WaveTrak Buttons: the Button Bank

WaveTrak has numerous buttons on virtually every card for performing a variety of functions. These buttons are described with the cards they live on in the 'WaveTrak Cards' chapter. As mentioned in that chapter, the Button Bank card is a repository for dozens of buttons specifically designed for data acquisition and signal processing. This chapter describes what each of these standard buttons does and how to use them. In general, you will *copy* the button you need and paste a duplicate into the *background* of a trace card, moving it to convenient location on the card. Consult your HyperCard manual on how to copy buttons. *Don't cut buttons from the Button Bank* because WaveTrak depends on some of them for normal operation.

Tip:

The most common mistake is to paste a button into the card layer rather than the background layer of a trace card. With the next acquisition, the button disappears because it was only pasted onto a single trace card, rather than being common to all traces. See your HyperCard manual if you are not clear about background and card layers. Also, when you paste a button into the trace background, because it was the most recent addition, it will probably sit "on top" of the fields revealed by the Show Wave Info button. Choose the 'Send Farther' menu command several times to send it behind the fields.

You will eventually want to modify and customize your buttons. Their scripts are relatively short and straightforward with many comments to help you understand how they work. We encourage you to open and examine the scripts and modify them to suit your needs.

The existing buttons form a good foundation upon which to expand your collection. Duplicate the button that most closely resembles the function you need, and build your new button from the existing script. You can fully exploit the power of WaveTrak's data acquisition toolbox and signal processing functions by familiarizing yourself with HyperTalk, the standard HyperCard scripting language. If you create buttons that you think might enjoy widespread interest in the scientific or engineering community, let us know, so we can organize a larger collection of functions to be shared among WaveTrak users.

This chapter begins to discuss in more detail the inner workings of WaveTrak's data acquisition functions. The instructions executed by the 'Single' and 'Multiple' buttons are discussed in detail to teach you how to script your own functions so they will be compatible with the rest of the WaveTrak environment. If you are not yet ready to start scripting, skip over the details for now; we are confident that you will come back to this discussion to learn how to write your own functions when you see how easy this is to do using the WaveTrak data acquisition toolbox.

The buttons in the Button Bank are grouped into four sections: 'Acquisition', 'Measurement', 'Signal Generation' and 'Analysis & Wave Math'. Each group of buttons will be discussed in a separate section in this chapter. Note that you will need the WaveTrak AD version of the software to perform functions in the first three groups.

Acquisition

The buttons in this group have one property in common: they digitize signals and create trace cards for storing the results. As a result, *they must be pasted into a trace card* and will function only from there. All new traces are appended after the last trace under the current root. As the button names imply, 'Single...' buttons acquire a single wave and create a single trace card, whereas 'Multiple...' buttons acquire as many as 16 waves simultaneously from up to 16 A/D channels, placing the results in consecutive traces.

1. Single A/D

   Function:
   This button acquires a single wave from the A/D channel selected in the pop-up menu on the trace card.

XCMDs: `AcqWave`

Comments:
The sampling rate and number of points per wave are taken from the current entries in the Scope card, and defaults are copied from the 'Hardware parameters' and 'Readings' fields in the System Parameters card. The 'Full-scale' data (line 4 in the Hardware parameters field) are taken from the 'External A/D gain' field in the System Parameters card corresponding to the selected channel.

2. Multiple

Function:
The 'Multiple' button is similar to the Single A/D function described above, except that it acquires data from up to 16 A/D channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.

Comments: Traces are added consecutively following the last card and marked with the channel number. This version assumes that the converter is jumpered for differential input and therefore acquires a maximum of 8 A/D channels.

3. Single A/D Thresh

Function:
Acquires a single wave from the A/D channel selected in pop-up menu in trace card. Begins acquisition when the signal crosses threshold as in Scope card.

Parameters:
• `triggerLevel`: analog threshold in mV.
• `slope`: 1 will trigger on positive threshold crossing, 0 on negative.
• `timeout`: waits for a threshold crossing for `timeout` seconds before returning with an error.

XCMDs: `AcqWaveThresh`

Comments:
Use this button if you want to trigger an acquisition at a particular level on your wave, simulating the trigger level on an oscilloscope. The `AcqWaveThresh` XCMD requires that the threshold level be passed as a binary integer (i.e. between -2048 and 2047 for a 12 bit converter) and not as a mV value. The `translateToBinary` handler in the stack script converts

real mV to the equivalent binary count given the minimum and maximum full scale mV range and the current binary coding scheme contained in the **ADCbits** global (see the chapter on Handlers for more information).

4. Multiple Thresh

Function:
Similar to Single A/D Thresh, but acquires several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `triggerLevel`: analog threshold in mV.
• `slope`: 1 will trigger on positive threshold crossing, 0 on negative.
• `timeout`: waits for a threshold crossing for this many seconds before returning with an error.

XCMDs: `AcqWaveThresh`

Comments:
The trigger level is detected at the first channel (i.e. at startMUX). After a threshold crossing, several channels are acquired as in the 'Multiple' button.

5. Single A/D + DAC

Function:
Acquires a single wave from the A/D channel selected in the pop-up menu in the trace card. Generates a simultaneous analog pulse on one D/A channel.

Parameters:
• `DACchannel`: the D/A channel where the step will be generated.
• `DACpre`: analog level of pre-pulse in mV.
• `prePulse`: duration of pre-pulse in µs.
• `DACpulse`: analog level of pulse in mV.
• `pulseWidth`: duration of pulse in µs.
• `DACpost`: baseline analog level after pulse in mV.

XCMDs: `AcqWaveDAC`

Comments:

This button allows you to stimulate a system with an analog signal and simultaneously digitize the response. The analog output consists of two phases: a pre-pulse and a pulse. The duration and level of each can be independently set. The pre-pulse is useful for conditioning a system prior to the stimulus (the pulse). If all you need is a single analog pulse then set the pre-pulse duration (`prePulse`) to zero. Note that, unlike the threshold function above, the D/A level is passed as a mV value; it is first corrected according to the external D/A gain from **DACGainTable**. *The durations of the pre-pulse and the pulse must be integral multiples of the sampling interval*. See `AcqWaveDAC` for details. Connect the D/A output to the selected A/D channel to capture and examine the analog pulses. `DACpost` will be the final analog level after the pulses are delivered and will persist after the button completes execution.

6. Multiple + DAC

Function:
Similar to Single A/D + DAC but acquires several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `DACchannel`: the D/A channel where the pulses will be generated.
• `DACpre`: analog level of pre-pulse in mV.
• `prePulse`: duration of pre-pulse in µs.
• `DACpulse`: analog level of pulse in mV.
• `pulseWidth`: duration of pulse in µs.
• `DACpost`: baseline analog level after pulse in mV.

XCMDs: `AcqWaveDAC`

Comments:
The start of the first analog step will coincide with the first point of the first channel.

7. Single A/D + DAC2

Function:

Similar to Single A/D + DAC except that the pulse parameters are taken from the Hardware Parameters field in the System Parameters card instead of being typed into the button script.

Parameters:
• `DACchannel`: the D/A channel where the step will be generated.

XCMDs: `AcqWaveDAC`

Comments:
Use this example to modify other buttons if you prefer setting your acquisition parameters via the Hardware Parameters field instead of typing them into the button scripts.

8. Single A/D on TTL

Function:
Acquires a single wave from the A/D channel selected in the pop-up menu in the trace card. Acquisition begins when an edge is detected on one of the TTL input lines.

Parameters:
• `TTLbit`: the bit number of the TTL input where the triggering edge will be detected (0-7).
• `edge`: trigger on rising (1) or falling (0) edge.
• `timeout`: waits for an edge for this many seconds before returning with a timeout error.

XCMDs: `AcqWaveOnTTL`

Comments:
Similar to Single A/D Thresh, but looks for a trigger in the form of a digital signal instead of an analog threshold crossing. Use this button to acquire signals when an external TTL pulse (which could be generated by the on-board counter/timer chip) triggers a response you wish to capture.

9. Multiple on TTL

Function:
Similar to Single A/D on TTL but acquires several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `TTLbit`: the bit number of the TTL input where the triggering edge will be detected (0-7).
• `edge`: trigger on rising (1) or falling (0) edge.
• `timeout`: waits for an edge for this many seconds before returning with a timeout error.

XCMDs: `AcqWaveOnTTL`

10. Single A/D pre TTL

Function:
Acquires a single wave from the A/D channel selected in the pop-up menu in the trace card. Acquisition begins when an edge is detected on one of the TTL input lines. In addition, pre-triggering allows you to capture a segment *before* the edge occurs.

Parameters:
• `TTLbit`: the bit number of the TTL input where the triggering edge will be detected (0-7).
• `edge`: trigger on rising (1) or falling (0) edge.
• `timeout`: waits for an edge for `timeout` seconds before returning with a timeout error.
• `preTrig`: length of segment to be captured before trigger edge (in µs).

XCMDs: `AcqWavePreTTL`

Comments:
Similar to Single A/D on TTL, but allows you to capture a portion of your signal before the trigger. For example, you may have an external analog threshold detector which will trigger an acquisition. You are also interested in what happened immediately before the detector was triggered. Use this button to capture a segment before the TTL edge transition.

WaveTrak Buttons

11. Multiple pre TTL

Function:
Similar to Single A/D pre TTL but acquires several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `TTLbit`: the bit number of the TTL input where the triggering edge will be detected (0-7).
• `edge`: trigger on rising (1) or falling (0) edge.
• `timeout`: waits for an edge for this many seconds before returning with a timeout error.
• `preTrig`: length of segment captured before trigger edge (in µs).

XCMDs: `AcqWavePreTTL`

12. Single A/D + TTL

Function:
Acquires a single wave from the A/D channel selected in the pop-up menu in the trace card. Generates a simultaneous TTL pulse on one TTL output. Pre-triggering allows a segment to be captured before the onset of the TTL pulse.

Parameters:
• `TTLbit`: the bit number of the TTL output where pulse will be generated (0-7).
• `preTrig`: length of segment captured before TTL pulse (in µs).
• `pulseWidth`: duration of TTL pulse (in µs).

XCMDs: `AcqWaveTTL`

Comments:
Use this button to stimulate a system and acquire the response. Pre-triggering (i.e. the acquisition is triggered before the pulse is generated) allows you to capture what happened immediately preceding the TTL pulse. The durations

of `preTrig` and `pulseWidth` are adjusted to integral multiples of the sampling interval.

13. Multiple + TTL

Function:
Similar to Single A/D + TTL but acquires several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `TTLbit`: the bit number of the digital output where pulse will be generated (0-7).
• `preTrig`: length of segment captured before TTL pulse (in µs).
• `pulseWidth`: duration of TTL pulse (in µs).

XCMDs: `AcqWaveTTL`

14. Single A/D + Timer

Function:
Acquires a single wave from the A/D channel selected in the pop-up menu in the trace card. Generates a simultaneous TTL pulse on one counter/timer output. Pre-triggering allows a segment to be captured before the onset of the TTL pulse.

Parameters:
• `timerChannel`: the number of the timer channel where pulse will be generated (1-4).
• `preTrig`: length of segment captured before TTL pulse (in µs).
• `pulseWidth`: duration of TTL pulse (in µs).

XCMDs: `AcqWaveTimer`

Comments:
This button is very similar to Single A/D + TTL, but offers more control over the timing of the digital pulse. Pulses are generated at the corresponding OUT pin of the timer chip (see your MacADIOS II manual). Because the pulse is generated directly by the AM9513 counter/timer chip, `preTrig` and `pulseWidth` need not be integral multiples of the sampling interval. In fact,

unlike with Single A/D + TTL, the pulse can extend past the end of the acquisition window. We recommend that you use XCMDs that generate TTL pulses using the timer chip unless you have other reasons to use the TTL output port.

15. Multiple + Timer

Function:
Similar to Single A/D + Timer but acquires several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `timerChannel`: the number of the timer channel where pulse will be generated (1-4).
• `preTrig`: length of segment captured before TTL pulse (in µs).
• `pulseWidth`: duration of TTL pulse (in µs).

XCMDs: `AcqWaveTimer`

16. SingleAvg A/D + TTL

Function:
Averages a number of waves from a single A/D channel selected in the pop-up menu in the trace card. Generates a simultaneous TTL pulse on one digital output. Pre-triggering allows a segment to be captured before the TTL pulse is generated.

Parameters:
• `TTLbit`: the bit number of the digital output where pulse will be generated (0-7).
• `preTrig`: length of segment captured before TTL pulse (in µs).
• `pulseWidth`: duration of TTL pulse (in µs).
• `nAvg`: number of waves to be averaged
• `period`: averaging period (time between averages, in µs)
• `lock`: if FALSE, command-period will terminate averaging; if TRUE, Mac will be locked out until entire series is complete.

XCMDs: `AvgWaveTTL`

Comments:

This button is an extension of Single A/D + TTL. The acquisition and pulse generation are identical except that several waves are averaged before the result is returned. The time between acquisitions (and also the period between pulses) is set by `period`, and the number of waves to be averaged by `nAvg`. If you need maximum precision (to within 1 µs) in the timing of the period, set `lock` to TRUE; however, your Mac will be locked out until the series is complete. If you anticipate long averages where you might need to prematurely terminate the series, then set `lock` to FALSE; this will allow command-period to stop the averaging and return with the result accumulated up to that point. The disadvantage is that the jitter between periods might be as high as ±500 µs (the pulse width and sampling intervals will be accurate however).

17. MultipleAvg + TTL

Function:
Similar to SingleAvg A/D + TTL but averages several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `TTLbit`: the bit number of the digital output where pulse will be generated (0-7).
• `preTrig`: length of segment captured before TTL pulse (in µs).
• `pulseWidth`: duration of TTL pulse (in µs).
• `nAvg`: number of waves to be averaged
• `period`: averaging period (time between averages, in µs)
• `lock`: if FALSE, command-period will terminate averaging; if TRUE, Mac will be locked out until entire series is complete.

XCMDs: `AvgWaveTTL`

18. SingleAvg A/D + Timer

Function:
Averages a number of waves from a single A/D channel selected in the pop-up menu in the trace card. Generates a simultaneous TTL pulse on one output of the counter/timer chip. Pre-triggering allows a segment to be captured before the TTL pulse is generated.

Parameters:
• `timerChannel`: the number of the timer channel where pulse will be generated (1-3).
• `preTrig`: length of segment captured before TTL pulse (in µs).

• `pulseWidth`: duration of TTL pulse (in µs).
• `nAvg`: number of waves to be averaged
• `period`: averaging period (time between averages, in µs)
• `lock`: if FALSE, command-period will terminate averaging; if TRUE, Mac will be locked out until entire series is complete.

XCMDs: `AvgWaveTimer`

Comments:
This button is similar to SingleAvg A/D + TTL, but offers more control over the timing of the digital pulse. Pulses are generated at the corresponding OUT pin of the timer chip (see your MacADIOS II manual). Because the pulse is generated directly by the AM9513 counter/timer chip, `preTrig` and `pulseWidth` need not be integral multiples of the sampling interval; the pulse can extend past the end of the acquisition window. The time between acquisitions (and also the period between pulses) is set by `period`, and the number of waves to be averaged by `nAvg`. If you need maximum precision (to within 1 µs) in the timing of the period, set `lock` to TRUE; however, your Mac will be locked out until the series is complete. If you anticipate long averages where you might need to prematurely terminate the series, then set `lock` to FALSE; this will allow command-period to stop the averaging and return with the result accumulated up to that point. The disadvantage is that the jitter between periods might be as high as ±500 µs (the pulse width and sampling intervals will be accurate however). We recommend that you use the `AvgWaveTimer` XCMD for averaging and delivering single TTL pulses.

19. MultipleAvg + Timer

Function:
Similar to SingleAvg A/D + Timer but averages several channels simultaneously.

Parameters:
• `startMUX`: first A/D channel in series.
• `endMUX`: last A/D channel (inclusive) in series.
• `timerChannel`: the number of the timer channel where pulse will be generated (1-3).

- `preTrig`: length of segment captured before TTL pulse (in μs).
- `pulseWidth`: duration of TTL pulse (in μs).
- `nAvg`: number of waves to be averaged
- `period`: averaging period (time between averages, in μs)

• `lock`: if FALSE, command-period will terminate averaging; if TRUE, Mac will be locked out until entire series is complete.

XCMDs: `AvgWaveTimer`

20. Single A/D+Timer+Baseline

Function:
Acquires a single wave from the A/D channel selected in the pop-up menu in the trace card. Generates a simultaneous TTL pulse on one counter/timer output. Pre-triggering allows a segment to be captured before the TTL pulse is generated. Computes the mean analog level of segment (whose length is set by `preTrig`) prior to TTL pulse and stores it in the 'Trace baseline' entry in the 'Readings' field.

Parameters:
• `timerChannel`: the number of the timer channel where pulse will be generated (1-4).
• `preTrig`: length of segment captured before digital pulse (in µs).
• `pulseWidth`: duration of digital pulse (in µs).

XCMDs: `AcqWaveTimer`

Comments:
This button is very similar to Single A/D + Timer, but, in addition, computes the baseline before any stimulus pulse is delivered. This function is useful if you have an unstable DC baseline and need to compute the amplitude of your signal with respect to this changing DC level; a new level will be captured immediately prior to every acquisition. Note that the baseline drawn in the display window automatically reflects the one that is measured and entered into line 2 of the 'Readings' field.

21. Autocycle

This button is a template for creating functions that repeatedly perform acquisitions at predefined intervals. In its present version, the 'Autocycle' button will repeatedly execute the script of the 'Single A/D' button (by sending it a mouseUp message) at the interval (in seconds) shown in the pop-up menu below it in the trace card (see Fig. 6-3). Comments in the script tell you where you can place your own instructions to be executed repeatedly. To abort the autocycle function, press and hold down the mouse button. Due to the significant overhead involved in creating and linking trace cards, and writing the data to disk, the minimum autocycling period is about 2 sec on a Mac IIfx. This could be longer if you have many points/wave, a large stack or a slower CPU or hard disk.

Measurement

The measurement buttons report on the state of signals applied to either the analog or digital inputs. Although they do not acquire traces, these buttons do require information from pop-up menus in the trace cards and some of them write results to the REPORT field in the current trace. Therefore, the measurement buttons should only be used from within a trace card.

1. Show Single Mean

   Function:
   This button measures the mean level at the A/D channel selected in the pop-up menu in the trace card, and displays the result (in mV) in the message box. The current **sampleInterval** and **npoints** are used to acquire a signal, then compute the mean.

   XCMDs: `AcqMean`

   Comment: this button uses the `translateToReal` handler in the stack script, which does the opposite of `translateToBinary` (see 'Single A/D Thresh' above). When a binary integer (the computed mean) is passed along with the full scale analog values and binary coding, this handler returns the equivalent real value in mV (see the chapter on Handlers for more information).

2. Show Multiple Means

   Function:
   This button is similar to Show Single Mean except that mean levels from several A/D channels are measured. The result is written to the REPORT field in the current trace card.

Parameters:
- `startMUX`: first A/D channel in series.

• `endMUX`: last A/D channel (inclusive) in series.

XCMDs: `AcqMean`

3. Read Input Byte

Function:
This button reads the current 8-bit pattern at the digital input port and writes the result (0-255) in the message box.

XCMDs: `ReadTTLbyte`

Comment: you will most likely add other steps to isolate the values of one or more specific bits at the input port and act depending on their values.

4. Read Input bit

Function:
This button reads the value of a single bit from the digital input port and writes the result (0 or 1) in the message box.
Parameters: a dialog box asks you which bit to read.

XCMDs: `ReadTTLbit`

Comment: you will most likely add other steps to act differently depending on the value of the bit.

5. Show RMS

Function:
This button measures the RMS (root-mean-square) level of the signal the A/D channel selected in the pop-up menu in the trace card, and displays the result (in mV) in the message box. The current **sampleInterval** and **npoints** are used to acquire a signal, then compute the RMS value.

XCMDs: `AcqWave, GetWaveStats`

Comment: the `GetWaveStats` XCMD computes a variety of useful statistics on waves; see the chapter on WaveTrak XCMDs/XFCNs for details.

6. Show AC RMS

   Function:
   This button measures the AC RMS value (i.e. RMS with the DC level removed, also equal to the standard deviation of all points in a wave) of the signal at the A/D channel selected in the pop-up menu in the trace card, and displays the result (in mV) in the message box. The current **sampleInterval** and **npoints** are used to acquire a signal.

   XCMDs: `AcqWave, GetWaveStats`

   Comment: the `GetWaveStats` XCMD computes a variety of useful statistics on waves; see the chapter on WaveTrak XCMDs/XFCNs for details. For example, the AC RMS value is useful for computing the AC noise of a system whose output is floating on a non-zero DC level.

Signal Generation

This collection of buttons generates a variety of digital and analog signals using the on-board D/A converter, digital outputs and the counter/timer chip outputs.

1. StartPulseTrain

   Function:
   Instructs the counter/timer chip to generate a continuous train of positive (active high) TTL pulses.

   Parameters:
   • `period`: time between pulses (µs)
   • `pulseWidth`: pulse width (µs)
   • `counter`: which counter to generate pulses (1-5)

XCMDs: `StartPulseTrain`.

Comment: Because the counter/timer chip can generate repetitive digital signals on its own, the pulse train will continue after the button completes execution, until the StopPulseTrain button is clicked. The `StartPulseTrain` XFCN uses the next lower numbered counter channel internally, to help generate TTL pulses with the greatest possible accuracy (e.g. if you request counter 2 in the parameters, counter 1 is used internally and its output will be a TTL lo. Counter 1 uses counter 5). However, because the AM9513A chip has 16-bit counters, when long periods are requested (i.e. seconds), the resolution of very narrow pulses (i.e. tens of µs) will be limited. If the true pulse width and/or period differ from those requested, a silent error (60) will be returned in **XCMDErr**. The value of the XFCN returns a comma-delimited list containing the true width of the pulse, the true period, and the available resolution for both pulse width and period (all in µs). If either the true pulse width or the true period would have differed by more than 30% from that requested in the parameters, the pulse train is not generated and the function returns with an error (76). See the description of the `StartPulseTrain` XFCN in the chapter on WaveTrak XCMDs/XFCNs for more details.

2. StopPulseTrain

Function:
Stops pulse generation that was begun by StartPulseTrain.

Parameters:
• `counter`: which counter to turn off (1-5)

XCMDs: `StopPulseTrain`.

Comment: Because the `StartPulseTrain` XFCN uses the next lower numbered counter channel (e.g. if you request counter 2 in the parameters, counter 1 is used internally and its output will be a TTL lo. Counter 1 uses counter 5), StopPulseTrain disarms both the counter requested in the parameters list and the next lower numbered counter as well.

3. Write Output bit

Function:
Sets (1) or resets (0) a bit in the TTL output port.

Parameters: a dialog box asks you which bit to change.

XCMDs: `WriteTTLbit`

Comment: you will most likely include the bit set/reset function as part of a larger script.

4. Write Output byte

   Function:
   Writes a byte (8 bits at a time) to the TTL output port.

   Parameters: a dialog box asks you what value to write.

   XCMDs: `WriteTTLbyte`

   Comment: you will most likely include this function as part of a larger script. The byte is represented as an unsigned 8 bit value, ranging from 0 (all bits 0) to 255 (all bits 1).

5. DAC pulse.

   Function:
   Generates an analog pulse at the D/A converter output. Simultaneously toggles one of the TTL output bits.

   Parameters:
   • `DACchannel`: the D/A channel to be pulsed (0 or 1)
   • `DACpulseLevel`: the analog level of the pulse (mV)
   • `pulseWidth`: pulse width in µs
   • `DACpostLevel`: analog level after pulse
   • `TTLbit`: toggle bit number (0-7, or -1 for none)

   XCMDs: `DACpulse`

   Comment: A TTL bit is toggled for the same duration as the analog pulse; this is useful for triggering other equipment such as a scope to capture the event. You can place the `DACpulse` command in a loop to generate a repetitive analog pulse train.

Analysis and Wave Math - Single Traces

These buttons demonstrate WaveTrak's powerful collection of analysis, mathematical and digital signal processing functions. Many of these buttons operate on single traces, and some give examples of analyzing a series of traces. The latter should be pasted into the root card background. These buttons serve a dual purpose: they perform very useful operations as they are, but even more importantly, they help teach you how to write your own signal processing scripts using the extensions provided in the WaveTrak toolbox. We encourage you not only to use these buttons, but also to open them up and examine how easy it is to script complex operations.

The techniques used in numerical processing of sampled signals (such as the FFT) are extremely powerful tools. Many of these buttons rely on the FFT (filters, spectral analysis), and we recommend that you familiarize yourself with some of the fundamentals so you can fully exploit the capabilities of these functions.

The following buttons operate on single traces and must be pasted into the background layer of trace cards.

1. Make Sine

   Function:
   Example of how to create two sine waves and draw them in the display window.

   Parameters:
   • `cycles`: no. of cycles in each sine wave
   • `phase`: phase in degrees
   • `offset`: offset in mV
   • `resultType`: -12 is a signed 12-bit wave

   XCMDs: `InitWaveSin`

   Comment: This button is an example of how to use some of WaveTrak's math functions. Waves acquired by the on-board A/D converter are always 12-bit binary; the only choice you have is whether to encode them as signed (-

2048 to +2047) or unsigned (0 to 4095) binary integers. Mathematically synthesized waves can be represented in a number of different formats, either as floating point or binary numbers. The latter can have different bits of precision (8, 12, 16 and so on), and can be signed or unsigned. The `resultType` parameter allows to select what type of result you want, depending on your application. See the scripting chapter for more details on WaveTrak data types.

2. Make Sine & Filter

Function:
Creates a sine wave, adds noise, then lo-pass filters (log roll-off) the combination according to the parameters set in 'Digital Filter Parameters' card.

Parameters:
• `cycles`: no. of cycles in each sine wave
• `phase`: phase in degrees
• `offset`: offset in mV
• `resultType`: -12 is a signed 12-bit wave

XCMDs: `InitWaveSin, InitWaveNoise, FilterWaveFFTloLog, AddWaves, AddWaveK`

Comment: This button is a more elaborate example of how to combine several functions to manipulate waves. It also demonstrates the power of digital filtering and its ability to extract signal from noise. Experiment with different filter parameters to get the response you need. The degree of filtering will depend heavily on the filter parameters in relation to the sampling rate of the existing trace. Don't forget that any functions (such as filters) using the FFT can only process waves consisting of `n` points, where `n` is an integral power of 2.

3. Peak Frequency

Function:
Determines the dominant frequency in the signal by computing the peak component in the power spectrum, and writes the result in the message box. Number of points must be a power of 2.

Parameters:
• `dB`: TRUE: compute frequency components in dB (log scale); FALSE: compute on a linear scale.

• `floor`: limit very small dB components to this value
• `windowType`: window function by which wave is multiplied before the FFT (1 = Hanning, 2 = Parzen, 3 = Welch).

XCMDs: `Window, AmplSpectrum, GetWaveStats, AddWaveK`

Comment: The frequency resolution will be limited by the sampling rate. With certain signals (such as trains of narrow pulses where the repetition frequency falls between the resolving limits of the sampling rate), the fundamental peak may in fact be smaller (though its area will be greater) than some harmonics, and this button may return an incorrect result. That's why it's always a good idea to examine the original signal in the time domain and its entire spectrum with the 'Freq Spectrum' command.

Tip:

Frequency spectra commonly have some values that are very small, so it's usually most convenient to compute them in dB, i.e. on a log scale. Windowing reduces "spectral leakage" (i.e. spurious frequency components) with waves that are discontinuous at their beginning and end.

---

4. Freq Spectrum

Function:
Computes the frequency (amplitude) spectrum of the trace and plots the result in the display window. Number of points must be a power of 2.

Parameters:
• `dB`: TRUE: plot frequency spectrum in dB (log scale); FALSE: plot on a linear scale.
• `floor`: limit very small dB components to this value.
• `windowFlag`: TRUE: window data before computing spectrum.
• `windowType`: window function by which wave is multiplied before the

FFT (1=Hanning, 2=Parzen, 3=Welch).

XCMDs: `Window, AmplSpectrum`

Comment: Frequency spectra commonly have some values that are very small, so it's usually most convenient to display them in dB, i.e. on a log scale. Even then, some components can be so small that they would result in most of the spectrum being vertically compressed at the top to accommodate all the tiny values. Limiting the smallest components with the `floor` parameter results in a more readable plot; -80 dB is a good choice, since in the real world, the noise level of most sources is rarely 80 dB below the signal. Windowing reduces "spectral leakage" (i.e. spurious frequency components) with waves that are discontinuous at the beginning and end.

5. Wave Stats

Function:
Computes various statistics (data type, mean DC level, DC and AC RMS, min & max values, peak-to-peak, encoded size) on the wave and writes result to REPORT field.

XCMDs: `GetWaveStats`

Comment: see the chapter on XCMDs/XFCNs for more details on the `GetWaveStats` XFCN.

6. Trace Area

Function:
Computes area under wave, prints result in message box

Parameters:
• `startTime`, `endTime`: compute area between these two time points.

XCMDs: `AreaWave`

Comment: area is computed around value saved as baseline (line 2 in Readings field). Resulting dimensions are Xunit•Yunit. This button only works for data in the time domain. Pass -1 in `endTime` to extend the computation to the end of the wave.

7. Trace Abs Area

Function:
Computes absolute (rectified) area under wave, prints result in message box.

Parameters:
• `startTime`, `endTime`: compute area between these two time points.

XCMDs: `AreaWave`

Comment: area is computed around value saved as baseline (line 2 in Readings field). Values below baseline are reflected above before summing (rectified). Resulting dimensions are Xunit•Yunit. This button only works for data in the time domain. Pass -1 in `endTime` to extend the computation to the end of the wave.

8. Integrate

Function:
Integrates trace about baseline, result normalized to ±2000 integer counts.

Parameters:
• `overlayFlag`: TRUE: overlay original and integrated waves in display window.
• `normalize`: TRUE: instruct `IntegrateWave` to return a normalized result (± 1 range), which is then scaled up to ±2000.

XCMDs: `IntegrateWave, MultWaveK`

Comment: This button only works for data in the time domain.

9. Differentiate

Function:
Differentiates trace, overlays original and differentiated waves.

Parameters:
• `overlayFlag`: TRUE: overlay original and differentiated waves in display window.

XCMDs: `DifferentiateWave`

Comment: This button only works for data in the time domain. Square waves will differentiate into very sharp impulses which might not all be plotted in the normal view. To ensure that you don't miss any sharp transients, increase the number of points plotted by choosing "Display points..." under the Analysis menu. Waves without sharp steps will differentiate into very low amplitude waves; you will need to magnify the view to examine their derivatives in detail.

10. Threshold Detect

Function:
Threshold detects a trace: all values ≥ `threshold` become 1 and all values < `threshold` become 0. Overlays original and threshold detected waves.

Parameters:
• `overlayFlag`: TRUE: overlay original and threshold detected waves in display window.
• `hysteresis`: true threshold will be adjusted by ±hysteresis/2 with each crossing.
• `threshold`: requested with a dialog box (mV).

XCMDs: `ThresholdWave`

Comment: The combination of `ThresholdWave` and `WaveToEventTable` are very useful for examining at which time point(s) along a wave an event occurred. You can combine other operations before thresholding (see the 'Peak Detector' button as an example). The `hysteresis` parameter is used to eliminate multiple threshold crossings with noisy waves, at the expense of some uncertainty in the time where the true threshold was crossed. Pass 0 if you don't want any hysteresis applied to the threshold level.

11. Lo-pass/Hi-pass Filter

    Function:
    Digitally filters the trace using the FFT. Overlays original and filtered waves.

Parameters:
• `overlayFlag`: TRUE: overlay original and filtered waves in display window.

XCMDs: `FilterWaveFFTloLog, FilterWaveFFThiLog`

Comment: Since the FFT is used for filtering, the number of points in the wave must be an integral power of 2. These buttons implement filters with a log roll-off; set the -3 dB points and the slope in the Digital Filters card. The `FilterWaveFFTloLin` and `FilterWaveFFThiLin` XFCNs implement linear roll-off filters. Experiment with each to see which type of filter gives you the best results. The degree of filtering will depend heavily on the filter parameters in relation to the sampling rate of the trace.

12. Lo-pass/Hi-pass + window

Function:
Digitally filters the trace using the FFT. Waves are multiplied by a window function before the FFT. Windowing is removed by dividing by the same window. Overlays original and filtered waves.

Parameters:
• `windowType`: window function by which wave is multiplied before the FFT (1=Hanning, 2=Parzen, 3=Welch).
• `overlayFlag`: TRUE: overlay original and filtered waves in display window.

XCMDs: `Window, FilterWaveFFTloLog, FilterWaveFFThiLog`

Comment: These two buttons are identical to those in 11 except that the wave is multiplied by a window function before filtering. Filtering a wave that is discontinuous at the beginning and end (i.e. the difference between the first and last points is large) tends to create spurious oscillations. Windowing will dramatically reduce such oscillations. One drawback is that the beginning and end of the data can be distorted due to floating point round-off errors when the window is applied and then removed. Experiment with various filter combinations to see which one gives you the best results. Since the FFT is used for filtering, the number of points in the wave must be an integral power of 2. The degree of filtering will depend heavily on the filter parameters in relation to the sampling rate of the trace.

13. Peak Detector

    Function:
    Returns a list of peaks in REPORT field. Overlays original and filtered waves.

    XCMDs: `FilterWaveFFTloLog, DifferentiateWave, WaveToEventList, WaveToYtable`

    Comment: This button is an good example of how WaveTrak functions can be combined to perform complex analyses. Peak detection is not trivial. Filtering the wave eliminates the many false peaks that would be generated if the raw data were simply differentiated. Keep in mind also that filtering will also heavily influence what a peak really is. Examine closely the tops of the sample sine wave in trace 1 before and after filtering with this button, and see how the reported peaks can differ slightly from what you would expect depending on the filter parameters. Notice also the advantage of having a floating point result after filtering; this allows the peak detector to pick out a point even though the integer representation yields several points having the same value at the top of the sine wave.

14. FIR Filter

    Function:
    Digitally filters the trace by implementing a finite impulse response non-recursive digital filter. Overlays original and filtered waves.

    Parameters:
    • `overlayFlag`: TRUE: overlay original and filtered waves in display window.

    XCMDs: `FilterWaveFIR`

    Comment: The FIR filter must be designed using another application, such as Igor Filter Design Lab (WaveMetrics, Lake Oswego, OR). The filter coefficients are stored in a field in the Digital Filters Card, and are copied to the global variable **FIRCoeffs**, which is passed to the filter XFCN. Any type of filter can be designed and its characteristics will be determined by its coefficients. See the discussion on the *Digital Filters Card* in the 'WaveTrak Cards' chapter, and the `FilterWaveFIR` XFCN in the 'WaveTrak XCMDs and XFCNs' chapter for more details.

Analysis and Wave Math - Traces under a Root

The following are examples of how to create buttons that operate not on a single trace, but on a group of traces under one root. Traces are normally grouped by their marks. These buttons must be pasted into the background of *root* cards, and not trace cards.

1. Plot Abs Area

   Function:
   Computes the absolute area under the wave for a series of traces with the same mark and plots result vs. time of trace acquisition on the Plot Card. Areas are normalized to a maximum of 1.

   Parameters:
   • `startTime`, `endTime`: compute area between these two time points in each trace.

   XCMDs: `AbsAreaWave`

   Comment: Use this button as a template and replace `AbsAreaWave` with other XFCNs to compute and plot other wave parameters.

2. Abs Area to Disk

   Function:
   Computes the absolute area under the wave for a series of traces with the same mark and writes the result in tab-delimited format to a disk file. Areas are normalized to a maximum of 1.

   Parameters:
   • `startTime`, `endTime`: compute area between these two time points in each trace.

   XCMDs: `AbsAreaWave`

   Comment: Use this button as a template and replace `AbsAreaWave` with other XFCNs to write reports to disk files. These files can then be opened by other applications such as spreadsheets or statistical programs for further analysis.

In Summary

- Pre-programmed WaveTrak buttons are stored in the Button Bank card.

- Buttons will operate normally only from root or trace cards. Therefore, *do not press buttons from within the Button Bank!*

- Some buttons were designed to work from root cards (those that operate on more that one trace), while others work only from trace cards (those that operate on a single trace).

- Copy (*don't cut*) and paste the buttons you need into the *background* layer of either a root or trace card.

- Use the existing button scripts as templates to script your own custom functions.

- You may need the optional WaveTrak AD version of the software to perform data acquisition and signal generation operations. Contact Ortex Systems for details.