TDTRP User Guide

Use MATLAB to directly talk to TDT hardware

Updated 2026-10-07

© 2016-2026 Tucker-Davis Technologies, Inc. (TDT). All rights reserved.

Tucker-Davis Technologies
11930 Research Circle
Alachua, FL 32615 USA
Phone: +1.386.462.9622
Fax: +1.386.462.5365

Notices

The information contained in this document is provided "as is," and is subject to being changed, without notice. TDT shall not be liable for errors or damages in connection with the furnishing, use, or performance of this document or of any information contained herein.

The latest versions of TDT documents are always online at https://www.tdt.com/docs/

Table of contents

TDTRP User Guide 4

- Requirements 4
- About TDTRP 4
- TDTRP Functionality 4
- Examples Using TDTRP 10

TDTPA5 User Guide 12

- Requirements 12
- About TDTPA5 12
- TDTPA5 Functionality 12
- Examples Using TDTPA5 17

TDTZBUS User Guide 18

- Requirements 18
- About TDTZBUS 18
- TDTZBUS Functionality 18
- Examples Using TDTZBUS 23

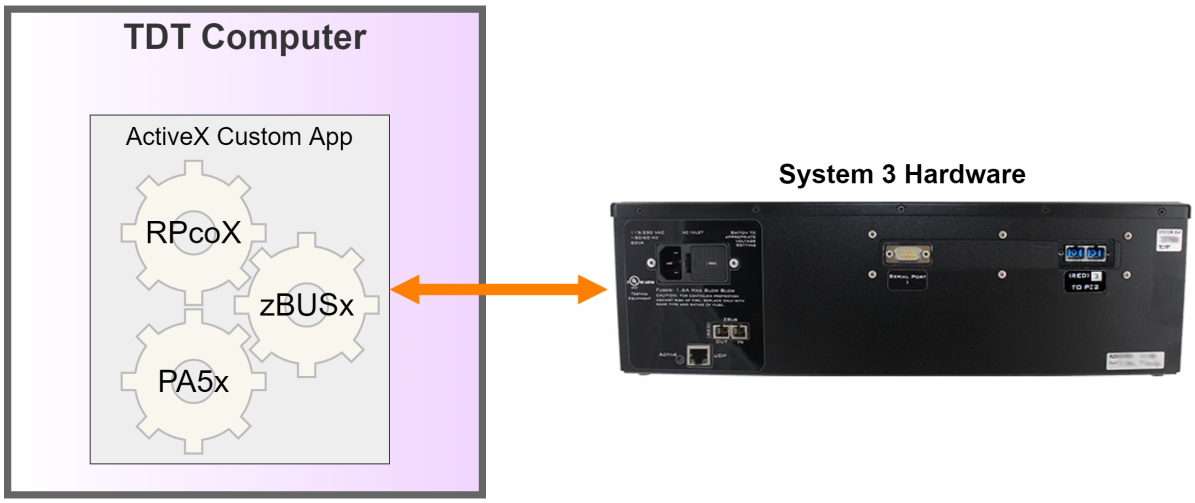
TDTRP User Guide

Use TDTRP to directly communicate with TDT hardware from MATLAB or Python.

Requirements

- TDT Drivers
- TDT ActiveX package
- Python only: `pywin32`

About TDTRP



TDTRP is a wrapper for the RPcoX ActiveX control that installs with the TDT ActiveX package. It provides direct access to TDT hardware (connect, load/run circuit, read/write tags on the hardware, send software triggers) and is generally easier to use than calling the RPcoX object directly.

TDTRP installs with the ActiveX controls into:

```
C:\TDT\ActiveX\ActXExamples\Matlab
C:\TDT\ActiveX\ActXExamples\py
```

or you can download it in the ActiveX Controls example zip.

TDTRP Functionality

This section includes methods, examples, and comparisons to RPcoX.

> **Note**
>
> See ActiveX User Reference for the complete RPcoX programming guide.

Creating a TDTRP object

```
obj = TDTRP(CIRCUITPATH, DEVICETYPE, 'parameter', value, ...)
```

`obj` is what you want the structure to be called for your code. In this document obj will be 'TDT' after we make our first call to TDTRP below. However, obj can be any unique variable name.

MATLAB	Python	Values
`'INTERFACE'`	`interface`	String, interface type. Can be `'USB'`, `'USB3'`, or `'GB'` (default for fiber optic interface)
`'NUMBER'`	`number`	Integer, device number as enumerated by zBusMon (default 1)
`'FS'`	`fs`	Float, sampling rate to run the circuit at (otherwise uses circuit default)

MATLAB

```
TDT = TDTRP('C:\TDT\ActiveX\ActXExamples\RP_Files\Continuous_Acquire.rcx', 'RZ6')
```

`TDT` is the returned object. The target device is an RZ6 processor. I only have one, so no need to set the 'NUMBER' parameter, and it has a fiber optic interface built into it, so it uses the default 'GB' interface.

Python

```
from TDTRP import TDTRP

TDT = TDTRP(r'C:\TDT\ActiveX\ActXExamples\RP_Files\Continuous_Acquire.rcx', 'RZ6')
```

The Python wrapper takes the same arguments as keywords: `interface`, `number` and `fs`. It can also be used as a context manager, which halts the circuit on the way out even if something raises.

RPcoX Method Comparison

```
RP = actxserver('RPco.X');
RP.ConnectRZ6('GB',1)
RP.LoadCOF('C:\TDT\ActiveX\ActXExamples\RP_Files\Continuous_Acquire.rcx');
e = RP.Run

if e == 0
    error('error running circuit')
else
    disp('Circuit ready to run')
end
```

Setting the mode with TDTRP

```
TDT.halt % stop any existing processing chain

TDT.load % load the processing chain
```

```
TDT.run % run the processing chain
```

MATLAB

```
>> circuit = 'C:\TDT\ActiveX\ActXExamples\RP_Files\Continuous_Acquire.rcx';

>> TDT.halt;

>> TDT.load(circuit);

>> TDT.run;

% Calling TDTRP() does the equivalent of calling halt, load, and run
% load and run the circuit onto the specified device
```

Python

```
circuit = r'C:\TDT\ActiveX\ActXExamples\RP_Files\Continuous_Acquire.rcx'

TDT.halt()
TDT.load(circuit)
TDT.run()

# Creating a TDTRP does the equivalent of halt, load and run
```

RPcoX Method Comparison

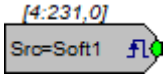
```
RP.Halt
circuit ='C:\TDT\ActiveX\ActXExamples\RP_Files\Continuous_Acquire.rcx';
e = RP.LoadCOF(circuit); % Loads circuit

if e == 0
    error('Error loading circuit')
else
    disp('Circuit ready to run')
end

e = RP.Run

if e == 0
    error('Error running circuit')
else
    disp('Circuit ready to run')
end
```

Software Triggers



Use software triggers to pulse a `TrigIn` component in RPvdsEx for one sample.

```
TDT.trg(TRIGNUM) % TRIGNUM is the software trigger index (1-10)
```

TRIGNUM determined by number after 'Soft' in the `TrigIn` component.

MATLAB

```
TDT.trg(1);
```

Python

```
TDT.trg(1)
```

RPcoX Method

```
RP.SoftTrg(1);
```

Writing to Parameter Tags

```
TDT.write(TAGNAME, VALUE, 'parameter', value, ...)
```

`TAGNAME` is a string. `VALUE` is the single value or array that you want to push onto the tag. Whether the tag is pointed to a scalar component or buffer will determine what is a valid write. Examples of these paradigms are below in the examples below.

MATLAB	Python	Values
`'FORMAT'`	`fmt`	String, destination format (array only) options are 'F32' (default), 'I32', 'I16', 'I8'
`'OFFSET'`	`offset`	Scalar, offset into buffer (array only). Default is 0

Parameter Tag pointing to a scalar value

MATLAB

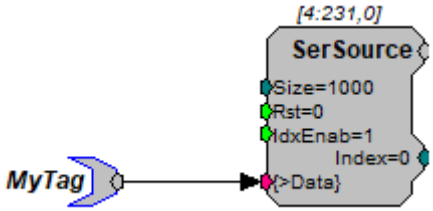
```
% writing a single scalar value to a ConstF component
TDT.write('MyTag', 3.14);
```

Python

```
# writing a single scalar value to a ConstF component
TDT.write('MyTag', 3.14)
```

RPcoX Method

```
RP.SetTagVal('MyTag', 3.14); % for single value
```



Parameter Tag pointing to memory buffer

MATLAB

```
% writing an array of values to a SerSource, which is a serial buffer
TDT.write('MyTag', 1:10);
```

Python

```
# writing an array of values to a SerSource, which is a serial buffer
TDT.write('MyTag', range(1, 11))
```

RPcoX Method

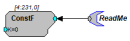
```
RP.WriteTagVEX('MyTag', 0, 'F32', 1:10); % for array of values
```

Reading from Parameter Tags

```
TDT.read(TAGNAME,'parameter',value,...)
```

Similar to the TDT.write method, what the tag is pointed to will determine what a valid read is. A scalar tag must be on a scalar component. An array must be on a buffer.

MATLAB	Python	Values
`'SOURCE'`	`source`	String, source format (array only) options are 'F32' (default), 'I32', 'I16', 'I8'
`'DEST'`	`dest`	String, destination format (array only) options are 'F64', 'F32' (default), 'I32', 'I16', 'I8'
`'SIZE'`	`size`	Scalar, number of words to read (array only). Default is the entire buffer
`'OFFSET'`	`offset`	Scalar, offset into buffer (array only). Default is 0
`'NCHAN'`	`nchan`	Scalar, number of channels in buffer (array only). Used for de-interlacing data. Default is 1



Parameter Tag pointing to a scalar value

Parameter tags always point *towards* a component port.

MATLAB

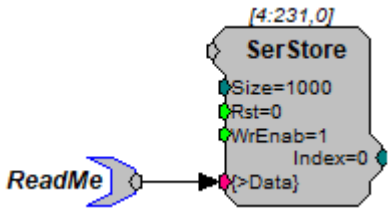
```
% read single value parameter tags
ScalarVal = TDT.read('ReadMe');
```

Python

```
# read single value parameter tags
scalar_val = TDT.read('ReadMe')
```

RPcoX Method

```
RP.GetTagVal('ReadMe'); % for single value
```



Parameter Tag pointing to memory buffer

MATLAB

```
% This will read all values stored in the SerStore serial buffer
ArrVals = TDT.read('ReadMe');
```

Python

```
# This will read all values stored in the SerStore serial buffer
arr_vals = TDT.read('ReadMe')
```

RPcoX Method

```
RP.ReadTagVEX('ReadMe',0,1000,'F32','F32',1); % for 1 chan array of 1000 values
```

Examples Using TDTRP

You can find examples of ActiveX code using TDTRP installed with ActiveX in:

```
C:\TDT\ActiveX\ActXExamples\Matlab
C:\TDT\ActiveX\ActXExamples\py
```

or you can download it in the ActiveX Controls example zip.

TDTPA5 User Guide

Use TDTPA5 to control a PA5 Programmable Attenuator from MATLAB or Python.

Requirements

- TDT Drivers
- TDT ActiveX package
- Python only: `pywin32`

About TDTPA5

TDTPA5 is a wrapper for the PA5x ActiveX control that installs with the TDT ActiveX package. It provides direct access to the attenuator (connect, set and read attenuation, write to the front panel display, set User mode parameters) and is generally easier to use than calling the PA5x object directly.

TDTPA5 installs with the ActiveX controls into:

```
C:\TDT\ActiveX\ActXExamples\Matlab
C:\TDT\ActiveX\ActXExamples\py
```

or you can download it in the ActiveX Controls example zip.

TDTPA5 Functionality

> **Note**
>
> See PA5 Programmable Attenuator in the ActiveX User Reference for the complete PA5x programming guide.

Creating a TDTPA5 object

Connecting also resets the attenuator, so you always start from a known state.

Input Parameter	Values
`'INTERFACE'`	String, interface type. Can be `'USB'` or `'GB'` (default for fiber optic interface)
`'NUMBER'`	Integer, device number as enumerated by zBusMon (default 1)
`'ATTEN'`	Float, attenuation to set once connected (default 0)

MATLAB

```
PA5 = TDTPA5();

% or, connecting to the second PA5 over USB and starting at 20 dB
PA5 = TDTPA5('INTERFACE', 'USB', 'NUMBER', 2, 'ATTEN', 20);
```

Python

```
from TDTPA5 import TDTPA5

pa5 = TDTPA5()

# or, connecting to the second PA5 over USB and starting at 20 dB
pa5 = TDTPA5(interface='USB', number=2, atten=20)
```

PA5x Method Comparison

```
PA5 = actxserver('PA5.x');
if PA5.ConnectPA5('GB', 1) == 0
    error('Unable to connect')
end
PA5.Reset
```

Setting and Reading Attenuation

Attenuation is a floating point value from 0.0 to 120.0 dB. Values outside that range are rejected before the call reaches the hardware.

MATLAB

```
PA5.atten(20);         % set 20 dB of attenuation

current = PA5.atten;   % read it back
```

Python

```
pa5.atten(20)          # set 20 dB of attenuation

current = pa5.atten()  # read it back
```

PA5x Method Comparison

```
PA5.SetAtten(20);

% SetAtten sets an error flag rather than returning failure, so the
% error string is what actually tells you whether it worked
err = PA5.GetError;
if ~isempty(err)
    disp(err)
end

current = PA5.GetAtten;
```

Front Panel Display

The display is eight characters wide. Longer text is truncated rather than rejected. Position is 0 (left, the default) through 7 (right).

MATLAB

```
PA5.show('20 dB');
PA5.show('CAL', 4);   % starting at position 4
```

Python

```
pa5.show('20 dB')
pa5.show('CAL', 4)    # starting at position 4
```

PA5x Method Comparison

```
PA5.Display('20 dB', 0);
```

Resetting

Restores the factory defaults: attenuation 0.0, step size 3.0, update dynamic.

MATLAB

```
PA5.reset;
```

Python

```
pa5.reset()
```

PA5x Method Comparison

```
PA5.Reset;
```

User Attenuation Mode

User values change what the front panel displays and how the dial behaves. They do **not** change the actual attenuation applied to the signal. Use `atten` for that. They are intended for quickly comparing several PA5s from their front panels.

Parameter	Meaning
`'BASE'`	Base attenuation (0.0 .. 120.0 dB). Use when several speakers vary in output so each displays the same attenuation.
`'STEP'`	Dial step size (0.0 .. 120.0 dB)
`'REFERENCE'`	Reference dB level (0.0 .. 120.0 dB). Lets the display show smaller, or negative, numbers.
`'UPDATE'`	`'DYNAMIC'` changes attenuation continuously as the dial turns; `'MANUAL'` only applies it when the dial is pressed.
`'ABSMIN'`	Minimum attenuation the dial will allow, to prevent accidentally playing something very loud.

MATLAB

```
PA5.user('BASE', 5);
PA5.user('STEP', 5);
PA5.user('REFERENCE', 120);
PA5.user('ABSMIN', 20);
PA5.user('UPDATE', 'DYNAMIC');
```

Python

```
pa5.user('BASE', 5)
pa5.user('STEP', 5)
pa5.user('REFERENCE', 120)
pa5.user('ABSMIN', 20)
pa5.user('UPDATE', 'DYNAMIC')
```

PA5x Method Comparison

```
% the raw control takes numeric parameter codes
PA5_USERPAR_BASE = 1;
PA5_USERPAR_STEP = 2;
PA5_USERPAR_REFERENCE = 3;
PA5_USERPAR_UPDATE = 4;
PA5_USERPAR_ABSMIN = 5;
PA5_USERUPDATE_DYNAMIC = 0;

PA5.SetUser(PA5_USERPAR_BASE, 5);
PA5.SetUser(PA5_USERPAR_STEP, 5);
PA5.SetUser(PA5_USERPAR_REFERENCE, 120);
PA5.SetUser(PA5_USERPAR_ABSMIN, 20);
PA5.SetUser(PA5_USERPAR_UPDATE, PA5_USERUPDATE_DYNAMIC);
```

Checking for Errors

Returns the last error string, or an empty string if there was none. `atten` already checks this for you and warns.

MATLAB

```
msg = PA5.error;
if ~isempty(msg)
    PA5.show(msg);
end
```

Python

```
msg = pa5.error()
if msg:
    pa5.show(msg)
```

Examples Using TDTPA5

You can find examples installed with ActiveX in:

```
C:\TDT\ActiveX\ActXExamples\Matlab
C:\TDT\ActiveX\ActXExamples\py
```

or you can download them in the ActiveX Controls example zip.

Step a PA5 through a range of attenuations:

Python

```
import time
from TDTPA5 import TDTPA5

pa5 = TDTPA5()
for atten in range(0, 61, 10):
    pa5.atten(atten)
    pa5.show('{0} dB'.format(atten))
    print('attenuation now {0} dB'.format(pa5.atten()))
    time.sleep(1)
pa5.reset()
```

MATLAB

```
PA5 = TDTPA5();
for atten = 0:10:60
    PA5.atten(atten);
    PA5.show([num2str(atten) ' dB']);
    disp(['attenuation now ' num2str(PA5.atten) ' dB']);
    pause(1);
end
PA5.reset;
```

TDTZBUS User Guide

Use TDTZBUS to trigger and manage zBUS device racks from MATLAB or Python.

Requirements

- TDT Drivers
- TDT ActiveX package
- Python only: `pywin32`

About TDTZBUS

TDTZBUS is a wrapper for the zBUSx ActiveX control that installs with the TDT ActiveX package. It provides access to the zBUS (connect, fire the zBusTrigA and zBusTrigB triggers, flush and reset racks, look up device addresses and microcode versions) and is generally easier to use than calling the zBUSx object directly.

The main convenience is the triggers: the wrapper takes the trigger type by name rather than as a magic number, and supplies the minimum delay for you.

TDTZBUS installs with the ActiveX controls into:

```
C:\TDT\ActiveX\ActXExamples\Matlab
C:\TDT\ActiveX\ActXExamples\py
```

or you can download it in the ActiveX Controls example zip.

TDTZBUS Functionality

> **Note**
>
> See zBUS Device in the ActiveX User Reference for the complete zBUSx programming guide.

Creating a TDTZBUS object

Input Parameter	Values
`'INTERFACE'`	String, interface type. Can be `'USB'`, `'USB3'` or `'GB'` (default for fiber optic interface)

MATLAB

```
zBUS = TDTZBUS();

% or over USB 3.0
zBUS = TDTZBUS('INTERFACE', 'USB3');
```

Python

```
from TDTZBUS import TDTZBUS

zb = TDTZBUS()

# or over USB 3.0
zb = TDTZBUS(interface='USB3')
```

zBUSx Method Comparison

```
zBUS = actxserver('ZBUS.x');
if zBUS.ConnectZBUS('GB') == 0
    error('Unable to connect')
end
```

Triggers

zBusTrigA and zBusTrigB trigger several processors simultaneously, either in one rack or across every rack.

Argument	Values
rack	`0` for every rack (default), or a single rack number
type	`'PULSE'` (default), `'HIGH'` or `'LOW'`
delay	Delay in ms before the trigger, minimum 2 ms per rack

Confirming a trigger fired

From microcode v57 onward these calls return 0 even when the trigger fired correctly, so the wrapper does not report the return value. To confirm a trigger, link the `zTrig` output to a digital output on the device, or to a parameter tag you can read back.

To generate a single sample pulse, put an `EdgeDetect` after the `zTrig` component in your circuit.

MATLAB

```
zBUS.trigA;                 % pulse trigger A on every rack
zBUS.trigA(1);              % pulse trigger A on rack 1 only
zBUS.trigA(1, 'HIGH');      % hold rack 1 high
zBUS.trigA(1, 'LOW');       % and back low
zBUS.trigB(0, 'PULSE', 10); % trigger B, every rack, 10 ms delay
```

Python

```
zb.trigA()                  # pulse trigger A on every rack
zb.trigA(1)                 # pulse trigger A on rack 1 only
zb.trigA(1, 'HIGH')         # hold rack 1 high
zb.trigA(1, 'LOW')          # and back low
zb.trigB(0, 'PULSE', 10)    # trigger B, every rack, 10 ms delay
```

zBUSx Method Comparison

```
% 0 = pulse, 1 = high, 2 = low
zBUS.zBusTrigA(0, 0, 2);
zBUS.zBusTrigA(1, 0, 2);
zBUS.zBusTrigA(1, 1, 2);
zBUS.zBusTrigA(1, 2, 2);
zBUS.zBusTrigB(0, 0, 10);
```

Flushing and Resetting a Rack

`flush` clears the I/O lines to remove bad data from the buffers. `reset` resets the rack's logical connection to the computer, clearing data lines and restoring device connections. Both default to rack 1.

MATLAB

```
zBUS.flush(1);
zBUS.reset(1);
```

Python

```
zb.flush(1)
zb.reset(1)
```

zBUSx Method Comparison

```
zBUS.FlushIO(1);
zBUS.HardwareReset(1);   % returns 0 on success
```

Device Address and Version

Both take a device name and an optional device number (default 1). The wrapper maps the name to the numeric device ID the control expects.

`addr` returns 0 if there is no such device. Otherwise even numbers are position 1 of a rack and odd numbers position 2: 2 = rack 1 position 1, 3 = rack 1 position 2, 4 = rack 2 position 1.

`version` returns 0 if there is no such device, otherwise the microcode version.

MATLAB

```
address = zBUS.addr('PA5');        % address of PA5 #1
version = zBUS.version('RZ6', 2);  % microcode version of RZ6 #2

if zBUS.version('RP2') < 50
    disp('Update your microcode to run with this ActiveX')
end
```

Python

```
address = zb.addr('PA5')           # address of PA5 #1
version = zb.version('RZ6', 2)     # microcode version of RZ6 #2

if zb.version('RP2') < 50:
    print('Update your microcode to run with this ActiveX')
```

zBUSx Method Comparison

```
% the raw control takes numeric device IDs
address = zBUS.GetDeviceAddr(33, 1);       % 33 = PA5
version = zBUS.GetDeviceVersion(54, 2);    % 54 = RZ6

if zBUS.GetDeviceVersion(35, 1) < 50       % 35 = RP2
    disp('Update your microcode to run with this ActiveX')
end
```

Recognised device names, and the IDs they map to:

PA5	RP2	RL2	RA16	RV8	RX5	RX6	RX7	RX8	RZ2	RZ5	RZ6
33	35	36	37	38	45	46	47	48	50	53	54

Synchronising Rack Clocks

`sync` minimises drift between racks. The first rack listed is the master and the rest take their clock from it. A raw bitmask is also accepted.

> **Warning**
>
> This only works with the UB1/UZ4 USB 1.1 interfaces and always returns 0 on any other interface type. Connect the Sync lines with short BNC cables and T-connectors to minimise noise.

MATLAB

```
zBUS.sync([1 3]);   % rack 1 is master, rack 3 follows
zBUS.sync(5);       % the same thing as a bitmask
```

Python

```
zb.sync([1, 3])     # rack 1 is master, rack 3 follows
zb.sync(5)          # the same thing as a bitmask
```

zBUSx Method Comparison

```
zBUS.zBusSync(5);   % bitmask only
```

Checking for Errors

Returns the last error string, or an empty string if there was none. Note that an unsuccessful return is not always a zBUS error: asking for a device that does not exist legitimately returns 0.

MATLAB

```
if zBUS.version('PA5') == 0
    disp(zBUS.error);
end
```

Python

```
if zb.version('PA5') == 0:
    print(zb.error())
```

Examples Using TDTZBUS

You can find examples installed with ActiveX in:

```
C:\TDT\ActiveX\ActXExamples\Matlab
C:\TDT\ActiveX\ActXExamples\py
```

or you can download them in the ActiveX Controls example zip.

Report what is on the bus, then toggle trigger A:

Python

```python
import time
from TDTZBUS import TDTZBUS

zb = TDTZBUS()

for device in ('RZ2', 'RZ5', 'RZ6', 'RP2', 'RX6', 'RX8'):
    version = zb.version(device)
    if version:
        print('{0} #1 at address {1}, microcode v{2}'.format(
            device, zb.addr(device), version))

for _ in range(5):
    zb.trigA(0, 'HIGH')
    time.sleep(2)
    zb.trigA(0, 'LOW')
    time.sleep(2)
```

MATLAB

```matlab
zBUS = TDTZBUS();

devices = {'RZ2', 'RZ5', 'RZ6', 'RP2', 'RX6', 'RX8'};
for i = 1:numel(devices)
    version = zBUS.version(devices{i});
    if version > 0
        fprintf('%s #1 at address %d, microcode v%d\n', ...
            devices{i}, zBUS.addr(devices{i}), version);
    end
end

for i = 1:5
    zBUS.trigA(0, 'HIGH');
    pause(2);
    zBUS.trigA(0, 'LOW');
    pause(2);
end
```