKDIRSENSE enumeration as follows:

0 Joystick initiated moves are disabled. The joystick used for menu scrolling only.

1 Upwards/Right deflection of the joystick results in a positive motion (i.e. increased position count).

The following option applies only when the JSMode is set to Velocity Control Mode (1). If set to Jog Mode (2) or Go to Position Mode (3), the following option is ignored.

2 Upwards/Right deflection of the joystick results in a negative motion (i.e. decreased position count).

The following parameters apply only when the JSMode parameter is set to Go to Position Mode (3). If set to Velocity Control Mode (1) or Jog Mode (2), the following preset position parameters are ignored.

plPresetPos1 and plPresetPos2 return the associated position values in position steps from the home position.

In certain applications, it may be necessary to adjust the brightness of the LCD display on the top of the unit. The brightness is returned in the plDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again.

The panel parameters may be set by calling the SetKCubePanelParams method.

Piezo Motor Method GetKCubeTriggerParams

This method is applicable only to K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax
Function **GetKCubeTriggerParams** (plTrigChan1, plTrigChan2, plTrig1Mode As Long, plTrig1Polarity As Long, plTrig2Mode As Long, plTrig2Polarity As Long) As Long

Parameters
plTrigChan1 - The drive channel associated with trigger channel 1 (IO 1)
plTrigChan2 - The drive channel associated with trigger channel 2 (IO 2)
plTrig1Mode - IO 1 operating mode
plTrig1Polarity - The active state of IO 1 (i.e. logic high or logic low)
plTrig2Mode - IO 2 operating mode
plTrig2Polarity - The active state of IO 2 (i.e. logic high or logic low)

Returns
MG Return Code

Details
The KIM101 K-Cube inertial piezo motor controller has two bidirectional trigger ports (I/O 1 and I/O 2) that can be used as a general purpose digital input/output, or can be configured to output a logic level to control external equipment.
Either of them can be independently configured as an input or an output and the active logic state can be selected High or Low to suit the requirements of the application. Electrically the ports output 5 Volt logic signals and are designed to be driven from a 5 Volt logic.

When the port is used in the input mode, the logic levels are TTL compatible, i.e. a voltage level less than 0.8 Volt will be recognised as a logic LOW and a level greater than 2.4 Volt as a logic HIGH. The input contains a weak pull-up, so the state of the input with nothing connected will default to a logic HIGH. The weak pull-up feature allows a passive device, such as a mechanical switch to be connected directly to the input.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method returns the current operating parameters of the IO1 and IO2 connectors on the front panel of the unit.

The unit has 4 drive channels but only two trigger ports. The drive channel associated with each trigger port is returned in the plTrigChan1 and plTrigChan2 parameters.

The operating mode of the trigger is returned in the plTrig1Mode and plTrig2Mode parameters, which takes values from the TRIGMODE enumeration as follows:

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a relative or absolute move, or for starting a position count reset as follows:

0x00 DISABLED: The trigger IO is disabled

0x01 GPI: General purpose logic input (read through status bits using the LLGetStatusBits method).

0x02 RELMOVE: Input trigger for relative move.

0x03 ABSMOVE: Input trigger for absolute move.

0x04 RESETCOUNT: Input trigger for a count reset.

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output, or to indicate motion status or to produce a trigger pulse at configurable positions as follows:

0x0A GPO: General purpose logic output (set using the LLSetGetDigOPs method).

0x0B INMOTION: Trigger output active (level) when motor 'in motion'. The output trigger goes high (5V) or low (0V) (as set in the lTrig1Polarity and lTrig2Polarity parameters) when the stage is in motion.

0x0C MAXVELOCITY: Trigger output active (level) when motor at 'max velocity'.

0x10 FWDLIMIT - Trigger output active (level) when the FWD limit switch is activated.

0x11 REVLIMIT - Trigger output active (level) when the REV limit switch is activated.

0x12 EITHERLIMIT - Trigger output active (level) when the either the FWD or REV limit switch is activated.

0x0D POSSTEPS_FWD: Trigger output active (pulsed) at pre-defined positions moving forward (set using wStartPosFwd, lIntervalFwd, wNumPulsesFwd and lPulseWidth members). Only one Trigger port at a time can be set to this mode.

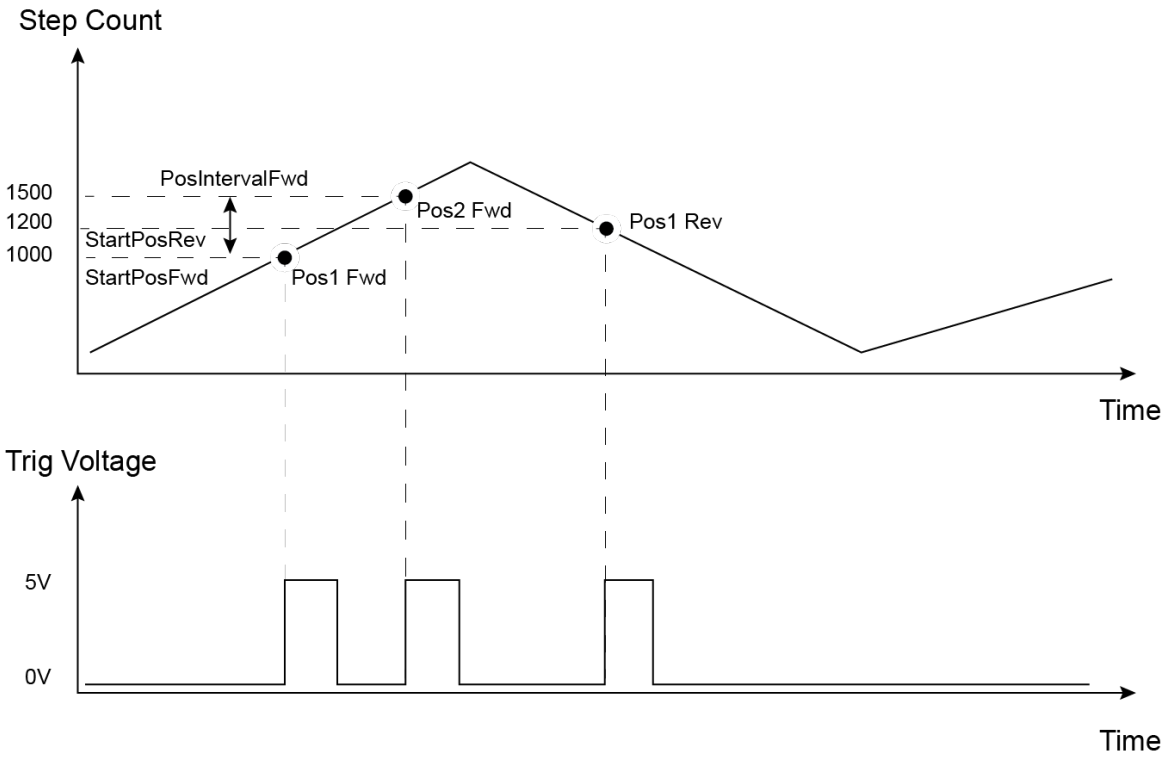
0x0E POSSTEPS_REV: Trigger output active (pulsed) at pre-defined positions moving backwards (set using wStartPosRev, lIntervalRev, wNumPulsesRev and lPulseWidth members). Only one Trigger port at a time can be set to this mode.

0x0F POSSTEPS_BOTH: Trigger output active (pulsed) at pre-defined positions moving forwards and backward. Only one Trigger port at a time can be set to this mode.

Trigger Out Position Steps

In the last three modes described above, the controller outputs a configurable number of pulses, of configurable width, when the actual position of the stage matches the position values configured as the Start Position and Position Interval - see SetKCubePosTriggerParams method. These mode allow external equipment to be triggered at exact position values.

Position triggering can be configured to be unidirectional (forward or revese only) or bidirectional (both). In bidirectional mode the forward and reverse pulse sequences can be configured separately. A cycle count setting (set in the SetKCubePosTriggerParams method, *lNumCycles* parameter) allows the uni- or bidirectional position triggering sequence to be repeated a number of times.



Example for a move from 0 to 2000 position steps.

In forward direction: The first trigger pulse occurs at 1000 steps (StartPosFwd), the next trigger pulse occurs after another 500 steps (PosIntervalFwd), the stage then moves to 2000 steps.

In reverse direction: The next trigger occurs when the stage gets to 12oo steps.

Please note that position triggering can only be used on one TRIG port at a time.

The operation of the position triggering mode is described in more detail in the SetKCubePosTriggerParams method.

The polarity of the trigger pulse is returned in the plTrig1Polarity and plTrig2Polarity parameters, which takes values from the TRIGPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

For details on setting the trigger parameters, see the SetKCubeTriggerParams method.

Piezo Motor Method *GetKCubePosTriggerParams*

This method is applicable only to K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax
Function GetKCubePosTriggerParams (lChanID As Long, plStartPosFwd As Long, plPosIntervalFwd As Long, plNumPulsesFwd As Long, plStartPosRev As Long, plPosIntervalRev As Long, plNumPulsesRev As Long, plPulseWidth As Long, plNumCycles As Long) As Long

Parameters
lChanID - the channel identifier
plStartPosFwd - Trigger pulse output position start forward [in position steps].
plPosIntervalFwd - Trigger pulse output position interval forward [in position steps].
plNumPulsesFwd - Number of forward output pulses during a move.
plStartPosRev - Trigger pulse output position start backward [in position steps].
plPosIntervalRev - Trigger pulse output position interval backward [in position steps].
plNumPulsesRev - Number of backward output pulses during a move.
plPulseWidth - Trigger output pulse width (from 1 µs to 100000 µs).
plNumCycles - Number of forward/reverse move cycles.

Returns
MG Return Code

Details

The K-Cube piezo motor controllers have two bidirectional trigger ports (I/O 1 and I/O 2) that can be set to be used as input or output triggers. This method obtains the current settings for the operating parameters used when the triggering mode is set to a trigger out position steps mode by calling the SetKCubeTriggerParams method.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration as follows:

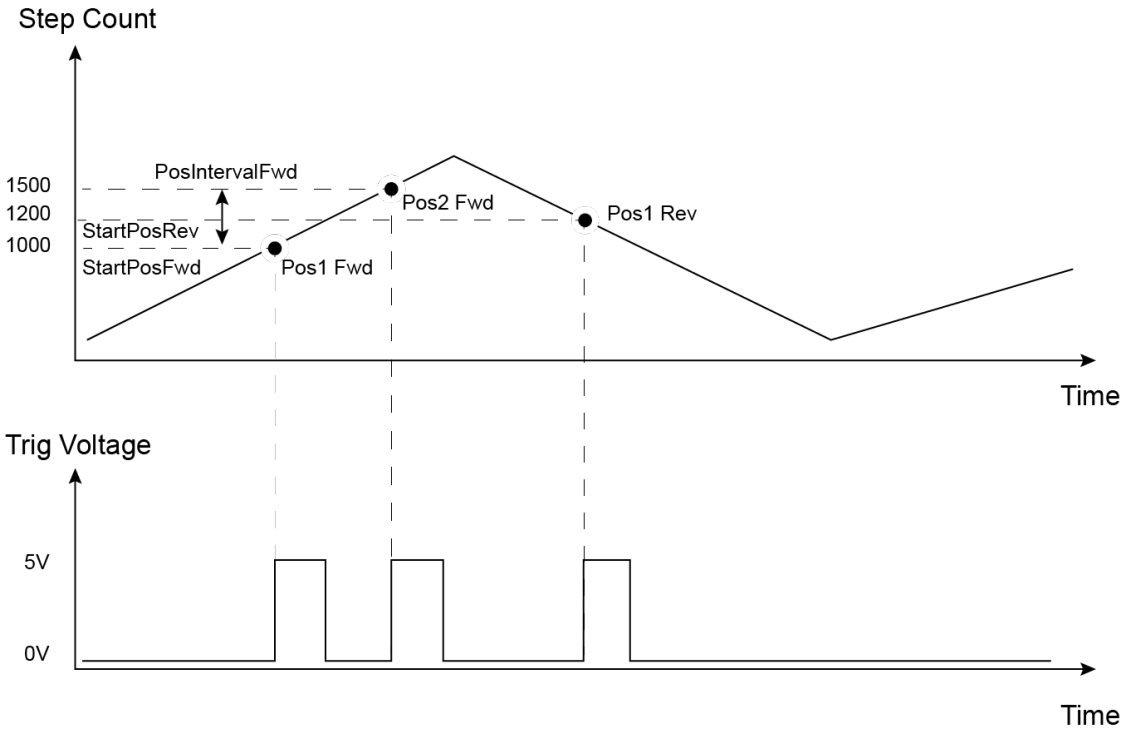
01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

As soon as position triggering is selected on either of the TRIGGER ports, the port will assert the inactive logic state, set in the SetKCubeTriggerParams method. As the stage moves in its travel range and the actual position matches the position returned in the plStartPosFwd parameter, the TRIG port will output its active logic state. The active state will be output for the length of time returned by the plPulseWidth parameter, then return to its inactive state and schedule the next position trigger point at the "plStartPosFwd value plus the value returned in the plPosIntervalFwd parameter. Thus when this second position is reached, the TRIG output will be asserted to its active state again. The sequence is repeated the number of times returned in the plNumPulsesFwd parameter.

When the number of pulses returned in the plNumPulsesFwd parameter has been generated, the trigger engine will schedule the next position to occur at the position returned in the plStartPosRev parameter. The same sequence as the forward direction is now repeated in reverse, except that the plPosIntervalRev and plNumPulsesRev parameters apply. When the number of pulses has been output, the entire forward-reverse sequence will repeat the number of times returned by plNumCycles parameter. This means that the total number of pulses output will be plNumCycles x *(plNumPulsesFwd + plNumPulsesRev)*.

Once the total number of output pulses have been generated, the trigger output will remain inactive.

When a unidirectional sequence is selected, only the forward or reverse part of the sequence will be activated.



Example for a move from 0 to 20 mm and back.

In forward direction: The first trigger pulse occurs at 10 mm (StartPosFwd), the next trigger pulse occurs after another 5 mm (PosIntervalFwd), the stage then moves to 20 mm.

In reverse direction: The next trigger occurs when the stage gets to 12 mm.

Note that the position triggering scheme works on the principle of always triggering at the next scheduled position only, regardless of the actual direction of movement. If, for example, a position trigger sequence is set up with the forward start position at 10 mm, but initially the stage is at 15 mm, the first forward position trigger will occur when the stage is moving in the reverse direction. Likewise, if the stage does not complete all the forward position trigger points, the reverse triggering will not activate at all. For normal operation it is assumed that all trigger points will be reached during the course of the movement.

The position triggering sequence can be stopped at any time by changing the triggering function of the port to one of the other options in the SetKCubeTriggerParams method (for example, general purpose output). If the function of the port is then changed back to position triggering, the sequence will re-start from the beginning.

To specify the position trigger parameter settings, see the SetKCubePosTriggerParams method.

Piezo Motor Method GetJogMode

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **GetJogMode**(lChanID As Long, plMode As Long) As Long

Parameters
lChanID - the channel identifier
plMode - the jogging mode

Returns
MG Return Code

Details
This method returns the jogging mode for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The jogging mode is returned in the *plMode* parameter, which takes values specified by the JOGMODE enumeration.
Jog commands can be issued by calling the MoveJog method, via the Motor Control GUI panel or by pressing the buttons on the

hardware unit. When a jog command is received, if the jog mode is set to 'Step' the motor moves by the step size specified in the SetJogStepSize method (or the GUI Settings panel). If the jog mode is set to 'Continuous', the motor continues to move until the jog signal is removed (i.e. the jog button is released) when the motor will stop immediately.

To set the jogging mode for a particular channel, see the SetJogMode method.

Piezo Motor Method GetJogOPParams

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **GetJogOPParams**(lChanID As Long, plStepRate As Long, plStepAccn As Long) As Long

Parameters
lChanID - the channel identifier
plStepRate - the velocity to move when a command is initiated
plStepAccn - the acceleration up to the step rate

Returns
MG Return Code

Details
This method returns various drive parameters which define the speed and acceleration of moves initiated in the following ways:

by clicking the Jog buttons on the GUI panel
via the panel buttons when 'Jogging' mode is selected
via software using the MoveJog, method.

Drive parameters for other moves are returned in the GetDriveOPParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The piezo motor moves by ramping up the drive voltage (to the value set in the SetDriveOPParams method) and then dropping quickly to zero, then repeating. One cycle is termed a step. The velocity to move when a jog command is initiated is set in the lStepRate parameter. The step rate is specified in steps/sec, in the range 1 to 2,000.

The acceleration up to the step rate of the drive command is specified in the lStepAccn parameter in the range 1 to 100,000 cycles/sec/sec.

The jog output parameters can be set by calling the SetJogOPParams method.

Piezo Motor Method *GetJogStepSize*

Visual Basic Syntax
Function **GetJogStepSize**(lChanID As Long, plJogSteps As Long) As Long

Parameters
lChanID - the channel identifier
plJogSteps - the size of step (in drive pulses) taken when the jog signal is initiated

Returns
MG Return Code

Details
Note: This method is applicable only if the Jog Mode is set to 'Step' in the SetJogMode method.
This method sets the distance to move when a jog command is initiated. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
A jog step consists of a number of drive pulses. The number of these pulses which make up a jog step is returned in the *plJogSteps* parameter in the range 1 to 2,000.

To set the jog step size for a particular channel, see the SetJogStepSize method.

Piezo Motor Method GetPositionSteps

Visual Basic Syntax
Function **GetPositionSteps**(lChanID As Long, plPosSteps As Long) As Long

Parameters
lChanID - the channel identifier
plPosSteps - the position counter value,in number of steps

Returns
MG Return Code

Details
This method returnes the position counter on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
The counter (position) value is returned in number of steps in the plPos*Steps* parameter. The returned position value is measured from the zeroed position.

To set the position steps for a particular channel, see the SetPositionSteps method.

Piezo Motor Method GetPotParams

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **GetPotParams** (lChanID As Long, plMinStepRate As Long, plMaxStepRate As Long) As Long

Parameters
lChanID – the channel identifier
plMinStepRate – the step rate associated with minimum deflection of the potentiometer
plMaxStepRate – the step rate associated with maximum deflection of the potentiometer

Returns
MG Return Code

Details

This method returns the speed of a move initiated by the potentiometer on the top panel of the hardware unit. The applicable channel is specified in the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The potentiometer slider is sprung such that when released it returns to it's central position. In this central position the piezo motor is stationary. As the slider is moved away from the centre, the motor begins to move. Bidirectional control of the motor is possible by moving the slider in both directions. The speed of the motor increases as a function of slider deflection. The present speed is returned as a step rate (drive pulses per second) in the lMinStepRate and lMaxStepRate parameters, in the range 1 to 2,000.

These parameters can be set by calling the SetPotParams method.

Piezo Motor Method *GetRelMoveDist*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **GetRelMoveDist**(lChanID As Long, plRelDist As Long) As Long

Parameters
lChanID - the channel identifier
plRelDist - the distance to move (in position steps) relative to the present position

Returns
MG Return Code

Details

This method is used to obtain the relative distance moved on receipt of an input trigger, when the trigger mode is set to RELMOVE - 02 in the SetKCubeTriggerParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1

02 - Channel 2
03 - Channel 3
04 - Channel 4

The relative distance to move is returned (in position steps, negative or positive) in the *plRelDist* parameter.

The relative move distance can be set by calling the SetRelMoveDist method.

Piezo Motor Method *Identify*

Visual Basic Syntax
Function **Identify**() As Long

Returns
MG Return Code

Details
This method allows the hardware unit associated with the ActiveX control instance to be identified. The associated unit is specified by the HWSerialNum property.
When the method is called, the front panel LEDs on the relevant hardware unit will flash for a short period.

Piezo Motor Method *LoadParamSet*

Visual Basic Syntax
Function **LoadParamSet**(bstrName As String) As Long

Parameters
bstrName – the name of the parameter set to be loaded

Returns
MG Return Code

Details
This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

Piezo Motor Method *LLGetStatusBits*

Visual Basic Syntax
Function **LLGetStatusBits**((lChanID As Long, plStatusBits As Long) As Long

Parameters
lChanID – the channel identifier
plStatusBits – the 32-bit integer containing the status flags.

Returns
MG Return Code

Details
This low level method returns a number of status flags pertaining to the operation of the piezo motor controller channel specified in the *lChanID* parameter. The *lChanID* parameter takes values specified by the HWCHANNEL enumeration.
These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description
0x00000010	5.	Motor shaft moving clockwise (1 - moving, 0 - stationary).
0x00000020	6.	Motor shaft moving counterclockwise (1 - moving, 0 - stationary)
0x00000040	7.	Shaft jogging clockwise (1 - moving, 0 - stationary).
0x00000080	8.	Shaft jogging counterclockwise (1 - moving, 0 - stationary).
0x00000100	9.	Motor connected (1 - connected, 0 - not connected).
0x00000200	10.	Motor homing (1 - homing, 0 - not homing).
0x00000400	11.	Motor homed (1 - homed, 0 - not homed).

0x00000800	12.	For Future Use

Piezo Motor Method *MoveAbsoluteStepsEx*

Visual Basic Syntax
Function **MoveAbsoluteStepsEx**(lChanID As Long, lAbsSteps As Long, bWait As Boolean) As Long

Parameters
lChanID - the channel identifier
lAbsSteps - the distance to move from the zero position, measured in number of steps
bWait - specifies the way in which the MoveAbsoluteStepsEx method returns

Returns
MG Return Code

Details
This method initiates a move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
The distance to move, relative to the zero position, is specified in number of steps by the lAbs*Steps* parameter.
If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveAbsoluteStepsEx returns only after the motor has completed its move. In either mode, a MoveComplete event is fired once the motor moves have been completed.
When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and respond to the *MoveComplete* event. This event driven approach allows a client application to service other tasks while the motors are moving.

Piezo Motor Method *MoveJog*

Visual Basic Syntax
Function **MoveJog**(lChanID As Long, lJogDir As Long,) As Long

Parameters
lChanID - the channel identifier
lJogDir - the direction in which the motor is jogged

Returns
MG Return Code

Details
This method is used to initiate a jog move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration. The step size is specified by the SetJogStepSize method.
Note. The move can also be initiated from the associated APTPZMOTOR GUI panel, or the jog buttons on the top panel of the hardware unit.
The *lJogDir* parameter specifies the direction of the move, either 'Forward' or 'Reverse' and takes values from the JOGDIR enumeration.

Piezo Motor Method *MoveRelativeStepsEx*

Visual Basic Syntax
Function **MoveRelativeStepsEx**(lChanID As Long, lRelSteps As Long, bWait As Boolean) As Long

Parameters
lChanID - the channel identifier
lRelSteps - the distance to move from the present position, measured in number of steps
bWait - specifies the way in which the MoveRelativeStepsEx method returns

Returns
MG Return Code

Details
This method initiates a move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
The distance to move, relative to the present position, is specified in number of steps by the l*RelSteps* parameter.
If the *bWait* parameter is set to 'False', the method returns as soon as the move has been initiated. If *bWait* is set to 'True', MoveRelativeStepsEx returns only after the motor has completed its move. In either mode, a MoveComplete event is fired once the motor moves have been completed.
When a client application needs to sequence motor moves, it is more efficient programming practice to set *bWait* to 'False' and

respond to the *MoveComplete* event. This event driven approach allows a client application to service other tasks while the motors are moving.

Piezo Motor Method *MoveVelocity*

Visual Basic Syntax
Function **MoveVelocity**(lChanID As Long, lDirection As Long) As Long

Parameters
lChanID - the channel identifier
lDirection - the direction sense to move

Returns
MG Return Code

Details
This method initiates a motor move on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
The direction to move is specified by the *lDirection* parameter, which in turn takes values from the JOGDIR enumeration.
When this method is called, the motor will move continuously in the specified direction, using the velocity parameters set in the SetDriveOPParams method, until the Stop method is called.

Piezo Motor Method *SaveParamSet*

Visual Basic Syntax
Function **SaveParamSet**(bstrName As String) As Long

Parameters
bstrName – the name under which the parameter set is to be saved

Returns
MG Return Code

Details
This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).
When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved. Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

Piezo Motor Method *SetAbsMovePos*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **SetAbsMovePos**(lChanID As Long, lAbsPos As Long) As Long

Parameters
lChanID - the channel identifier
lAbsPos - the absolute position to move to.

Returns
MG Return Code

Details

This method is used to set the absolute position to move on receipt of an input trigger, when the trigger mode is set to ABSMOVE - 03 in the SetKCubeTriggerParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1

02 - Channel 2
03 - Channel 3
04 - Channel 4

The absolute position to move is set (in position steps) in the *lAbsPos* parameter.

The relative move distance can be obtained by calling the GetAbsMovePos method.

Piezo Motor Method SetButtonParams

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **SetButtonParams**((lChanID As Long, lButMode As Long, lButPos As Long, lButPos2 As Long) As Long

Parameters
lChanID – the channel identifier
lButMode – the operating mode of the front panel buttons
lButPos1 – the absolute position associated with the left hand button
lButPos2 – the absolute position associated with the right hand button

Returns
MG Return Code

Details
This method sets the operation mode of the front panel buttons for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The buttons on the front of the unit can be used either to jog the motor, or to perform moves to absolute positions.
The mode of operation is specified in the *lButMode* parameter which in turn takes values from the BUTTONMODE enumeration as follows:

BUTTON_MODEJOG: In this mode, the front panel buttons are used to jog the motor. Once set to this mode, the move parameters for the buttons are taken from the 'Jog' parameters set via the 'Move/Jogs' settings tab or the SetJogStepSize and SetJogOPParams methods.

BUTTON_MODEPOSITION: In this mode, each button can be programmed with a different position value, such that the controller will move the motor to that position when the specific button is pressed.

The following parameters are applicable only if 'BUTTON_MODEPOSITION is selected in the 'lButMode' parameter. In each case, the position is set in number of steps, measured from the zero position.

lButPos1: The position to which the motor will move when the top button is pressed.

lButPos2: The position to which the motor will move when the bottom button is pressed.

The button parameters can be obtained by calling the GetButtonParams method.

Piezo Motor Method SetDriveOPParams

Visual Basic Syntax
Function **SetDriveOPParams**(lChanID As Long, lVoltage As Long, lStepRate As Long, lStepAccn As Long) As Long

Parameters
lChanID - the channel identifier
lVoltage - the maximum piezo drive voltage
lStepRate - the velocity to move when a command is initiated
lStepAccn - the acceleration up to the step rate

Returns
MG Return Code

Details

This method sets various drive parameters which define the speed and acceleration of moves initiated in the following ways:

by clicking in the position display
via the panel buttons when 'Go To Position' mode is selected
via software using the MoveVelocity, MoveAbsoluteStepsEx or MoveRelativeStepsEx methods.

Drive parameters for Jog moves are specified in the SetJogOPParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The maximum piezo drive voltage is specified in the lVoltage parameter, in the range 80V to 130V.

The piezo motor moves by ramping up the drive voltage to the value set in the lVoltage parameter and then dropping quickly to zero, then repeating. One cycle is termed a step. The velocity to move when a command is initiated is set in the lStepRate parameter. The step rate is specified in steps/sec, in the range 1 to 2,000.

The acceleration up to the step rate of the drive command is specified in the lStepAccn parameter in the range 1 to 100,000 cycles/sec/sec.

The present settings for the drive parameters can be obtained by calling the GetDriveOPParams method.

Piezo Motor Method SetJogMode

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **SetJogMode**(lChanID As Long, lMode As Long) As Long

Parameters
lChanID - the channel identifier
lMode - the jogging mode

Returns
MG Return Code

Details
This method sets the jogging mode for the channel specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The jogging mode is specified the *lMode* parameter, which takes values specified by the JOGMODE enumeration.
Jog commands can be issued by calling the MoveJog method, via the Motor Control GUI panel or by pressing the buttons on the hardware unit. When a jog command is received, if the jog mode is set to 'Step' the motor moves by the step size specified in the SetJogStepSize method (or the GUI Settings panel). If the jog mode is set to 'Continuous', the motor continues to move until the jog signal is removed (i.e. the jog button is released) when the motor will stop immediately.

To retrieve the jogging mode for a particular channel, see the GetJogMode method.

Piezo Motor Method SetJogOPParams

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **SetJogOPParams**(lChanID As Long, lStepRate As Long, lStepAccn As Long) As Long

Parameters
lChanID - the channel identifier
lStepRate - the velocity to move when a command is initiated
lStepAccn - the acceleration up to the step rate

Returns
MG Return Code

Details
This method sets various drive parameters which define the speed and acceleration of moves initiated in the following ways:

by clicking the Jog buttons on the GUI panel
via the panel buttons when 'Jogging' mode is selected
via software using the MoveJog, method.

Drive parameters for other moves are specified in the SetDriveOPParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The piezo motor moves by ramping up the drive voltage (to the value set in the SetDriveOPParams method) and then dropping quickly to zero, then repeating. One cycle is termed a step. The velocity to move when a jog command is initiated is set in the lStepRate parameter. The step rate is specified in steps/sec, in the range 1 to 2,000.

The acceleration up to the step rate of the drive command is specified in the lStepAccn parameter in the range 1 to 100,000 cycles/sec/sec.

The present settings for the jog output parameters can be obtained by calling the GetJogOPParams method.

Piezo Motor Method *SetJogStepSize*

Visual Basic Syntax
Function **SetJogStepSize**(lChanID As Long, lJogSteps As Long) As Long

Parameters
lChanID - the channel identifier
lJogSteps - the size of step (in drive pulses) taken when the jog signal is initiated

Returns
MG Return Code

Details
Note: This method is applicable only if the Jog Mode is set to 'Step' in the SetJogMode method.
This method sets the distance to move when a jog command is initiated. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
A jog step consists of a number of drive pulses. The number of these pulses which make up a jog step is specified in the *lJogSteps* parameter in the range 1 to 2,000.

To retrieve the jog step size for a particular channel, see the GetJogStepSize method.

Piezo Motor Method *SetKCubeChanEnableMode*

This method is applicable only to K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax
Function **SetKCubeChanEnableMode** (lMode As Long) As Long

Parameters
lMode - the channel mode, either single channels, or channel pairs

Returns
MG Return Code

Details

The K-Cube piezo motor controllers have four drive output channels, and in some applications (e.g. if the actuators are fitted to a 2-axis mirror mount), it may be advantageous to move two axes at the same time by moving the joystick diagonally. The method allows channels to be enabled individually, or in pairs.

The mode of operation is set in the *plMode* parameter, which in turn takes values from the CHANENABLEMODE Enumeration as follows:

00 - None, i.e. all channels disabled
01 - Channel 1 enabled
02 - Channel 2 enabled
03 - Channel 3 enabled
04 - Channel 4 enabled
05 - Channels 1 and 2 enabled
06 - Channels 3 and 4 enabled

To obtain the current setting for the channel enable mode, see the GetKCubeChanEnableMode method.

Piezo Motor Method *SetKCubeFeedbackSig*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **SetKCubeFeedbackSig**(lChanID As Long, lFBMode As Long, lEncoderConst As Long) As Long

Parameters
lChanID - the channel identifier
lFBMode - the type of digital input providing the feedback

The following parameter is not implemented at this time and is ignored.
lEncoderConst - the calibration constant for converting encoder counts to real world units

Returns
MG Return Code

Details

The USER IO connector on the rear panel exposes two pairs of four digital inputs. These inputs can be used by a drive channel to receive a signal from the actuator being driven, either a differential QEP encoder feedback signal, or the FWD and REV limit switch signals. This sub message sets up the QEP/Limit switch selection for a specified channel..

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The feedback mode is set in the *plFBMode* parameter, which takes values specified by the FEEDBACKMODE enumeration as follows:

00 - Disabled - The digital inputs are disabled
01 - LIMSWITCH - The inputs accept a signal when the limit switches are activated.

The following option is for future use and is not implemented at this time.

02 – ENCODER. The inputs accept a feedback signal from the encoder in the actuator

The following parameter is for future use and is not implemented at this time. The description is provided for information only.

If the *FBMode* parameter above is set to Encoder 02, the *lEncoderConst* parameter specifies the calibration constant for converting encoder counts to real world units (mm or degrees) for the actuator being driven.

The current settings for the Feedback signal mode can be obtained by calling the GetKCubeFeedbackSig method.

Piezo Motor Method *SetKCubeJogParams*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **SetKCubeJogParams**(lChanID As Long, lJogMode As Long, lJogStepSizeFwd As Long, lJogStepSizeRev As Long, lJogStepRate As Long, lJogStepAccn As Long) As Long

Parameters
lChanID - the channel identifier
lJogMode - the jogging mode
lJogStepSizeFwd- the number of pulses which make up a forward jog step
lJogStepSizeRev - the number of pulses which make up a reverse jog step

lJogStepRate - the velocity to move when a command is initiated
lJogStepAccn - the acceleration up to the step rate

Returns
MG Return Code

Details

This method sets various parameters which define the speed and acceleration of moves initiated in the following ways:

by clicking the jog buttons on the GUI panel

by moving the joystick on the unit when 'Jog Mode' mode is selected in the SetKCubePanelParams method.

via software using the MoveJog method.

It differs from the normal motor jog message in that there are two jog step sizes, one for forward and one for reverse. The reason for this is that due to the inherent nature of the PIA actuators going further in one direction as compared with another this will allow the user to potentially make adjustments to get fore and aft movement the same or similar.

Drive parameters for other moves are set in the SetDriveOPParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The jogging mode is set in the *lJogMode* parameter, which takes values specified by the JOGMODE enumeration as follows:

01 - Continuous
02 - Single Step

When a jog command is received, if the jog mode is set to 'Single Step' the motor moves by the step size specified in the JogStepSizeFwd and JogStepSizeRev parameters. If the jog mode is set to 'Continuous', the motor continues to move until the jog signal is removed (i.e. the joystick is released) when the motor will stop immediately.

A jog step consists of the number of drive pulses and the size of a jog step is specified by the number of pulses contained in a step. This is returned in the JogStepSizeFwd and JogStepSizeRev parameters in the range 1 to 2,000.

The piezo motor moves by ramping up the drive voltage (to the value set in the SetDriveOPParams method) and then dropping quickly to zero, then repeating. One cycle is termed a step. The velocity to move when a jog command is initiated is set in the lJogStepRate parameter. The step rate is specified in steps/sec, in the range 1 to 2,000.

The acceleration up to the step rate of the drive command is specified in the lJogStepAccn parameter in the range 1 to 100,000 cycles/sec/sec.

The current jog output parameters can be obtained by calling the GetKCubeJogParams method.

Piezo Motor Method SetKCubePanelParams

This method is applicable only to K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax
Function SetKCubePanelParams (lChanID As Long, lJSMode As Long, lJSMaxStepRate As Long, lJSDirSense As Long, lPresetPos1 As Long, lPresetPos2 As Long, lDispBrightness As Long,) As Long

Parameters
lChanID - the channel identifier
lJSMode - The operating mode of the top panel joystick
lJSMaxStepRate - The max velocity of a move initiated by the top panel joystick (i.e. the max step rate for full joystick deflection)

lJSDirSense - The direction sense of a move initiated by the top panel joystick
lPresetPos1 - The preset position 1 when operating in go to position mode

lPresetPos2 - The preset position 2 when operating in go to position mode
lDispBrightness - The display brightness when the unit is active

Returns
MG Return Code

Details
This method sets the operating parameters of the joystick on the top panel of the associated K-Cube unit.
The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The operating mode is set in the lJSMode parameter, which in turn takes values from the JOYSTICKMODE enumeration as follows:

1 Velocity Control Mode - Deflecting the joystick starts a move with the velocity proportional to the deflection. The maximum velocity (i.e. velocity corresponding to the full deflection of the joystick) is specified in the lJSMaxStepRate parameter following.

2 Jog Mode - Deflecting the joystick initiates a jog move, using the parameters specified by the SetKCubeJogParams message. Keeping the joystick deflected repeats the move automatically after the current move has completed.

3 Go To Position Mode - Deflecting the joystick starts a move from the current position to one of the two predefined “teach” positions. The teach positions are specified in number of steps from the zero position in the PresetPos1 and PresetPos2 parameters. In this mode, move the joystick left (Ch1 and 3) or up (Ch 2 and 4) to go to position 1, and right or down to go to position 2.

The lJSMaxStepRate parameter sets the max velocity of a move initiated by the top panel joystick (i.e. the max step rate for full joystick deflection), in the range 1 to 2000 position steps/sec.

The direction of a move initiated by the joystick is specified in the lJSDirSense parameter, which in turn takes values from the JOYSTICKDIRSENSE enumeration as follows:

0 Joystick initiated moves are disabled. The joystick is used for menu scrolling only.

1 Upwards/Right deflection of the joystick results in a positive motion (i.e. increased position count).

The following option applies only when the JSMode is set to Velocity Control Mode (1). If set to Jog Mode (2) or Go to Position Mode (3), the following option is ignored.

2 Upwards/Right deflection of the joystick results in a negative motion (i.e. decreased position count).

The following parameters apply only when the JSMode parameter is set to Go to Position Mode (3). If set to Velocity Control Mode (1) or Jog Mode (2), the following preset position parameters are ignored.

lPresetPos1 and lPresetPos2 are used to set the associated position values (in position steps from the home position).

In certain applications, it may be necessary to adjust the brightness of the LCD display on the top of the unit. The brightness is returned in the plDispBrightness parameter, as a value from 0 (Off) to 100 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again.

The panel parameters may be obtained by calling the GetKCubePanelParams method.

Piezo Motor Method SetKCubeTriggerParams

This method is applicable only to K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax

Function S**etKCubeTriggerParams** (lTrigChan1, lTrigChan2, lTrig1Mode As Long, lTrig1Polarity As Long, lTrig2Mode As Long, lTrig2Polarity As Long) As Long

Parameters
lTrigChan1 - The drive channel associated with trigger channel 1 (IO 1)
lTrigChan2 - The drive channel associated with trigger channel 2 (IO 2)
lTrig1Mode - IO 1 operating mode
lTrig1Polarity - The active state of IO 1 (i.e. logic high or logic low)
lTrig2Mode - IO 2 operating mode
lTrig2Polarity - The active state of IO 2 (i.e. logic high or logic low)

Returns
MG Return Code

Details

The KIM101 K-Cube inertial piezo motor controller has two bidirectional trigger ports (I/O 1 and I/O 2) that can be used as a general purpose digital input/output, or can be configured to output a logic level to control external equipment.
Either of them can be independently configured as an input or an output and the active logic state can be selected High or Low to suit the requirements of the application. Electrically the ports output 5 Volt logic signals and are designed to be driven from a 5 Volt logic.

When the port is used in the input mode, the logic levels are TTL compatible, i.e. a voltage level less than 0.8 Volt will be recognised as a logic LOW and a level greater than 2.4 Volt as a logic HIGH. The input contains a weak pull-up, so the state of the input with nothing connected will default to a logic HIGH. The weak pull-up feature allows a passive device, such as a mechanical switch to be connected directly to the input.

When the port is used as an output it provides a push-pull drive of 5 Volts, with the maximum current limited to approximately 8 mA. The current limit prevents damage when the output is accidentally shorted to ground or driven to the opposite logic state by external circuity.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method sets the operating parameters of the IO1 and IO2 connectors on the front panel of the unit.

The unit has 4 drive channels but only two trigger ports. The drive channel associated with each trigger port is specified in the lTrigChan1 and lTrigChan2 parameters.

The operating mode of the trigger is set in the lTrig1Mode and lTrig2Mode parameters, which take values from the TRIGMODE enumeration as follows:

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a relative or absolute move, or for starting a position count reset as follows:

0x00 DISABLED: The trigger IO is disabled

0x01 GPI: General purpose logic input (read through status bits using the LLGetStatusBits method).

0x02 RELMOVE: Input trigger for relative move.

0x03 ABSMOVE: Input trigger for absolute move.

0x04 RESETCOUNT: Input trigger for a count reset.

When used for triggering a move, the port is edge sensitive. In other words, it has to see a transition from the inactive to the active logic state (Low->High or High->Low) for the trigger input to be recognized. For the same reason a sustained logic level will not trigger repeated moves. The trigger input has to return to its inactive state first in order to start the next trigger.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output, or to indicate motion status or to produce a trigger pulse at configurable positions as follows:

0x0A GPO: General purpose logic output (set using the LLSetGetDigOPs method).

0x0B INMOTION: Trigger output active (level) when motor 'in motion'. The output trigger goes high (5V) or low (0V) (as

set in the lTrig1Polarity and lTrig2Polarity parameters) when the stage is in motion.

0x0C MAXVELOCITY: Trigger output active (level) when motor at 'max velocity'.

0x10 FWDLIMIT - Trigger output active (level) when the FWD limit switch is activated.

0x11 REVLIMIT - Trigger output active (level) when the REV limit switch is activated.

0x12 EITHERLIMIT - Trigger output active (level) when the either the FWD or REV limit switch is activated.

0x0D POSSTEPS_FWD: Trigger output active (pulsed) at pre-defined positions moving forward (set using wStartPosFwd, lIntervalFwd, wNumPulsesFwd and lPulseWidth members). Only one Trigger port at a time can be set to this mode.

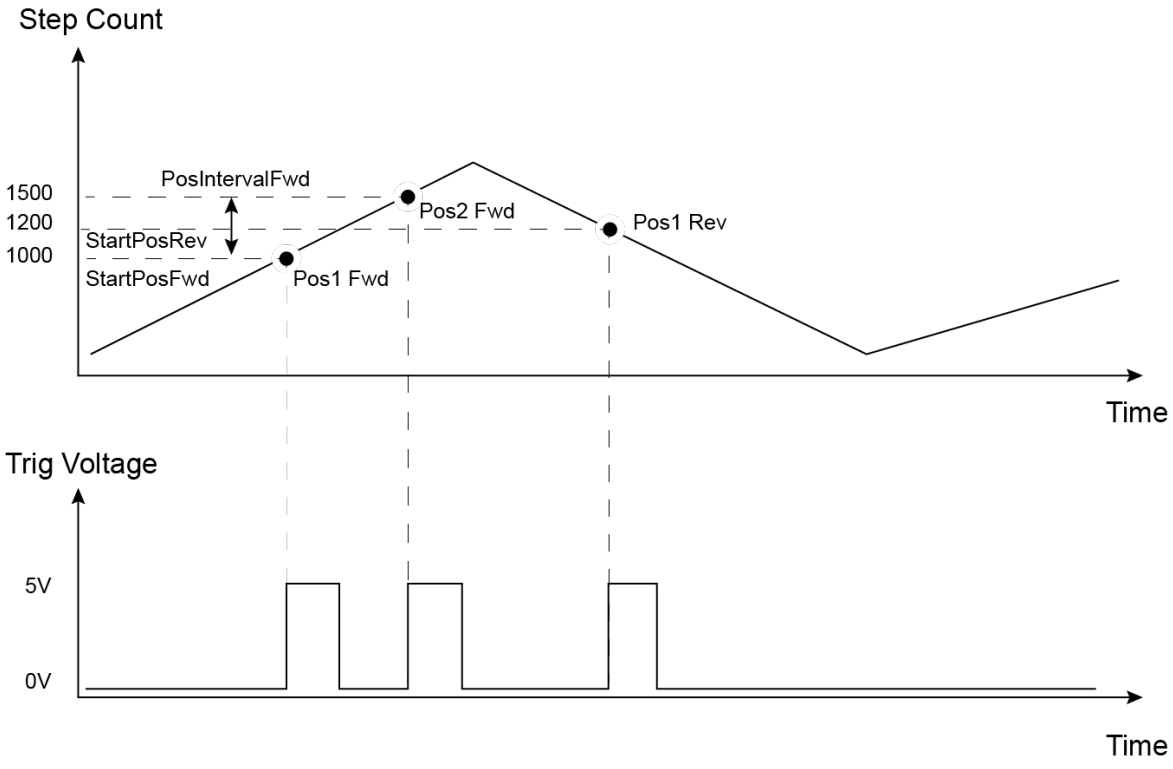
0x0E POSSTEPS_REV: Trigger output active (pulsed) at pre-defined positions moving backwards (set using wStartPosRev, lIntervalRev, wNumPulsesRev and lPulseWidth members). Only one Trigger port at a time can be set to this mode.

0x0F POSSTEPS_BOTH: Trigger output active (pulsed) at pre-defined positions moving forwards and backward. Only one Trigger port at a time can be set to this mode.

Trigger Out Position Steps

In the last three modes described above, the controller outputs a configurable number of pulses, of configurable width, when the actual position of the stage matches the position values configured as the Start Position and Position Interval - see SetKCubePosTriggerParams method. These mode allow external equipment to be triggered at exact position values.

Position triggering can be configured to be unidirectional (forward or revese only) or bidirectional (both). In bidirectional mode the forward and reverse pulse sequences can be configured separately. A cycle count setting (set in the SetKCubePosTriggerParams method, *lNumCycles* parameter) allows the uni- or bidirectional position triggering sequence to be repeated a number of times.



Example for a move from 0 to 2000 position steps.

In forward direction: The first trigger pulse occurs at 1000 steps (StartPosFwd), the next trigger pulse occurs after another 500 steps (PosIntervalFwd), the stage then moves to 2000 steps.

In reverse direction: The next trigger occurs when the stage gets to 12oo steps.

Please note that position triggering can only be used on one TRIG port at a time.

The operation of the position triggering mode is described in more detail in the SetKCubePosTriggerParams method.

The polarity of the trigger pulse is returned in the plTrig1Polarity and plTrig2Polarity parameters, which takes values from the TRIGPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

The current setting for the trigger parameters can be obtained by calling the GetKCubeTriggerParams method.

Piezo Motor Method SetKCubePosTriggerParams

This method is applicable only to K-Cube Series piezo motor controllers

See Also Example Code

Visual Basic Syntax
Function SetKCubePosTriggerParams (lChanID As Long, lStartPosFwd As Long, lPosIntervalFwd As Long, lNumPulsesFwd As Long, lStartPosRev As Long, lPosIntervalRev As Long, lNumPulsesRev As Long, lPulseWidth As Long, lNumCycles As Long) As Long

Parameters
lChanID - the channel identifier
lStartPosFwd - Trigger pulse output position start forward [in position steps].
lPosIntervalFwd - Trigger pulse output position interval forward [in position steps].
lNumPulsesFwd - Number of forward output pulses during a move.
lStartPosRev - Trigger pulse output position start backward [in position steps].
lPosIntervalRev - Trigger pulse output position interval backward [in position steps].
lNumPulsesRev - Number of backward output pulses during a move.
lPulseWidth - Trigger output pulse width (from 1 µs to 100000 µs).
lNumCycles - Number of forward/reverse move cycles.

Returns
MG Return Code

Details

The K-Cube piezo motor controllers have two bidirectional trigger ports (I/O 1 and I/O 2) that can be set to be used as input or output triggers. This method sets operating parameters used when the triggering mode is set to a trigger out position steps mode by calling the SetKCubeTriggerParams method.

The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration as follows:

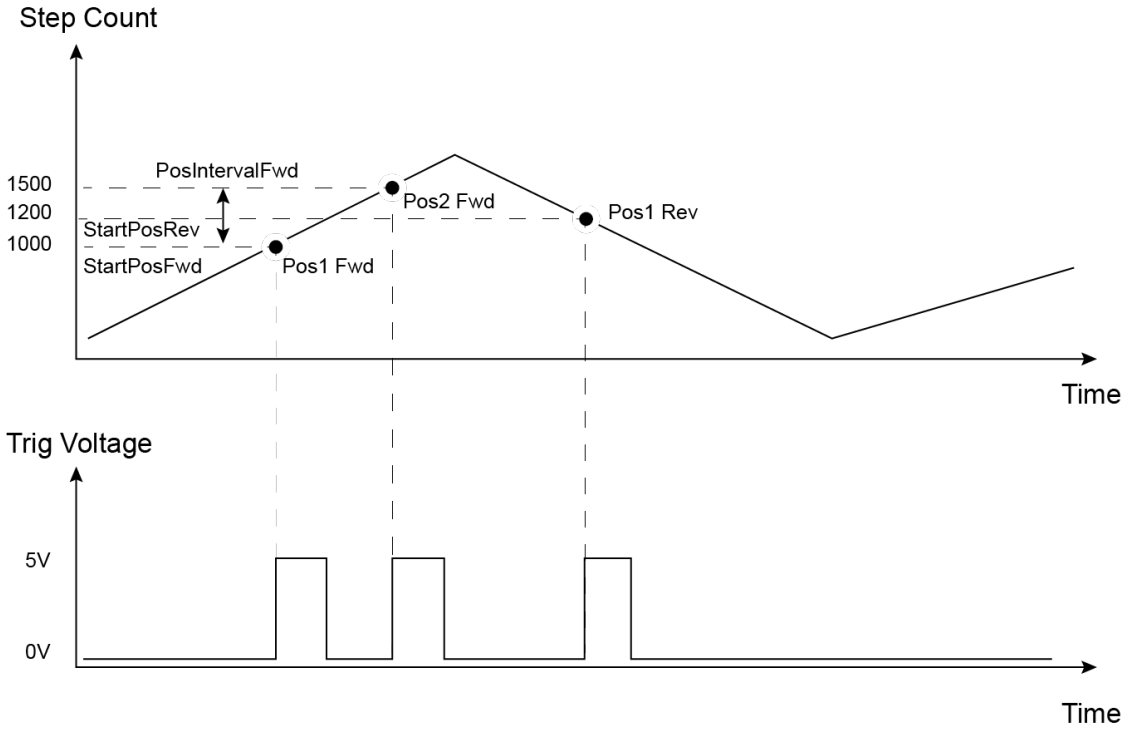
01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

As soon as position triggering is selected on either of the TRIGGER ports, the port will assert the inactive logic state, set in the SetKCubeTriggerParams method. As the stage moves in its travel range and the actual position matches the position set in the lStartPosFwd parameter, the TRIG port will output its active logic state. The active state will be output for the length of time specified by the lPulseWidth parameter, then return to its inactive state and schedule the next position trigger point at the "lStartPosFwd value plus the value set in the lPosIntervalFwd parameter. Thus when this second position is reached, the TRIG output will be asserted to its active state again. The sequence is repeated the number of times set in the lNumPulsesFwd parameter.

When the number of pulses set in the lNumPulsesFwd parameter has been generated, the trigger engine will schedule the next position to occur at the position specified in the lStartPosRev parameter. The same sequence as the forward direction is now repeated in reverse, except that the lPosIntervalRev and lNumPulsesRev parameters apply. When the number of pulses has been output, the entire forward-reverse sequence will repeat the number of times specified by lNumCycles parameter. This means that the total number of pulses output will be lNumCycles x *(lNumPulsesFwd + lNumPulsesRev).*

Once the total number of output pulses have been generated, the trigger output will remain inactive.

When a unidirectional sequence is selected, only the forward or reverse part of the sequence will be activated.



Example for a move from 0 to 20 mm and back.

In forward direction: The first trigger pulse occurs at 10 mm (StartPosFwd), the next trigger pulse occurs after another 5 mm (PosIntervalFwd), the stage then moves to 20 mm.

In reverse direction: The next trigger occurs when the stage gets to 12 mm.

Note that the position triggering scheme works on the principle of always triggering at the next scheduled position only, regardless of the actual direction of movement. If, for example, a position trigger sequence is set up with the forward start position at 10 mm, but initially the stage is at 15 mm, the first forward position trigger will occur when the stage is moving in the reverse direction. Likewise, if the stage does not complete all the forward position trigger points, the reverse triggering will not activate at all. For normal operation it is assumed that all trigger points will be reached during the course of the movement.

The position triggering sequence can be stopped at any time by changing the triggering function of the port to one of the other options in the SetKCubeTriggerParams method (for example, general purpose output). If the function of the port is then changed back to position triggering, the sequence will re-start from the beginning.

To obtain the current position trigger parameter settings, see the GetKCubePosTriggerParams method.

Piezo Motor Method SetPositionSteps

Visual Basic Syntax
Function **SetPositionSteps**(lChanID As Long, lPosSteps As Long) As Long

Parameters
lChanID - the channel identifier
lPosSteps - the position counter value, in number of steps

Returns
MG Return Code

Details
This method sets the position counter on the channel specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
The counter (position) value is specified in number of steps in the lPos*Steps* parameter. This method is normally used to zero

the piezo. When the motor is at the required zero position, this method is called to set the counter to zero. All absolute moves are then measured from the zeroed position.

To retrieve the position steps for a particular channel, see the GetPositionSteps method.

Piezo Motor Method SetPotParams

This message is applicable only to TIM101 T-Cube units

Visual Basic Syntax
Function **SetPotParams** (lChanID As Long, lMinStepRate As Long, lMaxStepRate As Long) As Long

Parameters
lChanID – the channel identifier
lMaxStepRate – the step rate associated with maximum deflection of the potentiometer

Returns
MG Return Code

Details

This method defines the speed of a move initiated by the potentiometer on the top panel of the hardware unit. The applicable channel is specified in the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration.

The potentiometer slider is sprung such that when released it returns to it's central position. In this central position the piezo motor is stationary. As the slider is moved away from the centre, the motor begins to move. Bidirectional control of the motor is possible by moving the slider in both directions. The speed of the motor increases as a function of slider deflection. The speed is entered as a step rate (drive pulses per second) in the lMaxStepRate parameter, in the range 1 to 2,000.

The current settings for these parameters can be obtained by calling the GetPotParams method.

Piezo Motor Method *SetRelMoveDist*

This method is applicable only to KIM101 K-Cube Series piezo motor controllers

Visual Basic Syntax
Function **SetRelMoveDist**(lChanID As Long, lRelDist As Long) As Long

Parameters
lChanID - the channel identifier
lRelDist - the distance to move (in position steps) relative to the present position

Returns
MG Return Code

Details

This method is used to set the relative distance (i.e. number of steps) to move on receipt of an input trigger, when the trigger mode is set to RELMOVE - 02 in the SetKCubeTriggerParams method.

The required channel is specified by the *lChanID* parameter, which in turn, takes values specified by the HWCHANNEL enumeration as follows:

01 - Channel 1
02 - Channel 2
03 - Channel 3
04 - Channel 4

The relative distance to move is specified (in position steps, negative or positive) in the *lRelDist* parameter.

The present setting for the relative move distance can be obtained by calling the GetRelMoveDist method.

Piezo Motor Method *StartCtrl*

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns
MG Return Code

Details
This method is used in conjunction with the HWSerialNum property, to activate an ActiveX control instance at runtime.
After the HWSerialNum property has been set to the required serial number (at design time or run time), this method can be called at run time to initiate the hardware enumeration sequence and activate the control.
Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

Example
```
Private Sub Form_Load()
' Start system.
frmSystem.MG17System1.StartCtrl
' Set serial number
' (use number of actual HW unit or simulated unit - see APTConfig for details)
APTPZMotor1.HWSerialNum = 65000001
' Start Piezo Motor control
APTPZMotor1.StartCtrl
End Sub
```

Piezo Motor Method *Stop*

Visual Basic Syntax
Function **Stop**(lChanID As Long) As Long

Parameters
lChanID - the channel identifier

Returns
MG Return Code

Details
This method is called to stop motor moves on the channel specified by the *lChanID* parameter, which in turn takes values specified by the HWCHANNEL enumeration.
Moves are stopped abruptly in a non-profiled manner, (i.e. the parameter *lStepAccn*, set using the SetDriveOPParams method, is ignored) and there is a risk that steps, and therefore positional integrity, could be lost.
This method can be called to stop prematurely, absolute or relative moves and, more importantly, is called to stop MoveVelocity moves.

Piezo Motor Method *StopCtrl*

Visual Basic Syntax
Function **StopCtrl**() As Long

Returns
MG Return Code

Details
This method is called to deactivate the operation of an ActiveX control instance.
When this method is called, communication to the associated hardware unit is stopped. It is efficient programming practice to call StopCtrl on any ActiveX control instances that are no longer required to be active. For example, this may be the case in a multiple window client application where multiple instances of an ActiveX control need to communicate at different times with the same hardware unit

Example
```
Private Sub Form_Unload(Cancel As Integer)
' Stop system control and unload forms.
frmSystem.MG17System1.StopCtrl
Unload frmSystem
APTPZMotor1.StopCtrl
Unload Me
End Sub
```

Piezo Motor Property *APTHelp*

Visual Basic Syntax

Property **APTHelp** As Boolean

Details

This property specifies the help file that will be accessed when the user presses the F1 key. If APTHelp is set to 'True', the main server helpfile MG17Base will be launched. If APTHelp is set to 'False', the helpfile is the responsibility of the application programmer.

Piezo Motor Property *HWSerialNum*

Visual Basic Syntax

Property **HWSerialNum** As Long

Details
This property specifies the serial number of the hardware unit to be associated with an ActiveX control instance.
The serial number must be set in the *HWSerialNum* property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.
Every APT control unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.
For example, if two or more piezo motor hardware units are connected to the PC, different instances of the APTPZMOTOR ActiveX Control can be included in a client application . If each of these Control instances is programmed with a unique hardware serial number, then they will communicate with their associated hardware units . In this way, multiple graphical control panels communicating with multiple hardware units can easily be incorporated into a custom client application.
After a serial number has been allocated, an ActiveX control instance must be activated at run time by calling the StartCtrl method.

Note. The factory programmed serial number also has a model type identifier incorporated. This ensures for example that the serial number for an APT Piezo Motor controller cannot be assigned to an APT NanoTrak ActiveX control. If a serial number is set that is either illegal or for which no hardware unit exists, then the ActiveX control will remain inactive, i.e. the control's GUI will appear 'dead'.
Example

```
Private Sub Form_Load()
' Start system.
frmSystem.MG17System1.StartCtrl
' Set serial number
' (use number of actual HW unit or simulated unit - see APTConfig for details)
MG17Piezo1.HWSerialNum = 65000001
' Start piezo control
MG17Piezo1.StartCtrl
End Sub
```

Introduction To Quad Detector Reader Programming

The ActiveX functionality for the Position Aligner (formerly Quad Detector Reader) is accessed via the APTQuad Control Object, and provides the functionality required for a client application to control a number of Position Aligner units.
Every hardware unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.
The serial number must be allocated using the HWSerialNum property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.
The methods of the Position Aligner unit can then be used to perform activities such as switching between Monitor, Open Loop and Closed Loop operating modes, setting the position demand parameters, reading the present beam position and setting the LED display intensity.
Note. The Quad Detector Reader T-Cube methods have been written to apply only to single channel units and the use of the *lChanID* parameter is not necessary.
For details on the operation of the position aligner, refer to the manual for the unit, which is available from www.thorlabs.com.

T-Cube PSD Autoalignment Unit Enumeration DISPMODE

The TPA101 and KPA101 PSD Autoalignment UnitS HAVE 2 display modes, which can be selected by calling the SetQuadDispMode method. This enumeration contains constants that specify the display mode.

Constant Name	Purpose

1. QUAD_DISPMODE_DIFF The display shows the X- and Y-axis difference signals
2. QUAD_DISPMODE_POS The display shows the Xpos and Ypos position demand outputs.

T-Cube PSD Autoalignment Unit Enumeration HEADTYPE

The T-Cube PSD Autoalignment unit may be used with a number of different detector/sensor heads, including those from third party manufacturers.

The detector head type is set by calling the SetDetectorHead method, and this enumeration contains constants that specify the type of detector head being used.

Constant	Name	Purpose
1.	HEAD_PDQ80A	PDQ80A Quadrant Detector Head.
2.	HEAD_PDQ30C	PDQ30C Quadrant Detector Head
3.	HEAD_PDP90A	PDp90A Lateral Effect Detector Head
4.	HEAD_OTHER	Third Party Detector Head.

T-Cube PSD Autoalignment Unit Enumeration QUAD_LVOUTPUTROUTE

The PSD Autoalignment Unit T-Cube outputs signals via the rear panel SMA connectors. By calling the SetSigOutputParams method, the unit can also be set to output signals via the T-Cube USB hub internal comms lines. This enumeration contains constants that specify the signal output mode.

Constant	Name	Purpose
1.	QUAD_ANALOGUEOUT_SMAONLY	Output signals via rear panel SMA connectors only.
2.	QUAD_ANALOGUEOUT_SMAHUB	Output signals via rear panel SMA connectors and the Hub internal comms lines.

T-Cube PSD Autoalignment Unit Enumeration QUAD_CLOSEDLOOPPOSDEMANDSENSE

This enumeration is applicable only to units running Firmware V1.0.3 or earlier
When the PSD Autoalignment Unit T-Cube is operated in ‘closed loop’ mode, certain application set ups may require the sense of the feedback signals to be inverted (this may be due to the choice of piezo amplifier/driver or the configuration of mirrors or other optical components).
The sense of the position demand signals to the piezo drivers can be set to positive or negative by calling the SetPosDemandParams method. If set to positive, the signals are positive when the beam is in the top and left hand quadrants of the detector array, and negative when in the bottom and right hand quadrants. If Negative is selected, then the sense is reversed.

Constant	Name	Purpose
1.	QUAD_CLOSEDLOOPPOSDEMAND_POSITIVE	Output signal sense positive.
2.	QUAD_CLOSEDLOOPPOSDEMAND_NEGATIVE	Output signal sense negative.

For units running firmware V1.0.4 or later, the position demand sense and gain value are entered as an integer from -10 to 10, see the SetPosDemandParams method for more details.

T-Cube PSD Autoalignment Unit Enumeration QUAD_DERIVFILTERENABLED

Note. This enumeration is not applicable to the TQD001.

The TQD101 PSD Autoalignment Unit T-Cube has a derivative low pass filter, which can be selected by calling the SetLoopFilterParams method. This enumeration contains constants that enable and disable the filter.

Constant	Name	Purpose
1.	TQD101_DERIVFILTER_ON	Low Pass Filter ON
2.	TQD101_DERIVFILTER_OFF	Low Pass Filter OFF

T-Cube PSD Autoalignment Unit Enumeration QUAD_NOTCHFILTERENABLED

Note. This enumeration is not applicable to the TQD001.

The TQD101 PSD Autoalignment Unit T-Cube has a notch filter, which can be selected by calling the SetLoopFilterParams method. This enumeration contains constants that enable and disable the filter.

Constant	Name	Purpose
1.	TQD101_NOTCHFILTER_ON	Notch Filter ON
2.	TQD101_NOTCHFILTER_OFF	Notch Filter OFF

T-Cube PSD Autoalignment Unit Enumeration QUAD_OPENLOOPPOSDEMANDS

If the PSD Autoalignment Unit T-Cube is operated in 'open loop' mode, the XPos and YPos position demand signals are either set to zero, or held at their last closed loop values, depending on the setting called by the SetPosDemandParams method.

This enumeration contains constants that specify the signal status.

Constant	Name	Purpose
1.	QUAD_OPENLOOPPOSDEMANDS_ZERO	Output signals set to zero.
2.	QUAD_OPENLOOPPOSDEMANDS_HELD	Output signals held at last closed loop values

K-Cube PSD Autoalignment Unit Enumeration QUADOPERMODE

The PSD Autoalignment Unit T-Cube and K-Cube have 3 operating modes, which can be selected by calling the SetQuadMode method. This enumeration contains constants that specify the operating mode.

Constant	Name	Purpose
1	QUADMODE_MONITOR	Monitor Mode
2	QUADMODE_OPENLOOP	Open Loop Mode
3	QUADMODE_CLOSEDLOOP	Closed loop Mode
4	QUADMODE_AUTOOPENCLOSEDLOOP	Closed loop Mode with automatic switch to open loop when SUM values falls below the value set in the *ISumMin* parameter of the SetKCubeTriggerParams method.

T-Cube PSD Autoalignment Unit Enumeration QUAD_TQD101CLOSEDLOOPPOSDEMANDSENSE

This enumeration is applicable only to TQD101 units

When the PSD Autoalignment Unit T-Cube is operated in 'closed loop' mode, certain application set ups may require the sense of the feedback signals to be inverted (this may be due to the choice of piezo amplifier/driver or the configuration of mirrors or other optical components).
The sense of the position demand signals to the piezo drivers can be set to positive or negative by calling the SetPosDemandParams method. If set to positive, the signals are positive when the beam is in the top and left hand quadrants of the detector array, and negative when in the bottom and right hand quadrants. If Negative is selected, then the sense is reversed.

Constant	Name	Purpose
32767	CLOSEDLOOP_TQD101POSDEMAND_POSITIVE	Output signal sense positive.
-32767	CLOSEDLOOP_TQD101POSDEMAND_NEGATIVE	Output signal sense negative.

K-Cube PSD Autoalignment Unit Enumeration *QUAD_TRIGGERCHANNELS*

The K-Cube position aligner has two bidirectional trigger ports (TRIG1 and TRIG2) that can be independently configured either as an input or an output and assigned a function by calling the SetKCubeTriggerParams method. This enumeration contains constants that specify the trigger port being addressed, Trig1 or Trig2.

Constant	Name	Purpose
1	QUAD_TRIG_CHAN1	Port Trig1
2	QUAD_TRIG_CHAN1	Port Trig2

K-Cube PSD Autoalignment Unit Enumeration *QUAD_TRIGGERPORTMODES*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating mode of the trigger.

Input Trigger Modes

When configured as an input, the TRIG ports can be used as a general purpose digital input, or for triggering a drive voltage change as follows:

Constant	Name	Purpose
0x00	QUAD_TRIG_DISABLED	The trigger IO is disabled
0x01	QUAD_TRIGIN_GPI	General purpose logic input (read through status bits using the LLGetStatusBits method).
0x02	QUAD_TRIGIN_LOOPOPENCLOSE	The trigger can be used to toggle the operating mode of the controller between open loop and closed loop modes.

Output Trigger Modes

When configured as an output, the TRIG ports can be used as a general purpose digital output.

Constant	Name	Purpose
0x0A	QUAD_TRIGOUT_GPO	General purpose logic output (set using the LLSetGetDigOPs method).
0x0B	QUAD_TRIGOUT_SUM	The state of the TRIG port is asserted depending on whether the SUM signal coming from the position sensor is inside the limits specified in the Trig Sum Min and Trig Sum Max parameters of the SetKCubeTriggerParams method.
0x0B	QUAD_TRIGOUT_DIFF	The state of the TRIG port is asserted depending on whether both the XDIFF and the YDIFF signals coming from the position sensor are below the Trig Diff Threshold parameter of the SetKCubeTriggerParams method.
0x0B	QUAD_TRIGOUT_SUMDIFF	This output mode is a 'logic AND' combination of the "Inside SUM range" and "Below Diff Threshold" conditions described above.

K-Cube PSD Autoalignment Unit Enumeration *QUAD_TRIGGERPORTPOLARITY*

When using the trigger ports on the front panel of the unit, this enumeration contains constants used by the SetKCubeTriggerParams method, that specify the operating polarity of the trigger.

Constant	Name	Purpose
0X01	QUAD_TRIGPOL_HIGH	The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).
0X02	QUAD_TRIGPOL_LOW	The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

T-Cube Quad Detector Method DeleteParamSet

See Also Example Code

Visual Basic Syntax

Function **DeleteParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be deleted

Returns

MG Return Code

Details

This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube Quad Detector Method DoEvents

See Also Example Code

Visual Basic Syntax

Function **DoEvents**() As Long

Returns

MG Return Code

Details

This method enables the APT server to allow message processing to take place within a client application. For example, if the client application is processing a lengthy loop and is making multiple calls to the server, this can often cause the client application to become non-responsive because other user interface message handling, such as button clicks, cannot be processed. Calling the DoEvents method allows such client application message handling to take place.

Note. The DoEvents method works in the same way as the DoEvents keyword found in Visual Basic.

T-Cube Quad Detector Method GetDetectorHead

See Also Example Code

Visual Basic Syntax

Function **GetDetectorHead**(plHeadType As Long) As Long

Parameters

plHeadType – the type of detector head being used

Returns

MG Return Code

Details.

The T-Cube PSD Autoalignment unit can be configured for use with a number of sensor/detector heads.

This method obtains the type of detector head being used. The detector head type is returned in the *plHeadType* parameter, which in turn accepts values from the HEADTYPE enumeration as follows:

1. HEAD_PDQ80A PDQ80A Quadrant Detector Head.
2. HEAD_PDQ30C PDQ30C Quadrant Detector Head
3. HEAD_PDP90A PDp90A Lateral Effect Detector Head

4. HEAD_OTHER Third Party Detector Head.

The type of detector head currently connected may be set by calling the SetDetectorHead method.

T-Cube Quad Detector Method GetDispIntensity

See Also Example Code
Visual Basic Syntax
Function **GetDispIntensity**(plIntensity As Long) As Long
Parameters
plIntensity - the display intensity (0 to 255)
Returns
MG Return Code
Details.

For TPA101 units
This method obtains the intensity of the various LEDs on the front of the Quad Detector associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.
The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest).
The display intensity may be set by calling the SetDispIntensity method.

For KPA101 units
This method returns the intensity of the OLED display on the top of the unit associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.
The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Note. 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after a pre-defined time interval has elapsed. The time interval is not exposed through message calls, rather is accessed via the GUI Settings panel. See the handbook for further details.

The display intensity may be set by calling the SetDispIntensity method.

T-Cube Quad Detector Method GetPIDParams

See Also Example Code

Visual Basic Syntax

Function **GetPIDParams**(plProp As Long, plInt As Long, plDiff As Long) As Long

Parameters

plProp - the value of the proportional constant

plInt - the value of the integration constant

plDiff - the value of the differential constant

Returns

MG Return Code

Details.

This method obtains the current settings for the proportional, integration and differential feedback loop constants and returns values the *plProp*, *plInt* and *plDiff* parameters respectively. They apply when the quad detector unit is operated in closed loop mode, and position demand signals are generated at the rear panel SMA connectors by the feedback loops. These position demand voltages act to move the beam steering elements (e.g. a piezo driven mirror) in order to centralize a beam at the centre of the PSD head.

When operating in closed loop mode, the proportional, integral and differential (PID) constants can be used to fine tune the

behaviour of the feedback loop to changes in the position demand output voltages. The feedback loop parameters need to be adjusted to suit the different types of sensor that can be connected to the system. The default values have been optimized for the PDQ80A sensor.

Proportional – This term provides the force used to drive the piezo to the demand position, reducing the positional error.

It accepts values in the range 1 to 100.

Integral – This term provides the 'restoring' force that grows with time, ensuring that the positional error is zero under a constant loading.

It accepts values in the range 0 to 100.

Derivative – This term provides the 'damping' force proportional to the rate of change of the position.

It accepts values in the range 0 to 100.

To obtain the current settings for the PID constants, see the SetPIDParams method.

Quad Control Method QUAD_GetKCubeTriggerParams

This method is applicable only to K-Cube Series position aligner units

See Also Example Code

Visual Basic Syntax
Function **GetKCubeTriggerParams** (lTrigChan As Long, plTrigMode As Long, plTrigPolarity As Long, plSumMin As Long, plSumMax As Long, plDiffThreshold As Long) As Long

Parameters
plTrigChan - the trigger port to be set, Trig1 or Trig2
plTrigMode - the trigger operating mode
plTrigPolarity - The active state of the trigger (i.e. logic high or logic low)
plSumMin - The lower limit when the trigger mode is set to TRIGOUT_SUM.
plSumMax - The upper limit when the trigger mode is set to TRIGOUT_SUM.
plDiffThreshold - The threshold when the trigger mode is set to TRIGOUT_DIFF

Returns
MG Return Code

Details

The K-Cube position aligner has two bidirectional trigger ports (TRIG1 and TRIG2) that can be independently configured either as an input or an output and assigned a function from the list of options described in the following section. The polarity (logic HIGH / LOW or rising / falling edge) can also be configured to suit the requirements of the equipment connected to these ports.

In the input operating modes the port is electrically configured as a TTL compatible logic input. When the port is driven with a voltage level below +0.8 V, it will read a logic LOW and when driven above +2.4V, it will read a logic HIGH. The ports have an internal weak pull-up resistor ensuring that a stable logic level is present on the inputs even when there is no driving source connected to it. This means that when unconnected the ports will read a logic HIGH. The internal pull-up also allows the direct connection of mechanical switches or other unpowered control devices.

In the output modes the port is electrically configured as a logic output using 5 Volt logic levels. The port is connected to the output driver logic with a 620 Ohm resistor in series; this resistor limits the maximum output current to approximately 8 mA and provides protection against the output being accidental short circuited to ground. The output can be used to drive the majority of digital inputs used on external equipment without any additional circuitry.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method obtains the present operating parameter settings of the trigger connectors on the front panel of the unit.

The applicable port (TRIG1 or TRIG2) is specified by the l*TrigChan* parameter, which takes values specified by the TRIGGERCHANNELS enumeration as follows:

1 = TRIG1

2 = TRIG2

The operating mode of the trigger is returned in the *plTrigMode* parameter, which in turn accept values from the TRIGGERPORTMODES enumeration as follows:

Input Trigger Modes

0x00 TRIG_DISABLED - The trigger IO is disabled. Selecting this option effectively results in the port returning to its default digital input configuration

0x01 TRIGIN_GPI - General purpose logic input. Other than being able to read the logic state of port there is no other functionality associated with it. The state of the port is returned in the periodic status update messages, or can be read by using the LLGetStatusBits method). In this mode the Triggering Polarity setting has no effect; the logic state of the input is returned as it is present on the port without inversion.

0x02 TRIGIN_LOOPOPENCLOSE - In this mode the port can be used to toggle the operating mode of the controller between open loop and closed loop modes. If the trigger polarity is selected to be "Active High", the operating mode toggles on the rising edge (LOW to HIGH transition) of the signal present on the TRIG input. Conversely, with "Active Low", the toggle takes place on the falling edge (HIGH to LOW transition).

Output Trigger Modes

0x0A TRIGOUT_GPO - In this operating mode the TRIG port functions as a simple digital output. The logic state of the output can be set using the LLSetGetDigOPs method. Other than being able to read the logic state of port there is no other functionality associated with it. The logic state of the output can be inverted by setting the Triggering Polarity parameter to "Low"; with this option selected the state of the output will be the opposite of the corresponding bit setting in the software call. The default state of the output in this mode is also the opposite of the option selected as the Triggering Polarity.

0x0B TRIGOUT_SUM - The state of the TRIG port is asserted depending on whether the SUM signal coming from the position sensor is inside the limits specified in the lSumMin and lSumMax parameters. If SUM is within the limits, the state will be the logic state selected in Triggering Polarity and conversely if it falls outside these limits, it will assume the opposite logic state. This mode can be used to detect the presence or absence of light falling on the position sensor; or that the optical power is within the expected limits. This option might be useful to signal a condition required for normal operation as under normal operating conditions the optical power is often expected to remain fairly constant. The plSumMin and plSumMax parameters are returned as a percentage of full scale, in the range 1% to 99%.

0x0C TRIGOUT_DIFF - The state of the TRIG port is asserted depending on whether both the XDIFF and the YDIFF signals coming from the position sensor are below the value set in the lDiffThreshold parameter. If both XDIFF and YDIFF are below the limit, the state will be the logic state selected in Triggering Polarity and conversely if either of them falls outside these limits, it will assume the opposite logic state. This mode can be used to signal whether or not the beam is close to the centre (beam aligned) position within a certain margin. In closed loop mode it also indicates that the controller is capable of tracking the changes in the beam position and maintain beam alignment. The plDiffThreshold parameter is returned as a percentage of full scale, in the range 1% to 99%.

0x0D TRIGOUT_SUMDIFF - This output mode is a 'logic AND' combination of the "Inside SUM range" and "Below Diff Threshold" conditions described above. Having to meet both conditions provides a more reliable indication of the normal closed loop operation when the beam is aligned and in the centre of the position sensor. In this scenario the SUM signal is within the expected limits (there is sufficient amount of light hitting the sensor) and both XDIFF and YDIFF are below a certain threshold (the beam is centralized). The second part of the condition, XDIFF and YDIFF below the threshold can also occur if the beam is blocked.

The polarity of the trigger pulse is returned in the plTrigPolarity parameter, which in turn accept values from the QUAD_TRIGGERPORTPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

For details on obtain the current trigger parameter settings, see the SetKCubeTriggerParams method.

T-Cube Quad Detector Method GetLoopFilterParams

See Also Example Code

Visual Basic Syntax
Function G**etLoopFilterParams**(pfDerivLPCutOffFreq As Float, pfNotchCentreFreq As Float, pfNotchQ As Float, plNotchFilterOn As Long, plDerivFilterOn As Long) As Long

Parameters
plerivLPCutOffFreq - the cut off frequency of the derivative low pass filter
pfNotchCentreFreq - the centre frequency of the notch filter
pfNotchQ - the Q factor of the notch filter
plNotchFilterOn - enables and disables the notch filter
plDerivFilterOn - enables and disables the derivative low pass filter

Returns
MG Return Code

Details
This method obtains the present settings for the derivative cut off frequency and the notch filter of the PID loop associated with a particular ActiveX Control instance.

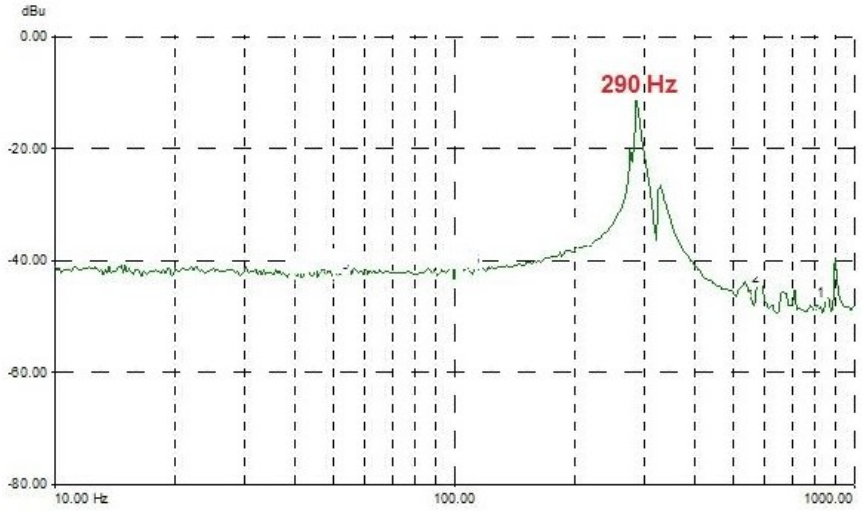
Derivative Low Pass Filter: The output of the derivative (differential) part of the PID controller can be passed through a tuneable low pass filter. Whilst the derivative component of the PID loop often improves stability (as it acts as a retaining force against abrupt changes in the system), it is prone to amplifying noise present in the system, as the derivative component is sensitive to changes between adjacent samples. To reduce this effect, a low pass filter can be applied to the samples. As noise often tends to contain predominantly high frequency components, the low pass filter can significantly decrease their contribution, often without diminishing the beneficial, stabilizing effect of the derivative action. In some applications enabling this filter can improve the overall closed loop performance.
The low pass filter is enable state is returned in the plDerivFilterOn parameter, which takes values from the QUAD_DERIVFILTERENABLED enumeration as follows:

TQD101_DERIVFILTER_ON = 1

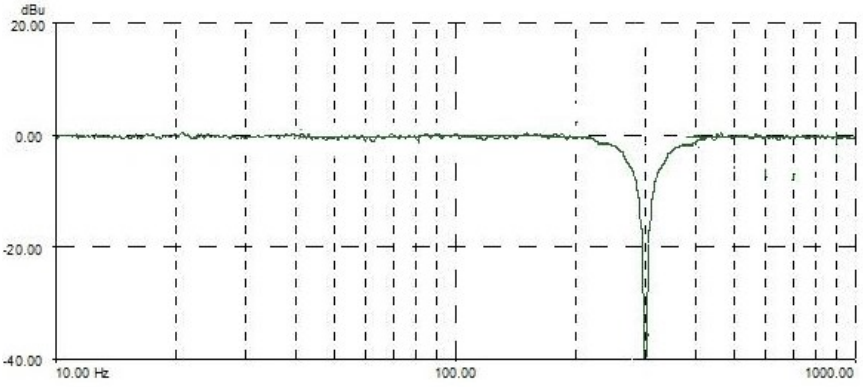
TQD101_DERIVFILTER_OFF = 2

The cut off frequency is returned in the plDerivLPCutOffFreq parameter, in the range 0 to 10,000.

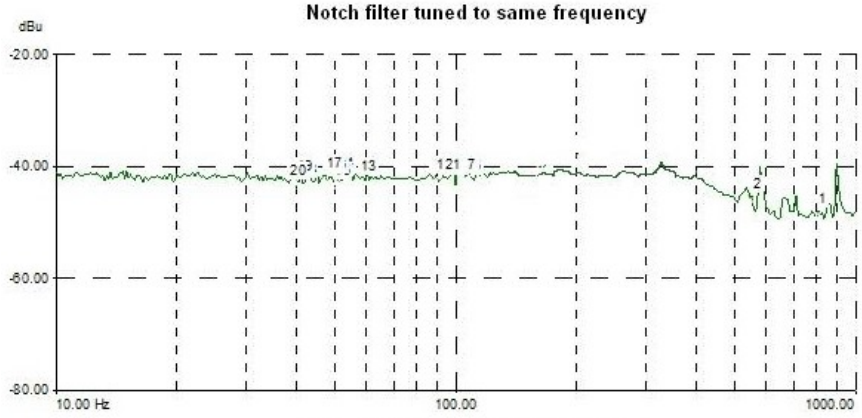
Notch Filter Settings - Due to their construction, most actuators are prone to mechanical resonance at well-defined frequencies. The underlying reason is that all spring-mass systems are natural harmonic oscillators. This proneness to resonance can be a problem in closed loop systems because, coupled with the effect of the feedback, it can result in oscillations. With some actuators (for example the ASM003), the resonance peak is either weak enough or at a high enough frequency for the resonance not to be troublesome. With other actuators (for example the PGM100) the resonance peak is very significant and needs to be eliminated for operation in a stable closed loop system. The notch filter is an adjustable electronic anti-resonance that can be used to counteract the natural resonance of the mechanical system.



Frequency response of actuator showing resonance at 290 Hz



Notch filter tuned to same frequency



The resonance is largely eliminated

As the resonance frequency of actuators varies with load in addition to the minor variations from product to product, the notch filter is tuneable so that its characteristics can be adjusted to match those of the actuator. In addition to its centre frequency, the bandwidth of the notch (or the equivalent quality factor, often referred to as the Q-factor) can also be adjusted. In simple terms, the Q factor is the centre frequency/bandwidth, and defines how wide the notch is, a higher Q factor defining a narrower ("higher quality") notch. Optimizing the Q factor requires some experimentation but in general a value of 5 to 10 is in most cases a good starting point.

The notch filter state is returned in the plNotchFilterOn parameter, which takes values from the QUAD_NOTCHFILTERENABLED enumeration as follows:

TQD101_NOTCHFILTER_ON = 1

TQD101_NOTCHILTER_OFF = 2

The centre frequency of the notch filter is returned in the pfNotchCentreFreq parameter, in the range 0 to 10000. The Q Factor is returned in the pf*NotchQ* parameter, in the range 0.1 to 100.

T-Cube Quad Detector Method GetPosDemandMinMax

See Also Example Code

Note. This method is applicable only when operating in closed loop mode.

Visual Basic Syntax

Function **GetPositionOutPuts**(pfXPosMin As Float, pfXPosMax As Float, pfYPosMin As Float, pfYPosMax As Float) As Long

Parameters

pfXPosMin – the minimum X axis position value

pfXPosMax – the maximum X axis position value

pfYPosMin – the minimum Y axis position value

pfYPosMax – the maximum Y axis position value

Returns

MG Return Code

Details.

When the Quad Detector T-Cube is operated in 'closed loop' mode, the XPos and YPos position demand signals are sent to the XDIFF and YDIFF SMA connectors. These signals are then routed to the positioning elements, (e.g. piezo drivers) to drive the positioning mirror to keep the beam central in the detector.

Under normal operating conditions, these signals are between –10V and +10V, however some applications may require the limits to be less than this.

This method is used to obtain the present settings for the XPos and YPos minimum and maximum limits and return values in the *pfXPosMin*, *pfXPosMax*, *pfYPosMin* and *pfYPosMax* parameters.

The XPos and YPos Min and max limits may be set by calling the SetPosDemandMinMax method.

T-Cube Quad Detector Method GetPositionOutputs

See Also Example Code

Note. This method is applicable only when operating in open loop mode.

Visual Basic Syntax

Function **GetPositionOutPuts**(pfXPos As Float, pfYPos As Float) As Long

Parameters

pfXPos – the X axis position output value

pfYPos – the Y axis position output value

Returns

MG Return Code

Details.

In addition to calling the SetPosDemandParams method, the position demand signals (on the XDIFF, YDIFF connectors) can also be set by calling this method.

When the quad detector unit is used with a beam steering device (e.g. a piezo mirror via piezo drivers), this method allows the beam to be positioned by entering a value (-10 V to +10V).

This method returns the currently applied settings for the open loop position demand signals (-10 V to +10V) in the *pfXPos* and *pfYPos* parameters.

The open loop position demand values may be set by calling the SetPositionOutputs method.

T-Cube Quad Detector Method GetPosDemandParams

See Also Example Code

Visual Basic Syntax
Function **GetPosDemandParams**(lOpenLoopPosDemands As Long, lXPosSense As Long, lYPosSense As Long) As Long

Parameters
plOpenLoopPosDemands - the status of the feedback outputs in open loop
plXPosSense - the sense of the X axis position demand signal
plYPosSense - the sense of the Y axis position demand signal

Returns
MG Return Code

Details.
This method returns values which specify the position demand (Xpos and Ypos) to values, depending on whether open loop or closed loop operating mode is selected.
When the Quad Detector T-Cube is operated in ‘open loop’ mode, the position demand signals (on the XDIFF, YDIFF and SUM connectors) can either be set to zero, or held at their latest closed loop values, according to the value entered in the *lOpenLoopPosDemands* parameter. The parameter accepts values from the QUAD_OPENLOOPPOSDEMANDS enumeration as follows:
If *plOpenLoopPosDemands* is set to QUAD_OPENLOOPPOSDEMANDS_ZERO, then the outputs are set to zero, when switching from Monitor or Closed loop mode to Open Loop mode.
If *plOpenLoopPosDemands* is set to QUAD_OPENLOOPPOSDEMANDS_HELD, then the outputs are held at the latest closed loop value, when switching from closed loop to open loop.
Due to the choice of piezo amplifier/driver or the configuration of mirrors (or other optical components) it is possible that certain application set ups may require the sense of the X and Y axis position demand signals to be inverted.
This is handled differently depending on the firmware version running on the unit.

When the Quad Detector T-Cube is operated in ‘closed loop’ mode, the sense of the position demand signals can be set to positive or negative by entering a value in the *XPosSense* and *YPosSense* parameters, which in turn take values from the QUAD_CLOSEDLOOPPOSDEMENDSENSE enumeration.
If *plXPosSense* is set to QUAD_CLOSEDLOOPPOSDEMAND_POSITIVE the signals are positive when the beam is in the left hand quadrants of the detector array, and negative when in the right hand quadrants.
If *plYPosSense* is set to QUAD_CLOSEDLOOPPOSDEMAND_POSITIVE the signals are positive when the beam is in the top quadrants of the detector array, and negative when in the bottom quadrants.
If *lXPosSense* or *lYPosSense* is set to QUAD_CLOSEDLOOPPOSDEMAND_NEGATIVE, then the sense is reversed.

Firmware V1.0.4 or later
When the Quad Detector T-Cube is operated in ‘closed loop’ mode, the sense of the position demand signals can be set to positive or negative.
Furthermore, in sensitive systems, e.g. scanning galvanometers, full feedback response could cause unstable behaviour, and some applications may require a the system response to be scaled down. The current settings for both the feedback sense and the gain can be returned as a value from '-10' to '10' in the *XPosSense* and *YPosSense* parameters:
If *plXPosSense* returns '10' the signals are positive when the beam is in the left hand quadrants of the detector array, and negative when in the right hand quadrants. The gain of the system is set to '1'.
If *plXPosSense* returns '-7' the signals are positive when the beam is in the right hand quadrants of the detector array, and negative when in the left hand quadrants. The gain of the system is set to '0.7'.
Similarly,
If *plYPosSense* returns '10' the signals are positive when the beam is in the top quadrants of the detector array, and negative when in the bottom quadrants. The gain is set to '1'.
If *plYPosSense* returns '-3' the signals are negative when the beam is in the bottom quadrants of the detector array, and positive when in the top quadrants. The gain is set to '0.3'.

TQD101
When the Quad Detector T-Cube is operated in ‘closed loop’ mode, the sense of the position demand signals can be set to

positive or negative by entering a value in the *XPosSense* and *YPosSense* parameters, which in turn take values from the QUAD_TQD101CLOSEDLOOPPOSDEMANDSENSE enumeration as follows:

If *IXPosSense* is set to CLOSEDLOOP_TQD101POSDEMANDSENSE_POSITIVE (i.e. 32767) the signals are positive when the beam is in the left hand quadrants of the detector array, and negative when in the right hand quadrants.
If *IYPosSense* is set to CLOSEDLOOP_TQD101POSDEMANDSENSE_POSITIVE (i.e. 32767) the signals are positive when the beam is in the top quadrants of the detector array, and negative when in the bottom quadrants.

If *IXPosSense* or *IYPosSense* is set to CLOSEDLOOP_TQD101POSDEMANDSENSE_NEGATIVE, (-32767) then the sense is reversed.

Note. The use of '0' is not allowed and will generate an error.
The current settings may be obtained by calling the SetPosDemendParams method

T-Cube Quad Detector Method GetQuadDispMode

See Also Example Code

Visual Basic Syntax

Function **GetQuadDispMode**(plMode As Long) As Long

Parameters

plMode - the display setting (Difference or Position)

Returns

MG Return Code

Details.

The main display on the GUI panel can be set to show X and Y axis difference signals from the detector array (Difference) or the Xpos and Ypos position demand output signals fed to the positioning elements. (Position).

This method returns the present display mode setting in the *lMode* parameter, which in turn takes values from the DISPMODEenumeration as follows.

If *plMode* is set to *QUAD_DISPMODE_DIFF*, the display represents the X and Y axis difference signals from the detector (i.e. the voltage outputs from the rear panel SMA connectors in Monitor Mode).

If *plMode* is set to *QUAD_DISPMODE_POS*, the display represents the position of the XPos and YPos position demand output signals fed to the positioning elements (i.e. the voltage outputs from the rear panel SMA connectors in OPEN or CLOSED loop mode).

The display mode may be set by calling the SetQuadDispMode method

T-Cube Quad Detector Method GetQuadMode

See Also Example Code
Visual Basic Syntax
Function **GetQuadMode**(plMode As Long) As Long
Parameters
plMode - the operating mode (Monitor, Open Loop or Closed Loop)
Returns
MG Return Code
Details.
The T-Cube Quad Detector can be set to operate in either open loop, closed loop or monitor mode. This method returns the current operating mode in the *lMode* parameter, which in turn takes values from the QUADOPERMODE enumeration as follows.

If *plMode* is set to *QUAD_MONITOR*, the unit operates in ‘Monitor’ mode. The X and Y axis difference signals and the sum signal from the photodiode array are fed through to the rear panel XDIFF, YDIFF and SUM SMA connectors for use in a monitoring application. The position of the beam in the photodiode array is represented on the Target display on the top panel of the unit.

If *plMode* is set to *QUAD_OPENLOOP*, the unit operates in 'open loop' mode. The X and Y axis position demand signals (Xpos and Ypos) are fed through to the rear panel XDIFF and YDIFF SMA connectors. These signals are either set to zero, or held at their last closed loop value, depending on the setting called by the SetPosDemandParams method. This mode is typically used when making manual adjustments to the turning mirrors in the system, to bring the beam position within the adjustment range of the quad detector system. See the operation section for a typical application

If *plMode* is set to *QUAD_CLOSEDLOOP*, the unit operates in closed loop' mode. The signal from the detector is interpreted by the unit, and the feedback circuit sends position demand signals (XOut and YOut) to the rear panel XDIFF and YDIFF connectors, which can be used to drive a pair of positioning elements (e.g. piezo controllers) in order to position the light beam within the center of the detector array.

If *plMode* is set to *QUAD_AUTOOPENCLOSEDLOOP*, the unit operates in closed loop' mode, until the SUM signal falls below the value set in the *lSumMin* parameter of the SetKCubeTriggerParams method.

A Note About Automatic Open Loop/Closed Loop Switching

The controller is capable of switching automatically between open loop and closed loop operating modes, depending on whether there is sufficient optical power required for closed loop operation. Automatic Switching mode can be selected by setting the lMode parameter to 4_*QUADMODE_AUTOOPENCLOSEDLOOP*. It can also be selected from the top panel settings menu or by clicking the Enable Auto Open Loop checkbox on the GUI Control Settings panel.

If during closed loop operation the SUM signal falls below the minimum specified in the *lSumMin* parameter of the SetKCubeTriggerParams method, the controller will switch back to open loop mode. If subsequently the SUM signal rises above the limit again, the controller will switch back to closed loop mode.

The automatic switchover works in conjunction with the "Position Demands In Open Loop Mode" option in the SetPosDemandParams method, that defines whether the controller will hold (freeze) the XPOS and YPOS outputs when switching over to open loop or set them to zero.

Automatic switchover might be advantageous in scenarios where the beam might be temporarily blocked, for example during experiments involving manual manipulation of optical components, particularly when the beam path is quite long and the beam steering actuator can deflect the beam so far that it falls outside the sensor area. In setups like this and with the controller in closed loop, blocking the beam can result in the feedback loop ramping the XPOS and/or YPOS outputs to saturation and steering the beam completely outside the sensor area. When this happens, restoring the beam will not normally restore the beam alignment as at this point the feedback algorithm does not even see the beam. However, with automatic switchover the loss of light will stop the closed loop operation, optionally freeze the last valid beam position and prevent the outputs ramping up as an unintentional consequence of the loss of feedback signals. Later when the beam is restored, closed loop operation will resume and continue control starting from the last valid beam position.

Note that because automatic switchover assumes the knowledge of the last valid closed loop beam position that is lost when the controller is powered down, this option cannot be persisted. For a similar reason, the controller will always power up in open loop mode.

T-Cube Quad Detector Method GetSigOutputParams

See Also Example Code

Visual Basic Syntax

Function **GetSigOutPutParams**(plSigRouting As Long) As Long

Parameters

plSigRouting – the comms routing for the position demand signals

Returns

MG Return Code

Details.

The Quad Detector T-Cube outputs the XDIFF and YDIFF or XPos and YPos signals via the rear panel SMA connectors. The unit can also be set to output these signals via the T-Cube USB hub internal comms lines. This method returns the current setting for the signal routing as specified by the *plSigRouting* parameter value, which in turn takes values from the QUAD_LVOUTPUTROUTEenumeration as follows.

If *plSigRouting* is set to QUAD_ANALOGUEOUT_SMAONLY, the X and Y signals are output only via the rear panel SMA connectors.

If *plSigRouting* is set to QUAD_ANALOGUEOUT_SMAHUB, the X and Y signals are output via the rear panel SMA connectors AND the USB hub internal comms lines.

The signal routing may be set by calling the SetSigOutputParams method.

Please see the handbook supplied with the unit for more details.

T-Cube Quad Detector Method LLGetStatusBits

See Also Example Code

Visual Basic Syntax

Function **LLGetStatusBits** (plStatusBits As Long) As Long

Parameters

plStatusBits – the 32-bit integer containing the status flags.

Returns

MG Return Code

Details

This low level method returns a number of status flags pertaining to the operation of the Quad Detector associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description
1. x00000001	1.	Position monitoring mode (1 - enabled, 0 - disabled).
2. x00000002	2	Open loop operating mode (1 - enabled, 0 - disabled).
0x00000004	3.	Closed loop operating mode (1 - enabled, 0 - disabled).
	4. to 32	For Future Use

Note. The present operating mode can also be obtained by calling the GetQuadMode method.

T-Cube Quad Detector Method LoadParamSet

See Also Example Code

Visual Basic Syntax

Function **LoadParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be loaded

Returns

MG Return Code

Details

This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube Quad Detector Method ReadPositionDemands

See Also Example Code

Note. This method is applicable only when operating in closed loop mode.

Visual Basic Syntax

Function **ReadPositionDemands**(pfXPos As Float, pfYPos As Float) As Long

Parameters

pfXPos – the present X position demand value

pfYPos – the present Y position demand value

Returns

MG Return Code

Details.

When the unit is operated in closed loop mode, the XPos and YPos position demand signals are routed to the rear panel SMA connectors for use in driving a piezo positioning mirror or other optical device.

This method returns the present values for the XPos and YPos position demand signals (-10 V to +10V) in the *pfXPos* and *pfYPos* parameters.

T-Cube Quad Detector Method ReadSumDiffSignals

See Also Example Code

Note. This method is applicable only when operating in open loop mode.

Visual Basic Syntax

Function **ReadSumDiffSignals**(pfSum As Float, pfXDiff As Float, pfYDiff As Float) As Long

Parameters

pfSum – the present SUM value (0 to 10)

pfXDiff – the present X axis difference (XDIFF) signal value (-10 to 10)

pfYDiff – the present Y axis difference (YDIFF) signal value (-10 to 10)

Returns

MG Return Code

Details.

The TQD001 Quad Detector T-Cube has been designed to operate optimally with the PDQ80A Quad Detector. The detector consists of a 4-segment photodiode sensor array, which provides ‘Bottom minus Top’ (YDIFF) and ‘Left minus Right’ (XDIFF) difference signals, together with the SUM of the signals (total beam power) from all four quadrants of the photodiode array.

When the Quad Detector T-Cube is operated in ‘open loop’ mode, the difference signals (on the XDIFF, YDIFF and SUM connectors) can either be set to zero, or held at their latest closed loop values, (by calling the SetPosDemandParams method)

Alternatively, a particular output position may be specified by calling the SetPositionOutputs method.

This method reads the present values for the SUM, XDIFF and YDIFF signals, and returns values in the *pfSum*, *pfXDiff*, and *pfYDiff* parameters.

T-Cube Quad Detector Method SaveParamSet

See Also Example Code

Visual Basic Syntax

Function **SaveParamSet**(bstrName As String) As Long

Parameters

bstrName – the name under which the parameter set is to be saved

Returns

MG Return Code

Details

This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved. Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

T-Cube Quad Detector Method SetDetectorHead

See Also Example Code

Visual Basic Syntax

Function **SetDetectorHead**(lHeadType As Long) As Long

Parameters

lHeadType – the type of detector head being used

Returns

MG Return Code

Details.

The T-Cube PSD Autoalignment unit can be configured for use with a number of sensor/detector heads.

This method allows the user to set the type of detector head being used. The detector head type is set in the *lHeadType* parameter, which in turn accepts values from the HEADTYPE enumeration as follows:

1. HEAD_PDQ80A PDQ80A Quadrant Detector Head.
2. HEAD_PDQ30C PDQ30C Quadrant Detector Head
3. HEAD_PDP90A PDp90A Lateral Effect Detector Head
4. HEAD_OTHER Third Party Detector Head.

The type of detector head currently connected may be obtained by calling the GetDetectorHead method.

T-Cube Quad Detector Method SetDispIntensity

See Also Example Code

Visual Basic Syntax
Function **SetDispIntensity**(lIntensity As Long) As Long

Parameters

lIntensity - the display intensity (0 to 255)

Returns
MG Return Code

Details.

For TPA101 units
This method sets the intensity of the various LEDs on the front of the Quad Detector associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.
The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest).
The current setting for display intensity may be obtained by calling the GetDispIntensity method.

For KPA101 units
This method sets the intensity of the OLED display on the top of the unit associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.
The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest). The display can be turned off completely by entering a setting of zero, however, pressing the MENU button on the top panel will temporarily illuminate the display at its lowest brightness setting to allow adjustments. When the display returns to its default position display mode, it will turn off again

Note. 'Burn In' of the display can occur if it remains static for a long time. To prevent this, the display is automatically dimmed after a pre-defined time interval has elapsed. The time interval is not exposed through message calls, but can be accessed via the GUI Settings panel. See the handbook for further details.

The current setting for display intensity may be obtained by calling the GetDispIntensity method.

Quad Control Method SetKCubeTriggerParams

This method is applicable only to K-Cube Series position aligner units

See Also Example Code

Visual Basic Syntax
Function **SetKCubeTriggerParams** (lTrigChan As Long, lTrigMode As Long, lTrigPolarity As Long, lSumMin As Long, lSumMax As Long, lDiffThreshold As Long) As Long

Parameters
lTrigChan - the trigger port to be set, Trig1 or Trig2
lTrigMode - the trigger operating mode
lTrigPolarity - The active state of the trigger (i.e. logic high or logic low)
lSumMin - The lower limit when the trigger mode is set to TRIGOUT_SUM.
lSumMax - The upper limit when the trigger mode is set to TRIGOUT_SUM.
lDiffThreshold - The threshold when the trigger mode is set to TRIGOUT_DIFF

Returns
MG Return Code

Details

The K-Cube position aligner has two bidirectional trigger ports (TRIG1 and TRIG2) that can be independently configured either as an input or an output and assigned a function from the list of options described in the following section. The polarity (logic HIGH / LOW or rising / falling edge) can also be configured to suit the requirements of the equipment connected to these ports.

In the input operating modes the port is electrically configured as a TTL compatible logic input. When the port is driven with a voltage level below +0.8 V, it will read a logic LOW and when driven above +2.4V, it will read a logic HIGH. The ports have an internal weak pull-up resistor ensuring that a stable logic level is present on the inputs even when there is no driving source connected to it. This means that when unconnected the ports will read a logic HIGH. The internal pull-up also allows the direct connection of mechanical switches or other unpowered control devices.

In the output modes the port is electrically configured as a logic output using 5 Volt logic levels. The port is connected to the output driver logic with a 620 Ohm resistor in series; this resistor limits the maximum output current to approximately 8 mA and provides protection against the output being accidental short circuited to ground. The output can be used to drive the majority of digital inputs used on external equipment without any additional circuitry.

Warning: do not drive the TRIG ports from any voltage source that can produce an output in excess of the normal 0 to 5 Volt logic level range. In any case the voltage at the TRIG ports must be limited to -0.25 to +5.25 Volts.

This method sets the operating parameters of the trigger connectors on the front panel of the unit.

The applicable port (TRIG1 or TRIG2) is specified by the *ITrigChan* parameter, which takes values specified by the TRIGGERCHANNELS enumeration as follows:

1 = TRIG1

2 = TRIG2

The operating mode of the trigger is specified in the *ITrigMode* parameter, which in turn accept values from the TRIGGERPORTMODES enumeration as follows:

Input Trigger Modes

0x00 TRIG_DISABLED - The trigger IO is disabled. Selecting this option effectively results in the port returning to its default digital input configuration

0x01 TRIGIN_GPI - General purpose logic input. Other than being able to read the logic state of port there is no other functionality associated with it. The state of the port is returned in the periodic status update messages, or can be read by using the LLGetStatusBits method). In this mode the Triggering Polarity setting has no effect; the logic state of the input is returned as it is present on the port without inversion.

0x02 TRIGIN_LOOPOPENCLOSE - In this mode the port can be used to toggle the operating mode of the controller between open loop and closed loop modes. If the trigger polarity is selected to be "Active High", the operating mode toggles on the rising edge (LOW to HIGH transition) of the signal present on the TRIG input. Conversely, with "Active Low", the toggle takes place on the falling edge (HIGH to LOW transition).

Output Trigger Modes

0x0A TRIGOUT_GPO - In this operating mode the TRIG port functions as a simple digital output. The logic state of the output can be set using the LLSetGetDigOPs method. Other than being able to read the logic state of port there is no other functionality associated with it. The logic state of the output can be inverted by setting the Triggering Polarity parameter to "Low"; with this option selected the state of the output will be the opposite of the corresponding bit setting in the software call. The default state of the output in this mode is also the opposite of the option selected as the Triggering Polarity.

0x0B TRIGOUT_SUM - The state of the TRIG port is asserted depending on whether the SUM signal coming from the position sensor is inside the limits specified in the lSumMin and lSumMax parameters. If SUM is within the limits, the state will be the logic state selected in Triggering Polarity and conversely if it falls outside these limits, it will assume the opposite logic state. This mode can be used to detect the presence or absence of light falling on the position sensor; or that the optical power is within the expected limits. This option might be useful to signal a condition required for normal operation as under normal operating conditions the optical power is often expected to remain fairly constant. The lSumMin and lSumMax parameters are specified as a percentage of full scale, in the range 1% to 99%.

0x0C TRIGOUT_DIFF - The state of the TRIG port is asserted depending on whether both the XDIFF and the YDIFF signals coming from the position sensor are below the value set in the lDiffThreshold parameter. If both XDIFF and YDIFF are below the limit, the state will be the logic state selected in Triggering Polarity and conversely if either of them falls outside these limits, it will assume the opposite logic state. This mode can be used to signal whether or not the beam is close to the centre (beam aligned) position within a certain margin. In closed loop mode it also indicates that the controller is capable of tracking the changes in the beam position and maintain beam alignment. The lDiffThreshold parameter is specified as a percentage of full scale, in the range 1% to 99%.

0x0D TRIGOUT_SUMDIFF - This output mode is a 'logic AND' combination of the "Inside SUM range" and "Below Diff Threshold" conditions described above. Having to meet both conditions provides a more reliable indication of the normal closed loop operation when the beam is aligned and in the centre of the position sensor. In this scenario the SUM signal is within the expected limits (there is sufficient amount of light hitting the sensor) and both XDIFF and YDIFF are below a certain threshold (the beam is centralized). The second part of the condition, XDIFF and YDIFF below the threshold can also occur if the beam is blocked.

The polarity of the trigger pulse is specified in the lTrigPolarity parameter, which in turn accept values from the QUAD_TRIGGERPORTPOLARITY enumeration as follows:

0x01 The active state of the trigger port is logic HIGH 5V (trigger input and output on a rising edge).

0x02 The active state of the trigger port is logic LOW 0V (trigger input and output on a falling edge).

To obtain the current trigger parameter settings, see the GetKCubeTriggerParams method.

T-Cube Quad Detector Method SetLoopFilterParams

See Also Example Code

Visual Basic Syntax
Function **SetLoopFilterParams**(fDerivLPCutOffFreq As Float, fNotchCentreFreq As Float, fNotchQ As Float, lNotchFilterOn As Long, lDerivFilterOn As Long) As Long

Parameters
lDerivLPCutOffFreq - the cut off frequency of the derivative low pass filter
fNotchCentreFreq - the centre frequency of the notch filter
fNotchQ - the Q factor of the notch filter
lNotchFilterOn - enables and disables the notch filter
lDerivFilterOn - enables and disables the derivative low pass filter

Returns
MG Return Code

Details
This method is used to set the derivative cut off frequency and the notch filter for the PID loop associated with a particular ActiveX Control instance.

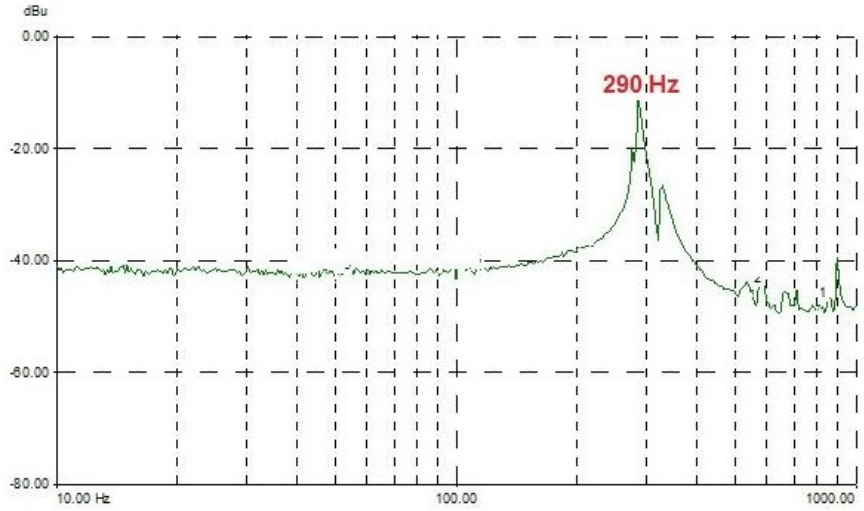
Derivative Low Pass Filter: The output of the derivative (differential) part of the PID controller can be passed through a tuneable low pass filter. Whilst the derivative component of the PID loop often improves stability (as it acts as a retaining force against abrupt changes in the system), it is prone to amplifying noise present in the system, as the derivative component is sensitive to changes between adjacent samples. To reduce this effect, a low pass filter can be applied to the samples. As noise often tends to contain predominantly high frequency components, the low pass filter can significantly decrease their contribution, often without diminishing the beneficial, stabilizing effect of the derivative action. In some applications enabling this filter can improve the overall closed loop performance.
The low pass filter is enabled in the lDerivFilterOn parameter, which takes values from the QUAD_DERIVFILTERENABLED enumeration as follows:

TQD101_DERIVFILTER_ON = 1

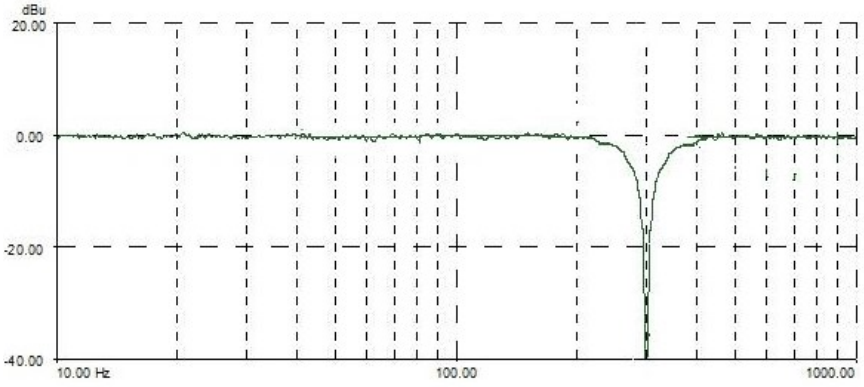
TQD101_DERIVFILTER_OFF = 2

The cut off frequency is set in the lDerivLPCutOffFreq parameter, in the range 0 to 10,000.

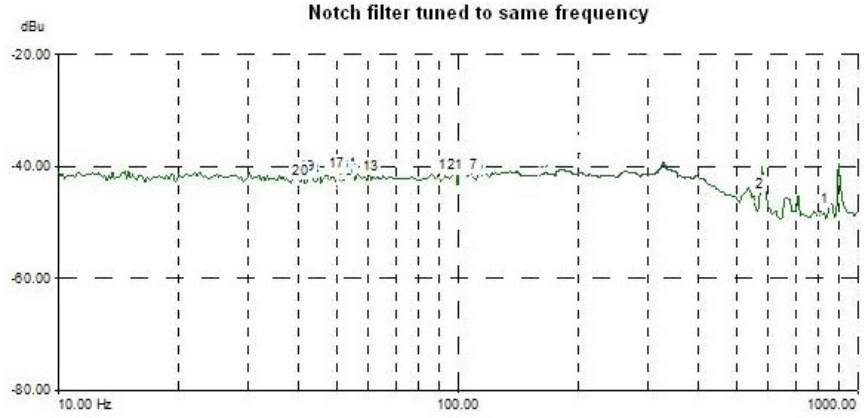
Notch Filter Settings - Due to their construction, most actuators are prone to mechanical resonance at well-defined frequencies. The underlying reason is that all spring-mass systems are natural harmonic oscillators. This proneness to resonance can be a problem in closed loop systems because, coupled with the effect of the feedback, it can result in oscillations. With some actuators (for example the ASM003), the resonance peak is either weak enough or at a high enough frequency for the resonance not to be troublesome. With other actuators (for example the PGM100) the resonance peak is very significant and needs to be eliminated for operation in a stable closed loop system. The notch filter is an adjustable electronic anti-resonance that can be used to counteract the natural resonance of the mechanical system.



Frequency response of actuator showing resonance at 290 Hz



Notch filter tuned to same frequency



The resonance is largely eliminated

As the resonance frequency of actuators varies with load in addition to the minor variations from product to product, the notch filter is tuneable so that its characteristics can be adjusted to match those of the actuator. In addition to its centre frequency, the bandwidth of the notch (or the equivalent quality factor, often referred to as the Q-factor) can also be adjusted. In simple terms, the Q factor is the centre frequency/bandwidth, and defines how wide the notch is, a higher Q factor defining a narrower ("higher quality") notch. Optimizing the Q factor requires some experimentation but in general a value of 5 to 10 is in most cases a good starting point.

The notch filter is enabled in the lNotchFilterOn parameter, which takes values from the QUAD_NOTCHFILTERENABLED enumeration as follows:

TQD101_NOTCHFILTER_ON = 1

TQD101_NOTCHILTER_OFF = 2

The centre frequency of the notch filter is set in the fNotchCentreFreq parameter, in the range 0 to 10000. The Q Factor is set in the f*NotchQ* parameter, in the range 0.1 to 100.

T-Cube Quad Detector Method SetPIDParams

See Also Example Code
Visual Basic Syntax
Function **SetPIDParams** (lProp As Long, lInt As Long, lDiff As Long) As Long
Parameters
lProp - the value of the proportional constant
lInt - the value of the integration constant
lDiff - the value of the differential constant
Returns
MG Return Code
Details.
This method sets the proportional, integration and differential feedback loop constants to the value specified in the *lProp*, *lInt* and *lDiff* parameters respectively. They apply when the quad detector unit is operated in closed loop mode, and position demand signals are generated at the rear panel SMA connectors by the feedback loops. These position demand voltages act to move the beam steering elements (e.g. a piezo driven mirror) in order to centralize a beam at the centre of the PSD head.
When operating in closed loop mode, the proportional, integral and differential (PID) constants can be used to fine tune the behaviour of the dual feedback loops to adjust the response of the position demand output voltages. The feedback loop parameters need to be adjusted to suit the different types of sensor that can be connected to the system. The default values have been optimized for the PDQ80A sensor.
Proportional – This term provides the force used to drive the piezo to the demand position, reducing the positional error.
It accepts values in the range 0 to 10000.
Integral – This term provides the 'restoring' force that grows with time, ensuring that the positional error is eventually reduced to zero.
It accepts values in the range 0 to 10000.
Derivative – This term provides the 'damping' force proportional to the rate of change of the position.
It accepts values in the range 0 to 10000.
To obtain the current settings for the PID constants, see the GetPIDParams method.

T-Cube Quad Detector Method SetPosDemandMinMax

See Also Example Code

Note. This method is applicable only when operating in closed loop mode.

Visual Basic Syntax

Function **SetPositionOutPuts**(fXPosMin As Float, fXPosMax As Float, fYPosMin As Float, fYPosMax As Float) As Long

Parameters

fXPosMin – the minimum X axis position value

fXPosMax – the maximum X axis position value

fYPosMin – the minimum Y axis position value

fYPosMax – the maximum Y axis position value

Returns

MG Return Code

Details.

When the Quad Detector T-Cube is operated in 'closed loop' mode, the XPos and YPos position demand signals are sent to the XDIFF and YDIFF SMA connectors. These signals are then routed to the positioning elements, (e.g. piezo drivers) to drive the positioning mirror to keep the beam central in the detector.

Under normal operating conditions, these signals are between –10V and +10V, however some applications may require the limits to be less than this.

This method is used to set the XPos and YPos minimum and maximum limits to the values entered in the *fXPosMin*, *fXPosMax*, *fYPosMin* and *fYPosMax* parameters.

The present settings for the XPos and YPos limits may be obtained by calling the GetPosDemandMinMax method.

T-Cube Quad Detector Method SetPosDemandParams

See Also Example Code

Visual Basic Syntax
Function **SetPosDemandParams**(lOpenLoopPosDemands As Long, lXPosSense As Long, lYPosSense As Long) As Long

Parameters
lOpenLoopPosDemands - the status of the feedback outputs in open loop
lXPosSense – the sense of the X axis position demand signal
lYPosSense - the sense of the Y axis position demand signal

Returns
MG Return Code

Details.
This method sets the position demand outputs (XPos and YPos) to different values, depending on whether open loop or closed loop operating mode is selected.
When the Quad Detector T-Cube is operated in 'open loop' mode, the position demand signals (on the XDIFF and YDIFF connectors) can either be set to zero, or held at their last closed loop value, according to the value entered in the *lOpenLoopPosDemands* parameter.
The parameter accepts values from the QUAD_OPENLOOPPOSDEMANDS enumeration as follows:
If *lOpenLoopPosDemands* is set to QUAD_OPENLOOPPOSDEMANDS_ZERO, then the outputs are set to zero when switching to Open Loop mode from Monitor or Closed loop mode.
If *lOpenLoopPosDemands* is set to QUAD_OPENLOOPPOSDEMANDS_HELD, then the outputs are held at the last closed loop value when switching from closed loop to open loop mode.

Due to the choice of piezo amplifier/driver or the configuration of mirrors (or other optical components) it is possible that certain application set ups may require the sense of the X and Y axis position demand signals to be inverted.
This is handled differently depending on the unit, and on the firmware version running.

TQD001 Firmware V1.0.3 or earlier
When the Quad Detector T-Cube is operated in 'closed loop' mode, the sense of the position demand signals can be set to positive or negative by entering a value in the *XPosSense* and *YPosSense* parameters, which in turn take values from the QUAD_CLOSEDLOOPPOSDEMENDSENSE enumeration.

If *lXPosSense* is set to QUAD_CLOSEDLOOPPOSDEMAND_POSITIVE the signals are positive when the beam is in the left hand quadrants of the detector array, and negative when in the right hand quadrants.
If *lYPosSense* is set to QUAD_CLOSEDLOOPPOSDEMAND_POSITIVE the signals are positive when the beam is in the top quadrants of the detector array, and negative when in the bottom quadrants.
If *lXPosSense* or *lYPosSense* is set to QUAD_CLOSEDLOOPPOSDEMAND_NEGATIVE, then the sense is reversed.

TQD001 Firmware V1.0.4 or later
When the Quad Detector T-Cube is operated in 'closed loop' mode, the sense of the position demand signals can be set to positive or negative.
Furthermore, in sensitive systems, e.g. scanning galvanometers, full feedback response could cause unstable behaviour, and some applications may require a the system response to be scaled down. Both the feedback sense and the gain can be set by entering a value from '-10' to '10' in the *XPosSense* and *YPosSense* parameters as follows:
If *lXPosSense* is set to '10' the signals are positive when the beam is in the left hand quadrants of the detector array, and negative when in the right hand quadrants. The gain of the system is set to '1'.
If *lXPosSense* is set to '-7' the signals are positive when the beam is in the right hand quadrants of the detector array, and negative when in the left hand quadrants. The gain of the system is set to '0.7'.
Similarly,
If *lYPosSense* is set to '10' the signals are positive when the beam is in the top quadrants of the detector array, and negative when in the bottom quadrants. The gain is set to '1'.
If *lYPosSense* is set to '-3' the signals are negative when the beam is in the bottom quadrants of the detector array, and positive when in the top quadrants. The gain is set to '0.3'.

TQD101
When the Quad Detector T-Cube is operated in 'closed loop' mode, the sense of the position demand signals can be set to positive or negative by entering a value in the *XPosSense* and *YPosSense* parameters, which in turn take values from the

QUAD_TQD101CLOSEDLOOPPOSDEMANDSENSE enumeration as follows:

If *lXPosSense* is set to CLOSEDLOOP_TQD101POSDEMANDSENSE_POSITIVE (i.e. 32767) the signals are positive when the beam is in the left hand quadrants of the detector array, and negative when in the right hand quadrants.
If *lYPosSense* is set to CLOSEDLOOP_TQD101POSDEMANDSENSE_POSITIVE (i.e. 32767) the signals are positive when the beam is in the top quadrants of the detector array, and negative when in the bottom quadrants.

If *lXPosSense* or *lYPosSense* is set to CLOSEDLOOP_TQD101POSDEMANDSENSE_NEGATIVE, (-32767) then the sense is reversed.

Note. The use of '0' is not allowed and will generate an error.
The current settings may be obtained by calling the GetPosDemendParams method

T-Cube Quad Detector Method SetPositionOutputs

See Also Example Code

Note. This method is applicable only when operating in open loop mode.

Visual Basic Syntax

Function **SetPositionOutPuts**(fXPos As Float, fYPos As Float) As Long

Parameters

fXPos – the X axis position output value

fYPos – the Y axis position output value

Returns

MG Return Code

Details.

In addition to calling the SetPosDemandParams method, the position demand signals (on the XDIFF, YDIFF connectors) can also be set by calling this method.

When the quad detector unit is used with a beam steering device (e.g. a piezo mirror via piezo drivers), this method allows the beam to be positioned by entering a value (-10 V to +10V) in the *fXPos* and *fYPos* parameters.

The open loop position demand values currently applied may be obtained by calling the GetPositionOutputs method.

T-Cube Quad Detector Method SetQuadDispMode

See Also Example Code

Visual Basic Syntax
Function **SetQuadDispMode**(lMode As Long) As Long

Parameters
lMode - the display setting (Difference or Position)

Returns
MG Return Code

Details.
The main display on the GUI panel can be set to show X and Y axis difference signals from the detector array (Difference) or the Xpos and Ypos position demand output signals fed to the positioning elements. (Position).
This method sets the display mode to the value specified in the *lMode* parameter, which in turn takes values from the DISPMODEenumeration as follows.
If *lMode* is set to *QUAD_DISPMODE_DIFF*, the display represents the X and Y axis difference signals from the detector (i.e. the voltage outputs from the rear panel SMA connectors in Monitor Mode).

If *lMode* is set to *QUAD_DISPMODE_POS*, the display represents the position of the XPos and YPos position demand output signals fed to the positioning elements (i.e. the voltage outputs from the rear panel SMA connectors in OPEN or CLOSED loop mode).
The current setting for the display mode may be obtained by calling the GetQuadDispMode method

Quad Control Method SetQuadMode

See Also Example Code

Visual Basic Syntax
Function **SetQuadMode**(lMode As Long) As Long

Parameters
lMode - the operating mode (Monitor, Open Loop or Closed Loop)

Returns
MG Return Code

Details.
The T-Cube Quad Detector can be set to operate in either open loop, closed loop or monitor mode. This method sets the operating mode for the unit, as specified in the *lMode* parameter, which in turn takes values from the QUADOPERMODE enumeration as follows.
If *lMode* is set to 1_*QUADMODE_MONITOR*, the unit operates in 'Monitor' mode. The X and Y axis difference signals and the sum signal from the photodiode array are fed through to the rear panel XDIFF, YDIFF and SUM SMA connectors for use in a monitoring application. The position of the beam in the photodiode array is represented on the Target display on the top panel of the unit.

If *lMode* is set to 2_*QUADMODE_OPENLOOP*, the unit operates in 'open loop' mode. The X and Y axis position demand signals (Xpos and Ypos) are fed through to the rear panel XDIFF and YDIFF SMA connectors. These signals are either set to zero, or held at their last closed loop value, depending on the setting called by the SetPosDemandParams method. This mode is typically used when making manual adjustments to the turning mirrors in the system, to bring the beam position within the adjustment range of the quad detector system.

If *lMode* is set to 3_*QUADMODE_CLOSEDLOOP*, the unit operates in closed loop' mode. The signal from the detector is interpreted by the unit, and the feedback circuit sends position demand signals (XOut and YOut) to the rear panel XDIFF and YDIFF connectors, which can be used to drive a pair of positioning elements (e.g. piezo controllers) in order to position the light beam within the center of the detector array.

If *lMode* is set to 4_*QUADMODE_AUTOOPENCLOSEDLOOP*, the unit operates in closed loop' mode, until the SUM signal falls below the value set in the *lSumMin* parameter of the SetKCubeTriggerParams method - see below for more details.

A Note About Automatic Open Loop/Closed Loop Switching

The controller is capable of switching automatically between open loop and closed loop operating modes, depending on whether there is sufficient optical power required for closed loop operation. Automatic Switching mode can be selected by setting the lMode parameter to 4_*QUADMODE_AUTOOPENCLOSEDLOOP.* It can also be selected from the top panel settings menu or by clicking the Enable Auto Open Loop checkbox on the GUI Control Settings panel.

If during closed loop operation the SUM signal falls below the minimum specified in the lSumMin parameter of the SetKCubeTriggerParams method, the controller will switch back to open loop mode. If subsequently the SUM signal rises above the limit again, the controller will switch back to closed loop mode.

The automatic switchover works in conjunction with the "Position Demands In Open Loop Mode" option in the SetPosDemandParams method, that defines whether the controller will hold (freeze) the XPOS and YPOS outputs when switching over to open loop or set them to zero.

Automatic switchover might be advantageous in scenarios where the beam might be temporarily blocked, for example during experiments involving manual manipulation of optical components, particularly when the beam path is quite long and the beam steering actuator can deflect the beam so far that it falls outside the sensor area. In setups like this and with the controller in closed loop, blocking the beam can result in the feedback loop ramping the XPOS and/or YPOS outputs to saturation and steering the beam completely outside the sensor area. When this happens, restoring the beam will not normally restore the beam alignment as at this point the feedback algorithm does not even see the beam. However, with automatic switchover the loss of light will stop the closed loop operation, optionally freeze the last valid beam position and prevent the outputs ramping up as an unintentional consequence of the loss of feedback signals. Later when the beam is restored, closed loop operation will resume and continue control starting from the last valid beam position.

Note that because automatic switchover assumes the knowledge of the last valid closed loop beam position that is lost when the controller is powered down, this option cannot be persisted. For a similar reason, the controller will always power up in open loop mode.

T-Cube Quad Detector Method SetSigOutputParams

See Also Example Code

Visual Basic Syntax

Function **SetSigOutPutParams**(lSigRouting As Long) As Long

Parameters

lSigRouting – the comms routing for the position demand signals

Returns

MG Return Code

Details.

The Quad Detector T-Cube outputs the XDIFF and YDIFF or XPos and YPos signals via the rear panel SMA connectors. The unit can also be set to output these signals via the T-Cube USB hub internal comms lines. This method sets the signal routing as specified by the *lSigRouting* parameter value, which in turn takes values from the QUAD_LVOUTPUTROUTEenumeration as follows.

If *lSigRouting* is set to QUAD_ANALOGUEOUT_SMAONLY, the X and Y signals are output only via the rear panel SMA connectors.

If *lSigRouting* is set to QUAD_ANALOGUEOUT_SMAHUB, the X and Y signals are output via the rear panel SMA connectors AND the USB hub internal comms lines.

The current setting for the signal routing may be obtained by calling the GetSigOutputParams method.

Please see the handbook supplied with the unit for more details.

T-Cube Quad Detector Method StartCtrl

See Also Example Code

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns

MG Return Code

Details

This method is used in conjunction with the HWSerialNum property, to activate an ActiveX control instance at runtime.

After the HWSerialNum property has been set to the required serial number (at design time or run time), this method can be called at run time to initiate the hardware enumeration sequence and activate the control.

Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

Example

```
Private Sub Form_Load()

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

APTQuad1.HWSerialNum = 89000001
```

```
' Start Quad Detector Reader control

APTQuad1.StartCtrl

End Sub
```

T-Cube Quad Detector Method StopCtrl

See Also Example Code

Visual Basic Syntax

Function **StopCtrl**() As Long

Returns

MG Return Code

Details

This method is called to deactivate the operation of an ActiveX control instance.

When this method is called, communication to the associated hardware unit is stopped. It is efficient programming practice to call StopCtrl on any ActiveX control instances that are no longer required to be active. For example, this may be the case in a multiple window client application where multiple instances of an ActiveX control need to communicate at different times with the same hardware unit

Example

```
Private Sub Form_Unload(Cancel As Integer)

' Stop Quad control and unload forms.

APTQuad1.StopCtrl

Unload Me

End Sub
```

Introduction To TEC Controller Programming

The ActiveX functionality for the TEC Controller is accessed via the APTTEC Control Object, and provides the functionality required for a client application to control a number of T-Cube TEC Controller units.
Every hardware unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.
The serial number must be allocated using the HWSerialNum property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.
The methods of the T-Cube TEC Controller can then be used to perform activities such as switching between display modes, reading the present TEC element temperature, and setting the LED display intensity.

For details on the operation of the TEC controller, refer to the manual for the unit, which is available from www.thorlabs.com.

TEC Controller Enumeration DISPMODE

This enumeration contains constants that specify the display mode of the front panel of the unit.

Constant	Name	Purpose
0	*DISPMODE_TACT*	the display shows the actual temperature of the TEC element
1.	*DISPMODE_TSET*	the display shows the demanded set point value.
2.	*DISPMODE_DELTA*	the display shows the difference between the actual temperature (TAct) and the set point temperature (TSet)..
3.	*DISPMODE_ITEC*	the display shows the current (in Amps) sourced into the TEC element by the controller.

TEC Controller Enumeration TEMPSENSOR

This enumeration contains constants that specify the type of TEC element controlled by the unit.

Constant	Name	Purpose
0	*SENSOR_IC_AD59X*	TEC element is a AD59x IC type transducer
1.	*SENSOR_THERM20KOHM*	TEC element is a 20kOhm thermistor.
2.	*SENSOR_THERM200KOHM*	TEC element is a 200kOhm thermistor.

T-Cube TEC Controller Method DeleteParamSet

See Also Example Code

Visual Basic Syntax

Function **DeleteParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be deleted

Returns

MG Return Code

Details

This method is used for housekeeping purposes, i.e. to delete previously saved settings which are no longer required. When calling *DeleteParamSet*, the name of the parameter set to be deleted is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (100056) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube TEC Controller Method DisableTEC

See Also Example Code

Visual Basic Syntax

Function **DisableTEC**() As Long

Returns

MG Return Code

Details

This method disables the output of the TEC Controller associated with the ActiveX control instance.

The associated unit is specified by the HWSerialNum property.

The output can be enabled by calling the EnableTEC method.

T-Cube TEC Controller Method DoEvents

See Also Example Code

Visual Basic Syntax

Function **DoEvents**() As Long

Returns

MG Return Code

Details

This method enables the APT server to allow message processing to take place within a client application. For example, if the client application is processing a lengthy loop and is making multiple calls to the server, this can often cause the client application to become non-responsive because other user interface message handling, such as button clicks, cannot be processed. Calling the DoEvents method allows such client application message handling to take place.

Note. The DoEvents method works in the same way as the DoEvents keyword found in Visual Basic.

T-Cube TEC Controller Method EnableTEC

See Also Example Code

Visual Basic Syntax

Function **EnableTEC**() As Long

Returns

MG Return Code

Details

This method enables the output of the TEC Controller associated with the ActiveX control instance.

The associated unit is specified by the HWSerialNum property.

The output can be disabled by calling the DisableTEC method.

T-Cube TEC Controller Method GetCurrentLim

See Also Example Code

Visual Basic Syntax

Function **GetCurrentLim**(pfCurrent As Float) As Long

Parameters

pfCurrent - Returns the maximum current limit (in Amps)

Returns

MG Return Code

Details

This method returns the maximum current that the TEC controller associated with the ActiveX control instance can source into the TEC element. Values are returned in the *pfCurrent* parameter in Amps, and must be within the range –1A to +1A.

The associated unit is specified by the HWSerialNum property.

The value can be set by calling the SetCurrentLim Method.

T-Cube TEC Controller Method GetDispIntensity

See Also Example Code

Visual Basic Syntax

Function **GetDispIntensity**(plIntensity As Long) As Long

Parameters

plIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method obtains the intensity of the LED display on the front of TEC controller associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The intensity is returned in the *plIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The display intensity may be set by calling the SetDispIntensity method.

T-Cube TEC Controller Method GetDispMode

See Also Example Code

Visual Basic Syntax

Function **GetDispMode**(plMode As Long) As Long

Parameters

plMode - the display setting (TAct, TDelta or ITec)

Returns

MG Return Code

Details.

The LED display window on the front of the unit can be set to display four different values; the actual temperature of the TEC element (TAct), the difference between the actual temperature and the set point (TDelta), the applied current (ITec), or the demanded set point value (TSet).

This method obtains the display mode for the TEC controller associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The mode is returned in the *plMode* parameter, which in turn takes values from the DISPMODE enumeration as follows.

If *plMode is set to DISPMODE_TACT*, the display shows the actual temperature of the TEC element.

If *plMode is set to DISPMODE_TSET*, the display shows the demanded set point value.

T-Cube TEC Controller Method GetPIDParams

See Also Example Code

Visual Basic Syntax

Function **GetPIDParams**(plProp As Long, plInt As Long, plDiff As Long) As Long

Parameters

plProp - the value of the proportional constant

plInt - the value of the integration constant

plDiff - the value of the differential constant

Returns

MG Return Code

Details.

This method obtains the proportional, integration and differential feedback loop constants and returns values in the *plProp*, *plInt* and *plDiff* parameters respectively. These parameters tune the feedback loop and thereby determine the response characteristics of the system.

When the TEC temperature is read (either from the GUI display or by calling the ReadTECTemp method) a discrepancy may exist between the value obtained and the demanded set point. This difference can be minimized by careful adjustment of the PID parameters.

Proportional – This term provides the force used to drive the temperature to the demand value, reducing the positional error.

It accepts values in the range 1 to 100.

Integral – This term provides the 'restoring' force that grows with time, ensuring that the temperature error is zero under a constant loading.

It accepts values in the range 0 to 100.

Derivative – This term provides the 'damping' force proportional to the rate of change of the temperature.

It accepts values in the range 0 to 100.

The PID constants can be set by calling the SetPIDParams method.

T-Cube TEC Controller Method GetTempSensor

See Also Example Code

Visual Basic Syntax

Function **GetTempSensor** (plType As Long) As Long

Parameters

plType - the type of TEC element

Returns

MG Return Code

Details

This method returns the type of TEC element associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The *plType* parameter specifies the type of element controlled by the unit. It takes values from the TEMPSENSOR enumeration:

If an AD59x transducer is selected, the temperature is set and displayed in °C.

If a 20kOhm or 200kOhm thermistor is selected, the temperature is set and displayed in kOhms.

The sensor type can be set or changed by calling the SetTempSensor method.

T-Cube TEC Controller Method GetTempSetpoint

See Also Example Code

Visual Basic Syntax

Function **GetTempSetpoint** (pfSetpoint As Float) As Long

Parameters

pfSetpoint - the current setting of the temperature setpoint

Returns

MG Return Code

Details

This method returns the temperature setpoint of the TEC controller associated with the ActiveX control instance. The value is returned in the *pfSetpoint* parameter.

Note. The value range, and the units in which the setpoint is specified are dependent upon the 'Sensor Type' selected (via the Settings panel or by calling the SetTempSensor method). If an IC type sensor is selected, the set point temperature is displayed in °C and can be set between 5 and 50°C. If a thermistor sensor is selected, the set point is displayed in kΩ. For a 20 kΩ sensor the range is 0 to 20 kΩ; for a 200 kΩ sensor the range is 0 to 200 kΩ.

Note. The maximum set point value is also dependent upon current limit parameter, which is set by calling the SetCurrentLimit method (or in the Settings panel). The setting value can be obtained by calling the GetCurrentLim method.

The associated unit is specified by the HWSerialNum property.

The temperature setpoint can be set by calling the SetTempSetpoint method.

T-Cube TEC Controller Method Identify

See Also Example Code

Visual Basic Syntax

Function **Identify**() As Long

Returns

MG Return Code

Details

This method allows the hardware unit associated with the ActiveX control instance to be identified. The associated unit is specified by the HWSerialNum property.

When the method is called, the front panel display on the relevant hardware unit will flash for a short period.

T-Cube TEC Controller Method LLGetStatusBits

See Also Example Code

Visual Basic Syntax
Function **LLGetStatusBits** (plStatusBits As Long) As Long

Parameters
plStatusBits – the 32-bit integer containing the status flags.

Returns
MG Return Code

Details
This low level method returns a number of status flags pertaining to the operation of the TEC controller associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.
These flags are returned in a single 32 bit integer parameter and can provide additional useful status information for client application development. The individual bits (flags) of the 32 bit integer value are described in the following table.

Hex Value	Bit Number	Description
0x00000001	1.	TEC output enabled state (1 - enabled, 0 - disabled).
	2. to 4	For Future Use
0x00000010	5	Display mode (1 – TAct, 0 - else).

0x00000020	6	Display mode (1 – TSet, 0 - else).
0x00000040	7	Display mode (1 – TDelta, 0 - else).
0x00000080	8	Display mode (1 – ITec, 0 - else).
	9 to 30	For Future Use
0x40000000	31	Error
0x80000000	32	For Future Use

T-Cube TEC Controller Method LoadParamSet

See Also Example Code

Visual Basic Syntax

Function **LoadParamSet**(bstrName As String) As Long

Parameters

bstrName – the name of the parameter set to be loaded

Returns

MG Return Code

Details

This method is used to load the settings associated with a particular ActiveX Control instance. When calling *LoadParamSet*, the name of the parameter set to be loaded is entered in the *bstrName* parameter. If the parameter set doesn't exist (i.e. bstrName parameter or serial number of the hardware unit is not recognised) then an error code (10004) will be returned, and if enabled, the Event Log Dialog is displayed.

T-Cube TEC Controller Method ReadTECCurrent

See Also Example Code

Visual Basic Syntax

Function **ReadTECCurrent** (pfCurrent As Float) As Long

Parameters

pfCurrent - the returned current sourced to the TEC element

Returns

MG Return Code

Details

This method returns the current sourced into the TEC element associated with the ActiveX control instance. The value is returned in Amps in the *pfCurrent* parameter.

The associated unit is specified by the HWSerialNum property.

T-Cube TEC Controller Method ReadTECTemp

See Also Example Code

Visual Basic Syntax

Function **ReadTECTemp** (pfTemp As Float) As Long

Parameters

pfTemp - the returned temperature of the TEC element

Returns

MG Return Code

Details

This method returns the actual temperature of the TEC element associated with the ActiveX control instance.

Note. The units in which the setpoint is specified are dependent upon the 'Sensor Type' selected (via the Settings panel or by calling the SetTempSensor method). If an IC type sensor is selected, the set point temperature is displayed in °C. If a thermistor sensor is selected, the set point is displayed in kΩ.

Note: Depending on the work environment and application, the value returned may be different to that obtained by the GetTempSetpoint method. The PID parameters may need to be adjusted (via the settings panel or by calling the SetPIDParams method) to minimize the difference (Tdelta) between the actual temperature and the demanded set point.

The associated unit is specified by the HWSerialNum property.

T-Cube TEC Controller Method SaveParamSet

See Also Example Code

Visual Basic Syntax

Function **SaveParamSet**(bstrName As String) As Long

Parameters

bstrName – the name under which the parameter set is to be saved

Returns

MG Return Code

Details

This method is used to save the settings associated with a particular ActiveX Control instance. These settings may have been altered either through the various method calls or through user interaction with the Control's GUI (specifically, by clicking on the 'Settings' button found in the lower right hand corner of the user interface).

When calling the *SaveParamSet* method, the *bstrName* parameter is used to provide a name for the parameter set being saved. Internally the APT server uses this name along with the serial number of the associated hardware unit to mark the saved parameter set. In this way, it is possible to use the SAME name to save the settings on a number of different Controls (i.e. hardware units) having differing serial numbers. It is this functionality that is relied on by the APTUser application when the user loads and saves graphical panel settings.

T-Cube TEC Controller Method SetCurrentLim

See Also Example Code

Visual Basic Syntax

Function **SetCurrentLim**(fCurrent As Float) As Long

Parameters

fCurrent - Sets the maximum current limit (in Amps)

Returns

MG Return Code

Details

This method sets the maximum current that the TEC controller associated with the ActiveX control instance can source into the TEC element. Values are set in the *fCurrent* parameter in Amps, and must be within the range –1A to +1A.

The associated unit is specified by the HWSerialNum property.

The present setting can be obtained by calling the GetCurrentLim Method.

T-Cube TEC Controller Method SetDispIntensity

See Also Example Code

Visual Basic Syntax

Function **SetDispIntensity**(lIntensity As Long) As Long

Parameters

lIntensity - the display intensity (0 to 255)

Returns

MG Return Code

Details.

This method sets the intensity of the LED display on the front of TEC controller associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The intensity is set in the *lIntensity* parameter, as a value from 0 (Off) to 255 (brightest).

The current setting for display intensity may be obtained by calling the GetDispIntensity method.

T-Cube TEC Controller Method SetDispMode

See Also Example Code

Visual Basic Syntax

Function **SetDispMode**(lMode As Long) As Long

Parameters

lMode - the display setting (TAct, TDelta or ITec)

Returns

MG Return Code

Details.

The LED display window on the front of the unit can be set to display four different values; the actual temperature of the TEC element (TAct), the difference between the actual temperature and the set point (TDelta), the applied current (ITec), or the demanded set point value (TSet).

This method sets the display mode for the TEC controller associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The mode is specified in the *lMode* parameter, which in turn takes values from the DISPMODE enumeration as follows.

If *lMode is set to DISPMODE_TACT*, the display shows the actual temperature of the TEC element.

If *lMode is set to DISPMODE_TSET*, the display shows the demanded temperature set point value.

T-Cube TEC Controller Method SetPIDParams

See Also Example Code

Visual Basic Syntax

Function **SetPIDParams**(lProp As Long, lInt As Long, lDiff As Long) As Long

Parameters

lProp - the value of the proportional constant

lInt - the value of the integration constant

lDiff - the value of the differential constant

Returns

MG Return Code

Details.

This method sets the proportional, integration and differential feedback loop constants to the value specified in the *lProp*, *lInt* and *lDiff* parameters respectively. These parameters tune the feedback loop and thereby determine the response characteristics of the system.

When the TEC temperature is read (either from the GUI display or by calling the ReadTECTemp method) a discrepancy may exist between the value obtained and the demanded set point. This difference can be minimized by careful adjustment of the PID parameters.

Proportional – This term provides the force used to drive the temperature to the demand value, reducing the positional error.

It accepts values in the range 1 to 100.

Integral – This term provides the 'restoring' force that grows with time, ensuring that the temperature error is zero under a constant loading.

It accepts values in the range 0 to 100.

Derivative – This term provides the 'damping' force proportional to the rate of change of the temperature.

It accepts values in the range 0 to 100.

To obtain the current settings for the PID constants, see the GetPIDParams method.

T-Cube TEC Controller Method SetTempSensor

See Also Example Code

Visual Basic Syntax

Function **SetTempSensor** (lType As Long) As Long

Parameters

lType - the type of TEC element

Returns

MG Return Code

Details

This method sets the type of TEC element associated with the ActiveX control instance. The associated unit is specified by the HWSerialNum property.

The *lType* parameter specifies the type of element controlled by the unit. It takes values from the TEMPSENSOR enumeration:

If an AD59x transducer is selected, the temperature is set and displayed in °C.

If a 20kOhm or 200kOhm thermistor is selected, the temperature is set and displayed in kOhms.

The present setting for the sensor type can be obtained by calling the GetTempSensor method.

T-Cube TEC Controller Method SetTempSetpoint

See Also Example Code

Visual Basic Syntax

Function **SetTempSetpoint** (fSetpoint As Float) As Long

Parameters

fSetpoint - the value of the temperature setpoint

Returns

MG Return Code

Details

This method specifies the temperature setpoint of the TEC controller associated with the ActiveX control instance. The value is set in the *fSetpoint* parameter.

Note. The value range, and the units in which the setpoint is specified are dependent upon the 'Sensor Type' selected (via the Settings panel or by calling the SetTempSensor method). If an IC type sensor is selected, the set point temperature is displayed in °C and can be set between 5 and 50°C. If a thermistor sensor is selected, the set point is displayed in kΩ. For a 20 kΩ sensor the range is 0 to 20 kΩ; for a 200 kΩ sensor the range is 0 to 200 kΩ.

Note. The maximum set point value is also dependent upon current limit parameter, which is set by calling the SetCurrentLimit method (or in the Settings panel). The setting value can be obtained by calling the GetCurrentLim method.

The associated unit is specified by the HWSerialNum property.

The current value for the temperature setpoint can be obtained by calling the GetTempSetpoint method.

T-Cube TEC Controller Method StartCtrl

See Also Example Code

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns

MG Return Code

Details

This method is used in conjunction with the HWSerialNum property, to activate an ActiveX control instance at runtime.

After the HWSerialNum property has been set to the required serial number (at design time or run time), this method can be called at run time to initiate the hardware enumeration sequence and activate the control.

Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

Example

```
Private Sub Form_Load()

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

APTTEC1.HWSerialNum = 87000001

' Start TEC control

APTTEC1.StartCtrl

End Sub
```

T-Cube TEC Controller Method StopCtrl

See Also Example Code

Visual Basic Syntax

Function **StopCtrl**() As Long

Returns

MG Return Code

Details

This method is called to deactivate the operation of an ActiveX control instance.

When this method is called, communication to the associated hardware unit is stopped. It is efficient programming practice to call StopCtrl on any ActiveX control instances that are no longer required to be active. For example, this may be the case in a multiple window client application where multiple instances of an ActiveX control need to communicate at different times with the same hardware unit

Example

```
Private Sub Form_Unload(Cancel As Integer)

' Stop TEC control and unload forms.

APTTEC1.StopCtrl

Unload Me

End Sub
```

T-Cube TEC Controller Property APTHelp

See Also Example Code

Visual Basic Syntax

Property **APTHelp** As Boolean

Returns

MG Return Code

Details

This property specifies the help file that will be accessed when the user presses the F1 key. If APTHelp is set to 'True', the main server helpfile APTServer will be launched. If APTHelp is set to 'False', the helpfile is the responsibility of the application programmer.

T-Cube TEC Controller Property HWSerialNum

See Also Example Code

Visual Basic Syntax

Property **HWSerialNum** As Long

Returns

MG Return Code

Details

This property specifies the serial number of the hardware unit to be associated with an ActiveX control instance.

The serial number must be set in the *HWSerialNum* property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.

Every APT control unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.

For example, if two or more T-Cube TEC Controller units are connected to the PC, different instances of the APTTEC ActiveX Control can be included in a client application. If each of these Control instances is programmed with a unique hardware serial number, then they will communicate with their associated hardware units. In this way, multiple graphical control panels communicating with multiple hardware units can easily be incorporated into a custom client application.

After a serial number has been allocated, an ActiveX control instance must be activated at run time by calling the StartCtrl method.

Note. The factory programmed serial number also has a model type identifier incorporated. This ensures for example that the serial number for an APT TEC controller cannot be assigned to an APT NanoTrak ActiveX control. If a serial number is set that is either illegal or for which no hardware unit exists, then the ActiveX control will remain inactive, i.e. the control's GUI will appear 'dead'.

Example

```
Private Sub Form_Load()

' Start system.

frmSystem.MG17System1.StartCtrl

' Set serial number

' (use number of actual HW unit or simulated unit - see APTConfig for details)

APTTEC1.HWSerialNum = 87000001

' Start APTTEC control

APTTEC1.StartCtrl

End Sub
```

Introduction to Flipper Programming

The ‘Motor’ ActiveX Control provides the functionality required for a client application to control one or more of the MFF series of Filter Flipper units.

A single Motor ActiveX Control instance can be used to control each flipper unit. To specify the particular unit being addressed, every unit is factory programmed with a unique 8-digit serial number. This serial number is key to the operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus. The serial number must be specified using the HWSerialNum property before an

ActiveX control instance can communicate with the hardware unit. This can be done at design time or at run time. Note that the appearance of the ActiveX Control GUI (graphical user interface) will change to the required format when the serial number has been entered.

The Methods and Properties of the Motor ActiveX Control can be used to perform activities such as setting the transit time, button operating modes and Digital IO Pulse width.

Where applicable, the target channel is identified in the *IChanID* parameter and on single channel units, this must be set to CHAN1_ID. See the HWCHANNEL enumeration for more details.

For details on the operation of the flipper unit, refer to the manual available from www.thorlabs.com.

Flipper Enumeration MFFOPERMODE

This enumeration contains constants that specify operating mode of the MFF filter flipper.

Constant	Name	Purpose
1.	MFFOPERMODE_IP_TOGGLEPOS	Sets IO connector to input and 'toggle position' mode. In this mode, the input signal causes flipper to move to other position).
2.	MFFOPERMODE_IP_GOTOPOS	Sets IO connector to input and 'goto position' mode. In this mode, the input signal dictates flipper position, POS 1 or POS 2. as dictated by the Button Input or Button Input (Swap Pos) parameters set in the SetMFFOperParams method.
3.	MFFOPERMODE_OP_TATPOS	Sets IO connector to output mode, where the O/P signal indicates the flipper is 'at position'.
4.	MFFOPERMODE_OP_MOVING	Sets IO connector to output mode, where the O/P signal indicates the flipper is in motion (i.e. between positions).

Flipper Enumeration MFFSIGMODE

This enumeration contains constants that specify signal mode of the MFF filter flipper.

Constant	Name	Purpose
01	MFFSIGMODE_IP_BUTTON	The connector can be short circuited (e.g. with button). If the Operating Mode is set to Input:Toggle Position then a short circuit causes the flipper to toggle position. If the Operating Mode is set to Input: Goto Position then a short circuit causes the flipper to move to Pos 1 and open circuit causes flipper to move to POS 2.
02	MFFSIGMODE_IP_LOGIC	The connector is set to logic input where a logic transition (edge) dictates flipper operation. If the Operating Mode above set to Input:Toggle Position, then a LO to HI edge causes flipper to toggle position. If the Operating Mode is set to Input: Goto Position, then a LO to HI edge causes the flipper to move to POS 1 and a HI to LO edge causes the flipper to move to POS 2.
04	MFFSIGMODE_IP_SWAP	This parameter can be 'Bitwise Ored' with either the button or the logic parameters above, such that the open circuit and short circuit or the edge functionality is swapped.
10	MFFSIGMODE_OP_LEVEL	The connector is set to a logic output where the logic transition (edge) represents flipper position. If the Operating Mode above is set to Output: At Position, then a LO to HI edge (HI level) indicates flipper is at POS 1 and a HI to LO edge (LO level) indicates the flipper is at POS 2. If the Operating Mode above is set to Output: InMotion, then a LO to HI edge (HI level) indicates the flipper is moving between positions and a HI to LO edge (LO level)

		indicates the flipper has stopped moving.
20	MFFSIGMODE_OP_PULSE	The connector is set to a logic output where a logic pulse indicates flipper operation. If the Operating Mode above is set to Output: At Position, then a logic HI pulse indicates flipper has reached a position. If the Operating Mode above is set to Output: InMotion, then a logic HI pulse indicates the flipper has started moving. The Pulse width is set in the Signal Width paramter below.
40	MFFSIGMODE_OP_INVERT	This parameter can be 'Bitwise Ored' with either the level (edge) or the pulse parameters above, such that the level or pulse functionality is swapped.

Filter Flipper Method GetMFFOperParams

See Also Example Code

Visual Basic Syntax
Function **GetMFFOperParams**(lChanID As Long, plTransitTime As Long, plDigIO1OperMode As Long, plDigIO1SigMode As Long, plDigIO1PulseWidth As Long, plDigIO2OperMode As Long, plDigIO2SigMode As Long, plDigIO2PulseWidth As Long,) As Long

lChanID - the channel identifier
plTransitTime - the time taken (in milliseconds) for the flipper to move from position 1 to position 2 and vice versa.
plDigIO1OperMode - the operating mode for DIG IO 1
plDigIO1SigMode - the functionality of the DIG IO 1 input/output signal.
plDigIO1PulseWidth - the pulse width for DIG IO 1 (when set to a Logic Pulse Output)
plDigIO2OperMode - the operating mode for DIG IO 2
plDigIO2SigMode - the functionality of the DIG IO 2 input/output signal.
plDigIO2PulseWidth - the pulse width for DIG IO 2 (when set to a Logic Pulse Output)

Returns
MG Return Code

Details
This method is used to obtain the present setting for various operating parameters that dictate the function of the flipper unit. The applicable channel is specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
Note: On single channel units such as the MFF Filter Flipper, the *lChanID* parameter should be set to CHAN1_ID.

The time taken for the flipper to move from position 1 to position 2 and vice versa is returned in the *plTransitTime* parameter, in milliseconds.

The following parameters dictate the function of the SMA Digital I/O connector DIGIO1 on the unit. The Digital I/O Pin 2 parameters behave in exactly the same way.

plDigIO1OperMode - This parameter returns the operating mode for DIG IO 1. It takes values specified by the MFFOPERMODE enumeration as follows:

Constant	Name	Purpose
1.	MFFOPERMODE_IP_TOGGLEPOS	Sets IO connector to input and 'toggle position' mode. In this mode, the input signal causes flipper to move to other position).
2.	MFFOPERMODE_IP_GOTOPOS	Sets IO connector to input and 'goto position' mode. In this mode, the input signal dictates flipper position, POS 1 or POS 2. as dictated by the Button Input or Button Input (Swap Pos) parameters set DigIOSigMode parameter below.
3.	MFFOPERMODE_OP_TATPOS	Sets IO connector to output mode, where the O/P signal indicates the flipper is 'at position'.
4.	MFFOPERMODE_OP_MOVING	Sets IO connector to output mode, where the O/P signal indicates the flipper is in motion (i.e. between positions).

plDigIO1SigMode - This parameter returns the functionality of the input/output signal. It takes values from the MFFSIGMODE enumeration

as follows:

Constant	Name	Purpose
01	MFFSIGMODE_IP_BUTTON	The connector can be short circuited (e.g. with button). If the Operating Mode is set to Input:Toggle Position then a short circuit causes the flipper to toggle position. If the Operating Mode is set to Input: Goto Position then a short circuit causes the flipper to move to Pos 1 and open circuit causes flipper to move to POS 2.
02	MFFSIGMODE_IP_LOGIC	The connector is set to logic input where a logic transition (edge) dictates flipper operation. If the Operating Mode above set to Input:Toggle Position, then a LO to HI edge causes flipper to toggle position. If the Operating Mode is set to Input: Goto Position, then a LO to HI edge causes the flipper to move to POS 1 and a HI to LO edge causes the flipper to move to POS 2.
04	MFFSIGMODE_IP_SWAP	This parameter can be 'Bitwise Ored' with either the button or the logic parameters above, such that the open circuit and short circuit or the edge functionality is swapped.
10	MFFSIGMODE_OP_LEVEL	The connector is set to a logic output where the logic transition (edge) represents flipper position. If the Operating Mode above is set to Output: At Position, then a LO to HI edge (HI level) indicates flipper is at POS 1 and a HI to LO edge (LO level) indicates the flipper is at POS 2. If the Operating Mode above is set to Output: InMotion, then a LO to HI edge (HI level) indicates the flipper is moving between positions and a HI to LO edge (LO level) indicates the flipper has stopped moving.
20	MFFSIGMODE_OP_PULSE	The connector is set to a logic output where a logic pulse indicates flipper operation. If the Operating Mode above is set to Output: At Position, then a logic HI pulse indicates flipper has reached a position. If the Operating Mode above is set to Output: InMotion, then a logic HI pulse indicates the flipper has started moving. The Pulse width is set in the Signal Width paramter below.
40	MFFSIGMODE_OP_INVERT	This parameter can be 'Bitwise Ored' with either the level (edge) or the pulse parameters above, such that the level or pulse functionality is swapped.

plDigIO1PulseWidth - The pulse width when the Digital Signal Mode described previously is set to Logic Pulse Output or Logic Pulse Output (Inverted).

Filter Flipper Method SetMFFOperParams

See Also Example Code

Visual Basic Syntax
Function **SetMFFOperParams**(lChanID As Long, lTransitTime As Long, lDigIO1OperMode As Long, lDigIO1SigMode As Long, lDigIO1PulseWidth As Long, lDigIO2OperMode As Long, lDigIO2SigMode As Long, lDigIO2PulseWidth As Long,) As Long

Parameters
lChanID - the channel identifier
lTransitTime - the time taken (in milliseconds) for the flipper to move from position 1 to position 2 and vice versa.
lDigIO1OperMode - the operating mode for DIG IO 1
lDigIO1SigMode - the functionality of the DIG IO 1 input/output signal.
lDigIO1PulseWidth - the pulse width for DIG IO 1 (when set to a Logic Pulse Output)
lDigIO2OperMode - the operating mode for DIG IO 2
lDigIO2SigMode - the functionality of the DIG IO 2 input/output signal.
lDigIO2PulseWidth - the pulse width for DIG IO 2 (when set to a Logic Pulse Output)

Returns
MG Return Code

Details
This method is used to set various operating parameters that dictate the function of the flipper unit. The applicable channel is

specified by the *lChanID* parameter, which takes values specified by the HWCHANNEL enumeration.
Note: On single channel units such as the MFF Filter Flipper, the *lChanID* parameter should be set to CHAN1_ID.

The time taken for the flipper to move from position 1 to position 2 and vice versa is specified in the *lTransitTime* parameter, in milliseconds.

The following parameters dictate the function of the SMA Digital I/O connector DIGIO1 on the unit. The Digital I/O Pin 2 parameters behave in exactly the same way.

lDigIO1OperMode - This parameter specifies the operating mode for DIG IO 1. It takes values specified by the MFFOPERMODE enumeration as follows:

Constant	Name	Purpose
1.	MFFOPERMODE_IP_TOGGLEPOS	Sets IO connector to input and 'toggle position' mode. In this mode, the input signal causes flipper to move to other position).
2.	MFFOPERMODE_IP_GOTOPOS	Sets IO connector to input and 'goto position' mode. In this mode, the input signal dictates flipper position, POS 1 or POS 2. as dictated by the Button Input or Button Input (Swap Pos) parameters set DigIOSigMode parameter below.
3.	MFFOPERMODE_OP_TATPOS	Sets IO connector to output mode, where the O/P signal indicates the flipper is 'at position'.
4.	MFFOPERMODE_OP_MOVING	Sets IO connector to output mode, where the O/P signal indicates the flipper is in motion (i.e. between positions).

lDigIO1SigMode - This parameter specifies the functionality of the input/output signal. It takes values from the MFFSIGMODE enumeration as follows:

Constant	Name	Purpose
01	MFFSIGMODE_IP_BUTTON	The connector can be short circuited (e.g. with button). If the Operating Mode is set to Input:Toggle Position then a short circuit causes the flipper to toggle position. If the Operating Mode is set to Input: Goto Position then a short circuit causes the flipper to move to Pos 1 and open circuit causes flipper to move to POS 2.
02	MFFSIGMODE_IP_LOGIC	The connector is set to logic input where a logic transition (edge) dictates flipper operation. If the Operating Mode above set to Input:Toggle Position, then a LO to HI edge causes flipper to toggle position. If the Operating Mode is set to Input: Goto Position, then a LO to HI edge causes the flipper to move to POS 1 and a HI to LO edge causes the flipper to move to POS 2.
04	MFFSIGMODE_IP_SWAP	This parameter can be 'Bitwise Ored' with either the button or the logic parameters above, such that the open circuit and short circuit or the edge functionality is swapped.
10	MFFSIGMODE_OP_LEVEL	The connector is set to a logic output where the logic transition (edge) represents flipper position. If the Operating Mode above is set to Output: At Position, then a LO to HI edge (HI level) indicates flipper is at POS 1 and a HI to LO edge (LO level) indicates the flipper is at POS 2. If the Operating Mode above is set to Output: InMotion, then a LO to HI edge (HI level) indicates the flipper is moving between positions and a HI to LO edge (LO level) indicates the flipper has stopped moving.
20	MFFSIGMODE_OP_PULSE	The connector is set to a logic output where a logic pulse indicates flipper operation. If the Operating Mode above is set to Output: At Position, then a logic HI pulse indicates flipper has reached a position. If the Operating Mode above is set to Output: InMotion, then a logic HI pulse indicates the flipper has started moving. The Pulse width is set in the Signal Width paramter below.
40	MFFSIGMODE_OP_INVERT	This parameter can be 'Bitwise Ored' with either the level (edge) or the pulse parameters above, such that the level or pulse functionality is swapped.

lDigIO1PulseWidth - The pulse width when the Digital Signal Mode described previously is set to Logic Pulse Output or Logic Pulse Output (Inverted).

Introduction To Solenoid Controller Programming

The ActiveX functionality for the Solenoid Controller is accessed via the Motor Control Object, and provides the functionality required for a client application to control a number of Solenoid Controller units.
Every hardware unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.
The serial number must be allocated using the HWSerialNum property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.
The methods of the T-Cube Solenoid Controller can then be used to perform activities such as switching the solenoid ON and OFF, changing the ON and OFF times and switching between operating modes.
Currently, there are only single channel T-Cube units available, but dual channel units may be available in the future and these methods have been written to be generic and apply equally to both single and dual channel units. The target channel is identified in the *lChanID* parameter and on the current single channel units, this must be set to CHAN1_ID. See the HWCHANNEL enumeration for more details.

For details on the operation of the solenoid controller, refer to the manual for the unit, which is available from www.thorlabs.com.

Introduction to System Control Programming

The System Control Object provides the functionality required the system to recognize the types of unit connected to the system, and the number of each type of unit.

Every hardware unit is factory programmed with a unique 8-digit serial number. This serial number is key to operation of the APT Server software and is used by the Server to enumerate and communicate independently with multiple hardware units connected on the same USB bus.

The serial number must be allocated using the HWSerialNum property, before an ActiveX control can communicate with the hardware unit. This can be done at design time or at run time.

The methods of the System Control can then be used to obtain the types and number of hardware units connected, and to recognize automatically when a unit is added or removed.

For details on the operation of the System control, refer to the Operating Guide.

System Control Enumeration APT_HW_TYPES_EX

This enumeration contains constants that specify the type of hardware unit.

Constant	Name	Purpose
11	USB_BSC001	1. Channel, 48V Benchtop Stepper Driver
12	USB_BSC101	1. Channel, 48V Benchtop Stepper Driver
13	USB_BSC002	2. Channel, 48V Benchtop Stepper Driver
14	USB_BDC001	1. Channel, 48V Benchtop DC Servo Driver
15	USB_BPC001	1. Channel, 75V Benchtop Piezo Driver
16	USB_BPC101	1. Channel, 75V Benchtop Piezo Driver
17	USB_BPC002	2. Channel, 75V Benchtop Piezo Driver
18	USB_BNT001	Benchtop NanoTrak Controller
19	USB_BMS001	1. Channel, 15V Benchtop Mini Stepper Driver
20	USB_BMS002	2. Channel, 15V Benchtop Mini Stepper Driver
21	CARD_SCC101	1. Channel, 48V Stepper Controller Card
22	CARD_DCC101	1. Channel, 48V DC Servo Controller Card
23	CARD_PCC101	1. Channel, 75V Piezo Controller Card (open/closed loop)
24	USB_ODC001	1. Channel, 15V OptoDCDriver, DC Servo controller
25	USB_OST001	1. Channel, 15V OptoSTDriver, Stepper Controller
26	MOD_MST601	Stepper Motor Control Module
27	MOD_MPZ601	Piezo Control Module
28	MOD_MNA601	NanoTrak Control Module

System Control Enumeration MG17_HW_TYPES

This enumeration contains constants that specify the type of hardware unit.

Constant	Name	Purpose
6	USB_STEPPER_DRIVE	Identifies a USB standalone stepper motor controller
7	USB_PIEZO_DRIVE	Identifies a USB standalone piezo controller
8.	USB_NANOTRAK	Identifies a USB standalone NanoTrak controller

System Control Event HWUnitAdded

This Event is not implemented.

System Control Event HWUnitRemoved

This Event is not implemented.

System Control Method GetHWInfo

See Also Example Code

Visual Basic Syntax

Function **GetHWInfo**(lSerialNum As Long, bstrModel As String, bstrSWVer As String, bstrHWNotes As String) As Long

Parameters

lSerialNum - the serial number specifying the hardware unit

bstrModel - the returned model number

bstrSWVer - the returned software version

bstrHWNotes - notes appertaining to the specified hardware unit

Returns

MG Return Code

Details

This method returns information about the hardware unit specified by the *lSerialNum* parameter.

The *bstrModel* parameter returns the model number.

The *bstrSWVer* parameter returns the software version currently installed.

The *bstrHWNotes* parameter returns a string which contains any notes relating to the specified hardware unit.

System Control Method GetHWSerialNum

See Also Example Code

Visual Basic Syntax

Function **GetHWSerialNum**(lHWType As Long, lIndex As Long, plSerialNum As Long) As Long

Parameters

lHWType - the type of hardware unit

lIndex - the index number specifying the hardware unit

plSerialNum - the returned serial number of the hardware unit

Returns

MG Return Code

Details

This method returns the serial number of the hardware unit specified by the *lHWType* and *lIndex* parameters.

The *lHWType* parameter takes values specified by the MG17_HW_TYPES enumeration.

The *lIndex* parameter is zero based and has a maximum value equivalent to the total number of hardware units of type *lHWType,* minus 1.

System Control Method GetHWSerialNumberEx

See Also Example Code

Visual Basic Syntax

Function **GetHWSerialNumEx**(lHWType As Long, lIndex As Long, plSerialNum As Long) As Long

Parameters

lHWType - the type of hardware unit

lIndex - the index number specifying the hardware unit

plSerialNum - the returned serial number of the hardware unit

Returns

MG Return Code

Details

This method returns the serial number of the hardware unit specified by the *lHWType* and *lIndex* parameters.

The *lHWType* parameter takes values specified by the APT_HW_TYPES_EX enumeration.

The *lIndex* parameter is zero based and has a maximum value equivalent to the total number of hardware units of type *lHWType,* minus 1, e.g. for a series of BSC101 units, the first unit has an ident of '0', the next an ident of '1' etc.

Note. For BSC103 and BPC103 units, this method applies to the daughter cards fitted and NOT the main unit. Each single channel daughter card has its own unique serial number set at the factory, and it is this number is returned in the *plSerialNum* parameter.

The serial number of a card is displayed in the upper right hand corner of the associated GUI panel as shown below.

System Control Method GetHWType

See Also Example Code

Visual Basic Syntax

Function **GetHWType**(lSerialNum As Long, plHWType As Long) As Long

Parameters

lSerialNum - the serial number specifying the hardware unit

plHWType - the type of hardware unit

Returns

MG Return Code

Details

This method returns the type of hardware unit for the hardware unit specified by the *lSerialNum* parameter.

The *plHWType* parameter takes values specified by the MG17_HW_TYPES enumeration..

System Control Method GetHWTypeEx

See Also Example Code

Visual Basic Syntax

Function **GetHWTypeEx**(lSerialNum As Long, plHWType As Long) As Long

Parameters

lSerialNum - the serial number specifying the hardware unit

plHWType - the type of hardware unit

Returns

MG Return Code

Details

This method returns the type of hardware unit for the unit serial number specified by the *lSerialNum* parameter.

The *plHWType* parameter takes values specified by the APT_HW_TYPES_EX enumeration.

Note. For BSC103 and BPC103 units, this method applies to the daughter cards fitted and NOT the main unit. Each single channel daughter card has its own unique serial number set at the factory, and it is this number that should be passed into the *lSerialNum* parameter. The *plHWType* parameter then returns a value from the APT_HW_TYPES_EX enumeration, relating to CARD_SCC101 (for stepper cards) CARD_DCC101 (for DC servo cards) or CARD_PCC101 (for piezo cards).

The serial number of a card is displayed in the upper right hand corner of the associated GUI panel as shown below.

System Control Method GetNumHWUnits

See Also Example Code

Visual Basic Syntax

Function **GetNumHWUnits**(lHWType As Long, plNumUnits As Long) As Long

Parameters

lHWType - the type of hardware unit

plNumUnits - the number of hardware units of the specified type connected

Returns

MG Return Code

Details

This method returns in the *plNumUnits* parameter, the number of hardware units of the type specified by the *lHWType* parameter.

The *lHWType* parameter takes values specified by the MG17_HW_TYPES enumeration.

System Control Method GetNumHWUnitsEx

See Also Example Code

Visual Basic Syntax

Function **GetNumHWUnitsEx**(lHWType As Long, plNumUnits As Long) As Long

Parameters

lHWType - the type of hardware unit

plNumUnits - the number of hardware units of the specified type connected

Returns

MG Return Code

Details

This method returns the number of hardware units of the type specified by the *lHWType* parameter.

The *lHWType* parameter takes values specified by the APT_HW_TYPES_EX enumeration.

Note. For BSC103 and BPC103 units, this method applies to the daughter cards fitted and NOT the main unit. For example, if a single BSC103 unit is fitted with 3 stepper motor daughter cards, the following method call;

MG17Motor1.GetNumHWUnitsEx(CARD_SCC101, plNumUnits)

results in a value of 3 being returned in the *plNumUnits* parameter.

Similarly, if 2 BSC103 units are connected to the PC, plNumUnits returns a value of 6.

Use the CARD_PCC101 enumeration constant when calling this method to retrieve the number of piezo cards for BPC103 units.

System Control Method StartCtrl

See Also Example Code

Visual Basic Syntax

Function **StartCtrl**() As Long

Returns

MG Return Code

Details

This method is called at run time to initiate the hardware enumeration sequence and activate the control.

Note that because StartCtrl is a runtime method call, ActiveX control instances do not communicate with physical hardware units at design time. This is required APT server operation in order to cater for the variety of program environments available and their individual ActiveX control instantiation mechanisms.

Return Codes

General

`MG17_UNKNOWN_ERR = 10000`
An unknown Server error has occurred.
`MG17_INTERNAL_ERR = 10001`
A Server internal error has occurred.
`MG17_FAILED = 10002`
A Server call has failed.
`MG17_INVALIDPARAM_ERR = 10003`
An attempt has been made to pass a parameter that is invalid or out of range. In the case of motor commands, this error may occur when a move is requested that exceeds the stage travel or exceeds the calibration data
`MG17_SETTINGSNOTFOUND = 10004`
An attempt has been made to save or load control parameters to the registry (using the SaveParamSet or LoadParamSet methods) when the unit serial number has not been specified.
MG17_DLLNOTINITIALISED = 10005
APT DLL not intialised.

PC System

MG17_DISKACCESS_ERR = 10050
An error has occurred whilst accessing the disk. Check that the drive is not full, missing or corrupted.
MG17_ETHERNET_ERR = 10051
An error has occurred with the ethernet connections or the windows sockets.
MG17_REGISTRY_ERR = 10052
An error has occurred whilst accessing the registry.
MG17_MEMORY_ERR = 10053
An internal memory allocation error or de-allocation error has occurred.
MG17_COM_ERR = 10054
An error has occurred with the COM system. Restart the program.
MG17_USB_ERR = 10055
An error has occurred with the USB communications.
MG17_NOTTHORLABSDEVICE_ERR = 10056
Not Thorlabs USB device error.

Rack and USB Units

MG17_SERIALNUMUNKNOWN_ERR = 10100
A serial number has been specified that is unknown to the server.
MG17_DUPLICATESERIALNUM_ERR = 10101
A duplicate serial number has been detected. Serial numbers are required to be unique.
MG17_DUPLICATEDEVICEIDENT_ERR = 10102
A duplicate device identifier has been detected.
MG17_INVALIDMSGSRC_ERR = 10103
An invalid message source has been detected.
MG17_UNKNOWNMSGIDENT_ERR = 10104
A message has been received with an unknown identifier.
MG17_UNKNOWNHWTYPE_ERR = 10105
An unknown hardware identifier has been encountered.
MG17_INVALIDSERIALNUM_ERR = 10106
An invalid serial number has been detected.
MG17_INVALIDMSGDEST_ERR = 10107
An invalid message destination ident has been detected.
MG17_INVALIDINDEX_ERR = 10108
An invalid index parameter has been passed.
MG17_CTRLCOMMSDISABLED_ERR = 10109
A software call has been made to a control which is not currently communicating with any hardware. This may be because the control has not been started or may be due to an incorrect serial number or missing hardware.
MG17_HWRESPONSE_ERR = 10110
A notification or response message has been received from a hardware unit. This may be indicate a hardware fault or that an illegal command/parameter has been sent to the hardware.
MG17_HWTIMEOUT_ERR = 10111
A time out has occurred while waiting for a hardware unit to respond. This may be due to communications problems or a hardware fault.
MG17_INCORRECTVERSION_ERR = 10112
Some functions are applicable only to later versions of embedded code.
This error is returned when a software call is made to a unit with an incompatible version of embedded code installed.
MG17_INCOMPATIBLEHARDWARE_ERR = 10115
Some functions are applicable only to later versions of hardware.
This error is returned when a software call is made to an incompatible version of hardware.
MG17_OLDVERSION_ERR = 10116
Older version of embedded code that can still be used

Motors

MG17_NOSTAGEAXISINFO = 10150
The GetStageAxisInfo method has been called when no stage has been assigned.
MG17_CALIBTABLE_ERR = 10151
An internal error has occurred when using an encoded stage.
MG17_ENCCALIB_ERR = 10152
An internal error has occurred when using an encoded stage.
MG17_ENCNOTPRESENT_ERR = 10153
A software call applicable only to encoded stages has been made to a non-encoded stage.
MG17_MOTORNOTHOMED_ERR = 10154
motor not homed error
MG17_MOTORDISABLED_ERR = 10155
motor disabled error
MG17_PMDMSG_ERR = 10156

PMD processor message error
MG17_PMDREADONLY_ERR = 10157
PMD based controller stage parameter 'read only' error

Piezos
MG17_PIEZOLED_ERR = 10200
Encoder not present error

NanoTraks
MG17_NANOTRAKLED_ERR = 10250
Encoder not present error
MG17_NANOTRAKCLOSEDLOOP_ERR = 10251
Closed loop error - closed loop selected with no feedback signal
MG17_NANOTRAKPOWER_ERR = 10252
Power supply error - voltage rails out of limits