# OpenDeveloper Reference Manual

Updated: 9/4/2015

OpenDeveloper Reference Manual

**Copyright**



**Licenses and Trademarks**

# Table of Contents

# Before You Begin:

## *Requirements*

TDT Drivers and the OpenEx Suite must be installed before installing OpenDeveloper.

The recommended operating system for all TDT systems is Windows 7®.

## *Organization of the Manual*

This manual is organized in the following sections:

- Overview
- TTankX
- TTankInterfaces
- TDevAcc

# Overview

OpenDeveloper allows direct access to data stored in TDT's tank format through custom applications written in programming languages such as MATLAB, C++ or any language that supports ActiveX controls. It also allows real-time access to hardware controlled by OpenWorkbench. OpenDeveloper uses the same interface to OpenEx servers (TTank and OpenWorkbench) as other OpenEx applications, such as OpenScope and OpenController.

**How the TTank Server Works**
The TTank server is a database engine allowing multiple clients to access local or networked tank data. Several applications can run simultaneously on a single or multiple computers and access information through the server. Selective filters are used to more efficiently extract data based on time stamp, channel number, sort code (neural spikes) or relation to an epoch event.

**How the OpenWorkbench Server Works**
Circuit files created in RPvdsEx include parameter tags that OpenWorkbench server can access for read/write operations. The circuit files are loaded onto the hardware by OpenWorkbench. While experiments are running, OpenWorkbench server allows client applications direct access to the parameter tags for real-time analysis and dynamic control of experimental parameters.

**OpenDeveloper ActiveX Controls**
There are three ActiveX controls that install with OpenDeveloper: TTankX, TDevAccX and TTankInterfaces.

- TTankX provides access to TTank servers. This is used for extracting data directly into a custom application for analysis.
- TDevAcc provides real-time access to hardware connected to OpenWorkbench server, as well as OpenWorkbench system mode (Record, Preview, Standby, and Idle).
- TTankInterfaces offers a set of graphical user interfaces (GUIs) for tank management. These are the same GUIs seen in OpenScope.

**Before Using OpenDeveloper**
TDT recommends that users become familiar with how OpenEx works before they use OpenDeveloper. In particular, users should examine how OpenWorkbench, OpenController and OpenScope function.

# TTankX

The TTankX ActiveX control is used to read tank data directly into a custom application for analysis and/or display. Selective filters are used to more efficiently extract data based on time stamp, channel number, sort code (neural spikes). Epoch indexing allows you to also create fast filters based on the epoch events stored in the tank. For example, extracting data around the occurrence of a stimulation event and only selecting data that occurred at particular stimulation parameter(s).

This section includes examples of common tasks and best practices, followed by a reference section of all available TTankX methods. Example code is in MATLAB, unless otherwise stated.

## *Getting Started*

Create an instance of the TTankX control and connect to a TTank server. This can be a TTank server installed on the local machine or a remote TTank server on another machine. The local TTank server is called 'Local'. We need to tell the server our name so it can manage its connections; we will use 'Me' in this example.

```
TT = actxcontrol('TTank.X');
TT.ConnectServer('Local','Me');
```

Open a data tank on that server. Typically, provide the entire path to the tank as the first parameter. The second parameter 'R' opens the tank as read-only. This is currently the only supported option.

```
TT.OpenTank('C:\TDT\OpenEx\Tanks\DEMOTANK2', 'R');
```

Select a block within that tank.

```
TT.SelectBlock('Block-1');
```

We are now ready to extract data from the selected block.

**Best Practice:** Avoid rapidly creating and destroying TTankX objects and/or server connections. This can slow down your application. If possible, only one instance of TTankX should be created per application.

## *Basics: Working with Data*

There are three types of data we can extract: continuous data, snippet data and epoch data. Continuous data are sampled at regular intervals from the start of the block to the end. Snippet data are a fixed number of points triggered by some event such as a threshold crossing. Epoch data are scalar values that correspond with triggered events.

We use different TTankX methods to read each of these event types.

Note that there is a fixed delay between when the data occurs and when it is available from the TTank server. This is the cache delay and its value is set in OpenWorkbench properties. The default delay is six seconds, but may be as high as 30 or as low as two. This delay allows the tank

server a buffer so that it can handle variances in data flow. The lower the delay, the closer to 'real-time' access you get, but the chance for tank errors while saving data increases.

## Continuous Data

The ReadWavesV function is used to read continuous data. It returns an array containing the waveforms; each column is one channel.

```
waves = TT.ReadWavesV('Wave');
```

Instead of a long argument list, the ReadWavesV function uses global parameters to determine what channel(s) and time ranges to return, among other things. Most global parameters are set with the SetGlobalV functions. The most commonly used are 'Channel', 'T1' and 'T2'. 'Channel' is the channel number to extract (the default value is 0, meaning all channels). 'T1' is the start time in seconds (default 0.0). 'T2' is the stop time in seconds (default 0.0, which means read until the end of the block). The following script will return a column of data containing channel 1 of the store 'Wave' between time t1=5s and t2=10s.

```
TT.SetGlobalV('Channel', 1);
TT.SetGlobalV('T1', 5);
TT.SetGlobalV('T2', 10);
waves = TT.ReadWavesV('Wave');
```

Often times, calling ReadWavesV with the default global parameters will return NaN (Not a Number) in MATLAB. This is because the data set requested exceeded the maximum amount of data the TTank server can return in any one call. The value of this limit is also a global parameter, 'WavesMemLimit'. The default value of WavesMemLimit is 33554432 bytes (32 MB) but can be increased by the user.

```
TT.SetGlobalV('WavesMemLimit', 1024^3);
```

This increases the maximum limit of data returned in any one call to the tank server to 1GB.

If ReadWavesV still returns NaN then the user must retrieve one channel at a time and concatenate them into one larger array in MATLAB.

```
% read first channel
TT.SetGlobalV('Channel', 1);
waves = TT.ReadWavesV('Wave');
% preallocate big array
big_array = [waves zeros(length(waves), nchan-1)];
% read the rest of the channels
for i = 2:nchan
    TT.SetGlobalV('Channel', i);
    waves = TT.ReadWavesV('Wave');
  big_array(:,i) = waves;
end
```

Now big_array contains all of the Wave data but with more calls to the TTank server. Similarly, the user can step through the block using T1 and T2 global parameters and concatenate into a large MATLAB array.

## Snippet Data

The ReadEventsSimple function is used to read snippet data (e.g. neural spikes) into MATLAB.

```
N = TT.ReadEventsSimple('eNeu');
```

Instead of returning all of the event information directly, the events are cached locally and the number of events that fit the parameters is returned (N). We use other functions to parse the waveform data (ParseEvV) or additional event information such as time stamps, sort codes or channel number (ParseEvInfoV) from the locally cached data.

The first two parameters of ParseEvV specify the starting index and number of events to return information from. In most cases we want to return waveform data from all of the cached events.

```
spikes = TT.ParseEvV(0, N);
```

The spikes array contains all of the waveform data. Each row is an event and the columns are the waveform data for that event. The data is ordered by time.

In addition to the waveform data, we want to know when the spikes occurred, what channel they occurred on and what their sort codes are. ParseEvInfoV can be used to extract this information from the cached events. The first two parameters are the same as ParseEvV and the last parameter is used to specify what type of information to retrieve about the events.

```
channels = TT.ParseEvInfoV(0, N, 4);
sortcodes = TT.ParseEvInfoV(0, N, 5);
timestamps = TT.ParseEvInfoV(0, N, 6);
```

Each call returns a single row vector with event information in the same order as they appear in the spikes array. A complete list of possible values for the third parameter of ParseEvInfoV can be found on page 30.

Like ReadWavesV, the ReadEventsSimple function uses global parameters instead of arguments. We can use the ‘SortCode’ global parameter to specify a sort code filter. The default value of 0 returns all sort codes.

```
TT.SetGlobalV('SortCode', 1);
N = TT.ReadEventsSimple('eNeu');
```

spikes = TT.ParseEvV(0, N);

The spikes array will now only contain events that have a sort code value of 1.

## Epoch Data

Epoch data consists of an onset timestamp, a value, and possibly an offset timestamp. Like snippet data, epoch data can be retrieved from the tank using ReadEventsSimple. However, we are usually more interested in other data (neural spikes, LFPs) that occurred around the epoch events, so we extract these interesting events relative to epoch timestamps.

TTankServer can create a local index based on epoch events that allows you to query a subset of records from the tank that meet specific epoch conditions. This means extracting events that occurred when an epoch was a certain value or range of values, or constructing histograms for a specific time period around an epoch onset timestamp.

Indexing allows data to be accessed relative to epochs. TTankServer uses a process called filtering to perform record querying. With typical database engines, SQL or a similar language is used to query records from a larger record set. TTankServer uses a parametric filtering methodology for fast, powerful querying capabilities.

The CreateEpocIndexing method is used to build these epoch indexes. This method must be called each time we select a new block.

```
TT.CreateEpocIndexing
```

Best Practice: Use a tilde prefixed to the block name when calling SelectBlock to automatically call CreateEpocIndexing when the block is selected (e.g. `TTX.SelectBlock('~Block-3');`)

Once an epoch index has been created, filtering calls can be made to limit the records (events) returned by TTankServer. By default no filters are applied, meaning all valid event records are returned when a Read* command is called.

Once the index is created you can quickly get epoch information relative to some event data. The code segment below reads all of the event data for channel one, and then queries to find out what the stimulus frequency (Freq) was when the 13th event occurred.

```
TT.SetGlobalV('Channel', 1);
N = TT.ReadEventsSimple('eNeu');
timestamps = TT.ParseEvInfoV(0, N, 6);
freq_value = TTX.QryEpocAtV('Freq', timestamps(13), 0);
```

QryEpocAtV returns the value associated with a specified epoch. One of four values is returned. Use the last argument to control which value is returned. The options are: the value of the epoch (0), the onset timestamp of the epoch event (1), the offset timestamp of the epoch event (2), or the filter status (3). CreateEpocIndexing must be called before QryEpocAtV.

## Using Epochs as Filters

After the epoch index is created, you can also issue filter commands before reading data so that the tank server only caches data that you are interested in.

First, use ResetFilters to ensure that you do not filter data that has already been filtered.

```
TTX.ResetFilters
```

Next, a filter is applied with SetFilterWithDescEx so that only event data that occurred when the ‘Freq’ epoch was a specific value will be retrieved in future calls.

```
TTX.SetFilterWithDescEx('Freq=4000')
```

Now a call to ReadEventsSimple returns only events that occur when Freq is 4000. Note that the global parameter 'Options' has to be been changed to 'FILTERED' from its default value of 'ALL', which would ignore the filter we just set.

```
TT.SetGlobalStringV('Options','FILTERED');
N = TTX.ReadEventsSimple('Snip')
```

SetFilterWithDescEx can be called with multiple conditions to apply multiple epoch filters simultaneously. The Boolean operators 'and' and 'or' can be used to combine multiple epoch filters in one statement.

```
TTX.SetFilterWithDescEx('Freq=1000 or Freq=4000');
N = TTX.ReadEventsSimple('Snip')
```

Only Snip events that occurred when Freq was 1000 or 4000 are cached in the tank server.

Suppose we want to generate a histogram of 'Snip' event timestamps around when the Freq epoch triggered. We are interested in a time period of one second before the Freq epoch triggered to 2 seconds after it triggered.

```
TTX.SetFilterWithDescEx('Freq=4000');
TTX.SetEpocTimeFilterV('Freq',-1,3);
N = TTX.ReadEventsSimple('Snip')
```

Now when we retrieve Snip data the timestamps will be adjusted so they appear from one second before the Freq event to 2 seconds after. We can create a histogram directly from the retrieved timestamps.

```
timestamps = TTX.ParseEvInfoV(0, N, 6);
hist(timestamps, 30);
```

The GetEpocsV function can be used to directly read epoch data. This function does not use the global parameters, so it has a longer argument list. In addition to the name of the epoch store, its other parameters are start time, stop time and maximum number of epochs to return. If stop time is 0 then all data until the end of the block is read.

```
epocs = TTX.GetEpocsV('Tick', 0, 0, 10000);
```

The epocs array contains four rows. The first row is the scalar value associated with the epoch. The second is the onset time (in seconds). The third is the offset time (in seconds). The fourth row tells you whether the epoch fits the current filter selection.

## Cleaning Up

When working with TTankX, always close your tanks and release your server connection when you are done.

```
TT.CloseTank;
TT.ReleaseServer;
```

These two lines of code should be added at the end of your code.

# Examples

Several working Matlab example files are provided with the OpenDeveloper installation. Currently three of these are documented below. The remaining examples are commented in the Matlab file.

**Bydefault, all TTankX examples are installed at:**

**C:\TDT\OpenEx\Examples\TTankX_Example\Matlab**

**TDT recommends starting with the TDT2mat.m and SEV2mat.m examples for extracting all block data into a matlab structure.**

The latest documentation is always available on the TDT website at:
http://www.tdt.com/downloads/sys3docs.htm.

## Example: Reconstructing Waveforms from Events

This example demonstrates the steps used to reconstruct waveforms from events. First, data is filtered based on epoch events. Next, the filtered data is extracted from the tank and waveforms are built from the events.

### *The example demonstrates:*

- Using global parameters.
- Filtering signal data based on epoch events.
- Matching up data sets with different sampling rates for later display.

### *Methods used:*

- SetGlobalV
- SetGlobalStringV
- ResetFilters
- SetFilterWithDescEx
- ReadWavesV
- GetValidTimeRangesV
- ReadWaveOnTimeRangeV

#### Example File

C:\TDT\OpenEx\Examples\TTankX_Example\Matlab\WaveReconstruction.m

#### Accessing the Tank

The first section of the Matlab script connects to the TTank ActiveX control and opens the server, tank, and block. See *TTankX, Getting Started*, page 5, for more information.

```
MyTank = 'C:\TDT\OpenEx\Tanks\DemoTank';
```

```
MyBlock = '~Block-2';
TTX = actxcontrol('TTank.X')
TTX.ConnectServer('Local','Me')
TTX.OpenTank(MyTank,'R')
TTX.SelectBlock(MyBlock)
```

### Building an Epoch Index

In this example, when the block is defined, a tilde is appended to the block name (such as `MyBlock = '~Block-2'`), serving as a shortcut to call the CreateEpocIndexing method. This method is used to build epoch indexes which allow data to be accessed relative to epochs.

### Using Global Parameters

Global parameters reduce the argument list for each method. TDT sets up default settings for these method calls, see *Global Parameters* for more information, page 23. To modify the global settings, a method call of either SetGlobalV or SetGlobalStringV is generated (depending on the global variable). The method call sets a global parameter (in this case, the parameter 'Channel' is set to 0 meaning all channels). To set the global parameter for using filtered data, the SetGlobalStringV method sets the parameter 'Options' to FILTERED.

```
TTX.SetGlobalV('Channel',0);
TTX.SetGlobalStringV('Options','FILTERED');
```

See page 23, for more information on global parameters.

Resetting filters ensures that you do not filter a subset of your data.

```
TTX.ResetFilters;
```

### Filtering and Processing Data

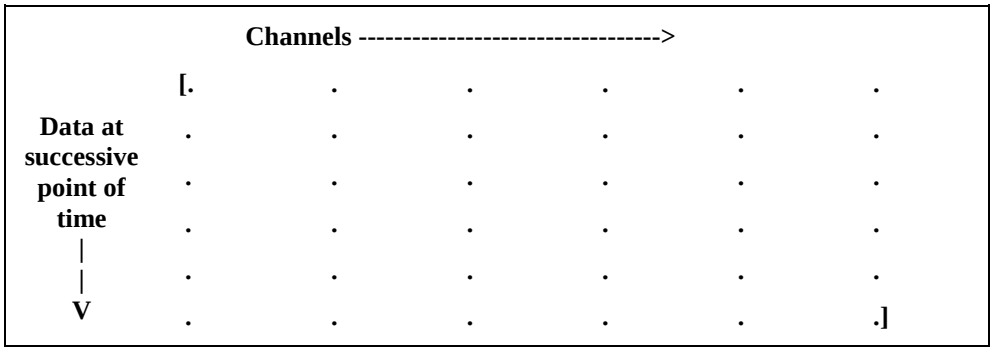
All filters are based on epochs. Epochs are scalar variables that are associated with fixed events, such as a behavioral response or stimulus presentation. In this example, the epochs are information about auditory stimuli, such as frequency and level. The next line sets the data filters. This command sets the data filters so that only data that occurred during epochs that had a Freq of 2000 and a Levl of 0 is read.

```
TTX.SetFilterWithDescEx('Freq=2000 and Levl=0')
```

ReadWavesV reads back the snippet data and forms a waveform from the points obtained for each channel.

```
filtSpikes = TTX.ReadWavesV('Snip');
```

ReadWavesV returns the events in a matrix. The diagram below illustrates the structure of the data matrix, with each column containing the waveform data for a channel and rows listing points in time.




In many cases, you'll want to view multiple events along the same axis. The next section of the code reads data for the Freq epoch so that it can be plotted with the snippet data already extracted. Before extracting epoch events, you'll need to consider two issues.

First, consider the sampling rates of the events of interest. For scalar data, the data is sampled in an asynchronous fashion, so there is no native sample rate. Unless specified, epoch events are extracted using a fixed sampling rate of 100 Hz. In this case, we need to generate a matrix that interleaves the Epoch event with zeros so that the Y dimension of the snippet and event data match.

Second, consider the difference in magnitude between the events. In many cases, the events will differ by orders of magnitude. Viewing the data on the same axis requires conversion of one value into another. In some cases, it may require scaling the values in the matrix. In other cases it may require replacing one value by another. In this case, we already know the value of the epoch (it is the epoch we filtered on) so we replace the value with one of the same magnitude.

Both of these issues can be resolved using some global variables. The global variable FillItem replaces each data point of the epoch with a fixed value. This fixed value must be specified by setting the global parameter FillValue. The global variable WaveSF sets the sampling frequency to 24414 (the sampling frequency of the Snip event) so that both data sets can be plotted on the same X-axis.

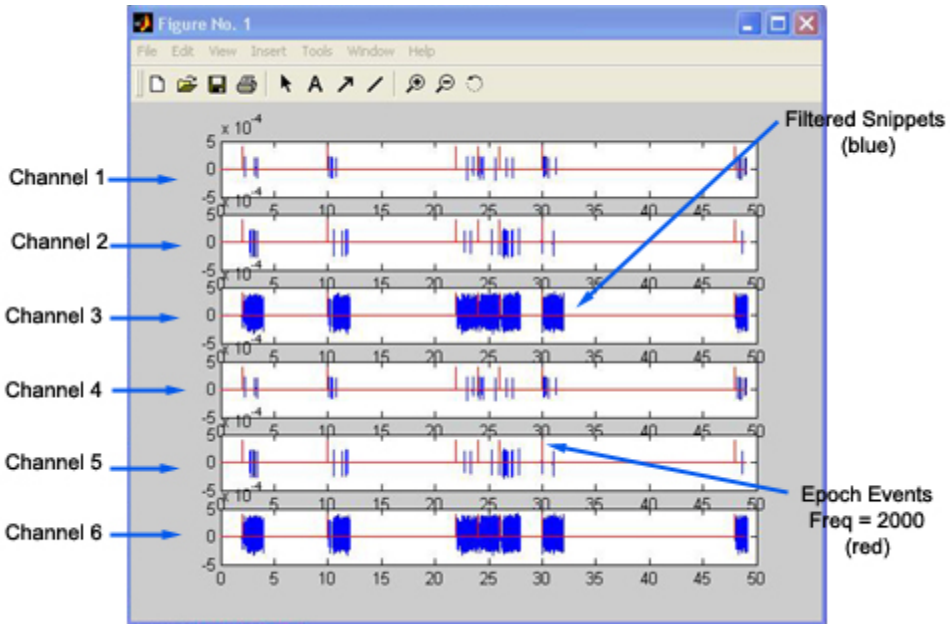
```
TTX.SetGlobals('FillItem=FixedNum;FillValue=0.0004');

TTX.SetGlobalV('WaveSF',24414)
```

After the global variables are set, ReadWavesV is used to read the epoch data.

```
filtFreqs = TTX.ReadWavesV('Freq');
```

After both data sets have been read and returned as Matlab matrices, standard Matlab scripting is used to set up an array for the time axis and to plot each of the six channels of snippet data (Snip) in a subplot. Superimposed upon that plot, is a plot of the occurrences of the frequency epoch (Freq) in red.




A second way to view the data is to compare filtered data from the same channel across epoch events. The next section of the script generates a second plot to display data for two channels, with a subplot for each event.

Keep in mind that the filters and parameters have not been changed since the last ReadWaves call. The data will once again be read after the same filter of 'Freq = 2000 and Levl = 0' is applied. One global parameter that must be reset is the FillItem parameter. Earlier, this parameter was set to FixedNum. To ensure that actual values are acquired the next time data is read, it must now be set to DataPoints.

```
TTX.SetGlobals('FillItem=DataPoints');
```

Next, the valid duration of each event (GetValidTimeRangesV) must be established and used to read back the snippet events for only those valid time ranges. In this example, the task is broken into three steps. First, GetValidTimeRangesV displays the time range values in the command window. Second, GetEpocsExV identifies the individual epochs to be read. Finally, ReadWavesOnTimeRangeV reads the data for each valid time range or epoch.

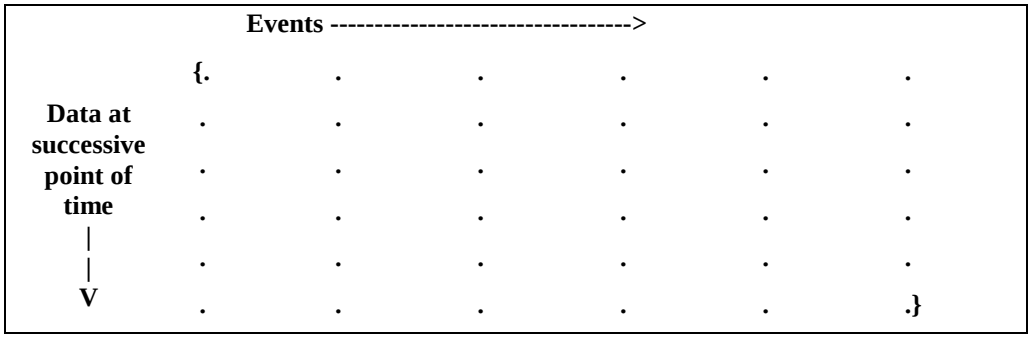
```
Ranges = TTX.GetValidTimeRangesV
TimeRanges = TTX.GetEpocsExV('Freq',0)
Chan1 = TTX.ReadWavesOnTimeRangeV('Snip',1);
Chan2 = TTX.ReadWavesOnTimeRangeV('Snip',2);
```

Note that GetValidTimeRangesV and GetEpocsExV are not required for the use of ReadWavesOnTimeRangeV. However, if GetEpocsExV is not used, epochs occurring in succession (without any gap) would be identified as a single time range and would be plotted in a single subplot.

Note: when viewing the output data in the Matlab command window, the user will notice that there are three time ranges that occur in succession (22-24 sec, 24-26 sec and 26-28 sec).

The data is read into two separate matrices. Each matrix contains the response from one channel. Each column of the matrix contains a different epoch event and the rows contain the data points

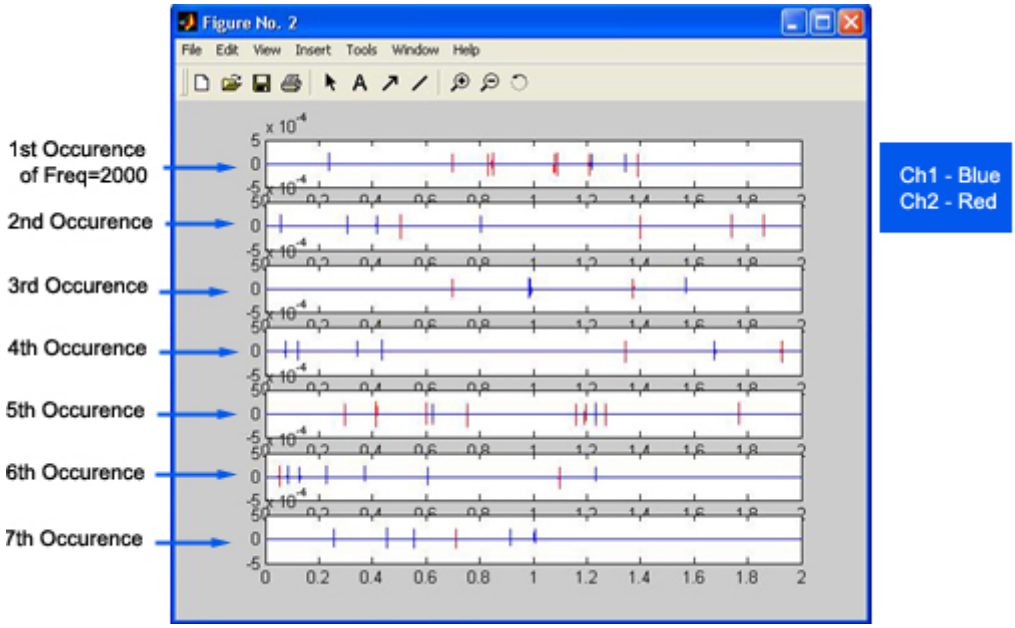
from the waveform for that channel. Note that, to plot the data correctly, times where the events did not occur are filled with zeros. Times where the events did occur contain the neural response of the unit. The diagram below illustrates the structure of one of the data matrices.




**Channel X Filtered On Epoch Event X**

This structure facilitates further analysis or plotting. For example, if you wanted to determine the average response from several stimulus presentations of the frequency and intensity, it would be simple to sum across the matrix and view the aggregate response.

In this example, standard Matlab scripting is used to plot the data. Each column of the channel one matrix (the first of seven valid time ranges during which Freq=2000) is displayed in a subplot with channel two data from the same time range superimposed in a different color on the same subplot.




Finally, the server is closed.

```
TTX.CloseTank
TTX.ReleaseServer
```

## Example: Using Filter Arrays

This example describes how to filter data from the OpenEx Tank. Data is filtered by querying the data tank using special events called Epochs. In OpenDeveloper, we use a three-step process to filter and load the data to local memory. The first part of the process is to set the filters. In this example, we use the ActiveX method call SetFilterArray that allows users to build a matrix of filter settings. Events are filtered into cells of a matrix, with each cell specifying a certain set of filters. The second part of the process is to read back the events and event information, which is done with ReadEventsSimple. The final part is to parse the event information for later analysis and display.

### *The example demonstrates:*

- Filtering signal data based on epoch events.
- Using global settings.
- Constructing filter arrays using multiple SetFilterArray calls.

### *Methods used:*

- SetFilterArray
- ReadEventsSimple
- ParseEvInfoV
- ParseEvV
- SetGlobals
- SetFilterWithDescEx

#### Example File

C:\TDT\OpenEx\Examples\TTankX_Example\Matlab\FilterArray.m

#### Accessing the Tank

The first section of the Matlab script connects to the TTank ActiveX control and opens the server, tank, and block. See *TTankX , Getting Started*, page 5, for more information.

```
MyTank = 'C:\TDT\OpenEx\Tanks\DemoTank';
MyBlock = '~Block-2';
TTX = actxcontrol('TTank.X')
TTX.ConnectServer('Local','Me')
TTX.OpenTank(MyTank)
TTX.SelectBlock(MyBlock)
```

#### Building an Epoch Index

In this example, when the block is defined, a tilde is appended to the block name (such as `MyBlock = '~Block-2';`), serving as a shortcut to call the CreateEpocIndexing method. This method is used to build epoch indexes which allow data to be accessed relative to epochs.

### Filtering and Processing Data

All filters are based on epochs. Epochs are scalar variables that are associated with fixed events, such as a behavioral response or stimulus presentation. In this example the epochs are information about auditory stimuli, such as frequency and level. The SetFilterArray command is used to set a filter and to give it an ID along a dimension. Three dimensions (0, 1, and 2) are allowed in all. Note that in other methods the dimension parameters will be defined as X, Y, and Z. Each filter has an ID along at least one dimension. Later the IDs will be used to plot the events in a grid, with each cell of the grid representing the conditions set by a filter.

The first filter is set with the condition Freq=1000 and is given ID 1 along the 0th dimension (x-Dimension). The first parameter in the argument specifies the dimension, the second specifies the ID of the filter along that dimension, followed by the filter itself. OpenEx allows users to query the data tank through an API using common SQL language with Boolean operations such as 'and' and 'or'. The last parameter sets a flag for exclusivity of the filter. Events that fit criteria of multiple filter settings can be assigned either to the lowest ID number (exclusivity) or to each filter for which it meets the criteria. Users should consider whether setting this flag will bias their analysis.

```
a = TTX.SetFilterArray(0,1,'Freq=1000',0)
b = TTX.SetFilterArray(0,2,'Freq=2000',0)
c = TTX.SetFilterArray(0,3,'Freq=4000',0)
d = TTX.SetFilterArray(0,4,'Freq=8000',0)
e = TTX.SetFilterArray(1,1,'Levl=0',0)
```

| | **Dimension 0 (X) ---------->** | | | | |
|---|---|---|---|---|---|
| **Dimension 1(Y)** **\| \| v** | | ID1 Freq=1000 | ID2 Freq=2000 | ID3 Freq=4000 | ID4 Freq=8000 |
| | ID1 Levl =0 | Freq=1000 and Levl=0 | Freq=2000 and Levl=0 | Freq=4000 and Levl=0 | Freq=8000 and Levl=0 |

### Reading Data

The filtered events in 'Snip' are read from channel one. Here we use the ReadEventsSimple call, which reads events from the tank into local memory. Users of previous versions of OpenDeveloper, note that ReadEventsSimple is a simplified version of the ReadEvents call. It has the same functionality as ReadEventsV, but uses global parameters instead of arguments.

### Using Global Parameters

The ReadEventsSimple method uses several global parameters whose default values are not changed in the example, such as SortCode (default 0, meaning all), T1 and T2 (both default 0, meaning full time span). Three globals, however, are changed – MaxReturn, Channel and Options. MaxReturn and Channel refer to the maximum number of events returned and channels extracted. By default, the global parameter Options is set to ALL, specifying that all events are extracted, not just filtered or new events. In this case, SetGlobalV is used to set the Options to FILTERED so that only events that meet the filter criteria will be read.

```
TTX.SetGlobals('Channel=1; MaxReturn=10000; Options=FILTERED');
```

See page 23, for more information on global parameters.

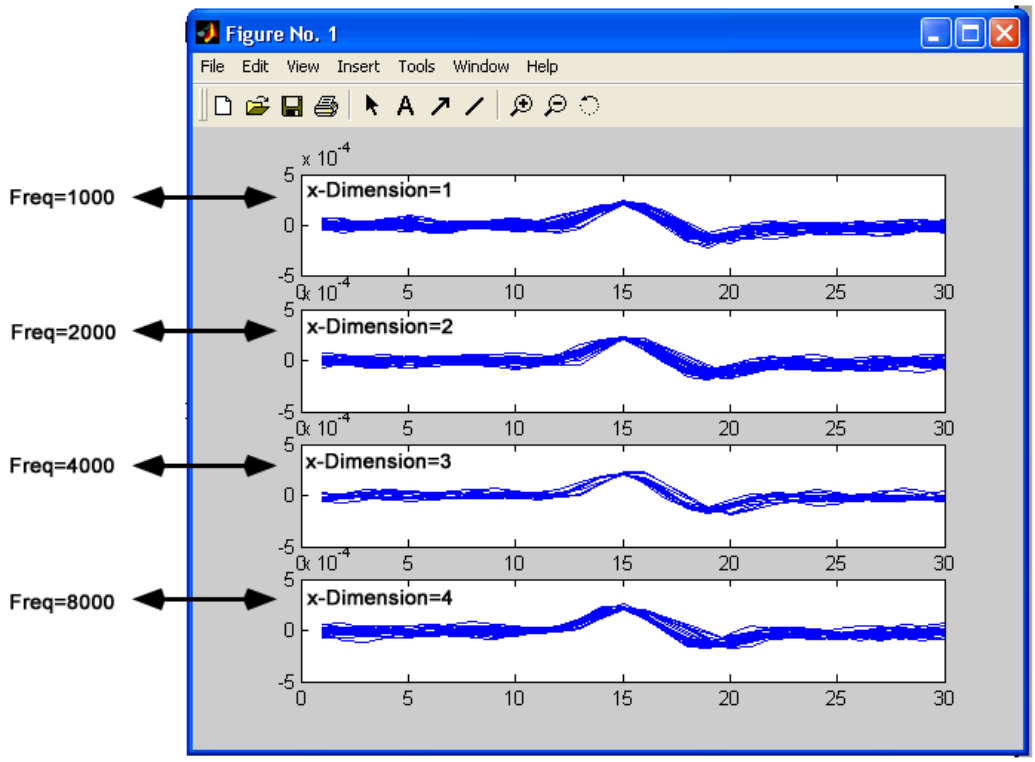
If users need to have more control over the parameters then they should use ReadEvents. ReadEventsSimple returns the number of events read. So it will return a maximum of 10000 filtered events from channel one.

```
X = TTX.ReadEventsSimple('Snip');
```

Next, the program loops through each event that was extracted from channel one of 'Snip'. Then the ID of each event along the 0th dimension is extracted using the ParseEvInfoV call. Finally, each event is extracted.

Within this loop, a grid of plots with one column and four rows (dimensions of the filter array) is formed. Then each event is plotted in that cell of the grid, which denotes its ID. For example, if an event has ID 2 along the 0th dimension, then it will be plotted in subplot two. At the end of the loop, subplot two will have only those events which have 0th dimension ID equal to 2, that is, those events which satisfy the Freq=2000 filter.

```
for t = 1:double(x)
  xid  = TTX.ParseEvInfoV(t-1,1,11);
   data = TTX.ParseEvV(t-1,0);
  …
end
```



For the next plot, the global parameters are set again, this time with channel being 0 or All, so that data from all the channels will be viewed.

```
TTX.SetGlobals('Channel=0; MaxReturn=10000; Options=FILTERED');
```

Next, the filter is set to Freq=2000, so that only events that occur when the value of the epoch ‘Freq’ is 2000 are extracted.

```
TTX.SetFilterWithDescEx('Freq=2000')
```

Again a filter array is created. However, this time events are differentiated by the channels on which they occurred. Note that each of these will be 'ANDed' with the previous filter, Freq=2000.

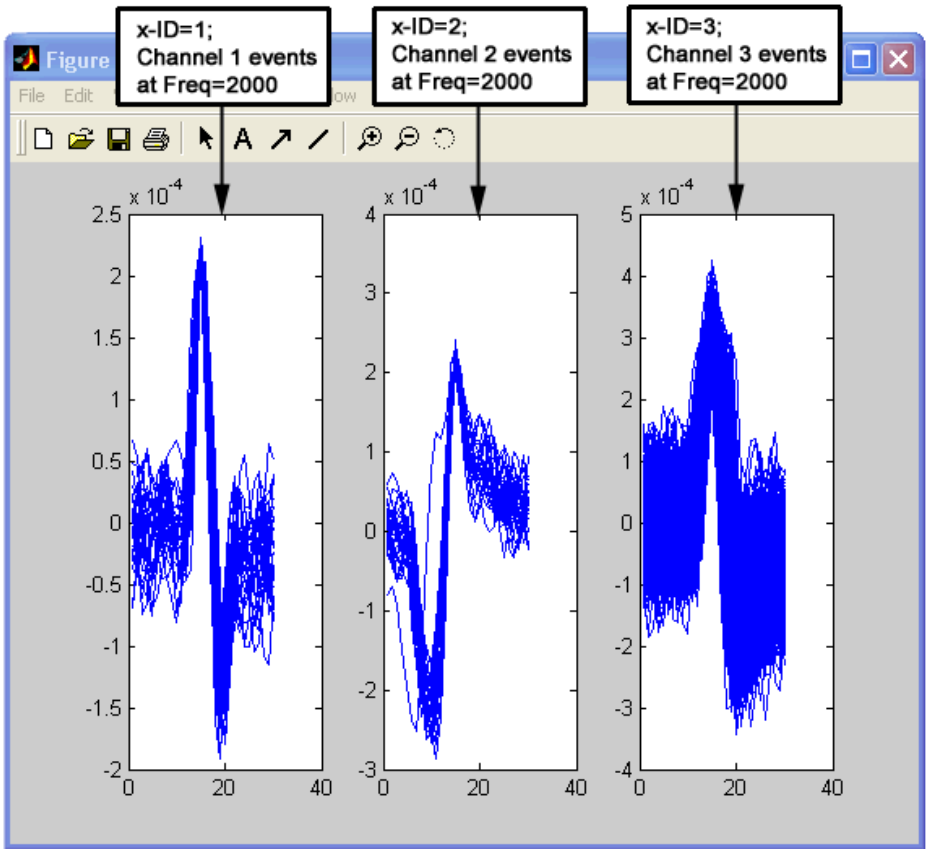
```
a = TTX.SetFilterArray(0,1,'Chan=1',0)
b = TTX.SetFilterArray(0,2,'Chan=2',0)
c = TTX.SetFilterArray(0,3,'Chan=3',0)
```

After setting up the new filters, the filtered events in 'Snip' from all channels are read again.

```
x = TTX.ReadEventsSimple('Snip');
```

Once more the program will loop through each event obtained, extract the x-dimension ID, extract the waveform itself, and plot the waveform in a subplot based on its ID. At the end of the loop, each subplot will have only those events which have X-dimension ID corresponding to that subplot number.

```
for t = 1:double(x)
    xid  = TTX.ParseEvInfoV(t-1,1,11);
    data = TTX.ParseEvV(t-1,0);
  …
end
```




### Closing the Tank

When all tasks are complete, the tank is closed and the server connection is released.

```
TTX.CloseTank;
TTX.ReleaseServer;
```

## Example: Plotting Data in an Inter-Spike Interval Histogram

This example demonstrates how to access tank data and parse events. Once the data has been read, events are used to plot the inter-spike intervals (ISI). The ISI histogram is a very commonly used plot in neurophysiology analysis for determining the number of distinct firing patterns (or neurons) that have been recorded.

### *The example demonstrates:*

- Reading time stamps for events.
- Using global variables.
- Using time stamp information to calculate and plot an inter-spike interval histogram.

### *Methods used:*

- SetGlobalV
- SetGlobalStringV
- ReadEventsSimple

#### Example File

C:\TDT\OpenEx\Examples\TTankX_Example\Matlab\InterSpikeInterval.m

#### Accessing the Data Tank

The first section of the Matlab script connects to the TTank ActiveX control and opens the server, tank, and block. See *TTank, Getting Started*, page 5, for more information.

```
MyTank = 'C:\TDT\OpenEx\Tanks\DemoTank';
MyBlock = '~Block-2';
TTX = actxcontrol('TTank.X')
TTX.ConnectServer('Local','Me')
TTX.OpenTank(MyTank,'R')
TTX.SelectBlock(MyBlock)
TTX.ResetFilters; % Reset all filters
```

#### Using Global Parameters and Processing Data

The ReadEventsSimple method in this example uses global parameters. ReadEventsSimple will use the default values for the global parameters unless they are specified using SetGlobalV or SetGlobalStringV (depending on the global variable). In this case, SetGlobalV is used to set the global variable Channel to include only data from channel 1.

**Note:** Global parameters reduce the argument list that must be specified for methods that use them. Unless specified, the default settings for these method calls are used (see *Global Parameters* for more information, page 23).

```
TTX.SetGlobalV('Channel',1);
```

See page 234, for more information on global parameters.

A simplified version of the ReadEvents method is used to read the Snip event from channel one. ReadEventsSimple has the same functionality as the older call ReadEventsV, except that it uses global variables instead of local arguments. The number of events read will be returned. The data itself and other information will now be available for parsing in local memory.

```
a = double(TTX.ReadEventsSimple('Snip'));
```

ParseEvInfoV is used to return the time stamp values for 10000 events. The first argument specifies the data number offset, the second specifies the number of events to parse, and the last one indicates what information to parse about the event. In this case, the number 6 specifies the time stamps. The data is returned in a matrix with a single row.

```
tstamps = TTX.ParseEvInfoV(0,10000,6);
```

### Creating an Inter-Spike Interval Histogram

The next section of code uses standard Matlab techniques to plot the data in an inter-spike interval histogram.

First, an array with 3500 zeroes is built to hold the 3500 bins of the histogram.

```
cache1 = zeros(1,3500);
```

Next, a loop is generated from 1 to the number of events obtained above.

```
for i = 1:a-1
```

Within the loop, the difference between each successive spike is computed.

```
    delta(i) = tstamps(i+1)-tstamps(i);
```

Each value is multiplied by 1000 and rounded off so that all values are in milliseconds and the bin width is 1 ms.

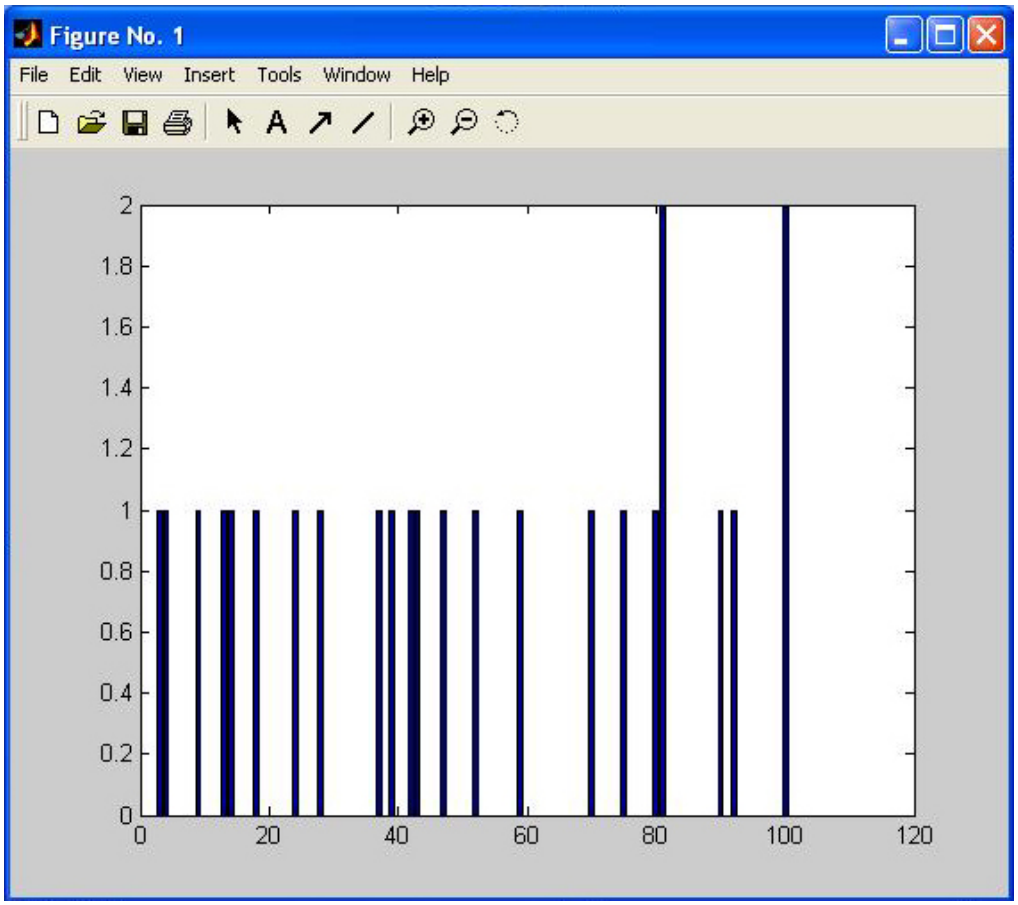
```
    bin = ceil(delta(i)*1000);
```

Next, the values must be sorted into the bins. To do this, increment the value of that element of cache1, which the time stamp falls into. For example, if the time stamp extracted is 0.0399 sec (39.9 ms), then bin = 40, and we increment the 40th element of cache1. So, the end result was that an event occurred at approximately 40 ms. Since our bin width is 1 ms, that event should fall into the 40th bin, and hence the 40th element of cache1 was incremented by 1.

```
    cache1(1,bin) = cache1(1,bin) + 1;
end
```

Next, the first 100 values of cache are plotted in a histogram yielding a plot of events with and inter-stimulus interval less than 100 ms.

```
bar(cache1(1:100));
```




The figure above shows an inter-stimulus interval plot generated using this example.

### Closing the Tank

When all tasks are complete, the tank should be closed and the connection to the server should be released.

```
TTX.CloseTank;
TTX.ReleaseServer;
```

# *Global Parameters*

Global parameters were included in release 1.54 of OpenDeveloper to minimize the number of variables in each method call. This minimizes errors in typing and allows users to set parameters only once for several calls. To make OpenDeveloper backward compatible, the method calls that use global parameters are defined differently. Global parameters can be set at any point in the program, and the new value will apply to any subsequent method that uses them. This allows users to set the value of a parameter across multiple methods and eliminates the need to set parameters in each method's argument list.

## Global Parameter Defaults

Global parameters are set with default values and need not be declared unless a different value is desired. The global parameters can be changed using *SetGlobals*, page 55, *SetGlobalV*, page 55, or *SetGlobalStringV*, page 55.

### AutoRefEpoch

**Default:** 1 (enabled)

**Description:** When using the call SetEpocTimeFilter, the time stamps of the events are referenced to the onset of that epoch event. This makes the construction of histograms easier. If this referencing of time stamps is not desired, then the AutoRefEpoc global parameter must be set to 0.

**Valid values:** 0 (disabled) or 1 (enabled)

### Channel

**Default:** 0 (all channels)

**Description:** Specifies that all channels will be used. Set to some number to specify a channel number.

**Valid values:** any non-negative integer

### FillItem

**Default:** 'DataPoints'

**Description:** Specifies that the returned matrix for ReadWavesV and similar methods will be filled with actual data points.

**Valid values:**

| | |
|---|---|
| 'Name' | Name of the event |
| 'Channel' | Channel number of the event |
| 'Sort' | Sort code of the event |
| 'Time' | Time stamp of the event |
| 'Freq' | Sampling rate of the event |
| 'xIndex' | Index along the x dimension (used with SetFilterArray) |
| 'yIndex' | Index along the y dimension (used with SetFilterArray) |
| 'zIndex' | Index along the z dimension (used with SetFilterArray) |
| 'FixedNum' | Arbitrarily specified number. The number can be specified using 'FillValue' |

### FillValue

**Default:** 1

**Description:** If 'FillItem' has been specified as 'FixedNum', this parameter specifies an arbitrary number to be inserted into the matrix at the occurrence of each event.

**Valid values:** any number

### MaxReturn

**Default:** 100000

**Description:** Specifies the maximum number of events to be returned. Used with ReadEventsSimple, GetEpocsExV, and GetValidTimeRanges, but not ReadWaves and similar methods.

**Valid values:** any positive integer

### Options

**Default:** 'ALL'

**Description:** Specifies what subset of data to cache

**Valid values:**

| *Value* | *Returns* |
|---|---|
| 'ALL' | all event records in range |
| 'NEW' | new events that occurred since last read.<br>Note: use this option to poll-read a block that is open for recording |
| 'SAME' | limit the read to the same access bounds as the previous read |
| 'JUSTTIMES' | list of event time stamps |
| 'DOUBLES' | couple with JUSTTIMES to get event time stamps as a list of doubles |
| 'NODATA' | only caches event information and not waveform data |
| 'FILTERED' | Only events that fit the currently specified filter(s) |
| 'ORDERED' | Orders output based on epoc filters |

### RespectOffsetEpoc

**Default:** 1 (enabled)

**Description:** This affects only buddy epochs or those epochs that have an offset. When set to 1 it will filter out the events that occur after the offset of the buddy epoch, otherwise it will include all events until the next onset.

**Valid values:** 0, 1

### SortCode

**Default:** 0 (any)

**Description:** Specifies the inclusion of spikes with all sort codes. Note: in many OpenEx applications, 0 is used for unassigned spikes. Here, 0 encompasses all sort codes.

**Valid values:** integers 0 to 31

If using a SortCode generated by OpenSorter, see *Known Anomalies*, page 79.

### T1

**Default:** 0.0

**Description:** When events are being extracted, this parameter specifies the starting time (in seconds) for that extraction.

**Valid values:** any non-negative value

### T2

**Default:** 0.0

**Description:** When events are being extracted, this parameter specifies the stopping time (in seconds) for the extraction. 0.0 is used to specify the end of the block, unless the number of events exceeds the MaxReturn.

**Valid values:** any non-negative value

### WavesMemLimit

**Default:** 33554432 (32 MB)

**Description:** Refers to the maximum memory (in bytes) that can be returned in a single call to the tank server. Used by ReadWavesV and ReadWavesOnTimeRangeV.

**Note:** If the WavesMemLimit is exceeded by a particular call, the method will return NaN or a negative return value.

**Valid values:** any positive integer

### WaveSF

**Default:** 0 (use event sampling rate, or 100 if there is no sampling frequency for the event, such as an epoch event)

**Description:** Specifies the sampling frequency, in Hz, used to sample a certain event. Used by ReadWavesV.

**Valid values:** any non-negative number

### WaveSFEvent

**Default:** 0 (none)

**Description:** Specifies an event whose sampling frequency is used to sample a certain event. Useful when you want to display multiple events with different sampling rates on the same plot with ReadWavesV or ReadWavesOnTimeRangesV.

**Valid values:** any event name or event code present in the block. Use SetGlobalStringV to set by event name, use SetGlobalV to set by event code (or to reset to 0).

# *Access Control* *TTank X*

## ConnectServer

**Description:** ConnectServer initiates a connection with a tank server. The connection adds a client to the server. Before exiting an application the program should release the connection by calling the ReleaseServer method.

**Prototype:**
```
Function ConnectServer(ServerName As String,
ClientName As String) As Long
```

**Arguments:**

| | |
|---|---|
| *ServerName* | name of the server, typically 'Local' |
| *ClientName* | name of the client application added to the server |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample connects to the Local server. The Client name is 'Me'.

```
TT = actxcontrol('TTank.X')
TT.ConnectServer('Local','Me')
```

**Related Calls:** ReleaseServer

## ReleaseServer

**Description:** ReleaseServer releases any connected server. This method should be called when the client is finished with the server, otherwise the server application will run until the client application is closed.

**Prototype:**
```
Function ReleaseServer()
```

**Related Calls:** ConnectServer

## OpenTank

**Description:** OpenTank opens a tank on the connected server for access of the specified type. The typical mode is 'R' for reading. ConnectServer must be called before OpenTank can be called. At the end of the client application use CloseTank to close the tank. To open a registered tank, use the tank name for the argument TankName. To open an unregistered tank, use the entire path to the tank.

**Prototype:**
```
Function OpenTank(TankName As String, AccessMode As
String) As Long
```

**Arguments:**

| | | | | |
|---|---|---|---|---|
| *TankName* | name of the tank to open | | | |
| *AccessMode* | 'R' (most common) | 'W' | 'C' | 'M' |
| | read | write | control | monitor |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample connects to the local server with the client name 'Me' and opens the unregistered tank named MyTank for reading.

```
TT.ConnectServer('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\Tanks\MyTank','R')
```

**Related Calls:** CloseTank, ConnectServer

### CloseTank

**Description:** CloseTank closes the open tank for a client.

**Prototype:** `Function CloseTank()`

**Related Calls:** OpenTank, ReleaseServer

### CheckTank

**Description:** CheckTank checks the current status of the tank.

**Prototype:** `Function CheckTank(TankName As String) As Long`

**Arguments:** *TankName* name of the tank

**Returns:** 67 (tank closed), 79 (tank open), 82 (tank in record mode)

**Related Calls:** OpenTank, CloseTank, GetStatus, GetError

### SelectBlock

**Description:** SelectBlock selects a block from the open tank for accessing. Before this is called ConnectServer and OpenTank must be called.

**Prototype:** `Function SelectBlock(BlockName As String) As Long`

**Arguments:** *BlockName* name of the block

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample connects to the server, opens a tank, and selects Block-1 from the opened tank.

```
TT.ConnectServer('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\Tanks\MyTank','R')
TT.SelectBlock('Block-1')
```

**Related Calls:** OpenTank, GetHotBlock, ConnectServer

# *Retrieving Records* *TTank X*

### ReadWavesV

**Description:** ReadWavesV reads continuous data or constructs a continuous waveform from a series of events (e.g. neural spikes) by filling zeroes in the samples where no event occurred. Scalar events are, by default, sampled at 100Hz.

The waveform can be down-sampled or up-sampled by setting the global parameter 'WaveSF' before calling ReadWavesV, or it can be sampled at the same frequency as an event specified in the global parameter 'WaveSFEvent'. This is useful for plotting two different events along the same timeline.

**Note:** When extracting streaming events, use WaveSF with caution. Inherent rounding errors make it unsuitable for downsampling this type of event. However, it can be used for rounding the sampling rate up or down to a nearby integer value or for upsampling data.

ReadWavesV does not work with 'CHAN' or 'SORT' filters.

**Prototype:** `Function ReadWavesV(EventName As String) As Variant`

**Arguments:** *EventName* four letter event name

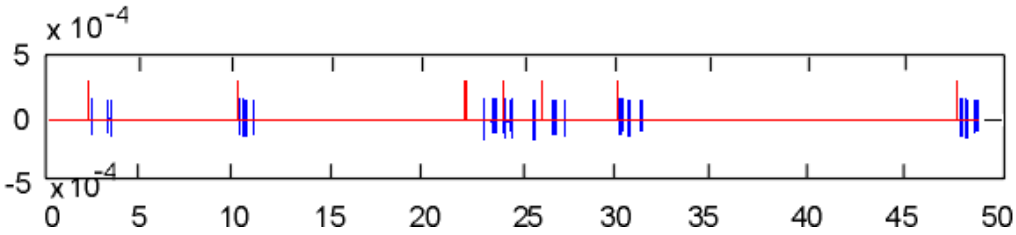
**Globals:** Channel, FillItem, FillValue, T1, T2, SortCode, Options, WaveSF, WaveSFEvent, WavesMemLimit

If using a SortCode generated by OpenSorter, see *Known Anomalies*, page 79.

**Returns:** single precision waveform array

**Sample Code:** This code sample sets a filter and uses SetGlobals to modify the WaveSFEvent and Channel global parameters, ensuring that the events can be plotted on the same timeline.

```
TT.SetGlobals('WaveSFEvent=Snip; Channel=1');
TT.SetFilterWithDescEx('Freq=2000');
wave = TT.ReadWavesV('Snip');
freq = TT.ReadWavesV('Freq');
plot(wave); hold on; plot(freq, 'r');
```




**Related Calls:** ReadWavesOnTimeRangeV, ReadEventsSimple

## ReadEventsSimple

**Description:** ReadEventsSimple reads the event records for the specified EventName from the currently selected block in the currently open tank. The events are cached to local memory where they can be accessed using ParseEvV and ParseEvInfoV.

**Prototype:**
```
Function ReadEventsSimple(EventName As String) As Long
```

**Arguments:** *EventName* four letter event name

**Globals:** Channel, T1, T2, SortCode, Options, MaxReturn

If using a SortCode generated by OpenSorter, see *Known Anomalies*, page 79.

**Returns:** number of events read

**Sample Code:** This code sample will set channel and time global parameters and cache all Snip events that fit those parameters.

```
TT.SetGlobals('Channel=1; T1=5; T2=10')
N = TT.ReadEventsSimple('Snip')
```

**Related Calls:** ParseEvV, ParseEvInfoV, ReadWavesV, ReadWavesOnTimeRangeV

## ReadEventsV

**Description:** ReadEventsV is similar to ReadEventsSimple but uses input arguments instead of global parameters. These additional arguments allow you to limit the access to a particular channel, sort code, and/or time range. The Options argument allows the user to select additional access modes.

**Prototype:**
```
Function ReadEventsV(MaxRet As Long, TankCode As
String, Channel As Long, SortCode As Long, T1 As
Double, T2 As Double, Options As String) As Long
```

**Arguments:**

| | | |
|---|---|---|
| Long | *MaxRet* | maximum number of events to be returned<br><br>**Note:** if the maximum number is returned it usually indicates that there were more events to be read. |
| String | *TankCode* | name of event in four character string form<br><br>**Note:** there is no error checking for valid TankCodes; incorrectly typed (TankCode is case sensitive) or nonexistent codes will return NaN or -1. |
| Long | *Channel* | return only records for this channel, or 0 for all channels |
| Long | *SortCode* | return only records with this sort code, or 0 to disregard sort codes<br><br>If using a SortCode generated by OpenSorter, see *Known Anomalies*, page 79. |
| Double | *T1* | return events with time stamp greater than or equal to T1<br><br>**Note:** use T1 = 0.0 to return events from the start of the block. |
| Double | *T2* | return events with a time stamp less than T2<br><br>**Note:** specify T2 = 0.0 to return events to the end of the block. |
| String | *Options* | See Options in global parameters. Options can be combined in a comma separated list like:<br><br>“JUSTTIMES,DOUBLES”. |

**Returns:** number of events cached to local memory

**Sample Code:** This code sample reads up to 1000 'Snip' events for channel 13 in Block-45.

```
TT.ConnectServer('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\Tanks\SomeTank','R')
TT.SelectBlock('Block-45')
NumRecs=TT.ReadEventsV(1000,'Snip',13,0,0,0,'ALL')
TT.CloseTank
TT.ReleaseServer
```

**Related Calls:** ReadEventsSimple, ReadWavesOnTimeRangeV

## ParseEvV

**Description:** ParseEvV retrieves some or all waveform data for event records cached in local memory by a call to ReadEventsSimple or ReadEventsV. The RecIndex parameter is used to specify the first record to access and is zero based. The function will return zero when the RecIndex is specified beyond the end of the returned list. Using the RecIndex and nRecs parameters you can retrieve waveform data for a number of records with just one call.

**Prototype:** `Function ParseEvV(RecIndex As Long, nRecs As Long) As Variant`

**Arguments:**

| | | |
|---|---|---|
| | *RecIndex* | Starting index of record(s) for which information is to be retrieved (0 based) |
| | *nRecs* | Number of records for which waveform data is to be retrieved. Pass 0 or 1 to get a single row of data for a single record. |

**Returns:** data in format found in tank

The data is a matrix with the columns being the waveform data and the rows being the indexed records. If nRecs = 0 the waveform data is returned in a row array.

**Sample Code:** Creates an index of Block-45 from MyTank and reads up to 1000 of the Snip events from time 0 to 47. ParseEvV retrieves the data for the first 10 records.

```
TT.ConnectSever('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\Tanks\MyTank','R')
TT.SelectBlock('Block-45')
Nrecs = TT.ReadEventsV(1000,'Snip',0,0,0,47,'ALL')
WaveData = TT.ParseEvV(0,10)
```

**Related Calls:** ParseEv, ParseEvInfoV, ReadWavesV, ReadEventsSimple

## ParseEvInfoV

**Description:** ParseEvInfoV is used to retrieve information from events cached using ReadEventsSimple or ReadEventsV, but not waveform data. Using the RecIndex and nRecs parameters you can retrieve information about a number of records with just one call.

**Prototype:** `Function ParseEvInfoV(RecIndex As Long, nRecs As Long, nItem As Long) As Variant`

**Arguments:**

| | | |
|---|---|---|
| Long | *RecIndex* | starting index of record(s) for which information is to be retrieved (0 based) |
| Long | *nRecs* | number of records for which information is to be retrieved |
| | | The number specified is the number of rows returned. Use 0 to have data returned in a single dimensional array or as a scalar. If RecIndex + nRecs exceeds the end of the cached records, the extra rows will be returned with zeros. |
| Long | *nItem* | item code for the information item to be returned |
| | | Use 0 to have all items returned as columns in the order shown below or select one of the following: |

| Item Code | Returns | Item Code | Returns |
|---|---|---|---|
| 1 | size of waveform data in bytes | 8 | data format code |
| 2 | event type | 9 | waveform sample rate in Hz (requires attached wavefrom data) |
| 3 | event code | 10 | not used (returns 0) |
| 4 | channel number | 11 | X dimension filter ID |
| 5 | sorting number | 12 | Y dimension filter ID |
| 6 | time stamp | 13 | Z dimension filter ID |
| 7 | scalar value (valid when no waveform data is attached) | 14 | fill Item |

**Returns:** The variant form of the data is a matrix with the columns being the data item or items and rows being the indexed records. The exact format of the returned data is dependent on which arguments are passed as 0s.

The possible return scenarios are:

{ nRecs > 0 and nItem > 0 } returns a row matrix containing the requested value for nRecs records

{ nRecs > 0 and nItem = 0 } returns a row/column matrix with nRecs rows by 10 columns containing all the information values

{ nRecs = 0 and nItem > 0 } returns a single scalar with the specified value for the specified record index

{ nRecs = 0 and nItem = 0 } returns a row matrix containing the 10 data items for the specified record index

**Sample Code:** Creates an index of Block-45 from MyTank and reads up to 1000 Snip events from time 0 to 47. The ParseEvInfoV returns the time stamp of each event.

```
TT.ConnectServer('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\Tanks\MyTank','R')
TT.SelectBlock('Block-45')
Nrecs=TT.ReadEventsV(1000,'Snip',0,0,0,47,'ALL')
TimeStamps=TT.ParseEvInfoV(0,Nrecs,6)
% to get the channel number for record index 17 call
chan = TT.ParseEvInfoV(17, 0, 4)
% to get all the information items for index 17 call
info = TT.ParseEvInfoV(17, 0, 0)
% to get all the info for all the records call
allinfo = TT.ParseEvInfoV(0, Nrecs, 0)
```

**Related Calls:** ParseEv, ParseEvV, ReadWavesV, ReadEventsSimple

### ReadWavesOnTimeRangeV

**Description:** This call returns events that occur in valid time ranges. (See *GetValidTimeRangesV*, page 36, and example below). Note that waves are built from the events. The events are returned in a variant with each column representing a valid time range. Note that only one channel can be processed at a time, because the events for a single channel are returned in a 2-D matrix.

**Prototype:**
```
Function ReadWavesOnTimeRangeV(EventName As String,
Channel As Long) As Variant
```

**Arguments:** *EventName* four letter event name

*Channel* channel number

**Globals:** T1, T2, FillItem, FillValue, WaveSF, SortCode, WavesMemLimit

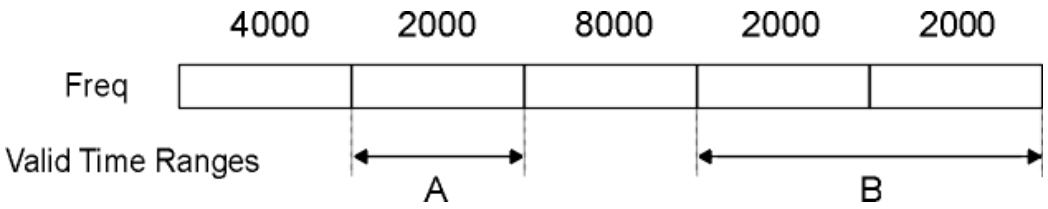
If using a SortCode generated by OpenSorter, see *Known Anomalies*, page 79.

**Returns:** matrix of waveform data with each column representing the events in a single valid interval of time

The length of the longest valid time interval determines the number of rows. If one of the valid time ranges is longer than the others, it will fill zeros at the end of the others.

**Sample Code:** The sample code will return a matrix with two columns, one for valid time range A and the second for B. The first column will be filled with zeroes at the end, so that it is the same length as the second column, which contains a longer valid time range. To get back all the epoch periods as separate columns, use GetEpocsExV.

MATLAB
```
filt = TT.SetFilterWithDescEx('Freq=2000')
waves = TT.ReadWavesOnTimeRangeV('Snip', 1)
```




**Related Calls:** GetEpocsExV, GetValidTimeRangesV

**Note:** Python users, see page 59.

# *Epochs and Filtering* *TTank X*

### CreateEpocIndexing

**Description:** A memory based epoch index must be created for the selected block before a client application can take advantage of high speed data indexing and filtering capabilities. After selecting a block for access, using SelectBlock, call CreateEpocIndexing to instruct TTankServer to build the epoch index. This call must be made each time a new block is selected.

A tilde (~) in front of the block name of the SelectBlock method will automatically generate an epoch index for that block.

**Prototype:** `Function CreateEpocIndexing() As Long`

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample indexes all the epoch events for the selected block.

```
TT.ConnectServer('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\Tanks\MyTank','R')
TT.SelectBlock('Block-45')
TT.CreateEpocIndexing
% Tilde Example
TT.ConnectServer('Local','Me')
TT.OpenTank('MyTank','R')
TT.SelectBlock('~Block-45')
```

**Related Calls:** SelectBlock

## GetEpocCode

**Description:** GetEpocCode can be used to build a list of epoch code strings currently in the memory index. The memory index must be built using CreateEpocIndexing. You must first call using an index of 0, to get the first epoch code, then increase the index until null is returned.

**Prototype:** `Function GetEpocCode(Index As Long) As String`

**Arguments:**

| | | |
|---|---|---|
| Long | *Index* | index number of the epoch code |

**Returns:**

String — epoch code string, such as SwpN or FREQ

**Related Calls:** CreateEpocIndexing

## GetEpocsV

**Description:** GetEpocsV returns epoch event information from a time region of the block. This variant form includes a list of four doubles that represent the values of the epoch, the start time, the stop time, and the filter status. The size of the variant in bytes is 32 times the number of epochs returned. MaxEpocs determines the maximum number of epochs to be returned. If the number of epochs is greater than the MaxEpocs only the maximum number will be returned.

**Prototype:** `Function GetEpocsV(TankCode As String, T1 As Double, T2 As Double, MaxEpocs As Long) As Variant`

**Arguments:**

| | | |
|---|---|---|
| String | *TankCode* | event name for the epoch<br><br>Note: Note: there is no error checking for valid TankCodes; incorrectly typed (TankCode is case sensitive) or nonexistent codes will return NaN or -1. |
| Double | *T1* | specifies the starting time in the block |

| | | |
|---|---|---|
| | | Only epochs with start times equal to or greater than T1 will be returned. Use 0 to get all epochs from the start of the block. |
| Double | *T2* | specifies the maximum time to return |
| | | Only epochs with a start time less than T2 will be returned. Use 0 to get all epochs in a block. |
| Long | *MaxEpocs* | maximum number of epochs to return |

Also uses the following Global parameters: RepectOffsetEpoc

**Returns:**

| | | |
|---|---|---|
| VARIANT | *row of doubles* | row of values with information about each epoch returned |
| | | The values are ordered like: [ Epoch Value ][ Start Time ][ Stop Time ][ Filter Selection ] ... next epoch. |
| | | For ONset strobe epochs the start times are returned. |
| | | For OFFset strobe epochs the stop times are returned. |
| | | For buddy epochs, the start times and stop times are returned. |

**Sample Code**

**Description:** Creates an index of Block-45 from MyTank and gets up to the first 1000 frequency epochs.

MATLAB

```
TT.ConnectServer('Local','Me')
TT.OpenTank(' C:\TDT\OpenEx\Tanks\MyTank','R')
TT.SelectBlock('Block-45')
TT.CreateEpocIndexing
MyEpocs = TT.GetEpocsV('Freq',0,0,1000)
```

**Related Calls:** GetEpocsExV, GetValidTimeRangesV, QryEpocAtV

## GetEpocsExV

**Description:** This call will return the valid epoch events that pass through any preceding filters. It will return a part of, or the entire epoch duration, depending on the mode argument.

**Prototype:**

```
Function GetEpocsExV(EpocName As String, Mode As Long) As Variant
```

**Arguments:**

| | | |
|---|---|---|
| String | *EpocName* | four letter epoch name |
| Long | *Mode* | specifies which part of the epoch to return: |
| | 0 | all epochs whose onsets occur within the filter epoch |
| | 1 | all epochs, occurring at least in part within the filter epochs |
| | 2 | only the parts of the epochs that occur within the filter epochs |

Also uses the following Global parameters: MaxReturn, RespectOffsetEpoc, T1, T2

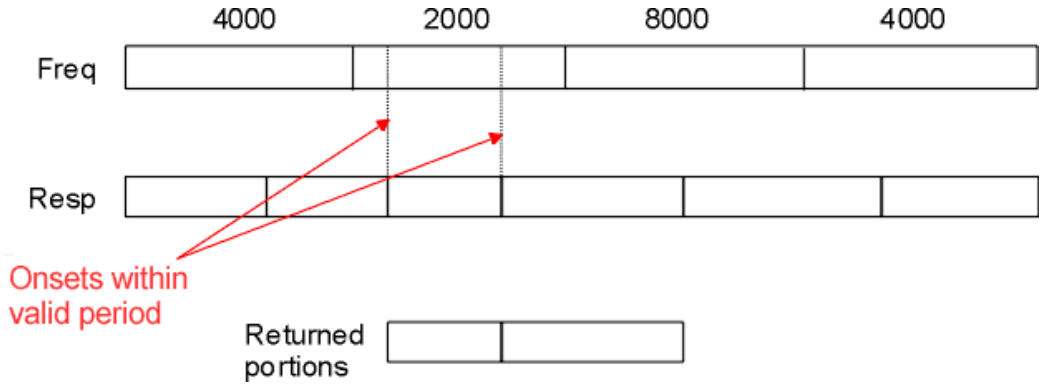
**Returns:**

Variant — Variant with each column representing a valid epoch. The first row of each column denotes the value of the epoch. The second and third rows contain the start and stop time of each valid epoch duration. Note that these start and stop times may vary based on the mode specified in the arguments.

**Sample Code**

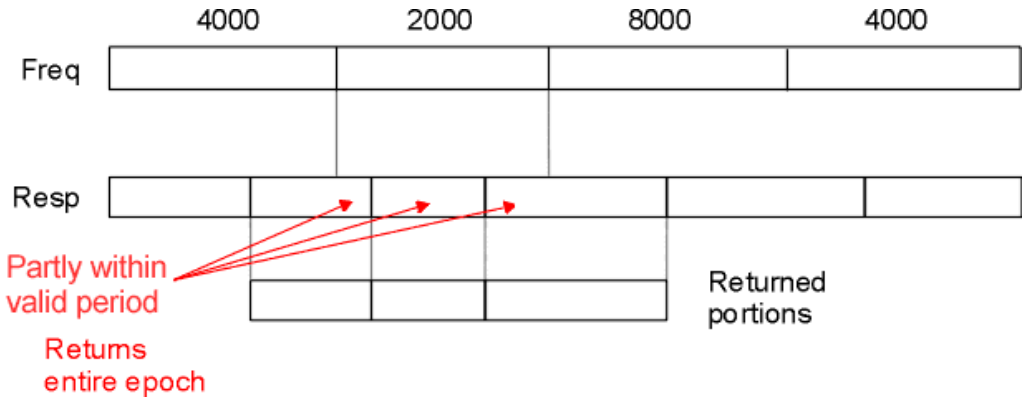
**Description:** This code sample sets a filter of Freq=2000 and returns only those epochs whose onsets occur within the valid epoch.

MATLAB
```
filt = TT.SetFilterWithDescEx('Freq=2000');
response = TT.GetEpocsExV('Resp',0);
```



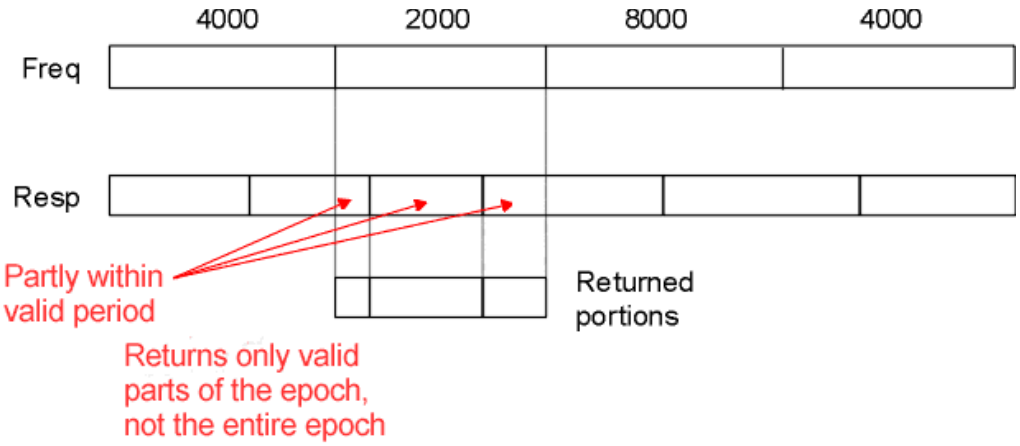

Again, a filter of Freq=200 is set. However, this time the mode is set to return only those parts of the epochs, which occur within the valid epochs.

MATLAB
```
filt = TT.SetFilterWithDescEx('Freq=2000');
response = TT.GetEpocsExV('Resp',1);
```




In this case, only those epochs, which occur at least in part within the valid epochs are returned.

MATLAB
```
filt = TT.SetFilterWithDescEx('Freq=2000');
response = TT.GetEpocsExV('Resp',2);
```




**Related Calls:** CreateEpocIndexing, GetValidTimeRangesV

## GetFilterTolerance

**Description:** Returns the tolerance of the filter. The tolerance is the margin of error allowed in evaluating the conditions of the filter. For example, if filter tolerance is set to 0.001, and the filter is specified as Freq=2000, any epoch with values between 1998 and 2002 will pass through the filter. The default value is 1e-7.

**Prototype:**
```
Function GetFilterTolerance() As Long
```

**Returns:** tolerance of the filter, or -1 in the absence of a tolerance, e.g. the tank has not been accessed

**Related Calls:** SetFilterTolerance

## GetValidTimeRangesV

**Description:** This call returns the valid time ranges based on the preceding filters. If no filters are specified, then it will return the entire duration of the block as a single valid time range.

**Prototype:**
```
Function GetValidTimeRangesV() As Variant
```

This method uses the following Global parameters: MaxReturn, RespectOffsetEpoc

**Returns:**

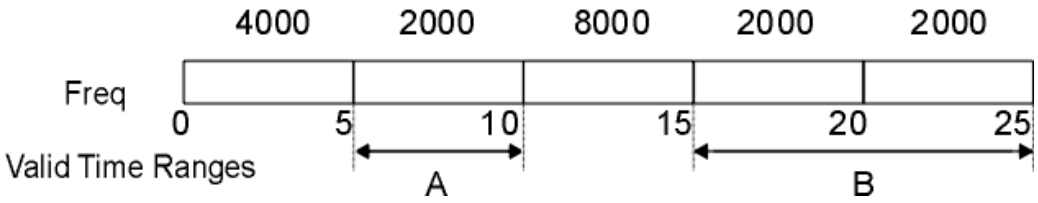
Variant — The time ranges are returned in the form of a variant with each column representing a single valid time range. There are always two rows: the first row containing the start time of the valid time range, and the second row containing its stop time.

**Sample Code:** This code sample will return a variant with two columns, each column containing the start and stop times for a valid time range.

```
filt = TT.SetFilterWithDescEx('Freq=2000')

tranges = TT.GetValidTimeRangesV
```

```
tranges =       5       15
               10       25
```




**Related Calls:** GetEpocsExV, ReadWavesOnTimeRangeV

## QryEpocAtV

**Description:** QryEpocAtV returns information about an epoch event that is active during a particular time point. One of four values is returned: the value of the epoch, the start of the epoch event, the end of the epoch event, or the filter status. CreateEpocIndexing must be called before QryEpocAtV.

**Prototype:**
```
Function QryEpocAtV(TankCode As String, rTime As Double, ReqItem As Long) As Variant
```

**Arguments:**

| | | |
|---|---|---|
| String | *TankCode* | four character epoch name<br>**Note:** TankCode is case sensitive. Incorrect values will result in the method returning NaN or a negative return value. |
| Double | *rTime* | requested time, time at which active epoch is to be found |
| Long | *ReqItem* | requested item type:<br>0 epoch value<br>1 start time of epoch<br>2 stop time of epoch<br>3 filter status (0 or 1) |

**Returns:** returns one of four items: value, start, stop, or filter status of epoch event (see ReqItem in the argument statement above)

**Sample Code:** Creates an index of Block-45 from MyTank and queries the value of epoch Freq at time 12.45 seconds.

```
TT.ConnectServer('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\MyTank','R')
TT.SelectBlock('~Block-45')
Epoch = TT.QryEpocAtV('Freq', 12.45, 0)
```

**Related Calls:** GetEpocsExV, GetValidTimeRangesV

## ResetFilters

**Description:** Resets the filters to no filtering. This function must be called whenever a new filtering criterion is to be invoked. For example, if you want all records concurrent with epoch FREQ = 4000 to be returned first and then all event records concurrent with FREQ > 8000 to be returned second, you must call ResetFilters in between the two accesses or all records with either of these criteria true will be returned on the second access.

Refer to the *Using Epochs as Filters* section, page 8, for more information and filtering examples.

**Prototype:** `Function ResetFilters()`

**Related Calls:** SetFilterWithDesc, SetFilterWithDescEx, SetFilterArray, SetEpocTimeFilterV

## SetEpocTimeFilterV

**Description:** SetEpocTimeFilterV sets a filter based on the specified offset and duration. The time filter is applied relative to the onset of the specified epoch. The functioning of this call is affected by the RespectOffsetEpoc global parameter. By default, it will respect the duration of buddy epochs. So, if the duration specified is more than the length of the buddy epoch, events that occurred after the offset of the buddy epoch will not pass the filter. To ignore the buddy epoch offset, set RespectOffsetEpoc to 0.

Each snippet can only be given one timestamp. If the duration window overlaps with multiple epochs such that a particular event passes multiple filters, the lowest timestamp that matches these filters will be returned for that event.

Refer to the *Using Epochs as Filters* section, page 8, for more information and filtering examples.

**Prototype:**
```
Function SetEpocTimeFilterV(EpocName As String,
Offset As Double, Duration As Double) As Long
```

**Arguments:**

| | |
|---|---|
| *EpocName* | four letter epoch name |
| *Offset* | time in seconds from the onset of the epoch to the onset of the filter (can be a negative value) |
| *Duration* | duration of the filter in seconds (0 specifies the entire duration of the epoch) |

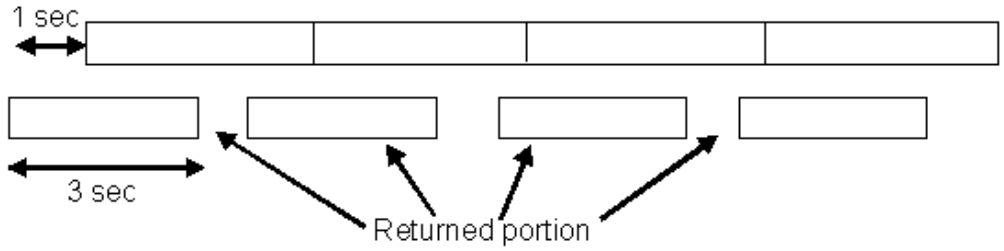
Also uses the following Global parameters: AutoRefEpoch, RespectOffsetEpoc

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample implements a filter that begins one second before the epoch onset and has a duration of three seconds. Note that in this case, zeroes will fill the one second interval before the start of the data.

```
Filt = TT.SetEpocTimeFilterV('Freq',-1, 3)
```

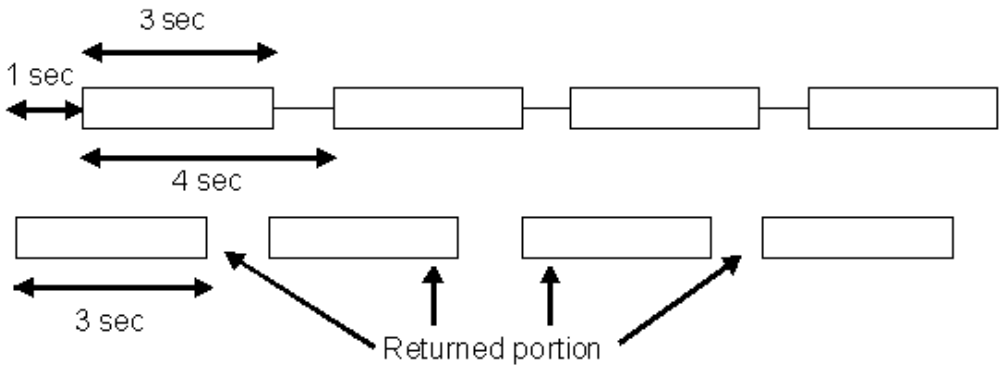
**Note:** Python users, see page 59.




```
Filt = TT.SetEpocTimeFilterV('Freq',-1, 4)
```

In this case, because the duration of the buddy epoch is three seconds, the filter passes only three of the four seconds specified. To include the entire four seconds, the RespectOffsetEpoc parameter must be set to 0. Also, note that four

seconds is the maximum that can be obtained, because that is the duration between the onset of adjacent epochs.




**Note:** To return the filtered data, a record retrieving call (such as ReadEventsV or ReadWavesV) must be used with the Options global parameter set to FILTERED.

**Related Calls:** SetRefEpocV, GetEpocsExV, GetValidTimeRangesV

## SetFilterWithDesc

**Description:** SetFilterWithDesc specifies epoch filters. It functions the same as SetFilter except that the filter is specified as a string rather than four longs. Refer to *SetFilter*, page 59, for more information.

A filter description contains three parts, (1) epoch name, (2) operation specification and (3) the value(s). The format looks like: [EpochName] [OperationSpec] [Value(s)].

EpochName -- The epoch name is specified as four chars, such as FREQ, SwpN, or Puff. There are three special keywords that will invoke special non-epoch driven filtering, they are: TIME, CHAN, and SORT. These three keywords allow for full filtering function on the time stamp, channel number, and sort code for event records.

OperationSpec -- Each of the operations enumerated in the SetFilter description has a corresponding text character specification. These characters are '=', '<>', '>=', '<=', '>', and '<'. The meaning of each is based on their use in standard mathematical equations. One exception is that a value range, or the 'include, between' function, is specified using the '=' character. For example, FREQ = 1000:8000 is used to specify all FREQs between 1000 and 8000. To specify the 'outside' or not between function use the '<>' characters in the form: FREQ <> 1000:8000.

Value(s) -- The values parameter is always a decimal number, such as 12.3 or 768. If two numbers are needed (for specifying a range) use a colon between them, for example 4:44.

Refer to the *Using Epochs as Filters* section, page 8, for more information and filtering examples.

**Prototype:**
```
Function SetFilterWithDesc(FiltDesc As String) As Long
```

**Arguments:** *FilterDesc* see description above

**Returns:** number of epoch blocks that met the filter condition

**Sample Code:** This code sample sets filters to select event records concurrent with |EyeX| < 1.0 and |EyeY| < 1.0

```
TT.SelectBlock('MyBlock-45')
TT.CreateEpocIndexing
TT.SetFilterWithDesc('EyeX = -1.0:1.0')
TT.SetFilterWithDesc('EyeY = -1.0:1.0')
```

Related Calls: SetFilterWithDescEx, SetFilterArray, SetEpocTimeFilterV, SetFilter

## SetFilterWithDescEx

**Description:** Similar to SetFilterWithDesc, sets multiple filters in a single string. If multiple calls are made, then the last call will overwrite the previous filters. Filters can be logically chained together using ANDs and/or ORs, up to 5000 characters. Use '!=' instead of '<>' for a not equal comparison.

**Prototype:**

```
Function SetFilterWithDescEx (Conditions As String) As Long
```

**Arguments:**

| | |
|---|---|
| *Conditions* | string defining the filter |

**Sample Code:** This code sample uses a single string to set a filter for stimulus frequency, acquisition channel number, and stimulus level.

```
Filter = TT.SetFilterWithDescEx('Freq=4000 AND CHAN<5 AND Levl=2')
```

## SetFilterArray

**Description:** Assigns a filter along one of three dimensions, and gives it an ID along that dimension. The dimensions are usually assigned the number 0, 1, or 2. They are often also referred to as X, Y, and Z dimensions.

The IDs along these dimensions are called X-ID, Y-ID, and Z-ID respectively. An event that meets the criteria set by one of these filters is assigned the appropriate filter ID. Multiple instances of this call will generate an array of filters, which can then be used to sort and display data.

This method includes an exclusive flag, allowing the user to determine whether events can be assigned IDs for multiple filters. If the exclusive flag is enabled (1), only the lowest ID number will be used for each event.

Refer to the *Using Epochs as Filters* section, page 8, for more information and filtering examples.

**Prototype:**

```
Function SetFilterArray(Dimension As Long, ID As Long, Conditions As String, Exclusive As Boolean) As Long
```

**Arguments:**

| | |
|---|---|
| *Dimension* | dimension for which the filter is set; specified as 0, 1, or 2 for X, Y, or Z respectively |
| *ID* | filter ID, specified as any number from 1 to 256 |
| *Conditions* | string defining the filter, for example: 'Freq=4000 and Level=2' |

**Note:** Boolean operators ‘and’ and ‘or’ are allowed within the filter string.

*Exclusive* flag specifying whether an event which meets the criteria for multiple filters is assigned to more than one ID

1 = exclusive (first ID)

0 = not exclusive (multiple ID’s)

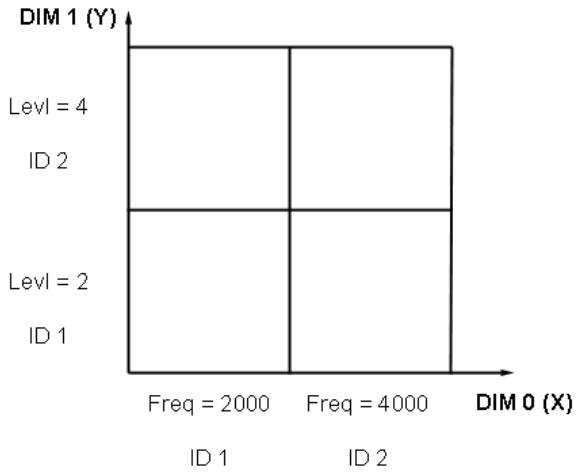
Also uses the following global parameters: RespectOffsetEpoc

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This sample code sets up an array of non-exclusive filters along the X and Y dimensions. Note that X, and Y dimensions are denoted by 0 and 1 respectively. After events have passed through the filter array, their X and Y dimension filter ID properties will be set according to the filter arguments.

```
a = TT.SetFilterArray(0,1,'Freq=2000',0)
b = TT.SetFilterArray(0,2,'Freq=4000',0)
c = TT.SetFilterArray(1,1,'Levl=2',0)
d = TT.SetFilterArray(1,2,'Levl=4',0)
```

When data is read, an event will be returned only if it passes filters specifying at least one of the cells of a grid (as pictured below). In other words, if an event passes the filters Freq=4000 and Levl=2, then it will be assigned the X ID 2, and Y ID 1 and will be returned in the corresponding cell. If an event passes Levl=4, but no other filter, then it will not be returned. If an event fits into multiple cells of the filter array, then it will be read multiple times, each time with a different set of X and Y IDs. Note that this is not possible when the exclusive flag is set to 1. In that case, the event would be returned only once, with the lowest set of IDs possible.




**Related Calls:** SetFilterWithDescEx, SetEpocTimeFilterV, GetValidTimeRangesV

## SetFilterTolerance

**Description:** Sets the tolerance of the filter. The tolerance is the margin of error allowed in evaluating the conditions of the filter. For example, if filter tolerance is set to

0.001, and the filter is specified as Freq=2000, any epoch with values between 1998 and 2002 will pass through the filter. The default value is 1e-7.

**Prototype:**
```
Function SetFilterTolerance(Tolerance As Double) As Long
```

**Arguments:** *Tolerance* tolerance of filters

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample sets the filter tolerance to 0.00001.

```
tolerance = TT.SetFilterTolerance(0.00001)
```

**Related Calls:** GetFilterTolerance

## SetRefEpocV

**Description:** This function will set a reference epoch, such that all events subsequently returned will have time stamps relative to the onset of the specified epoch. This is particularly useful for plotting histograms. This method can be called by the user, but it is also called automatically by the SetEpocTimeFilterV method.

When using SetEpocTimeFilterV, the AutoRefEpoch global parameter (enabled by default) will cause the epoch set by SetRefEpocV to be overwritten by the epoch set by SetEpocTimeFilterV. To prevent this, set AutoRefEpoch to 0. Also note that, if a negative offset has been specified in the SetEpocTimeFilterV arguments, it is possible to get a negative time stamp value.

These calls are typically used before ReadEventsV and do not affect the results of the GetValidTimeRangesV or GetEpocsExV calls.

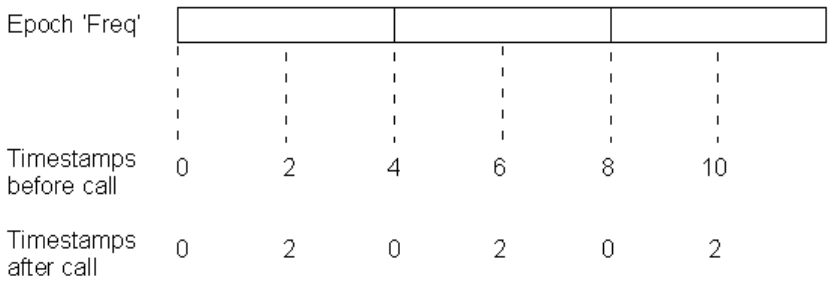
**Prototype:**
```
Function SetRefEpocV(EpocName As String) As Long
```

**Arguments:** *EpocName* four letter name of an epoch event

**Returns:** 0 (fails), 1 (succeeds)

Sample Code: This code sample specifies the Freq epoch as the reference epoch. The time stamps of all events returned after this call will be relative to this epoch, as seen in the diagram below.

```
a = TT.SetRefEpocV('Freq')
```




Related Calls: SetEpocTimeFilterV

**Note:** Python users, see page 59.

# Annotation Methods TTank X

The annotation methods are used to manipulate the note list in each block, and for setting epoch notes programmatically. To read the note timestamps for each epoch, see ParseEvInfoV page 30 (option 9 is used to access note indices). To filter the data based on note values, see SetFilterWithDescEx page 40.

## AppendNote

**Description:** Adds a new text note and returns the note index of the new note.

**Prototype:**
```
Function AppendNote(BSTR noteText) As long
```

**Arguments:** *noteText* text string for newly created note

**Returns:** index of newly created note

**Sample Code:**
```
noteIndex = TT.AppendNote('new note text')
```

## GetNote

**Description:** Returns the note text string associated with the specified note index.

**Prototype:**
```
Function GetNote(long noteIndex) As BSTR
```

**Arguments:** *noteIndex* note index of desired note string

**Returns:** note string, or ‘’ if no note was set

**Sample Code:**
```
noteOneText = TT.GetNote(1)
```

## ReplaceNote

**Description:** Replaces note text at the specified note index.

**Prototype:**
```
Function ReplaceNote(long noteIndex, BSTR
newNoteString) As long
```

**Arguments:** *noteIndex* note index

*newNoteString* new note string

**Returns:** 0 (fails), -1 (succeeds)

**Sample Code:**
```
TT.ReplaceNote(noteOneIndex, 'new note 1')
```

## SetNoteIndex

**Description:** Adds a note into an epoc store at the epoch data point closest to the specified timestamp, towards 0. Returns the actual timestamp for the new note.

**Note:** CreateEpocIndexing must be called after the block is opened for this to work.

**Prototype:** `Function SetNoteIndex(BSTR storeName, double timestamp, long noteIndex) As double`

**Arguments:**

| | |
|---|---|
| *storeName* | store name |
| *timestamp* | time stamp of desired note location |
| *noteIndex* | note index |

**Returns:** actual not location if successful, -1 on failure

**Sample Code:**

```
TT.CreateEpocIndexing;
TT.SetNoteIndex('Tick', 10, noteIndex);
```

# *Sorting Methods* *TTank X*

The sorting methods are used to create a sortID that is stored in the tank, and specify which sortID to use when filtering using the 'SortCode' parameter. Multiple sortIDs can be saved into the tank. The default is TankSort. OpenSorter can create sortIDs and users writing custom analysis routines can save their sort codes into the tank using the SaveSortCodes method.

The format necessary for the methods is a row vector which, when passed to the SaveSortCodes method, will associate each index and its paired neighbor as a set of an event index and sort code.

For example, the vector [5, 2, 6, 1, 7, 2, 8, 1] will have four event indexes 5,6,7,8 and four sort codes 2,1,2,1. When passed to SaveSortCodes the method will classify these indexes with their paired sort codes and save them to a user defined sortID.

## GetEvTsqIdx

**Description:** Returns an index array (in long) which contains the events distribution in the *.tsq file. This method should be called immediately after ReadEventsV. The options parameter in the ReadEventsV call MUST be set to "IDXPSQ" in order for the GetEvTsqIdx method to return the correct index array.

**Prototype:** `Function GetEvTsqIdx() As Variant`

**Returns:** array of indicies

**Sample Code:**

```
N = TT.ReadEventsV(10000,'Snip',1,0,0,0,'IDXPSQ')
```

IndexArray = TT.GetEvTsqIdx

## SaveSortCodes

**Description:** This method saves all sorting information to a user defined sortID.

**Prototype:** `Function SaveSortCodes(BSTR SortName, BSTR SnipName, long IdxChan, BSTR SortCondition, VARIANT SortCodeArray) As Long`

**Arguments:**

| | |
|---|---|
| *SortName* | name of the newly created sortID |
| *SnipName* | name of the event in four character string form |
| *IdxChan target* | channel that the sortID saves the events and sort codes to |

| | | |
|---|---|---|
| | *SortCondition* | user-defined string (such as the algorithm used) that can later be retrieved using the method GetSortCondition or viewed in OpenSorter |
| | *SortCodeArray* | vector that contains each event index and its paired sortcode as described above; must be of type int32 |

**Returns:** 0 (fails), 1 (succeeds)

## GetSortCondition

**Description:** Returns the defined sort condition associated with the specified sortID.

**Prototype:**
```
Function GetSortCondition(BSTR SortName, BSTR SnipName, long IdxChan) As BSTR
```

| | | |
|---|---|---|
| **Arguments:** | *SortName* | name of sortID to retrieve sort condition for |
| | *SnipName* | name of the event in four character string form |
| | *IdxChan* | target channel of desired sort condition string |
| | *SortCondition* | user-defined string (such as the algorithm used) that can later be retrieved using the method GetSortCondition or viewed in OpenSorter |
| | *SortCodeArray* | vector that contains each event index and its paired sortcode as described above |

**Returns:** sort condition string, or ‘’ if no sort conditions were set

## DeleteSortCode

**Description:** This method is used to delete a single channel’s sort codes from the desired sortID. This is equivalent to right-clicking a sorted channel in OpenSorter, clicking Delete and removing the check from that channel. Note that only a single channel may be deleted in one call to this method and no sortID can be deleted. You may use the GetSortChanMap method to verify that the target channel’s sort codes were indeed removed.

**Prototype:**
```
Function DeleteSortCode(BSTR SortName, BSTR SnipName, long IdxChan) As Long
```

| | | |
|---|---|---|
| **Arguments:** | *SortName* | name of sortID to retrieve sort condition for |
| | *SnipName* | name of the event in four character string form |
| | *IdxChan* | target channel of desired sort condition string |

**Returns:** 0 (fails), 1 (succeeds)

## GetSortChanMap

**Description:** This method returns a 1024 point vector which indicates which channel(s) of the specified sortID and event name are sorted (1) or unsorted (0). Note that this vector matrix begins its index at channel 0 which does not exist. You may format the returned vector matrix to exclude the first entry or simply just ignore it. If this method is called after a DeleteSortCode call to a specified channel, the same sortID will return a 0 for that channel’s index (again remember that the first index is disregarded).

**Prototype:**
```
Function GetSortChanMap(BSTR SortName, BSTR SnipName) As Variant
```

**Arguments:** *SortName* name of sortID to retrieve sort condition for

*SnipName* name of the event in four character string form

**Returns:** 1024 column array indicating which channels are sorted

## SetUseSortName

**Description:** SetUseSortName sets the sort file used for OpenDeveloper calls that retrieve events like ReadEventsV. The sort code file will be set if the event name matches and the desired channel has a sort named sortID. If this function is not used , the event name does not match, or of the sort name sortID is not present, this function has no effect and the original sort file from the online tank sort is used.

**Prototype:**
```
Function SetUseSortName(sortID As String) As Long
```

**Arguments:** *sortID* sort ID given in OpenSorter (the original online tank sort is always named TankSort)

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This sample reads Snip events of channel = 1and sort code = 1 from the sort set saved as “Sort1”.

```
SetSort1 = TT.SetUseSortName('Sort1')

Filter = TT.SetFilterWithDescEx('sort=1')

AllSort1 =
TT.ReadEventsV(1000,'Snip',1,0,0.0,0.0,'FILTERED')
```

The following example assumes a tank with online sort code named TankSort and sorts generated in OpenSorter named Sort1, Sort2 and Sort3.

For a hypothetical data set, the TankSort sort file has 100 Snip events for each sort code; 1, 2 and 3. Since a different sorting criterion was used for Sort1, this sort file has 50 Snip events for sort code 1 and 2 and 200 events for sort code 3.

If ReadEventsV (Snip, sort code 1) is called before applying SetUseSortName, 100 events will be returned. If SetUseSortName (Sort1) is called before calling ReadEventsV (Snip, sort code 1), the sort code file for Sort1 will be applied and 50 events will be returned.

If using a SortCode generated by OpenSorter, see *Known Anomalies*, page 79.

# *Information Access* *TTank X*

## CurBlockMemo

**Description:** CurBlockMemo returns the memo associated with the currently selected block. If no memo was specified, a null string is returned.

**Prototype:** `Function CurBlockMemo() As String`

**Returns:** returns memo in string or null string if no block is currently selected or if no memo was specified when the block was created

## CurBlockName

**Description:** CurBlockName returns the name of the currently selected block.

**Prototype:** `Function CurBlockName() As String`

**Returns:** block name or null string if no block is currently selected

## CurBlockNotes

**Description:** CurBlockNotes returns notes associated with the currently selected block. The notes for each store in the block include: the store name, number of points, sample frequency, number of channels and other information.

**Note:** This method is not supported by Legacy Tanks.

**Prototype:** `Function CurBlockNotes() As String`

Returns:

| | | |
|---|---|---|
| String | *StoreName* | name of each store |
| | *Enabled* | enable status of store |
| | *CircType* | circuit type |
| | *NumChan* | number of channels |
| | *StrobeMode* | onset/offset strobe |
| | *StrobeBuddy* | buddy epoch if applicable |
| | *SecTag* | secondary tag information if applicable |
| | *NumPoints* | number of points |
| | *DataFormat* | data format (0: float, 1: 32-bit integer, 2: short, 3: byte) |
| | *SampleFreq* | sample frequency |

## CurBlockStartTime

**Description:** CurBlockStartTime returns the start time of the selected block in seconds. The returned value is the elapsed time in seconds from 12:00 AM January 1st, 1970 to the start of the block. Pass the result through FancyTime to convert the result into a date/time string.

**Prototype:** `Function CurBlockStartTime() As Double`

**Returns:** block start time in seconds

**Sample Code:** This code sample returns the current block start time and then passes the result through FancyTime to return a more readable value.

```
start = TT.CurBlockStartTime
formstart = TT.FancyTime(start ,'D/O/Y H:M:S.U')
```

**Related Calls:** CurBlockStopTime, FancyTime

## CurBlockStopTime

**Description:** CurBlockStopTime returns the stop time of the selected block in seconds. The returned value is the elapsed time in seconds from 12:00 AM January 1st, 1970 to the end of the block. Pass the result through FancyTime to convert the result into a date/time string.

**Prototype:** `Function CurBlockStopTime() As Double`

**Returns:** block stop time in seconds

**Sample Code:** This code sample returns the current block stop time and then passes the result through FancyTime to return a more readable value.

```
stop = TT.CurBlockStopTime
formstop = TT.FancyTime(stop ,'D/O/Y H:M:S.U')
```

**Related Calls:** CurBlockStartTime, FancyTime

## FancyTime

**Description:** FancyTime converts a time in double format to string format based on the user's specifications. The input argument Time is assumed to be the total elapsed time from 12:00 AM January 1st, 1970 up to the event of interest.

**Prototype:** `Function FancyTime(Time As Double, Format As String) As String`

**Arguments:** *Time* tank time in double format

*Format* format for returned value

| Year | Month | Day | Hours | Minutes | Seconds | frac/Sec | D of W |
|---|---|---|---|---|---|---|---|
| Y | O | D | H | M | S | U | W |

**Returns:** time in string format with user specified formatting

**Sample Code**: This code sample returns the time in the format 'Date, Time, Day of Week' e.g. '01/Dec/2010 10:04:23.63 Fri' Using characters such as '/', ':' '.' and ' ' further delineate the string.

```
start = TT.CurBlockStartTime
formstart = TT.FancyTime(start ,'D/O/Y H:M:S.U W')
```

## GetCodeSpecs

**Description:** GetCodeSpecs (get code specifications) queries the block and returns the event record specifications for the event code specified. If successful the following properties within the TTankX control are assigned values:

EvChannel -- channel for first record found

EvDataSize -- size of record in 32 bit chunks

EvDForm -- waveform data format code (see *DFromToString* for more information)

EvSampFreq -- sampling frequency of waveform data

EvType -- record type code (see *EvTypeToString* for more information)

**Prototype:** `Function GetCodeSpecs(EvCode As Long) As Long`

**Arguments:** *EvCode* event code in long format

**Returns:** 0 (fails), 1 (succeeds)

**Related Calls:** ParseEvInfoV

## GetEnumServer

**Description:** GetEnumServer returns servers that are enumerated (registered) on your computer. 0 is returned when no more servers are found. Use this function to build a list of enumerated servers on your computer. To get the first server (typically 'Local') use an index of 0. Then increase the index until null is returned.

**Prototype:** `Function GetEnumServer(Index As Long) As String`

**Arguments:** *Index* server index (zero based)

**Returns:** name of server at specified index, or null string if no server at that index

**Sample Code:** This code sample gets the server name at index zero.

```
servname = TT.GetEnumServer(0)
```

**Related Calls:** GetEnumTanks, QueryBlockName, GetHotBlock

## GetEnumTank

**Description:** GetEnumTank is used to build a list of tanks enumerated (registered) on the connected server. To get the first tank use an index of 0. Then call with increasing indexes until null is returned.

**Prototype:** `Function GetEnumTank(Index As Long) As String`

**Arguments:** *Index* position in the registry (zero based)

**Returns:** name of tank at specified index, or null string if no tank at that index

**Sample Code:** This code sample gets the tank at index 0 of the registry.

```
tankname = T.GetEnumTank(0)
```

**Related Calls:** GetEnumServer, QueryBlockName, GetHotBlock

## QueryBlockName

**Description:** QueryBlockName returns the block name for a given block index. This function can be used to build a list of blocks within a tank. The first call must be made with BlockNumber of 0, then the index can be increased until null is returned.

**Prototype:** `Function QueryBlockName(BlockNumber As Long) As String`

**Arguments:** *BlockNumber* block number (zero based)

**Returns:** name of block at specified index, or null string if no block at that index

**Sample Code:** This code sample returns the name of the 45th block and then selects it for access. If there are less than 45 blocks in the tank a null string is returned.

```
block = TT.QueryBlockName(45)
TT.SelectBlock(block)
```

**Related Calls:** GetEnumTank, GetEnumServer, GetHotBlock

## GetError

**Description:** GetError retrieves any pending error string or null if there is no error pending.

**Prototype:**
```
Function GetError() As String
```

**Returns:** error message string or null

**Sample Code:** This code checks for pending error

```
if TT.OpenTank('C:\TDT\OpenEx\Tanks\MyTank','R')==0
  errmess = TT.GetError
end
```

## GetEventCodes

**Description:** Returns a list of valid long integer event codes for the selected block that match the specified event type.

**Prototype:**
```
Function GetEventCodes(EvType As Long) As Variant
```

**Arguments:** *EventType* event type code or 0 for all event types.

**Returns:** lists of all the codes

**Sample Code:** This code displays all stores that match the format of the 'Tick' data store

```
N = TT.ReadEventsSimple('Tick');
evtype = TT.ParseEvInfoV(0, 0, 2);
evcodes = TT.GetEventCodes(evtype);
for i = 1:length(evcodes)
  TT.CodeToString(evcodes(i))
end
```

## GetGlobalStringV

**Description:** This call will return the current string value of the specified global parameter. Note that this call supports only string parameters.

**Prototype:**
```
Function GetGlobalStringV(GlobalName As String) As
String
```

**Arguments:** *GlobalName* global parameter name

**Returns:** current value of the specified global parameter

**Sample Code:** This code sample returns the value of the global parameter, Options.

```
TT.GetGlobalStringV('Options')
```

**Related Calls:** GetGlobalV, SetGlobalV, SetGlobalStringV, SetGlobals, ResetGlobals

**Note:** Python users, see page 59.

## GetGlobalV

**Description:** This call will return the current value of the specified global parameter.

**Prototype:**
```
Function GetGlobalV(GlobalName As String) As Long
```

**Arguments:** *GlobalName* global parameter name

**Returns:** current value of the specified global parameter

**Sample Code:** This code sample returns the current value for the global parameters, Channel and T2.

```
TT.GetGlobalV('Channel')
TT.GetGlobalV('T2')
```

**Related Calls:** SetGlobalV, SetGlobalStringV, SetGlobals, ResetGlobals

**Note:** Python users, see page 59.

## GetHotBlock

**Description:** Returns the block that is being recorded into the opened tank. If no block is open for recording a null string is returned.

**Prototype:**
```
Function GetHotBlock() As String
```

**Returns:** name of block being recorded into

**Sample Code:** This code sample connects to the local server, opens a tank for reading and returns the block name of the block currently recorded to (if any).

```
TT.ConnectServer('Local','Me')
TT.OpenTank('C:\TDT\OpenEx\Tanks\DEMOTANK2','R')
recblock = TT.GetHotBlock
```

**Related Calls:** GetEnumTanks, GetEnumServer, QueryBlockName

## GetSortName

**Description:** GetSortName retrieves the sort IDs present for the given event name in the currently selected block. The sort IDs are returned in alphabetical order as the input argument idxSortID is incremented. If no sort IDs are present, an empty string is returned at index zero.

**Prototype:**
```
Function GetSortName(eventName As String, idxSortID
As Long) As String
```

**Arguments:** *eventName* event name

*idxSortID* sort ID

**Returns:** sortID for the given index and event name

**Sample Code:** This code sample displays each sort ID for the Snip event in the currently selected block.

```
idx = 0;
```

```
sortid = 'temp';
while  ~isempty(sortid)
  sortid = TT.GetSortName('Snip',idx)
  idx = idx+1;
end
```

## GetStatus

**Description:** Used to obtain state information about an open tank.

**Prototype:** `Function GetStatus(StatCode As Long) As Long`

**Arguments:** *StatCode* status code for item to be retrieved

| Defined Variable | Function | Integer |
|---|---|---|
| STAT_TANKSTATE | tank state (R,W,M,C) | 0 |
| STAT_CACHEUSAGE | percentage of cache in use | 1 |
| STAT_CACHEDEPTH | amount of memory allocated in cache | 2 |
| STAT_CACHECOLLIDE | number of collisions in the cache | 3 |
| STAT_ORDERERROR | number of ordering errors | 4 |
| STAT_EVRATE | event rate (number of events stored to tank per second) | 5 |
| STAT_DATARATE | data rate (number of bytes stored to tank per second) | 6 |

**Returns:** requested value or -1 if operation failed

**Sample Code:** This code sample returns the event rate for the opened tank.

```
TT.SelectBlock('Block-1')
evrate = TT.GetStatus(5)
```

## GetTankItem

**Description:** GetTankItem returns the path or the tank version for the tank provided in TankName. This function is only valid with enumerated or registered tanks.

**Prototype:** `Function GetTankItem(TankName As String, ItemCode As String) As String`

**Arguments:** *TankName* name of the enumerated tank

*ItemCode* ‘PT’ returns path to tank

‘VERSION’ returns the version of the tank or a null string if the tank does not exist

**Returns:** *ItemCode* Returns

‘PT’ Path to the given tank

‘VERSION’ ‘20’ new format tank

‘10’ legacy tank

‘’ tank does not exist

**Sample Code:** This code sample returns the path to the given registered tank.

```
path = TT.GetTankItem('DemoTank2','PT')
```

# *Misc Utilities* *TTank X*

## AddTank

**Description:** AddTank creates a new data tank.

**Prototype:**
```
Function AddTank(TankName As String, FilePath As String) As Long
```

**Arguments:**

| | |
|---|---|
| *TankName* | name of the new tank |
| *FilePath* | path to the new tank location<br>**Note:** prefix the path with 'REGISTER@' in order to register the tank at that path |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:**
```
% create tank DEMOTANK3 without registering it
TT.AddTank('DEMOTANK3','C:\TDT\OpenEx\Tanks')
% or create and register DEMOTANK3
TT.AddTank('DEMOTANK3','REGISTER@C:\TDT\OpenEx\Tanks')
```

## StringToEvCode

**Description:** StringToEvCode converts a four character string to its corresponding event code in long integer format. This call is used to obtain epoch codes required for SetFilter. This is the complement of CodeToString.

**Prototype:**
```
Function StringToEvCode(EvCode As String) As Long
```

**Arguments:**

| | |
|---|---|
| *EvCode* | event code in four character string format |

**Returns:** the event code in long value format

**Related Calls:** CodeToString

## CodeToString

**Description:** CodeToString converts a long integer event code to a four character string. This is the complement of StringToEvCode.

**Prototype:**
```
Function CodeToString(EvCode As Long) As String
```

**Arguments:**

| | |
|---|---|
| *EvCode* | event code in long format |

**Returns:** A four character string for the event code

**Related Calls:** StringToEvCode

## EvTypeToString

**Description:** EvTypeToString returns a string description for event type codes. An event type is the type of event, such as snippet, strobe, or streamed data.

**Prototype:**
```
Function EvTypeToString(evTypeCode As Long) As String
```

**Arguments:** *evTypeCode* Event code stored in event header

**Returns:** A string description for the data format (below)

| Event Type | Input Hex | Returns |
|---|---|---|
| Unknown | 0x0000 | "Unknown" |
| Strobe ON | 0x0101 | "Strobe+" |
| Strobe OFF | 0x0102 | "Strobe-" |
| Scalar | 0x0201 | "Scalar" |
| Stream | 0x8101 | "Stream" |
| Snip | 0x8201 | "Snip" |
| Marker | 0x8801 | "Mark" |
| has associated waveform data | 0x8000 | "HasData" |

**Sample Code:** This code displays the string description of the data store ‘eNeu’

```
N = TT.ReadEventsSimple('eNeu');
evtype = TT.ParseEvInfoV(0, 0, 2);
TT.EvTypeToString(evtype)
```

## DFromToString

**Description:** Converts a data format code to a descriptive string.

**Prototype:** `Function DFromToString(DFormCode As Long) As String`

**Arguments:** *DFormCode* Data format code stored in event header

| Data Format | Input | Returns |
|---|---|---|
| Float | 0 | "Float" |
| Long | 1 | "Long" |
| Short | 2 | "Short" |
| Byte | 3 | "Byte" |
| Double | 4 | "Double" |

**Returns:** a string description for the data format (see above)

**Sample Code:** This code displays the data format of the data store ‘eNeu’

```
N = TT.ReadEventsSimple('eNeu');
d = TT.ParseEvInfoV(0, 0, 8);
TT.DFormToString(d)
```

## ResetGlobals

**Description:** This call will reset all the global parameters to their default values. See list of default values of global parameters.

**Prototype:** `Function ResetGlobals()`

**Related Calls:** SetGlobals, SetGlobalStringV , GetGlobalV, SetGlobalV, GetGlobalStringV, SetGlobals

## SetGlobalV

**Description:** This call will set a global parameter to the value specified. See list of global parameters.

**Prototype:** `Function SetGlobalV(GlobalName As String, GlobalValue As Long) As Long`

**Arguments:** *GlobalName* global parameter name

*GlobalValue* desired value of the global parameter

**Returns:** 0 (fails), 1 (succeeds)

+ Tip ... If 0 is returned, the name of the global parameter might have been entered incorrectly.

**Sample Code:** This code sample sets new values for the global parameters, Channel and T2, then returns the number of events.

```
TT.SetGlobalV('Channel', 1)
TT.SetGlobalV('T2', 10)
events = TT.ReadEventsSimple('Snip')
```

**Related Calls:** GetGlobalV, SetGlobalStringV, SetGlobals, ResetGlobals

**Note:** Python users, see page 59.

## SetGlobalStringV

**Description:** This call will set the string value of the specified string global parameter. Note that this call supports only string parameters ‘FillItem’ and ‘Options’.

**Prototype:** `Function SetGlobalStringV(GlobalName As String, GlobalValue As String) As Long`

**Arguments:** *GlobalName* global parameter name

*GlobalValue* desired value of the global parameter

**Returns:** 0 (fails), 1 (succeeds)

If 0 is returned, the name of the global parameter might have been entered incorrectly.

**Sample Code:** This code sample sets the global parameter, Options, to FILTERED.

```
TT.SetGlobalStringV('Options','FILTERED')
```

**Related Calls:** GetGlobalV, SetGlobalV, GetGlobalStringV, SetGlobals, ResetGlobals

**Note:** Python users, see page 59.

## SetGlobals

**Description:** This call allows the user to set multiple global parameters of different types in a single call. Each global parameter specified is separated by a semicolon and uses an equal sign to assign its desired value.

**Prototype:** `Function SetGlobals(Settings As String) As Long`

**Arguments:** *Settings* multiple parameter settings specified as a string

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:**
```
TT.SetGlobals('Options=FILTERED; Channel=1; T2=10');
```

**Related Calls:** ResetGlobals, SetGlobalStringV, GetGlobalV, SetGlobalV, GetGlobalStringV, ResetGlobals

**Note:** Python users, see page 59.

# C++ Methods

The 'V' methods that accept string inputs have counterparts that accept integers in place of those strings. These methods are used with the C++ programming language.

## ReadEvents

**Description:** Same as ReadEventsV except the EventCode and Options parameters are specified as longs. See *ReadEventsV*, pg 28, for more information.

**C Prototype:**
```
long ReadEvents(long MaxRet, long TankCode, long Channel, long SortCode, double T1, double T2, long Options);
```

**Arguments:**

*TankCode* — name of event in long format

*Options* — See *Options* in the *Global Parameters* section, page24, for more information. The table below converts the Options string to its corresponding hex value for this function. Options can be combined by summing their integer values together

| Options String | ReadEvents Options Input Hex |
|---|---|
| "ALL" | 0x0000 |
| "NEW" | 0x0001 |
| "SAME" | 0x0002 |
| "JUSTTIMES" | 0x0100 |
| "DOUBLES" | 0x0200 |
| "NODATA" | 0x0400 |
| "FILTERED" | 0x1000 |
| "ORDERED" | 0x2000 |

## ParseEv

**Description:** ParseEv retrieves waveform and event information about one record and returns the index for the next record.

**C Prototype:**
```
long ParseEv(long RecIndex, double* TimeStamp, long* Channel, long* SortCode, long* Npts, float* pData);
```

**Arguments:**

*RecIndex* — index of record to retrieve (zero based)

*TimeStamp* — pointer for time stamp, 0 to note return timestamp

*Channel* — pointer for channel number, 0 to not return channel

*SortCode* — pointer for sort code value, 0 to not return sort code

If using a SortCode generated by OpenSorter, see *Known Anomalies*, page 79.

| | | |
|---|---|---|
| | *Npts* | pointer for the number of points in the pData array, 0 to not return data |
| | *pData* | pointer to a memory buffer to store the raw waveform |

**Returns:** next index value, 0 if no more data, -1 if call failed

**Related Calls:** ReadEvents , ReadEventsV

## QryEpocAt

**Description:** QryEpocAt works the same as QryEpocAtV except that is requires TankCode is specified as a long and the requested item is returned as a long.

**C Prototype:**
```
long QryEpocAt(long TankCode, double rTime, long ReqItem, double* RetVal);
```

| | | |
|---|---|---|
| **Arguments:** | *TankCode* | epoch event, a four byte number |
| | *rTime* | requested time, time at which active epoch is to be found |
| | *ReqItem* | requested item type |
| | *RetVal* | pointer to the location that stores the returned value |

**Returns:** 0 (fails), 1 (succeeds)

**Related Calls:** CreateEpocIndexing

## SetEpocTimeFilter

**Description:** SetEpocTimeFilter is identical to SetEpocTimeFilterV but requires the epoc event code (long integer) as input instead of the epoc name as string.

**C Prototype:**
```
long SetEpocTimeFilter(long EpocCode, double Offset, double Dur);
```

| | | |
|---|---|---|
| **Arguments:** | *EpocCode* | event code of an epoch event |

## SetRefEpoc

**Description:** This call will set a reference epoch, such that all events subsequently returned will have time stamps relative to the onset of the specified epoch. This is particularly useful for plotting histograms. This method can be called by the user, but it is also called automatically by the SetEpocTimeFilter method.

When using SetEpocTimeFilter, the AutoRefEpoch global parameter (enabled by default) will cause the epoch by SetRefEpoc to be overwritten by the epoch set by SetEpocTimeFilter. To prevent this, set AutoRefEpoch to 0. Also note that, if a negative offset has been specified in the SetEpocTimeFilter arguments, it is possible to get a negative time stamp value.

These calls are typically used before ReadEvents and do not affect the results of the GetValidTimeRanges or GetEpocsEx calls.

**C Prototype:**
```
long SetRefEpoc(long EpocCode);
```

| | | |
|---|---|---|
| **Arguments:** | *EpocCode* | event code of an epoch event |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample sets a reference epoch using the epoch's numeric code.

```
a = TT.SetRefEpoc(4367)
```

**Related Calls:** SetEpocTimeFilter

### SetFilter

**Description:** SetFilter functions the same as SetFilterWithDesc except that the filter is specified numerically rather than with a string. Input values are used to set boundaries for filtering out epoch events.

Multiple calls to SetFilter are cumulative and are logically ORed if applied to the same epoch or logically ANDed if applied across different epochs. To reset all filters make a call to ResetFilters.

**C Prototype:**
```
long SetFilter(long TankCode, long TestCode, double V1, double V2);
```

**Arguments:**

*TankCode* — epoch event code as 4 bytes long, determined using StringToEvCode or GetEventCodes

*TestCode* — a single value that sets the criteria value for the filter

letters shown correspond to their ASCII value

associated numbers for use with MATLAB are:

| | | |
|---|---|---|
| 'E' | 69 | equal to |
| 'N' | 78 | not equal to |
| 'G' | 71 | greater than or equal |
| 'L' | 76 | less than or equal |
| 'A' | 65 | above, greater than |
| 'B' | 66 | below, less than |
| 'I' | 73 | include, between these values |
| 'O' | 79 | outside of those values |

*V1* — primary value used in an equation

*V2* — secondary value used an equation

The V2 value is used with "I" and "O" to define the range of the filter.

**Returns:** number of epoch blocks removed by filtering

**Sample Code:** This code sample sets a filter for FREQ = 2000.

```
TT.SelectBlock('Block-45')
TT.CreateEpocIndexing
ecode = TT.StringToEvCode('FREQ')
TT.SetFilter(ecode, 69, 2000, 0)
```

**Related Calls:** GetEventCode, GetEpocCode, CreateEpocIndexing

# *Special Note for Python Users*

Functions with a ‘V’ suffix that accept string values as inputs will not work with Python because the string data type is poorly defined. The affected methods have equivalent methods suffixed with a ‘B’ that are identical to their ‘V’ counterparts but allow input of the BSTR String type.

These functions are compatible with languages that have tighter data type restrictions such as Python.

**The functions are:**

- GetGlobalStringB
- GetGlobalB
- SetGlobalStringB
- SetGlobalB
- SetGlobalsB
- ReadWavesOnTimeRangeB
- SetEpocTimeFilterB
- SetRefEpocB

# TTankInterfaces

## About the TTankInterfaces

TTankInterfaces includes four graphical user interfaces (GUIs) for displaying, modifying, and accessing data tanks through TDT's server applications.

**ServerSelect**
This interface allows users to modify and access server names.

**TankSelect**
This interface allows users to modify and access tank names.

**BlockSelect**
This interface allows users to access and modify blocks within an active tank.

**EventSelect**
This interface allows users to access event properties.

The operations of these components are linked through their event handlers. Developers can coordinate events fired by individual components to develop interactive applications similar to TDT's OpenScope, a client application in the OpenEx software suite. An example program developed in Visual Basic is included to provide an illustration of how these components are used together.

## TTankInterfaces Example

This example is installed with OpenEx Software Suite and can be found in the following path:

C:\TDT\OpenEx\Examples\TTankX_Example\Matlab\TTankInterfacesExample\

### About the Example

This example demonstrates a simple sequence of connecting the different TTankInterfaces ActiveX controls together so that a change in one interface is passed through all the interfaces. For example, if you change the active tank it updates the block selection window. Selection of a particular block lists the events stored in that block.

The example GUI interface was designed using Matlab's GUI editor. This program runs in Matlab 7 or greater. It uses the four TTankInterfaces: ServerSelect, TankSelect, BlockSelect, and EventSelect.

**Note:** Matlab contains its own naming scheme for COM objects, thus the ServerSelect, TankSelect, BlockSelect, and EventSelect COM objects are named activex1, activex2, activex3, and activex4 respectively.

The program responds to the following four events: the server is changed, the tank is changed, the block is changed, and/or an event is changed. A RunAnalysis button is included to illustrate how to add TTank function calls to the GUI interface. When pressed, the button calls the

ReadEventsSimple function which returns the total number of events for all channels of the currently selected Block and EventID.

### *Several files are provided for TTankInterfacesExample:*

**Main.m -** This file creates the GUI interface. Run this file in Matlab to run the example.

**RunAnalysis.m -** This file provides the function call for the RunAnalysis button in the GUI interface.

**TTankInterfacesExample.fig -** This file defines the GUI interface and contains the COM objects.

**Note:** The following files are auto generated from the TTankInterfacesDemo.fig file

**TTankInterfacesExample.m -** Contains the event listener functions for actions that occur in the GUI interface.

**TTankInterfacesExample_activex1 -** Describes the TTankInterfaces.ServerSelect COM object.

**TTankInterfacesExample_activex2 -** Describes the TTankInterfaces.TankSelect COM object.

**TTankInterfacesExample_activex3 -** Describes the TTankInterfaces.BlockSelect COM object.

**TTankInterfacesExample_activex4 -** Describes the TTankInterfaces.EventSelect COM object.

## ServerChanged

When the ServerChanged event occurs the function activex1_ServerChanged is called. This function then calls functions associated with TankSelect, the next interface in the group. TankSelect calls the UseServer function which returns all the tanks on that server and refreshes the TankSelect screen. The code is shown below.

```
function activex1_ServerChanged(hObject, eventdata, handles)

% Process Server selection info for TankSelect
handles.activex2.UseServer = eventdata.NewServer;
handles.activex2.Refresh;

% Update global variable CurrentServer
global CurrentServer;
CurrentServer = eventdata.NewServer;
```

## TankChanged

When the TankChanged event occurs, the function activex2_TankChanged is run. This function then calls functions associated with BlockSelect, the next interface in the group. BlockSelect calls the following functions: UseServer, UseTank, and Refresh. These functions return all the blocks on that tank and refresh the BlockSelect screen. In addition, if the current tank is changed, the currently selected block and event are deselected. The code is shown below.

```
function activex2_TankChanged(hObject, eventdata, handles)

% Process Server and Tank selection info for BlockSelect
handles.activex3.UseServer = eventdata.ActServer;

% Deselects the previously selected Block if the current Tank is
% changed
handles.activex3.ActiveBlock = '';
handles.activex3.Refresh;

% Deselects the previously selected Event and clears the event
% list if the current Tank is changed
handles.activex4.UseBlock = '';
handles.activex4.ActiveEvent = '';
handles.activex4.Refresh;

% Update global variable CurrentTank
global CurrentTank;
CurrentTank = eventdata.ActTank;
```

## BlockChanged

When the BlockChanged event occurs, the function activex3_BlockChanged is run. This function then calls functions associated with EventSelect, the next interface in the group. EventSelect calls the following functions: UseServer, UseTank, UseBlock, and Refresh. These functions return all the events in that block and refresh the EventSelect interface. In addition, if the current block is changed, the currently selected event is deselected. The code is shown below.

```
function activex3_BlockChanged(hObject, eventdata, handles)

% Process Server, Tank, and Block selection info for EventSelect
handles.activex4.UseServer = eventdata.ActServer;
handles.activex4.UseTank = eventdata.ActTank;
handles.activex4.UseBlock = eventdata.ActBlock;

% Deselects the previously selected Event if the current Block is
% changed
handles.activex4.ActiveEvent = '';
handles.activex4.Refresh;
```

## ActEventChanged

When the ActEventChange event occurs the function activex4_ActEventChanged is run. This function then stores the selected Event in the global variable CurrentEvent before calling the refresh function. Once an event has been selected, the RunAnalysis button can be used to return the total number of events from the currently selected Block. The code is shown below.

```
function activex4_ActEventChanged(hObject, eventdata, handles)
% Process Event Selection and refresh
global CurrentEvent;
CurrentEvent = eventdata.NewActEvent;
handles.activex4.Refresh;
```

## RunAnalysis

When the RunAnalysis button is pressed, the program RunAnalysis.m is called. This program reads the block data into a MATLAB structure using TDT2mat.m.

# TDevAcc

## *About TDevAcc*

TDevAcc is a series of methods for accessing and controlling hardware through an OpenWorkbench server. TDevAcc can be used to develop client applications similar to TDT's OpenController application.

TDevAcc provides access to System 3 real-time processing devices during an experiment. Client applications developed using TDevAcc can control circuit parameters, retrieve information from device buffers, and read device tags in real-time. Keep in mind that this unprotected access must be used carefully. An entire OpenEx experiment could 'crash' if a flawed attempt to access tags is executed. TDevAcc supports a modified tag access protocol similar to the one used by the RPco.X interface. Developers should be very familiar with the RPco.X interface, and RPvdsEx circuit design and use, before attempting to use TDevAcc.

TDevAcc uses targets to implement real-time control. When OpenWorkbench is running, client applications developed using TDevAcc can call the OpenWorkbench server and access OpenWorkbench targets. OpenWorkbench targets include the device name, or the name assigned to a hardware device within OpenWorkbench, and the parameter tag, or the name of a tag created in the circuit in RPvdsEx. The device name and parameter tag are used together and are separated by a period to create a target, such as Amp1.LPFreq. The target identifies and provides access to a specific parameter tag within a circuit running on a specific real-time processing device.

Before using the TDevAcc methods, users should have a strong understanding of RPvdsEx circuits and OpenEx methodology along with a background in programming with TDT ActiveX controls.

Users should be mindful of using good 'closed loop' access when working with TDevAcc. This means always releasing your servers.

### *A typical server access session for a client consists of five main steps:*

1. Run the OpenWorkbench application.
2. Load an OpenWorkbench configuration file.
3. Call ConnectServer -- Called to connect to the OpenWorkbench server. The connection is terminated with CloseConnection.
4. Perform any number of operations with the OpenWorkbench server.
5. Call CloseConnection -- Called to release the OpenWorkbench server.

### *A standard MATLAB routine might look like the routine below:*

```
DA = actxcontrol('TDevAcc.X')
DA.ConnectServer('Local')
 %Your code
DA.CloseConnection
```

# Organization of TDevAcc Methods

***TDevAcc methods can be divided into three basic groups:***

- **Setup and Control** -- The methods in this group are used to setup access to OpenWorkbench and control OpenWorkbench system modes.
- **Hardware Data Access** -- The methods in this group are used to read or write data to hardware device components.
- **Hardware Information Retrieval** -- The methods in this group are used to access information, such as status or sample frequency, about a device.

# Setup and Control — TDevAcc X

## ConnectServer

**Description:** ConnectServer initiates a connection with an OpenWorkbench server. The connection adds a client to the server. A project has to be loaded in order for Connect Server to return 1 and it will fail if OpenProject is not loaded or if the loaded OpenProject does not have a valid Workbench configuration file.

**Prototype:**
```
Function ConnectServer(ServerName As String) As Long
```

**Arguments:**

| | | |
|---|---|---|
| String | *ServerName* | name of the server, 'Local' is most common |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample connects to the Local server.

```
DA = actxcontrol('TDevAcc.X')
DA.ConnectServer('Local')
```

## CheckServerConnection

**Description:** CheckServerConnection verifies if OpenWorkbench is loaded and if the OpenWorkbench server has loaded a circuit file to the OpenWorkbench application. The method will return a 1 if OpenWorkbench is correctly loaded and configured and in Standby, Preview, or Record mode. It returns 0 if OpenWorkbench is not loaded, configured, or if OpenWorkbench is in Idle mode.

**Prototype:**
```
Function CheckServerConnection() As Long
```

**Returns:** 0 (Idle), 1 (Standby, Preview, or Record)

**Sample Code:** This code sample checks the connection to the server and returns a message if the client is not connected to the server.

```
DA.ConnectServer('Local')
if DA.CheckServerConnection==0
  display('Client application not connect to server')
end
```

## GetSysMode

**Description:** GetSysMode (get system mode) returns the state of OpenWorkbench as a long. This call can be used in conjunction with SetSysMode to control the operational mode of your entire OpenEx system. The various modes of OpenWorkbench, including Idle, Standby, Preview, and Record; are described in the OpenEx Manual.

**Prototype:** `Function GetSysMode() As Long`

**Returns:** 0 (Idle), 1 (Standby), 2 (Preview), 3 (Record)

**Sample Code:** This code sample opens a connection to the OpenWorkbench server. If the OpenWorkbench mode is Record (3) the routine is run.

```
if DA.ConnectServer('Local')==1 then
 if DA.GetSysMode==3 then
  %Start Routine
 end
end
```

## SetSysMode

**Description:** SetSysMode (set system mode) sets the state of OpenWorkbench through the system mode. The possible modes include: Idle, Standby, Preview, and Record.

**Prototype:** `Function SetSysMode(NewMode As Long) As Long`

**Arguments:** *NewMode* sets the mode of the system: 0 (Idle), 1 (Standby), 2 (Preview), 3 (Record)

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample opens a connection to the OpenWorkbench server. If the OpenWorkbench mode is not Record (3) mode, SetSysMode places the OpenWorkbench in Record (3) mode.

```
if DA.ConnectServer('Local')==1
  if DA.GetSysMode ~= 3
  DA.SetSysMode(3)
 end
end
```

## SetTankName

**Description:** SetTankName sets the active tank if OpenWorkbench is loaded and in Idle or Standby mode. If you are setting the value to a registered tank, you need only provide the tank name for the argument TankName. Otherwise, provide the entire path to the tank.

**Prototype:** `Function SetTankName(TankName as String) As Long`

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample opens a connection to the OpenWorkbench server and sets the active tank to the registered tank DemoTank2.

```
DA.ConnectServer('Local')
DA.SetTankName('DemoTank2')
```

**Note:** to use an unregistered tank, use the absolute path to the tank

```
DA.SetTankName('C:\TDT\OpenEx\Tanks\DemoTank2')
```

### GetTankName

**Description:** GetTankName returns name of the active tank if OpenWorkbench is loaded. If the tank has not yet been specified in OpenWorkbench, GetTankName returns the null string.

**Prototype:**
```
Function GetTankName() As String
```

**Returns:** name of the active tank

**Sample Code:** This code sample opens a connection to the OpenWorkbench server and gets the name of the active tank.

```
DA.ConnectServer('Local')
DA.GetTankName
```

### CloseConnection

**Description:** CloseConnection closes the connection to the OpenWorkbench server.

**Prototype:**
```
Function CloseConnection()
```

**Sample Code:** This code sample opens a connection to the OpenWorkbench server then closes it after the client application is finished.

```
DA.ConnectServer('Local')
% Your Code
DA.CloseConnection
```

## *Hardware Data Access  TDevAcc X*

### SetTargetVal

**Description:** SetTargetVal (set target value) sends a value to a target and is used to modify a parameter tag within an RCO circuit. It can also be used to set the attenuation value of a PA5. See *About TDevAcc*, page 65 for more information on targets.

**Prototype:**
```
Function SetTargetVal(Target As String, Val As
Double) As Long
```

**Arguments:**

| | |
|---|---|
| *Target* | target name in DevName.TagName format |
| *Val* | value to assign to the target |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code sample sets the value of target Acq1.Thresh to 5.

```
DA.SetTargetVal('Acq1.Thresh',5)
```

This code sample sets the attenuation of target PA5_1 to 20.

```
TD.SetTargetVal('PA5_1.Atten', 20)
```

## GetTargetVal

**Description:** GetTargetVal retrieves a target value and is used to read a parameter tag within an RCO circuit or the attenuation value of a PA5. See About TDevAcc, page 65 for more information on targets.

**Note:** this function will return 0.0 if the target specified is invalid or cannot be read. You may want to validate the target using GetTargetType before accessing it with GetTargetVal.

**Prototype:**
```
Function GetTargetVal(Target As String) As Double
```

**Arguments:** *Target* target name in DevName.TagName format

**Returns:** value read from the target

**Sample Code:** This code sample returns the cycle usage of device Acq1.

```
DA.GetTargetVal('Acq1.zCycUse')
```

This code sample returns the attenuation value of the PA5_1.

```
TD.GetTargetVal('PA5_1.Atten')
```

## WriteTarget

**Description:** WriteTarget is used to send data to a memory buffer located on a processor device.

WriteTarget functions similarly to WriteTargetVEX but is designed for Legacy users. New users should refer to the WriteTargetVEX function.

**Note:** The floating point data array pData must be cast as a single for use with Matlab.

**Prototype:**
```
Function WriteTarget(Target As String, nOS As Long,
nWords As Long, pData As Single) As Long
```

**Arguments:**

| | | |
|---|---|---|
| String | *Target* | name of parameter tag |
| Long | *nOS* | offset within buffer to begin write, given in 32bit words |
| Long | *nWords* | number of 32-bit words to write |
| Float FAR* | *pData* | floating point array holding data to load to RPx memory |

**Returns:** 0 (fails), 1 (succeeds)

## WriteTargetV

**Description:** WriteTargetV is used to send data to a memory buffer located on a processor device. WriteTargetV functions similarly to WriteTargetVEX but is designed for Legacy users. New users should refer to the WriteTargetVEX function.

**Note:** The Variant data array vData must be cast as a single for use with Matlab.

**Prototype:**
```
Function WriteTargetV(Target As String, nOS As Long,
(vData As Variant) As Single) As Long
```

**Arguments:**

| | | |
|---|---|---|
| String | *Target* | name of parameter tag |
| Long | *nOS* | number of points to offset in buffer before starting write |

| | | |
|---|---|---|
| Variant | *vData* | data array with the samples (this array must be cast as singles) |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code writes the matrix coefs to the target Acq1.Filter1Coef

```
coefs = [0.206572 0.413144 0.206572 -0.369527
0.195816];

coefs = single(coefs);

coefs = DA.WriteTargetV('Acq1.Filter1Coef',0,coefs)
```

## WriteTargetVEX

**Description:** WriteTargetVEX is used to send data to a memory buffer located on a processor device. The target is a parameter tag in a circuit running on a device and follows the form: DeviceName.ParameterTag. The DeviceName is the name given to the device by OpenWorkbench.

WriteTargetVEX is for use with programming languages that use dynamic data typing, such as Python, but is also compatible with Matlab.

**Note:** Unlike WriteTarget and WriteTargetV, WriteTargetVEX does not require the vData array to be cast as a single.

**Prototype:**

```
Function WriteTargetVEX(Target As String, nOS As
long, DstType As String, vData As Variant) As Long
```

**Arguments:**

| | |
|---|---|
| *Target* | name of parameter tag |
| *nOS* | number of points to offset in buffer before starting write |
| *DstType* | format for storing data |

| float (32-bit) | long (32-bit) | short (16-bit) | byte (8-bit) |
|---|---|---|---|
| 'F32' | 'I32' | 'I16' | 'I8' |

| | |
|---|---|
| *vData* | any data array |

**Returns:** 0 (fails), 1 (succeeds)

**Sample Code:** This code writes the matrix coefs to the target Acq1.Filter1Coef.

```
DA = actxcontrol('TDevAcc.X')

DA.ConnectServer('Local')

coefs = [0.206572 0.413144 0.206572 -0.369527
0.195816];

DA.WriteTargetVEX('Acq1.Filter1Coef',0,'F32',coefs)
```

## ZeroTarget

**Description:** Sets a parameter tag value to zero. When the parameter tag points to a memory buffer, all values in the buffer are set to zero.

**Prototype:**

```
Function ZeroTarget(Target As String) As Void
```

**Arguments:**

| | |
|---|---|
| *Target* | name of parameter tag |

**Sample Code:** This code resets a stimulus buffer using tag Stim.StimBuf.

```
DA = actxcontrol('TDevAcc.X')

DA.ConnectServer('Local')
```

```
DA.ZeroTarget('Stim.StimBuf')
```

## ReadTarget

**Description:** ReadTarget is used to read data from a target to a buffer on the PC.

ReadTarget functions similarly to ReadTargetVEX but is designed for Legacy users. New users should refer to the ReadTargetVEX function.

**Tech Notes:** ReadTargetV is used with the following components that have a data buffer: RamBuffer, SerialBuffer, AverageBuffer, LongDelay, LongDynDelay, ShortDelay, ShortDynDelay, Biquad, IIR,FIR, HrtfFir.

**Prototype:**
```
Function ReadTarget(Target As String, nOS As Long,
nWords As Long, pBuf As Single) As Long
```

**Arguments:**

| | | |
|---|---|---|
| String | *Target* | name of parameter tag |
| Long | *nOS* | number of points to offset in buffer before starting read |
| Long | *nWords* | number of 32-bit words to read (samples) |
| Float | *pBUF* | pointer to the buffer for storing the data |

**Returns:** 0 (fails), 1 (succeeds)

## ReadTargetV

**Description:** ReadTargetV is used to read data from a target to a buffer on the PC.

ReadTargetV functions similarly to ReadTargetVEX but is designed for Legacy users. New users should refer to the ReadTargetVEX function.

**Note:** The Variant data array returned is of type Single (F32).

**Tech Notes:** ReadTagV is used with the following components that have a data buffer: RamBuffer, Serial Buffer, Average Buffer, LongDelay, LongDynDelay, ShortDelay, ShortDynDelay, Biquad, IIR,FIR, HrtfFir.

**Prototype:**
```
Function ReadTargetV(Target As String, nOS As Long,
nWords As Long) As Single
```

**Arguments:**

| | | |
|---|---|---|
| String | *Target* | name of parameter tag |
| Long | *nOS* | number of points to offset in buffer before starting read |
| Long | *i* | number of 32-bit words to read (samples) |

**Returns:** array of type Single, or -1 or NaN if fails

## ReadTargetVEX

**Description:** ReadTargetVEX is used to read data from a target to a buffer on the PC.

ReadTargetVEX reads data from a parameter tag (Target) in a circuit running on a device that follows the form: DeviceName.ParameterTag. The DeviceName is the name given to the device by OpenWorkbench. The user must specify the source type (Srctype) of the parameter tag's data (F32, I32, I16, or 8-bit Integer) and the number of 32 bit words to read. It then converts it to one of five data formats (Double, Floating Point, Word, Integer, or 8-bit Integer) and stores it on a PC buffer.

If the data being read is shuffled, nWords is equivalent to the number of Samples in the Serial Buffer. If the data is compressed, nWords is equal to the number of points saved.

For example, if the data is compressed two-folded, 500 points of 1000 samples of the Serial Buffer have been read; nWords should be set to 500. For I8 format, a compression of 4, the number of points read from the buffer is 250 and nWords should be set to 250.

**Note:** ReadTargetVEX has been added to TDevAcc for use with programming languages that use dynamic data typing, such as Python, but is also compatible with Matlab.

**Prototype:**
```
Function ReadTargetVEX(Target As String, nOS As long, nWords As long, SrcType As String, DstType As String) As Variant
```

*Arguments:*

*Target* — name of parameter tag

*nOS* — number of points to offset in buffer before starting read

*nWords* — number of 32-bit words to read (samples)

*Srctype* — storage format type of data being read. Below is a list of the storage types

| float (32-bit) | long (32-bit) | short (16-bit) | byte (8-bit) |
|---|---|---|---|
| 'F32' | 'I32' | 'I16' | 'I8' |

*DstType* — format for storing data

| Double (64-bit) | float (32-bit) | long (32-bit) | short (16-bit) | byte (8-bit) |
|---|---|---|---|---|
| 'F64' | 'F32' | 'I32' | 'I16' | 'I8' |

**Returns:** array of the buffer contents, -1 or NaN if call failed

**Sample Code:** This sample reads five 32-bit words of data in single format from the parameter tag Acq1.Filter1Coef and returns it in double format.

```
DA = actxcontrol('TDevAcc.X')
DA.ConnectServer('Local')
coefs = DA.ReadTargetVEX('Acq1.Filter1Coef',0, 5, 'F32', 'F64')
```

# *Hardware Information Retrieval TDevAcc X*

## GetDeviceName

**Description:** GetDeviceName returns the name of the devices that OpenWorkbench is connected to. A null string is returned if the there is no device at that index. To get the first device name, use an index of 0. Then increase the index until a null string is returned.

**Prototype:**
```
Function GetDeviceName(Index As Long) As String
```

**Arguments:** *Index* — device index (zero based)

**Returns:** the device name at the specified index, or null string if no device at that index

## GetDeviceRCO

**Description:** GetDeviceRCO returns the full path name of the RCO file loaded to the specified device. A null string is returned if the device name is invalid or if no RCO file is loaded.

**Prototype:** `Function GetDeviceRCO(DeviceName As String) As String`

**Arguments:** *DeviceName* name of the device given by OpenWorkbench, e.g. 'Amp1'

**Returns:** the RCO file name and full path

## GetDeviceSF

**Description:** GetDeviceSF returns the exact sampling frequency of a device connected to the OpenWorkbench server.

**Prototype:** `Function GetDeviceSF(DeviceName As String) As Float`

**Arguments:** *DeviceName* name of the device given by OpenWorkbench, e.g. 'Amp1'

**Returns:** sampling frequency of the hardware device

## GetDeviceStatus

**Description:** GetDeviceStatus returns the status of a device connected to the OpenWorkbench server. The first three bits of the status are used by all programmable devices to indicate the following: a connection to the PC, a loaded RCX file, and a running circuit.

The target for GetDeviceStatus is the name of the hardware device. The device name is the name given to the device on the corresponding OpenWorkbench property sheet, for example, Amp1.

**Prototype:** `Function GetDeviceStatus(Target As String) As Long`

**Arguments:** *Target* name of the target (OpenWorkbench Device name)

**Returns:** connection status, first four bits check the status of the device

A bit-code value is set based on the status of the device.

**All devices:**

| Bit | Value (Enabled) | Status |
|---|---|---|
| 0 | 1 | Connected |
| 1 | 2 | Circuit loaded |
| 2 | 4 | Circuit running |

Using GetDeviceStatus simplifies the error checking routines (see below).

For best results, use bit-wise operations (0/1). Bits remain constant. Long values change as new bits are added to GetDeviceStatus().

If a circuit has previously been loaded to the device it will run when LoadCof fails, and the bit status of the device will read 0110(6) or 0111.

**RA16BA:**

The RA16BA has additional status values. Bit 4 indicates clipping is occurring on one or more channels. Bit 5 indicates that clipping has occurred since the last time GetDeviceStatus was called. Once GetDeviceStatus is called bit 5 is reset.

**Note:** When checking the status of the RA16BA, ensure that a preamplifier is properly connected and turned on. Connection status (Bit 0) will always return a 0 when a preamplifier is not properly connected. Bit 5 (amplifier clipped since last call) is reset after GetDeviceStatus is called.

| Bit | Value (Enabled) | Status |
|---|---|---|
| 0 | 1 | Connected |
| 1 | 2 | Circuit loaded |
| 2 | 4 | Circuit running |
| 3 | 8 | Battery status (RL2, RA16PA) |
| 4 | 16 | Clipping on one or more channels |
| 5 | 32 | Clipping occurred since last GetDeviceStatus |

**RV8:**

| Bit | Value (Enabled) | Status |
|---|---|---|
| 0 | 1 | Connected |
| 1 | 2 | Circuit loaded |
| 2 | 4 | Circuit running |
| 3 | 8 | N/A |
| 4 | 16 | N/A |
| 5 | 32 | N/A |
| 6 | 64 | System armed |
| 7 | 128 | Circuit running |
| 8 | 256 | Trigger enabled |
| 9 | 512 | Auto-clear DAC outs |
| 10 | 1024 | Tick out |
| 11 | 2048 | Clock out |
| 12 | 4096 | zTrigA |
| 13 | 8192 | zTrigB |
| 14 | 16384 | External trigger |
| 15 | 32768 | Multiple trigger |

## GetDeviceType

**Description:** GetDeviceType returns the type of device as a long integer. The target is the name of the hardware device as it appears in OpenWorkbench.

**Prototype:** `Function GetDeviceType(String Target) As Long`

**Arguments:** *Target* name of the target device in OpenWorkbench

**Returns:** a long that returns the DeviceType as a long integer (see table below)

**Tech Notes:**

| *Device Type* | *Value* |
|---|---|
| RP2 | 0 |
| RL2 | 1 |
| RA16 | 2 |
| RV8 | 3 |
| RM1 | 5 |
| RM2 | 6 |
| RX5 | 10 |
| RX6 | 11 |
| RX7 | 12 |
| RX8 | 13 |
| RZ2 | 15 |
| RZ5 | 18 |
| RZ6 | 19 |

## GetNextTag

**Description:** GetNextTag (get next parameter tag) returns the name of the parameter tag of a particular data type (such as integer or logical). The first call to the GetNextTag method must have a number other than 0 for DoFirst. All subsequent calls can pass the value 0 for DoFirst. The target for GetNextTag is the name of the hardware device. The DeviceName is the name given to the device by OpenWorkbench on the corresponding property sheet. Tags are indexed by alphabetical order.

**Prototype**:

```
Function GetNextTag(Target As String, ReqType As
Long, DoFirst As Long) As String
```

**Arguments:**

| | |
|---|---|
| *Target* | name of target, form for target is DeviceName |
| *ReqType* | data type associated with the target (see GetTargetType) |
| *DoFirst* | The DoFirst parameter allows the user to specify whether to return the first tag (DoFirst = value > 0) or the next successive tag in the circuit (DoFirst = 0). If a zero is passed initially, this method will return a null string, therefore, a nonzero value must be passed initially. |

**Returns:** returns the next parameter tag of that data type in the sequence

| *Data Type* | *Value* | *Ascii Map* |
|---|---|---|
| Data (buffer) | 68 | "D" |
| Integer | 73 | "I" |
| Logical | 76 | "L" |
| Coefficients | 80 | "P" |
| Float | 83 | "S" |

**Sample Code:** Retrieves all the parameter tags associated with the buffer data type.

```
DA = actxcontrol('TDevAcc.X');
```

```
DA.ConnectServer('Local');
% Must specify a nonzero index for the first tag
target= DA.GetNextTag('Acq1',68,1)
for i=0:10 % Search for up to 10 tags
 target=DA.GetNextTag('Acq1',68,0);
% Search through tags until no more are found and
display while ~strcmp(target, )
target=DA.GetNextTag('Acq1',68,0)
end
```

## GetTargetType

**Description:** GetTargetType returns the data type of the specified target. The target is a parameter tag in a circuit running on a device and follows the form: DeviceName.ParameterTag. The DeviceName is the name given to the device by OpenWorkbench on the corresponding property sheet. GetTargetType returns 0 if the target is invalid.

**Prototype:** `Function GetTargetType(Target As String) As Long`

**Arguments:** *Target* name of target, in the form DeviceName.ParameterTag

**Returns:** long that maps to an ASCII character

| *Data Type* | *Value* | *Ascii Char* |
|---|---|---|
| Data buffer / Delay line (DM) | 68 | "D" |
| Integer | 73 | "I" |
| Logical (1 or 0) | 76 | "L" |
| Coefficient buffer (PM) | 80 | "P" |
| Float (Single) | 83 | "S" |
| Undefined (e.g. latch output) | 65 | "A" |

## GetTargetSize

**Description:** This function returns the size of a DM or PM buffer or scalar tag in 32-bit words. The tag is specified using the standard target naming convention and will return either the allocated size of the buffer or a one if the target is a scalar. A zero is returned for an invalid target.

**Prototype:** `Function GetTargetSize(Target As String) As Long`

**Arguments:** *Target* name of target, in the form DeviceName.ParameterTag

**Returns:** Size of the buffer, 1 if tag is connected to a scalar value, 0 on error

# Examples

The example files below are installed with OpenEx Suite; however, the most up to date versions of examples are available in a downloadable ZIP file on the TDT website: http://www.tdt.com/files/examples/OpenDeveloperExamples.zip.

**TDT recommends starting with the TDT2mat.m and SEV2mat.m examples for extracting all block data into a matlab structure.**

## Recommended Examples

**Files:** OpenEx\Examples\TTankX_Example\Matlab\TDT2mat.m or TDT2mat.m

**Overview:** Demonstrates steps to extract Tank data into a Matlab structure.

**Files:** OpenEx\Examples\TTankX_Example\Matlab\SEV2mat.m or SEV2mat.m

**Overview:** Demonstrates steps to extract SEV data into a struct format.

**Files:** OpenEx\Examples\TTankX_Example\Matlab\TDTfilter.m or TDTfilter.m

**Overview:** Demonstrates steps to filter Tank Data.

**Files:** OpenEx\Examples\TTankX_Example\Matlab\ Raster_PSTH.m or Raster_PSTH.m

**Overview:** Demonstrates steps to display data as a PSTH raster plot.

**Files:** OpenEx\Examples\TTankX_Example\Matlab\rms.m or rms.m

**Overview:** Calculates RMS.

**Files:** OpenEx\Examples\TTankX_Example\Matlab\TDTdigitalfilter.m or TDTdigitalfilter.m

**Overview:** Demonstrates steps to apply a digital filter to streaming data.

**Files:** OpenEx\Examples\TTankX_Example\Matlab\TDTfft.m or TDTfft.m

**Overview:** Demonstrate steps to perform a frequency analysis of a data stream.

## Legacy Examples

**File:** OpenEx\Examples\TTankX_Example\Matlab\TTankInterfaces Example\Main.m

**Overview:** Demonstrates the TTankInterfaces (ServerSelect, TankSelect, BlockSelect, and EventSelect). A button is included which returns the total number of events for the currently selected event.

**Note:** This example must be run using Matlab 7 or greater.

**File:** OpenEx\Examples\TTankX_Example\Matlab\Example1.m or Example1.m

**Overview:** Demonstrates how data can be extracted from a tank and parsed.

**File:** OpenEx\Examples\TTankX_Example\Matlab\Example2.m or Example2.m

**Overview:** Demonstrates how to extract filtered data from the tank. Data is filtered based on epoch events then events are extracted and parsed for later analysis and display.

**File:** OpenEx\Examples\TTankX_Example\Matlab\FilterArray.m or FilterArray.m

**Overview:** Describes how to filter data from the OpenEx Tank.

**Files:** OpenEx\Examples\TTankX_Example\Matlab\InterSpikeInterval.m or InterSpikeInterval.m

**Overview:** Demonstrates how to access tank data and parse events then plot the inter-spike intervals (ISI).

**Files:** OpenEx\Examples\TTankX_Example\Matlab\WaveReconstruction.m or WaveReconstruction.m

**Overview:** Demonstrates the steps used to reconstruct waveforms from events.

# Known Anomalies

ReadEvents/ReadEventsV may miss some events if there is a long interval between events in a block. Instead of trying to read all the events in the block at once (with start and stop time = 0), loop through and read the events in steps of 100 second (or less) intervals. Sample Matlab code:

```
ts = [ ];
% assumes there will never be more than 1000 events in an
interval
maxevents = 1000;
% assumes a block will never be longer than 10000 seconds
maxtime = 10000;
% steps through block in 100 second intervals
steps = maxtime / 100;
for i = 1:steps
  % reads events in current 100 second interval
  events = ttank.ReadEventsV(maxevents, 'stor', 0, 0,
  ((i-1)*100), (i*100), 'ALL');
   if (events > 0)
    % if events were found, the timestamps are collected
    timestamps = ttank.ParseEvInfoV(0, events, 6);
    ts = cat(2, ts, timestamps(1,:));
  end
end
ts
```

---

Filtering methods, such as SetFilterArray, will always return 0 if a Store ID begins with any of the following characters: "-", "=", "(", ")", "<", ">", "!", a space or any number 0 to 9. When the TTank engine performs filtering of events in the Tank, the above characters will not be parsed correctly, and the store name will not be decoded properly.

---

The Global parameter SortCode, or argument SortCode for methods such as ReadEvents and ReadEventsV, cannot be used to define or condition the sort code of snippet events based on SortId results generated in OpenSorter. Even after SetUseSortName is called, the TTank server will use the default set of sort codes originally saved to the tank.

For example, the following Matlab code fails to cache Snip events with sort code 1, from the OpenSorter generated sort set “Sort1” and uses the default “TankSort” sort set instead.

```
SetSort = TT.SetUseSortName('Sort1')
SCode=1;
nEvents = TT.ReadEventsV(10000,'Snip',1,SCode,0.0,0.0,'All')
```

To use the alternate sort sets generated with OpenSorter, use the command SetFilterWithDescEx to set the sort code condition you want. Then use commands like ReadEventsV with the SortCode argument as 0, and the Options argument as 'FILTERED'. The Matlab code below will work to read Snip events of channel = 1and sort code = 1 from the sort set saved as "Sort1".

```
SetSort1 = TT.SetUseSortName('Sort1');
TT.SetFilterWithDescEx('sort=1');
AllSort1 = TT.ReadEventsV(10000,'Snip',1,0,0.0,0.0,'FILTERED')
```

---

The most recent anomalies updates are available on the Web at http://www.tdt.com/technotes/.

# Index