MATLAB® Instrument Driver

for PicoScope® 3000A/B Series

Programmer's Guide

ps3000abpg.matlab.en r2


pico
Technology

Contents


pico
Technology

1 Introduction

This document outlines the functions defined in the MATLAB® Instrument Driver for the PicoScope® 3000 Series A/B oscilloscopes.

2 License agreement

Grant of license. The material contained in this release is licensed, not sold. Pico Technology Limited ('Pico') grants a license to the person who installs this software, subject to the conditions listed below.

Access. The licensee agrees to allow access to this software only to persons who have been informed of and agree to abide by these conditions.

Usage. The software in this release is for use only with Pico products or with data collected using Pico products.

Copyright. The software in this release is for use only with Pico products or with data collected using Pico products. You may copy and distribute the SDK without restriction, as long as you do not remove any Pico Technology copyright statements. The example programs in the SDK may be modified, copied and distributed for the purpose of developing programs to collect data using Pico products.

Liability. Pico and its agents shall not be liable for any loss or damage, howsoever caused, related to the use of Pico equipment or software, unless excluded by statute.

Fitness for purpose. No two applications are the same, so Pico cannot guarantee that its equipment or software is suitable for a given application. It is therefore the user's responsibility to ensure that the product is suitable for the user's application.

Mission-critical applications. Because the software runs on a computer that may be running other software products, and may be subject to interference from these other products, this license specifically excludes usage in 'mission-critical' applications, for example life-support systems.

Viruses. This software was continuously monitored for viruses during production. However, the user is responsible for virus checking the software once it is installed.

Support. No software is ever error-free, but if you are dissatisfied with the performance of this software, please contact our technical support staff.

Upgrades. We provide upgrades, free of charge, from our web site at www.picotech.com. We reserve the right to charge for updates or replacements sent out on physical media.

Trademarks. *Pico Technology Limited* and *PicoScope* are internationally registered trademarks. *Pico Technology* is registered at the U.S. Patents and Trademarks Office. *MATLAB* is a registered trademark of The Mathworks, Inc. *Instrument Control Toolbox* is a trademark of The Mathworks, Inc. *Windows* is a registered trademark of Microsoft Corporation.

3 Company details

You can obtain technical assistance from Pico Technology at the following address:

Address: Pico Technology
James House
Colmworth Business Park
ST NEOTS
Cambridgeshire
PE19 8YP
United Kingdom

Phone: +44 (0) 1480 396 395
Fax: +44 (0) 1480 396 296

Email:

Technical Support: support@picotech.com
Sales: sales@picotech.com

Web site: www.picotech.com

4 MATLAB® Driver Package

MATLAB is a numerical computing environment and high-level language developed by Mathworks that is used in industry and academia for a wide variety of applications.

The PS3000a_MATLAB_IC_Generic_Driver zip file includes the following:

- a MATLAB Generic Instrument Driver that is used with the Instrument Control Toolbox
- scripts that demonstrate how to call various functions in order to capture data in block and streaming mode, as well as using the signal generator

For MATLAB product information, please contact:

The MathWorks
3 Apple Hill Drive
Natick, MA, 01760-2098
USA

Tel: +1 508-647-7000
Fax: +1 508-647-7101

E-mail: info@mathworks.com

Web: www.mathworks.com

4.1 Prerequisites

The following hardware and software will be required:

Hardware

One of the following devices:

- PicoScope 3204A or 3204B
- PicoScope 3205A or 3205B
- PicoScope 3206A or 3206B
- PicoScope 3207A or 3207B (must be connected to USB 3.0 port)
- PicoScope 3404A or 3404B
- PicoScope 3405A or 3405B
- PicoScope 3406A or 3406B

Software

- MATLAB (R2011b* or higher) – 32-bit/64-bit for Microsoft Windows
- Instrument Control Toolbox (v3.1 or later)
- PicoScope PS3000A Software Development Kit (R10.5.0.32 or later; filename beginning PS3000asdk)
  Download here or visit www.picotech.com > Software
  Be careful not to confuse this with the PS3000 SDK for the original PicoScope 3000 Series oscilloscopes (filename beginning PS3000sdk)
- Microsoft Windows SDK 7.1 (for MATLAB 64-bit only)
  Please also refer to "How do I install Microsoft Windows SDK 7.1?" on MATLAB CENTRAL

Note: Initial development was carried out with MATLAB 2011b before moving onto MATLAB 2012a (7.14).

4.2 Install procedure

Extract the contents of the PS3000a_MATLAB_IC_Generic_Driver.zip file to a directory on the computer. The following dynamic link library (DLL) files from the SDK will also be required:

- PS3000a.dll (rename from ps3000a.dll to PS3000a.dll if necessary)
- ps3000aWrap.dll
- PicoIpp.dll

For 64-bit versions, please ensure that you obtain these DLL files from the x64 directories in the root and Wrapper/Release directories in the SDK. The files from the ps3000a/x64 directory in the Instrument Driver Package will also be required.

In the root directory will be two folders:

- Functions
- ps3000a

The location of the SDK and the PS3000A folder must be added to the MATLAB Path (use the addpath command if necessary).

The Functions folder contains two files:

- adc2mv.m – converts ADC counts to millivolts
- mv2adc.m – converts millivolt values to ADC counts

The ps3000a folder contains the generic instrument driver, prototype files, example arbitrary waveform files and a number of scripts:

Driver

- PS3000a_IC_drv.mdd

Prototype Files

- PS3000aMFile.m
- ps3000aWrapMFile.m

The prototype files are used in place of header files to define functions, enumerations and structures.

The prototype files to use with 64-bit MATLAB can be found in the x64 directory located in the ps3000a directory of the Instrument Driver package. Copy these files into the same directory as the Instrument Driver file or add them to the MATLAB path, ensuring that the prototype files generated using the 32-bit DLL are moved elsewhere.

Example Scripts

The files below provide examples of using the driver to capture data using different modes and triggering:

- PS3000a_IC_Generic_Driver_1buffer_RunBlock.m
- PS3000a_IC_Generic_Driver_1buffer_RunBlock_Adv_Trig.m
- PS3000a_IC_Generic_Driver_2buffers_RunBlock.m
- PS3000a_IC_Generic_Driver_1buffer_RapidBlock.m
- PS3000a_IC_Generic_Driver_2buffers_Streaming_SimpleTrig.m

- PS3000a_IC_Generic_Driver_SigGen.m

Example AWG Files

In the ps3000a directory are two example files that can be used to generate a signal using the Arbitrary Waveform Generator:

- ramp_8192.txt
- sine_wave_4096.csv (created using PicoScope 6)

Functions

The file below allows you to change the power source of a 4-channel PicoScope 3000A/B device while it is connected:

- ps3000aChangePowerSource.m

In addition to the two folders, there is a PicoStatus.m file containing definitions of the error codes returned by the driver.

4.3 Running the examples

The example scripts can be run either from the MATLAB command window or from the script Editor window.

There are two ways to run a script:

1. Type the script name in the MATLAB command window and press the **ENTER** key.

2. Open the script in the MATLAB Editor and click the **Run** button.

Note 1: The m files are structured in cells, allowing you to run a small set of functions at a time.

Note 2: If a script crashes before finishing, you must call the disconnect code outlined below to close the device before running another script.

5 MATLAB Generic Instrument Driver

The MATLAB Generic Instrument Driver for the PicoScope 3000A/B Series consists of a number of properties and functions which are listed below.

5.1 Properties

Beside the standard Instrument Control Device properties, the following additional properties are stored internally within the driver for each instance of an object:

- `awgBufferSize` – size of the AWG waveform buffer ('B' models only).
- `channelCount` – the number of channels on the device.
- `firstRange` – the lowest voltage range supported by the device (50 mV).
- `lastRange` – the highest voltage range supported by the device (20 V).
- `maxValue` – the maximum ADC count value.
- `minValue` – the minimum ADC count value.
- `ps3000aEnums` – not used.
- `ps3000aStructs` – not used.
- `sigGenType` – the type of signal generator - None, Function Generator or AWG.
- `timebase` – the timebase index used by the device in block mode (not used).
- `unithandle` – the handle value assigned to the device.
- `unitserial` – the batch/serial number of the device.

5.2 Functions

Unless otherwise stated, please refer to the main *PicoScope 3000 Series (A API) Programmer's Guide* for a full description of the function calls as the Generic Instrument Driver functions mostly mirror those in the core API.

5.2.1 Callback functions

As MATLAB does not support C-style callback functions (warnings about 'FcnPtr' may appear when the main `PS3000a.dll` is loaded), API functions from the wrapper DLL (`ps3000aWrap.dll`) are used.

5.2.2 'handle' parameter

Note: as the Instrument Control Toolbox makes use of device objects to represent an instance of a device, the `handle` parameter is not used when calling the 'invoke' function call as in the API defined in the main *PicoScope 3000 Series (A API) Programmer's Guide*. The `unithandle` parameter is stored internally within the driver when a connection is made to a unit.

Instead of the `handle` parameter, the argument `obj,` which represents the device object, is used.

5.2.3 Function return values

The driver status code values returned are decimal values – use the `dec2hex()` MATLAB function to convert the value to a hexadecimal value, the definitions of which may be found in the main *PicoScope 3000 Series (A API) Programmer's Guide*.

In addition to the status, the output from a function call in MATLAB may consist of a number of parameters – this is due to the fact that MATLAB will output variables corresponding to arguments which are pointers in the API function call.

5.2.4 Connecting to/disconnecting from a PicoScope 3000A/B Oscilloscope

Connecting to a device

Use the `icdevice` command to create an instance of an object representing the PicoScope 3000A/B device. This loads the required library files.

It is possible to specify the batch/serial number of the device to connect to when creating the device object:

```
ps3000a_object = icdevice('PS3000a_IC_drv', 'TEST/001');
```

Alternatively, to proceed without specifying a batch/serial number:

```
ps3000a_object = icdevice('PS3000a_IC_drv', '');
```

To connect to the device, use the `connect` command:

```
connect(ps3000a_obj);
```

This Instrument Driver does not currently support multiple devices.

Disconnecting from a device

To disconnect a device from the PC use the `disconnect` command:

```
disconnect(ps3000a_obj);
```

For further information on the process of creating as well as connecting and disconnecting to/from a device object, please refer to the MATLAB Help files.

5.2.5 Function descriptions

5.2.5.1 Core API functions

The following functions based on the core API are provided by the MATLAB Generic Instrument Driver for the PicoScope 3000A/B Series. The *PicoScope 3000 Series (A API) Programmer's Guide* indicates which devices the functions can be used with.

- `status = ps3000aChangePowerSource(obj, powerstate)`
- `status = ps3000aCurrentPowerSource(obj, handle)`
- `status = ps3000aFlashLed(obj, start)`
- `[status, maximumVoltage, minimumVoltage] = ps3000aGetAnalogueOffset(obj, range, coupling, maximumVoltage, minimumVoltage)`
- `[status, ranges, length] = ps3000aGetChannelInformation(obj, info, ranges, length, channels)`
- `[status, maxDownSampleRatio] = ps3000aGetMaxDownSampleRatio(obj, numUnaggregatedSamples, downSampleRatioMode, segmentIndex)`
- `[status, maxSegments] = ps3000aGetMaxSegments(obj)`
- `[status, nCaptures] = ps3000aGetNoOfCaptures(obj, nCaptures)`
- `[status, nCaptures] = ps3000aGetNoOfProcessedCaptures(obj, nCaptures)`
- `[status, timeIntervalNs1, maxSamples1] = ps3000aGetTimebase(obj, timebase, noSamples, timeIntervalNs, oversample, maxSamples, segmentIndex)`
- `[status, timeIntNs1, maxSamples1] = ps3000aGetTimebase2(obj, timebase, noSamples, timeIntNs, oversample, maxSamples, segmentIndex)`
- `[status, timeUpper, timeLower, timeUnits] = ps3000aGetTriggerTimeOffset(obj, timeUpper, timeLower, timeUnits, segmentIndex)`
- `[status, time, timeUnits] = ps3000aGetTriggerTimeOffset64(obj, time, timeUnits, segmentIndex)`
- `[status, numSamples, overflow] = ps3000aGetValues(obj, startIndex, noOfSamples, downSampleRatio, downSampleRatioMode, segmentIndex, overflow)`
- `[status, noOfSamples, overflow] = ps3000aGetValuesBlk(obj, noOfSamples, fromSegmentIndex, toSegmentIndex, downSampleRatio, downSampleRatioMode, overflow)`
- `[status, noOfSamples, overflow] = ps3000aGetValuesOverlapped(obj, startIndex, noOfSamples, downSampleRatio, downSampleRatioMode, segmentIndex, overflow)`
- `[status, noOfSamples, overflow] = ps3000aGetValuesOverlappedBulk(obj, startIndex, noOfSamples, downSampleRatio, downSampleRatioMode, fromSegmentIndex, toSegmentIndex, overflow)`
- `[status, timesUpper, timesLower, timeUnits] = ps3000aGetValuesTriggerTimeOffsetBulk(obj, timesUpper, timesLower, timeUnits, fromSegmentIndex, toSegmentIndex)`
- `[status, times, timeUnits] = ps3000aGetValuesTriggerTimeOffsetBulk64(obj, times, timeUnits, fromSegmentIndex, toSegmentIndex)`
- `[status, ready] = ps3000aIsReady(obj, handle)`
- `[status, triggerEnabled, pWQEnabled] = ps3000aIsTriggerOrPulseWidthQualifierEnabled(obj, triggerEnabled, pWQEnabled)`
- `status = ps3000aMaximumValue(obj, handle)`
- `[status, nMaxSamples] = ps3000aMemorySegments(obj, nSegments)`
- `status = ps3000aMinimumValue(obj, handle)`
- `[status, noOfValues] = ps3000aNoOfStreamingValues(obj, handle)`
- `[status, open_async_status, serial] = ps3000aOpenUnitAsync(obj, serial)`
- `[status, progressPercent, complete] = ps3000aOpenUnitProgress(obj)`
- `status = ps3000aPingUnit(obj, handle)`

- `[status, timeIndisposedMs] = ps3000aRunBlock(obj, noOfPreTriggerSamples, noOfPreTriggerSamples, timebase, oversample, segmentIndex)`
- `[status, sampleInterval] = ps3000aRunStreaming(obj, sampleInterval, sampleIntervalTimeUnits, maxPreTriggerSamples, maxPostTriggerSamples, autoStop, downSampleRatio, downSampleRatioMode, overviewBufferSize)`
- `status = ps3000aSetBandwidthFilter(obj, channel, bandwidth)`
- `status = ps3000aSetChannel(obj, channel, enabled, type, range, analogueOffset)`
- `status = ps3000aSetDataBuffer(obj, channel, buffer, bufferLth, segmentIndex, mode)`
- `status = ps3000aSetDataBuffers(obj, channel, pBufferMax, pBufferMin, bufferLth, segmentIndex, ratioMode)`
- `[status, sampleTimePicoSeconds] = ps3000aSetEts(obj, mode, etsCycles, etsInterleave`
- `status = ps3000aSetEtsTimeBuffer(obj, buffer, bufferLth)`
- `status = ps3000aSetEtsTimeBuffers(obj, timeUpper, timeLower, bufferLth)`
- `status = ps3000aSetNoOfCaptures(obj, nCaptures)`
- `status = ps3000aSetPulseWidthQualifier(obj, conditions, nConditions, direction, lower, upper, type)`
- `status = ps3000aSetSigGenArbitrary(obj, offsetVoltage, pkToPk, startDeltaPhase, stopDeltaPhase, deltaPhaseIncrement, dwellCount, arbitraryWaveform, arbitraryWaveformSize, sweepType, operation, indexMode, shots, sweeps, triggerType, triggerSource, extInThreshold)`
- `status = ps3000aSetSigGenBuiltIn(obj, offsetVoltage, pkToPk, waveType, startFrequency, stopFrequency, increment, dwellTime, sweepType, operation, shots, sweeps, triggerType, triggerSource, extInThreshold)`
- `status = ps3000aSetSimpleTrigger(obj, enable, source, threshold, direction, delay, autoTrigger_ms)`
- `status = ps3000aSigGenSoftwareControl(obj, state)`
- `status = ps3000aStop(obj)`

5.2.5.2 Wrapper functions

5.2.5.2.1 AutoStopped

```
autoStop = AutoStopped(obj)
```

This function indicates if the device has stopped on collection of the number of samples specified in the call to the `ps3000aRunStreaming` function (if the `ps3000aRunStreaming` function's autostop flag is set).

Arguments:

- `obj` – the device object instance.

Returns:

- Non-zero – if streaming has autostopped.

5.2.5.2.2 AvailableData

```
[numSamples, startIndex] = AvailableData(obj)
```

This function returns the number of samples returned from the driver and shows the start index of the data in the buffer when collecting data in streaming mode.

Arguments:

- obj, the device object instance.

Returns:

- numSamples
  - 0 – Data is not yet available.
  - Non-zero – The number of samples returned from the driver.
- startIndex – On exit, an index to the first valid sample in the buffer (when data is available).

5.2.5.2.3 ClearTriggerReady

```
triggerCleared = ClearTriggerReady(obj)
```

This function clears the triggered and triggeredAt flags in relation to streaming mode capture.

Arguments:

- obj - the device object instance.

Returns:

- 1

5.2.5.2.4 GetStreamingLatestValues

```
status = GetStreamingLatestValues(obj)
```

This function facilitates communication with the driver to return the next block of values to your application when capturing data in streaming mode.

Arguments:

- obj - the device object instance.

Returns:

See ps3000aGetStreamingLatestValues() return values in the *PicoScope 3000 Series (A API) Programmer's Guide*.

5.2.5.2.5 IsReady

```
ready = IsReady(obj)
```

This function is used to poll the driver to verify that data is ready to be received. The ps3000aRunStreaming function must have been called prior to calling this function.

Arguments:

- obj - the device object instance.

Returns:

- 0 – Data is not yet available.
- Non-zero – Data is ready to be collected.

5.2.5.2.6 IsTriggerReady

```
[triggered, triggeredAt] = IsTriggerReady(obj)
```

This function indicates whether a trigger has occurred when collecting data in streaming mode, and the location of the trigger point in the buffer.

Arguments:

- `obj` - the device object instance

Returns:

- `triggered`
  - 0: The device has not triggered
  - Non-zero: the device has been triggered.
- `triggeredAt` – on exit, the index of the sample in the buffer where the trigger occurred.

5.2.5.3 MATLAB Instrument Driver specific functions

5.2.5.3.1 setAdvancedTrigger

```
status = setAdvancedTrigger(obj, channelProperties,
nChannelProperties, triggerConditions, nTriggerConditions, directions,
delay, auxOutputEnabled, autoTriggerMs)
```

This function sets up the advanced trigger functionality on the oscilloscope. For further information please refer to the main Programmer's guide for the following underlying functions:

- `ps3000aSetTriggerChannelConditions`
- `ps3000aSetTriggerChannelDirections`
- `ps3000aSetTriggerChannelProperties`
- `ps3000aSetTriggerDelay`

Arguments:

- `obj` – the device object instance.
- `channelProperties` – a structure array containing one or more structures specifying the conditions to be applied to each channel.
- `nChannelProperties` – the number of elements in the `channelProperties` structure.
- `triggerConditions` – a structure array containing one or more structures specifying the conditions to be applied to each channel.
- `nTriggerConditions` - the number of elements in the `triggerConditions` structure.
- `directions` – a structure containing enumerations specifying the direction in which the signal must pass through the threshold to activate the trigger for each channel.
- `delay` - the time between the trigger occurring and the first sample (in samples).
- `auxOutputEnabled` – not used.

- autoTriggerMs - the time in milliseconds for which the scope device will wait before collecting data if no trigger event occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.

Returns:

- status – a structure containing the status values returned by the calls to the underlying API functions.

5.2.5.3.2 setDefaults

```
status = setDefaults(obj, channelsettings)
```

This function sets the default settings for the oscilloscope's channels.

Arguments:

- obj – the device object instance.
- channelsettings – a structure array containing structures corresponding to each channel with the parameters enabled, DCCoupled and range.

Returns:

- status – a structure containing status codes relating to the calls to set the ETS mode and channels of the oscilloscope.

5.2.5.3.3 GetUnitInfo

```
status = GetUnitInfo(obj)
```

This function returns information on the specified device in the command line window.

NOTE: This function differs from that defined in the main *PicoScope 3000 Series (A API) Programmer's Guide* in that it displays all the unit information by making a series of calls to the corresponding API function using the info values listed in the main guide.

Arguments:

- obj – the device object instance.

Returns:

- status – an array containing the status codes returned from each call to the underlying ps3000aGetUnitInfo function.

5.3 Enumerations and structures

The prototype files supplied provide the user with the ability to access the enumerations and structures defined in the ps3000Api.h header file provided in the Software Development Kit.

The example scripts show that the prototype file is loaded and assigned to a number of variables:

```
[methodinfo, structs, enuminfo, ThunkLibName] = PS3000aMFile;
```

Enumerations

Enumerations are accessed from the enuminfo structure e.g.:

```
channelSettings(1).range = enuminfo.enPS3000ARange.PS3000A_1V;
```

Structures

Structures are accessed from the structs structure:

```
trig_ch_a_properties =
  structs.tPS3000ATriggerChannelProperties.members;
```

5.4 Instrument Control Test and Measurement Tool

At the time of writing, the Test and Measurement Tool is not fully compatible with the MATLAB Generic Instrument Driver for the PicoScope 3000 A/B Series.

6 Trademarks

MATLAB is a registered trademark of The Mathworks, Inc.

Pico Technology and *PicoScope* are internationally registered trademarks of Pico Technology. *Pico Technology* is registered at the U.S. Patents and Trademarks Office.


pico
Technology

Index


pico
Technology

Pico Technology

James House
Colmworth Business Park
ST. NEOTS
Cambridgeshire
PE19 8YP
United Kingdom
Tel: +44 (0) 1480 396 395
Fax: +44 (0) 1480 396 296
www.picotech.com

ps3000abpg.matlab.en r2 2014-05-20