PicoSource® AS108

8 GHz Agile Synthesizer

Programmer's Guide


pico
Technology

Contents


pico
Technology

1 Introduction

The **PicoSource AS108** from Pico Technology is an 8 GHz agile signal synthesizer. Output frequency, phase and level can be set to constant values, linearly swept values or arbitrary sequences of values.

This manual explains how to develop your own programs for setting up the PicoSource AS108 using the `picosynth.dll` dynamic link library. This provides support for the control of the signal generator from C-compatible programming languages and applications.

1.1 PC requirements

To ensure that your **PicoSource AS108** operates correctly with the driver, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown below:

Item	Specification
Operating system	All supported versions of Windows, 32-bit or 64-bit
Processor, memory, free disk space	As required by Windows
Ports	USB 2.0 or USB 3.0

1.2 Legal information

The material contained in this release is licensed, not sold. Pico Technology Ltd 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 these conditions and agree to abide by them.

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

Copyright. Pico Technology Ltd claims the copyright of, and retains the rights to, all material contained in this software. You may copy and distribute the software without restriction, as long as you do not remove any Pico Technology copyright statements.

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

Fitness for purpose. As no two applications are the same, Pico Technology cannot guarantee that its equipment or software is suitable for a given application. It is your responsibility, therefore, to ensure that the product is suitable for your application.

Mission-critical applications. This software is intended for use on a computer that may be running other software products. For this reason, one of the conditions of the license is that it excludes use in mission-critical applications, for example life support systems.

Viruses. This software was continuously monitored for viruses during production, but you are responsible for virus-checking the software once it is installed.

Support. If you are dissatisfied with the performance of this software, please contact our technical support staff, who will try to fix the problem within a reasonable time. If you are still dissatisfied, please return the product and software to your supplier within 14 days of purchase for a full refund.

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. *PicoSynth* is a trademark, and *Pico Technology, PicoSource* and *PicoSDK* are registered trademarks, of Pico Technology Ltd. *Windows* is a trademark or registered trademark of Microsoft Corporation.

1.3 Downloading and installing

To install the **PicoSynth 2** software, including the software development kit:

1. Go to **www.picotech.com** > **Downloads**
2. Select the **PicoSource** product range
3. Select the **PicoSource AS108** product
4. Download the **PicoSynth** installer
5. Run the **PicoSynth** installer

This will install the **PicoSynth** application as well as the drivers, library files and header files that you will need to develop your own applications.

The files will be installed in your `C:\Program Files\PicoSynth 2` folder, or `Program Files (x86)` for the 32-bit version.

1.4 Drivers

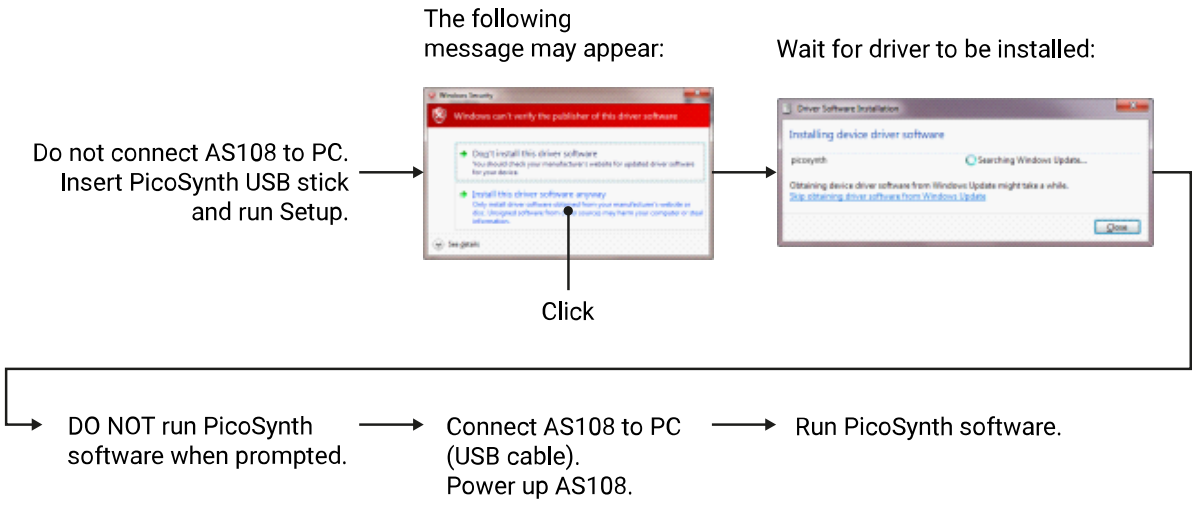
`picosynth.dll`: Your application will communicate with this library, which is supplied in 32-bit and 64-bit versions. The DLL exports the picosynth function definitions in `stdcall` format, which is compatible with a wide range of programming languages.

`ftdi.dll` or `ftd2xx64.dll`: `picosynth.dll` depends on this DLL, which is supplied in 32-bit and 64-bit versions. Always use versions of `picosynth.dll` and `ftdi.dll/ftd2xx64.dll` with matching word sizes: either both 32-bit or both 64-bit. The DLL must be on your dynamic linker path.

A static import library is also provided to simplify development. The import library may be statically linked and contains code to load our dynamic libraries.

1.5 Windows 7 setup

The following extra steps are required when installing on Windows 7.



1.6 Further information

For further information about your PicoSource AS108, see the following documents. All are available from www.picotech.com.

- *PicoSource AS108 Data Sheet*
- *PicoSource AS108 User's Guide*
- *PicoSource AS108 web pages*

2 Programming overview

2.1 Connecting to the device

Before opening a PicoSource AS108, you can optionally call:

- picosynthEnumerateUnits()

to obtain a list of all connected devices. If your application is expecting to find multiple devices connected, this is a way of obtaining their serial numbers and other data before choosing which device(s) to open.

In all cases, before using each AS108 device, you must open it by calling:

- picosynthOpenUnit()

This function returns a device ID called handle, which you then pass to all other functions in this API to specify which device you want to communicate with.

To confirm that a device is still connected and responding, you can call:

- picosynthPingUnit()

at any time while the device is open. You can also call:

- picosynthGetUnitInfo()

to obtain more detailed information about the device.

After device setup is complete, to release the handle and allow other applications to access the device, call:

- picosynthCloseUnit()

2.2 Operating modes

The AS108 can operate in the following modes:

Continuous wave (CW)
Generate a sine wave with programmable frequency, phase and level. Use either or both of the following functions:

- picosynthSetFrequency()
- picosynthSetPhase()

Amplitude modulation (AM)
Generate a sine wave with fixed frequency and level, modulated by either the external **FM/AM** input or the internal oscillator. Use the following function:

- picosynthSetAmplitudeModulation()

Frequency modulation (FM)
Generate a sine wave with fixed level, frequency-modulated by either the external **FM/AM** input or the internal oscillator. Use the following function:

- picosynthSetFrequencyModulation()

Phase modulation (PM)
Generate a sine wave with fixed frequency and level, phase-modulated by either the external **FM/AM** input or the internal oscillator. Use the following function:

- picosynthSetPhaseModulation()

Sweep
Generate a sine wave with a linear sweep of frequency, phase or level. Frequency and level can be swept at the same time, as can phase and level. Use one of the following functions:

- picosynthSetFrequencyAndLevelSweep()
- picosynthSetPhaseAndLevelSweep()

Arbitrary sequence
Generate a sine wave with an arbitrary (program-specified) sequence of (frequency, level) or (phase, level) states. These modes can be used to simulate modulation schemes such as ASK, FSK, QAM and PAM. Use one of the following functions:

- picosynthSetArbitraryFrequencyAndLevel()
- picosynthSetArbitraryPhaseAndLevel()

Output switched off
The **RF Output** continually generates a sine wave unless explicitly switched off. Use the following function:

- picosynthSetOutputOff()

2.3 Using a single device

Here is a typical sequence of operations for setting up the PicoSource AS108:

```
PICO_STATUS          status;
PICO_SOURCE_MODEL    model = PICO_SYNTH;
uint32_t             handle;
double               frequencyHz = 1e6;      // 1 MHz
double               powerLeveldBm = 10;     // 10 dBm

// Open the first available device:

status = picosynthOpenUnit(model, &handle, NULL);

// If all is well, status will be PICO_OK, handle will contain
// device ID.

// Set output to fixed frequency and level:

status = picosynthSetFrequency(handle, frequencyHz, powerLeveldBm);

// Now that we have finished talking to the unit, close it:

status = picosynthCloseUnit(handle);

// Note: Closing the unit does not switch off its output. To do that,
// call picosynthSetOutputOff() before closing.
```

2.4 Using multiple devices

The previous example can easily be modified to use multiple devices:

```
PICO_STATUS          status;
PICO_SOURCE_MODEL    model = PICO_SYNTH;
uint32_t             handle1, handle2;
uint8_t              serialNumber1[8] = '9000\0',
                     serialNumber2[8] = '9001\0';
double               frequencyHz = 1e6;      // 1 MHz
double               powerLeveldBm = 10;     // 10 dBm

// Open devices with serial numbers '9000' and '9001':

status = picosynthOpenUnit(model, &handle1, serialNumber1);
status = picosynthOpenUnit(model, &handle2, serialNumber2);

// If all is well, status will be PICO_OK, and handle1 and handle2
// will contain device IDs. Error-checking code is not shown.

// Set outputs to fixed frequency and level:

status = picosynthSetFrequency(handle1, frequencyHz, powerLeveldBm);
status = picosynthSetFrequency(handle2, frequencyHz, powerLeveldBm);

// Now that we have finished talking to the units, close them:

status = picosynthCloseUnit(handle1);
status = picosynthCloseUnit(handle2);

// Note: Closing a unit does not switch off its output. To do that,
// call picosynthSetOutputOff() before closing.
```

2.5 Triggering

The PicoSource AS108 can generate a free-running output or wait for an external trigger on the **Trig In** connector on the rear panel before generating an arbitrary sequence or a sweep.

The following functions support triggering:

- `picosynthSetArbitraryFrequencyAndLevel()`
- `picosynthSetArbitraryPhaseAndLevel()`
- `picosynthSetFrequencyAndLevelSweep()`
- `picosynthSetPhaseAndLevelSweep()`

Each of the above functions accepts a `triggerMode` argument that specifies the type of triggering. The following values can be used:

- `InternalTrigger`: beginning of sequence or sweep is internally controlled (free-running).
- `ExternalRisingEdgeStart`: start sequence or sweep on a rising edge of **Trig In**.
- `ExternalFallingEdgeStart`: start sequence or sweep on a falling edge of **Trig In**.
- `ExternalRisingEdgeStep`: advance to next sequence step or sweep step on a rising edge of **Trig In**.
- `ExternalFallingEdgeStep`: advance to next sequence step or sweep step on a falling edge of **Trig In**.

The AS108 always waits for the specified dwell time before responding to the next edge on **Trig In**. Any edges on **Trig In** that occur before the specified dwell time will be ignored.

Trigger output
The **Trig Out** connector on the rear panel automatically generates a signal for synchronizing external equipment with the internal sweep timing. The output is high (nominally 5 V) between sweeps, falling to 0 V (nominal) at the start of each sweep or arbitrary sequence for the duration of one sweep step or sequence step. When no sweep or sequence is active, the output remains low.

3 Function calls

3.1 picosynthCloseUnit()

```
PICO_STATUS picosynthCloseUnit
(
  uint32_t     handle
)
```

Purpose
Close the specified device. This allows other applications (including PicoSynth 2) to open it.

Closing the device does not switch off its output – the device will continue to generate the programmed output, whether fixed, swept or in an arbitrary sequence, until powered off. Furthermore, the device will resume generating the same output when powered on again.

Arguments
`handle`: the device identifier. After the function returns, `handle` is no longer valid.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.2 picosynthEnumerateUnits()

```
PICO_STATUS picosynthEnumerateUnits
(
  PICO_SOURCE_MODEL  model,
  uint8_t            * serials,
  uint16_t           * serialLth
)
```

Purpose
Return a list of all connected PicoSource devices of the specified type.

Arguments
`model`: the type of device to search for. Only `PICO_SYNTH` is currently supported.

`serials`: on entry, a buffer of length `serialLth` characters; on successful exit, an ASCII text string containing a comma-separated list of the serial numbers of all devices found:

```
<deviceString1>,<deviceString2>,<deviceString3>,...
```

`deviceString1, deviceString2` and so on are in the following format:

```
<serialNum>[<status>;<desc>;<usbType>;<hardwareID>;<softwareVer>]
```

where:

`serialNum` is an integer formatted as a string

`status` is "`OK`" if the device can be opened, or "`IN USE`" if it is being used by another application

`desc` is a human-readable string describing the device

`usbType` is the USB version required by the device

`hardwareID` is for Pico use only

`softwareVer` is for Pico use only

`serialLth`: on entry, the length of the `serials` buffer;
on successful exit, the length of the `serials` text string.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.3 picosynthGetUnitInfo()

```
PICO_STATUS picosynthGetUnitInfo
(
  uint32_t     handle,
  int8_t       * string,
  uint16_t     stringLength,
  uint16_t     * requiredSize,
  PICO_INFO    deviceInfo
)
```

Purpose
Read data about the specified device.

Arguments
`handle`: the device identifier.

`string`: on entry, a buffer of `stringLength` characters; on exit, an ASCII string containing the requested data.

`stringLength`: the length of the `string` buffer.

`requiredSize`: the number of characters that would be required to store the requested data in full. If this is greater than `stringLength`, the string in the `string` buffer will have been truncated.

`deviceInfo`: the type of information that you wish to read from the device. See PICO_INFO.

Returns
`PICO_OK (0)` if call was successful. Other PICO_STATUS values indicate errors or warnings.

3.3.1 PICO_INFO

The definitive list of `PICO_INFO` values is in the `PicoStatus.h` file, which is included in your PicoSynth SDK. The following values apply to PicoSource AS108 devices:

info			Example
0	PICO_DRIVER_VERSION	Version number of `picosynth.dll`	2.0.1.782
1	PICO_USB_VERSION	Type of USB connection to device: 1.1, 2.0 or 3.0	1.1
2	PICO_HARDWARE_VERSION	Hardware version of device	A31IIBHT
4	PICO_BATCH_AND_SERIAL	Batch and serial number of device	7698
9	PICO_FIRMWARE_VERSION_1	Primary firmware (FPGA code) version	1.5
14	PICO_DRIVER_PATH	Location of the driver in your file system	C:\picosynth.dll

3.4 picosynthOpenUnit()

```
PICO_STATUS picosynthOpenUnit
(
  PICO_SOURCE_MODEL  model,
  uint32_t           * handle,
  uint8_t            * serialNumber
)
```

Purpose
Open a PicoSource AS108 device. If more than matching device is connected, find each one in turn until all available units have been opened.

Arguments
`model`: the type of device to search for. Only `PICO_SYNTH` is currently supported.

`handle`: on successful exit, the device identifier is written to this location.

`serialNumber`: on entry, either `NULL` or a pointer to a null-terminated string. If `NULL`, opens the first available device. If a valid pointer, opens the device with the specified serial number.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.5 picosynthPingUnit()

```
PICO_STATUS picosynthPingUnit
(
  uint32_t     handle
)
```

Purpose
Report whether the specified device is present.

Arguments

handle:	the device identifier.

Returns

PICO_OK:	the device responded to the request.
PICO_NOT_RESPONDING:	the device did not respond.

3.6 picosynthSetAmplitudeModulation()

```
PICO_STATUS picosynthSetAmplitudeModulation
(
  uint32_t           handle,
  double             frequencyHz,
  double             powerLeveldBm,
  double             modulationDepthPercent,
  double             modulationRateHz,
  MODULATION_SOURCE  modulationSource,
  int16_t            enabled
)
```

Purpose
Generate a fixed-frequency carrier with amplitude modulation (AM).

Arguments
`handle`: device identifier.

`frequencyHz`: frequency of the carrier output in hertz.

`powerLeveldBm`: level of the carrier output in decibel-milliwatts.

`modulationDepthPercent`: depth of amplitude modulation from 0 (no modulation) to 100 (maximum modulation).

`modulationRateHz`: frequency of the internal modulating signal in hertz; used only if `modulationSource` is set to `Internal`.

`modulationSource`: source of the modulating signal:

- `Internal`: the built-in sine wave generator.
- `External`: the external **FM/AM** modulation input.

`enabled`: switch amplitude modulation on or off.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.7 picosynthSetArbitraryFrequencyAndLevel()

```
PICO_STATUS picosynthSetArbitraryFrequencyAndLevel
(
  uint32_t     handle,
  double       * arbitraryFrequencyHz,
  double       * arbitraryPowerLeveldBm,
  int32_t      numberOfPoints,
  double       dwellTimeUs,
  TRIGGER_MODE       triggerMode
)
```

Purpose
Generate an arbitrary sequence of (frequency,level) pairs. This can be used for simulating FSK and ASK modulation schemes.

Procedure
1. Create two lists – `* arbitraryFrequencyHz`, a list of frequencies, and `* arbitraryPowerLeveldBm`, a list of levels – each with `numberOfPoints` points.
2. Set `dwellTimeUs` to the time interval you require between successive points in the sweep.
3. Set `triggerMode` to the desired mode. You can make the synthesizer repeat the list with minimal delay between repeats, or wait for an external input before generating the whole list, or wait for an external input before advancing to the next step in the list.
4. Call `picosynthSetArbitraryFrequencyAndLevel()` with the above values.

Arguments
`handle`: device identifier.

`arbitraryFrequencyHz`: pointer to a list of frequencies, in hertz.

`arbitraryPowerLeveldBm`: pointer to a list of levels, in decibel-milliwatts.

`numberOfPoints`: number of phase and level values in the `arbitraryPhaseDeg` and `arbitraryPowerLeveldBm` arrays. Range: 1 to 9001.

`dwellTimeUs`: time interval between steps in the sequence, in microseconds.

`triggerMode`: how the sequence will be activated. See Triggering for possible trigger modes.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.8 picosynthSetArbitraryPhaseAndLevel()

```
PICO_STATUS picosynthSetArbitraryPhaseAndLevel
(
  uint32_t     handle,
  double       frequencyHz,
  double       * arbitraryPhaseDeg,
  double       * arbitraryPowerLeveldBm,
  int32_t      numberOfPoints,
  double       dwellTimeUs,
  TRIGGER_MODE       triggerMode
)
```

Purpose
Generate an arbitrary sequence of (phase,level) pairs. This can be used to simulate modulation schemes such as QPSK and QAM.

Procedure
1. Create two lists – `* arbitraryPhaseDeg`, a list of phases, and `* arbitraryPowerLeveldBm`, a list of levels – each with `numberOfPoints` points.
2. Set `frequencyHz` to the desired carrier frequency.
3. Set `dwellTimeUs` to the time interval you require between successive points in the sweep.
4. Set `triggerMode` to the desired mode. You can make the synthesizer repeat the list with minimal delay between repeats, or wait for an external input before generating the whole list, or wait for an external input before advancing to the next step in the list.
5. Call `picosynthSetArbitraryPhaseAndLevel()` with the above values.

Arguments
`handle`: device identifier.

`frequencyHz`: carrier frequency in hertz.

`arbitraryPhaseDeg`: pointer to a list of phase values, in degrees (0.0 to 360.0).

`arbitraryPowerLeveldBm`: pointer to a list of levels, in decibel-milliwatts.

`numberOfPoints`: number of phase and level values in the `arbitraryPhaseDeg` and `arbitraryPowerLeveldBm` arrays. Range: 1 to 9001.

`dwellTimeUs`: time interval between steps in the sequence, in microseconds.

`triggerMode`: how the sequence will be activated. See Triggering for possible trigger modes.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.9 picosynthSetFrequency()

```
PICO_STATUS picosynthSetFrequency
(
  uint32_t     handle,
  double       frequencyHz,
  double       powerLeveldBm
)
```

Purpose
Set the output frequency and level to fixed values.

Arguments
`handle`: device identifier.

`frequencyHz`: frequency of the output, in hertz.

`powerLeveldBm`: power level of the output, in decibel-milliwatts.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.10 picosynthSetFrequencyAndLevelSweep()

```
PICO_STATUS picosynthSetFrequencyAndLevelSweep
(
  uint32_t          handle,
  double            startFrequencyHz,
  double            stopFrequencyHz,
  double            startLevel,
  double            stopLevel,
  LEVEL_UNIT        levelUnit,
  double            dwellTimeUs,
  int32_t           pointsInSweep,
  SWEEP_HOP_MODE    mode,
  TRIGGER_MODE      triggerMode
)
```

Purpose
Sweep the frequency and level linearly between specified limits.

Arguments
`handle`: device identifier.

`startFrequencyHz`: initial frequency of the frequency sweep, in hertz.

`stopFrequencyHz`: final frequency of the frequency sweep, in hertz.

`startLevel`: initial level of the level sweep, in units of `levelUnit`.

`stopLevel`: final level of the level sweep, in units of `levelUnit`.

`levelUnit`: units in which `startLevel` and `stopLevel` are expressed – `VoltsRms`, `VoltsPkToPk`, `Dbm` or `MilliWatts`.

`dwellTimeUs`: time between changes in frequency and level, in microseconds.

`pointsInSweep`: number of steps from beginning to end of sweep.

`mode`: the type of sweep to generate:

- `SweepAndFlyback`: sweep from start values to stop values, then return to start values and repeat.
- `BidirectionalSweep`: sweep from start values to stop values, then return in the opposite direction, then repeat.
- `Hop`: jump repeatedly between start values and stop values with no intermediate steps.

`triggerMode`: how the sweep will be activated. See Triggering for possible trigger modes.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.11 picosynthSetFrequencyModulation()

```
PICO_STATUS picosynthSetFrequencyModulation
(
  uint32_t           handle,
  double             frequencyHz,
  double             powerLeveldBm,
  double             modulationDeviationHz,
  double             modulationRateHz,
  MODULATION_SOURCE  modulationSource,
  int16_t            enabled
)
```

Purpose
Generate a fixed-amplitude carrier with frequency modulation (FM).

Arguments
`handle`: device identifier.

`frequencyHz`: frequency of the output carrier in hertz.

`powerLeveldBm`: level of the output in decibel-milliwatts.

`modulationDeviationHz`: maximum deviation from the carrier frequency, in hertz.

`modulationRateHz`: frequency of the internal modulating signal in hertz; used only if `modulationSource` is set to `Internal`.

`modulationSource`: source of the modulating signal:

- `Internal`: the built-in sine wave generator.
- `External`: the external **FM/AM** modulation input.

`enabled`: switch frequency modulation on or off.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.12 picosynthSetOutputOff()

```
PICO_STATUS picosynthSetOutputOff
(
  uint32_t     handle
)
```

Purpose
Switch off the RF output of the specified device.

To switch the RF output back on, call any other function that generates an output.

Arguments
`handle`: the device identifier .

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.13 picosynthSetPhase()

```
PICO_STATUS picosynthSetPhase
(
  uint32_t     handle,
  double       phaseDeg
)
```

Purpose
Set the output phase to a fixed value. When the synthesizer starts generating a particular frequency, the phase is defined as zero.

Arguments
`handle`: device identifier.

`phaseDeg`: phase of the output, in degrees (0.0 to 360.0).

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.14 picosynthSetPhaseAndLevelSweep()

```
PICO_STATUS picosynthSetPhaseAndLevelSweep
(
  uint32_t          handle,
  double            frequencyHz,
  double            startPhaseDeg,
  double            stopPhaseDeg,
  double            startLevel,
  double            stopLevel,
  LEVEL_UNIT        levelUnit,
  double            dwellTimeUs,
  int32_t           pointsInSweep,
  SWEEP_HOP_MODE    mode,
  TRIGGER_MODE      triggerMode
)
```

Purpose
Sweep the phase and level linearly between specified limits.

Arguments
`handle`: device identifier.

`frequencyHz`: carrier frequency in hertz.

`startPhaseDeg`: initial phase of the frequency sweep, in degrees (0.0 to 360.0).

`stopPhaseDeg`: final phase of the frequency sweep, in degrees (0.0 to 360.0).

`startLevel`: initial level of the level sweep, in units of `levelUnit`.

`stopLevel`: final level of the level sweep, in units of `levelUnit`.

`levelUnit`: units in which `startLevel` and `stopLevel` are expressed – `VoltsRms`, `VoltsPkToPk`, `Dbm`, `MilliWatts`.

`dwellTimeUs`: time between changes in frequency and level, in microseconds.

`pointsInSweep`: number of steps from beginning to end of sweep.

`mode`: the type of sweep to generate:

- `SweepAndFlyback`: sweep from start values to stop values, then return to start values and repeat.
- `BidirectionalSweep`: sweep from start values to stop values, then return in the opposite direction, then repeat.
- `Hop`: jump repeatedly between start values and stop values with no intermediate steps.

`triggerMode`: how the sweep will be activated. See Triggering for possible trigger modes.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

3.15 picosynthSetPhaseModulation()

```
PICO_STATUS picosynthSetPhaseModulation
(
  uint32_t           handle,
  double             frequencyHz,
  double             powerLeveldBm,
  double             modulationDeviationDeg,
  double             modulationRateHz,
  MODULATION_SOURCE  modulationSource,
  int16_t            enabled
)
```

Purpose
Generate a fixed-amplitude carrier with phase modulation (PM).

Arguments
`handle`: device identifier.

`frequencyHz`: frequency of the output carrier in hertz.

`powerLeveldBm`: level of the output in decibel-milliwatts.

`modulationDeviationDeg`: maximum phase change, in degrees.

`modulationRateHz`: frequency of the internal modulating signal in hertz; used only if `modulationSource` is set to `Internal`.

`modulationSource`: source of the modulating signal:

- `Internal`: the built-in sine wave generator.
- `External`: the external modulation input.

`enabled`: switch phase modulation on or off.

Returns
`PICO_OK (0)` if call was successful. Other `PICO_STATUS` values indicate errors or warnings.

4 Reference

4.1 PICO_STATUS return values

Every function in this API returns a `PICO_STATUS` value. The default value is `PICO_OK (0)`, if the call was successful. This and other values are defined in the `PicoStatus.h` file included with your software.

4.2 Parameter limits

The following limits apply to all functions in this API.

Parameter	Unit	Min Value	Max Value
Frequency	kHz	300	8 192 000
Frequency step	kHz	0.000 1 (300 kHz to 125 MHz) 0.01 (> 125 MHz to 4 GHz) 0.02 (> 4 GHz)	8 192 000
Power level (into 50 Ω)	dBm	- 15	+15
	V RMS	0.039 8	1.26
	V pk-pk	0.112	3.56
	mW	0.031 6	31.6
Phase	deg	0	360
Number of points in sweep	1	2	10 001
Dwell time	µs	26	65 500
Modulation frequency	Hz	10	5 000
AM depth, 0 dBm carrier	%	5	90
AM depth, > 0 to 9 dBm carrier	%	5	50
FM deviation	%	0	2
	kHz	0	200

4.3 Numeric data types

Type	Bits	Signed or unsigned?
`int8_t`	8	signed
`int16_t`	16	signed
`uint16_t`	16	unsigned
`enum`	32	enumerated
`int32_t`	32	signed
`uint32_t`	32	unsigned
`float`	32	signed (IEEE 754 binary32)
`double`	64	signed (IEEE 754 binary64)
`int64_t`	64	signed
`uint64_t`	64	unsigned

4.4 Unit conversions

Some `picosynth` functions support multiple units of measurement. If you need to do your own conversions, use the following formulae:

$$P_{dBm} = 10 \log P_{mW} = 10 \log \left(1000 \times \frac{{E_{RMS}}^2}{50} \right) = 10 \log \left(1000 \times \frac{\left(E_{PP}/2\sqrt{2}\right)^2}{50} \right)$$

$$P_{mW} = 10^{P_{dBm}/10} = 1000 \times \frac{{E_{RMS}}^2}{50} = 1000 \times \frac{\left(E_{PP}/2\sqrt{2}\right)^2}{50}$$

$$E_{RMS} = \frac{E_{PP}}{2\sqrt{2}} = \sqrt{\frac{10^{P_{dBm}/10} \times 50}{1000}} = \sqrt{\frac{P_{mW} \times 50}{1000}}$$

$$E_{PP} = 2\sqrt{2}\, E_{RMS} = 2\sqrt{2} \times \sqrt{\frac{10^{P_{dBm}/10} \times 50}{1000}} = 2\sqrt{2} \times \sqrt{\frac{P_{mW} \times 50}{1000}}$$

where:

P_{dBm} = power in decibel-milliwatts

P_{mW} = power in milliwatts

E_{RMS} = RMS voltage

E_{PP} = peak-to-peak voltage


pico
Technology

UK headquarters:

Pico Technology
James House
Colmworth Business Park
St. Neots
Cambridgeshire
PE19 8YP
United Kingdom

Tel: +44 (0) 1480 396 395

sales@picotech.com
support@picotech.com

USA regional office:

Pico Technology
320 N Glenwood Blvd
Tyler
TX 75702
United States

Tel: +1 800 591 2796

sales@picotech.com
support@picotech.com

Asia-Pacific regional office:

Pico Technology
Room 2252, 22/F, Centro
568 Hengfeng Road
Zhabei District
Shanghai 200070
PR China

Tel: +86 21 2226-5152

pico.asia-pacific@picotech.com

Germany regional office and EU Authorized Representative:

Pico Technology GmbH
Im Rehwinkel 6
30827 Garbsen
Germany

Tel: +49 (0) 5131 907 62 90

info.de@picotech.com

www.picotech.com

as108pg-2