



## BrainWare User's Guide

سازمان



*BrainWare32 User's Guide – Version 6.1*

© 1998, 1999 Jan Schnupp and TDT, Inc. All rights reserved.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without the express written permission of TDT.

**Licenses and Trademarks**

Microsoft, MS-DOS, Windows and Windows 95 are registered trademarks of Microsoft Corporation.

Printed in U. S. A.

Printed on 3/22/99





# Contents

## Organization of the Manual 0-1

### **Chapter 1 Introduction 1-1**

#### ***What Is BrainWare? 1-1***

BrainWare Capabilities 1-1

Hardware Support 1-1

#### ***Before You Begin 1-2***

What You Need 1-2

#### ***Installing the Software 1-2***

Requirements 1-2

Installation 1-3

Hardware Configuration 1-3

#### ***Quick Start Guide to BrainWare 1-4***

### **Chapter 2 Using BrainWare 2-1**

#### ***Main Menu Overview 2-1***

The Toolbar 2-1

File Menu 2-4

Recording Menu 2-6

Stimulus Grid 2-7

Options Menu 2-9

Window Menu 2-10

#### ***Window Overviews 2-11***

The Electrode Channel Window 2-11

The Data View (or Data File, or Data Set) Window 2-13

Cutting clusters 2-20

The Spike Shape Window 2-24

The Post Stimulus Time Histogram (PSTH) Window 2-25

#### ***BrainWare Stimulus Modules 2-31***

SigGen Files as BrainWare Stimulus Modules 2-32

Using SigGen Variables with BrainWare 2-34

SigGen Variable Types 2-34

**Data Collection 2-36**

Analysis Options 2-37

Analysis of a previous recording 2-37

Exporting Data 2-38

Timing Issues 2-43

Running the D/A and A/D on the Same Clock (microsecond spike timing) 2-43

**Chapter 3 Stimulus DLLs 3-1**

Writing Stimulus DLLs for BrainWare 3-2

The Stimulus Cycle 3-4

Stimulus DLL Functions 3-6

Writing stimulus DLLs with Delphi 3-7

Writing stimulus DLLs with MSVC++ 3-10

**DLL Function Definitions 3-13**

computeStimulus 3-13

DLLReadyToBuildNextStimulus 3-16

DLLReadyForNextSweep 3-17

triggerDLLStimulus 3-18

initializeStimulusDLL 3-19

setDLLOptions 3-20

tidyUpStimulusDLL 3-21

numDefaultConditions 3-22

defaultStimParam 3-22

numStimParams 3-24

parameterName 3-24

DAparameters 3-25

getMessageFunc 3-28

**DDE (Dynamic Data Exchange) with BrainWare 3-30**

**DDE Basics 3-30**

Client and Server 3-30

Using DDE with BrainWare 3-31

**Supported DDE Topics and Items 3-32**

DDE topic "Settings" 3-32

DDE topic "StimulusTable" 3-32

***Using DDE to set up BrainWare's Stimulus Grid 3-34***



# Organization of the Manual

- **Chapter 1 Introduction**  
Installation of BrainWare, hardware configuration, and Quick Start.
- **Chapter 2 Learning the Basics**  
Presents an overview of the menus and toolbar.
- **Chapter 3 Using BrainWare**  
Presents detailed information on how to use BrainWare for sound presentation and data acquisition and analysis.
- **Chapter 4 Stimulus DLLs**  
Shows how to program Dynamic Link Libraries for producing stimuli in BrainWare.



# **Chapter 1 Introduction**

## **What Is BrainWare?**

BrainWare is a software package for sound presentation and extracellular electrophysiological recording, providing spike acquisition, sorting, and analysis.

## **BrainWare Capabilities**

BrainWare will play SigGen signals as well as user-programmed DLL (Dynamic Link Library) stimulus files. BrainWare provides facilities for multi-channel recordings and recordings from tetrodes.

BrainWare enables the operator to discriminate single unit action potentials from the digitized electrode signals by visually cutting clusters in a “feature space” representation of action potential candidates identified by the software. Recorded data is automatically structured in collections of “sweeps” (responses to repeated stimulus presentations). The response strength of each sweep collection is also automatically quantified and displayed on-line in the data file’s DataView Window.

Various modes of displaying the data are available. Commands to copy graphs and data to the clipboard and to export data to ASCII-readable files or binary files that can be read from programs like Matlab allows BrainWare to be used for data collection, organization, presentation and publication.

## **Hardware Support**

BrainWare supports TDT's System II instrumentation, including any combination of TDT's D/A converters, A/D converters (except the DD1), programmable attenuators, waveform generators, cosine switches and programmable filters. You can play simple, single channel stimuli or complex stimuli on two channels.

## ***Before You Begin***

### **What You Need**

*See your Microsoft Windows documentation.*

- Windows fundamentals

You should be comfortable with Windows basics: starting Windows; using the mouse; manipulating windows; opening, closing, and saving files.

*See the SigGen manual*

- Basic SigGen concepts

You should recognize the term *SigGen Index (SGI)* and understand how SigGen variables work.

## ***Installing the Software***

### **Requirements**

In order to run BrainWare, you must have the following:

- Microsoft Windows 95
- A monitor with at least VGA resolution graphics. Super VGA (1024 x 768) resolution graphics highly recommended
- TDT's AP2 Array Processor
- TDT's XBUS hardware including
  - D/A converter
  - A/D converter
  - PA4 Programmable Attenuator (optional; for controlling sound level)
  - WG1/2 Waveform Generator (optional; for producing continuous masking tones or noise).
  - PF1 Programmable Filter (optional, for filtering sounds)
- SigGen software for generating signal files

## Installation

### *To install BrainWare*

1. Make sure your TDT hardware (including AP2 Array Processor and XBUS devices) is installed and functioning properly.

Refer to the *System II Installation Guide*.

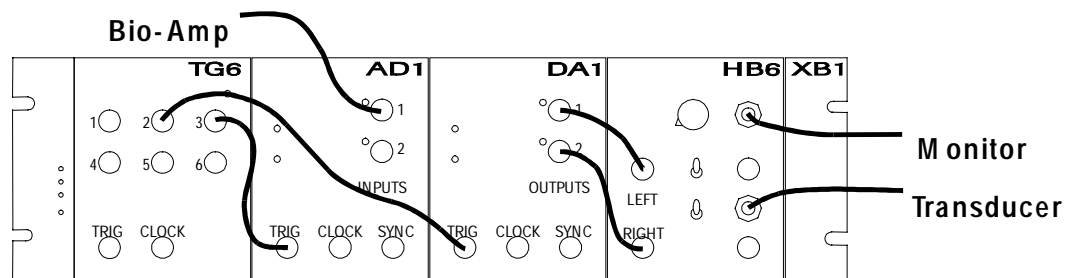
2. Insert the BrainWare diskette into drive A: or B:.
3. Run **setup.exe** to start the installation program.

## Hardware Configuration

When using SigGen signals, BrainWare triggering should be setup to use the TG6 for triggering as shown in the diagram below.

Note that your configuration will likely be more complicated and involve connecting the D/A output to a programmable attenuator (PA4) and an amplifier (such as the TDT HB6). This diagram is intended just to show the connection of the trigger line.

2. TG6 Triggering: The TG6 triggers both the D/A and the A/D. Connect TG6 channel 2 to the TRIG input on the D/A, and TG6 channel 3 to the TRIG input on the A/D.



## Quick Start Guide to BrainWare

The following is a step-by-step example of how to record a (very small) frequency / response area with BrainWare. It should help users unfamiliar with BrainWare to get up to speed very quickly.

### Run BrainWare

When you launch BrainWare the program will try to detect TDT System II hardware and initialize it. The status line at the bottom of the main window will tell you whether initialization was successful or not. If it was successful BrainWare will show a Stimulus Grid, and you can start producing search stimuli and record data. If no TDT hardware was detected or initialization failed there will be no Stimulus Grid, but you can still open previously recorded data files for reanalysis.

### Load SigGen Signal

On the Stimulus Grid Window click the "Configure Stimulus Modules" button. A pop-up menu will appear. Choose "Add Module". Choose the SigGen file "tune.sig". (Alternatively you could load one or several of the provided sample stimulus DLLs, like "calibton.dll". [Click here to find out more.](#))

### Hardware Connections

No TG6: BrainWare triggers the D/A through software and the A/D is triggered by the SYNC pulse from the D/A. Make sure that the D/A and A/D are connected as shown in the figure below.

TG6 Triggering (using SigGen files): The TG6 triggers both the D/A and the A/D. Connect TG6 channel 2 to the TRIG input on the D/A, and TG6 channel 3 to the TRIG input on the A/D.

Connect the output of your DA module to a transducer via appropriate amplifiers / headphone buffers, and, if desired, anti-aliasing filters.

Connect the output of your microelectrode amplifier to input 1 on your AD module. (For test runs and when trying to familiarize yourself with BrainWare it is useful to have spikes on a tape recorded that can be played to the AD module). For optimal operation you should try to adjust your microelectrode amp so as to achieve an amplitude between 2-4 Volts for the largest Spikes in the trace fed to the AD module.

### Search

Choose Recording | Single Channel | Channel 1 on the BrainWare main menu. An Electrode Channel Window appears.

Click the Search button on the main window toolbar. BrainWare should now present the stimulus selected in the Stimulus Grid in a continuous loop and read sweeps of data, which are shown as green traces in the Channel Window.

You can choose a different search stimulus simply by clicking on a different line on the Stimulus Grid, and advance your electrode until there appears to be driven activity in the Channel Window. Adjust the positive and negative trigger levels on the Channel Window so that action potentials cross both the upper and the lower red trigger line in the window, but keeping the noise band well inside the trigger levels. If you set the trigger levels too close to the noise you will record a lot of spurious activity that will be hard to sort. In most cases it is preferable to record only a few units with high signal / noise. You can switch the Channel Window to Hash display to see a raster plot of the last seven recorded sweeps. That may help identify promising electrode positions.

## Record Data

Once your electrode is at a promising position click the Record button on the toolbar. You will be prompted to give a file name for the new data file. BrainWare will suggest the file name "r00001e1.src" for "recording #1 on electrode #1". Click "OK".

BrainWare will create the new data file and record a number of data sweeps with the current stimulus. The number of sweeps recorded is set in the Sweep dialog (menu Recording | Sweep Params ...). A Dataset Window and a Feature Space window will appear to display the recorded data. A number of measurements ("features") have been taken from every spike that exceeded both thresholds during recording, and the Feature Space window shows a point for every recorded spike.

## Cut Clusters

Spikes from a single unit should have very similar action potential shapes and should therefore form clusters in the Feature Space window. Double click on the Feature Space window to start delineating a cluster in feature space. Drag and click once to finish delineating the new cluster. A data point should appear in the Dataset Window, indicating the response strength statistics for the new cluster. Click on the Dataset Window with the right mouse button. A pop-up menu appears. Select "Plot | On X Axis | freq [Hz]". Now try selecting a different stimulus on the Stimulus Grid and click on the Record button on the toolbar again.

With the right mouse button on the Dataset Window choose "Show | Psth" and "Show | Shapes" to view the post stimulus time histograms and action potential shapes associated with the data points in your file.

If the shapes suggest that your cluster is multi-unit, you may try to adjust the cluster boundaries in the Feature Space window. You may be able to improve discrimination by investigating other dimensions of feature space (click on the Feature Space window with the right mouse button and choose different "X Axis -" or "Y Axis Features").

Once you are happy with your discrimination (don't spend a lot of time on it, you can always change boundaries later in a reanalysis) you are ready to record the whole frequency response area. Before closing the current Dataset Window choose File | New on the main menu. Note that this creates a new data file that preserves the cluster boundaries you just set in the previous file. The old Dataset Window and all associated feature space, PSTH and shape windows will close and you will be prompted whether you want to save the data to disk. On the new Dataset Window with the right mouse button choose Plot | On X Axis | freq [dB], then Plot | On Y Axis | level [dB] and finally Plot | LogX.

### **Batch Record Data for Entire Response Area**

Now click on the Batch button on the toolbar. BrainWare will record data for all the stimulus combinations shown in your Stimulus Grid. The presentation order is normally "random interleaved", but other orders are possible (see menu Recording | Sweep Params). The data set window will produce symbols of different size to show the differences in response strengths. You may have to adjust the size of the symbols by "right-clicking" on the Dataset Window and choosing Options | "Surface" Plots ...

Once recording of the response area is done you could load other signal files to the Stimulus Grid to further investigate the units at your current electrode position. Once you are done recording from a particular position close the Dataset Window and start searching for the next position.

# Chapter 2 Using BrainWare

This chapter first presents an overview of the main menu choices and their specific functions. Then the window views of BrainWare are described to show you how to load a SigGen signal and configure BrainWare for playing, recording, and analyzing.

## Main Menu Overview

### The Toolbar



Clicking here is the same as the File|Open menu item



Clicking here is the same as the File|Save menu item



Copy graph in active child window to the clipboard

Fnt

Select default font for graph axis labels



Move to the previous dataset



Move to the next dataset



Move to the previous cluster



Move to the next cluster



Search

Present search stimuli



Record

Start the record protocol



Batch

Start the batch recording protocol



Pause

Pause the current protocol

R/L

Used to indicate which hemisphere the data are recorded from recording side

This button is used to note which side (hemisphere) of the brain the recordings were made from. For free-field receptive field mappings BrainWare may use this to express coordinates contralateral to the recorded side with negative degree values, as per convention.



Clicking here is the same as the File|Exit menu item

## Keyboard Shortcuts

arrow keys	browse through the currently active data file, selecting the next or previous dataset (left / right) or the next or previous cluster (up / down)
F1	Open Help pages (context sensitive online help)
Ctrl + C	<b>Copy</b> the graph from the current Dataset, FeatureSpace, PSTH, or SpikeShape window to the clipboard
Ctrl + E	<b>Export</b> all spike trains for all clusters for all open data files in format " <u>spike times as binary</u> "
Ctrl + L	<b>Loop Fill</b> <u>stimulus grid</u> . (The stimulus grid must be in edit mode).
Ctrl + R	In the <u>PSTH Window</u> , switches <b>Raster</b> Plot Mode on / off
Ctrl + S	<b>Save</b> the currently active datafile
Ctrl + Shift + S	<b>Save All</b> currently open datafiles
Ctrl + <Number>	Activate / Show <u>Electrode Channel Window</u> for corresponding electrode Channel
Ctrl + Shift + <Number>	Activate / Show <u>Electrode Channel Window</u> for corresponding <i>Tetrode</i>
Ctrl + Tab	Focus on next Child Window.

## File Menu

The File menu provides commands for creating new files, opening existing files, saving files, printing files, and exiting the application.

### Open

The File|Open command displays the Open File dialog box so you can select a spike record file for post recording analysis.

### Save

The File|Save command saves changes to the active spike data file to disk.

### Save As

Similar to the File|Save command, the File|Save As command allows you to save the active spike data file to disk, but under a new filename.

This command can also be used for exporting data to ASCII format, for subsequent analysis by other software packages.

In order to do this choose one of the several possible export formats in the “Save as type” box of the Save dialog window. The data will be saved in a file with extension “.txt” or “.bin”.

### Close

Close the current file. If any changes have been made you will be prompted to save the data before closing the file.

### Close All

This command this closes **all** open spike data files. You will be prompted to save any changes.

### Delete

This command deletes the current spike data file from disk. The user will be prompted to confirm that he/she really wants to delete the data file.

### New

This command is only available during recording and when data files are associated with active electrode channels. The command will associate new data files with **every** active electrode channel and copy the cluster boundaries from the current to the new data files. This makes it possible to continue recording from the same units at the same recording site, but into new data files.

**Associate**

Only available during recording, not at post-hoc analysis. Associates an existing, previously recorded data file with an electrode channel. Make sure you select (“activate”) the Channel Window before choosing File|Associate. Associate allows further data to be recorded and added to the file.

**Exit**

The File|Exit command exits BrainWare. You will be asked to confirm and prompted to save any modified data files.

## Recording Menu

The Recording menu is used to activate electrode channels and provides access to the data acquisition parameters dialogs.

The Recording Menu is not accessible if no acquisition hardware was detected at startup.

**Single Channel...** Activates an A/D converter channel for single channel data acquisition and opens a Channel Window.

**Tetrode Channel...** Activates four consecutive A/D converter channels for tetrode data acquisition and opens a Channel Window.

**Sweep params...** Opens the Sweep Parameters dialog box. Only available when data acquisition was detected and initialized at startup present.

### Use this dialog box to indicate:

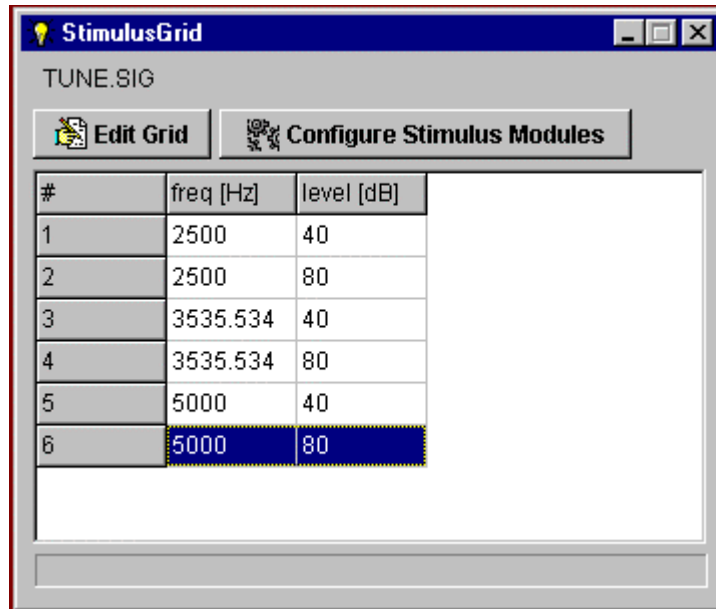
- **Repeat Rate:** the minimum inter-stimulus interval you require. Note that this is a guaranteed minimum, NOT a fixed value. BrainWare may take more time than indicated in the interval to get ready for the presentation of the next stimulus.
- **Sweep Length:** how long the sweeps of data recorded are supposed to be. BrainWare version 6.0 currently sets an arbitrary upper limit at 10 seconds.
- **# of repeats:** how many times you would like each stimulus to be presented in the next recording cycle.
- **Batch Presentation Order**
  1. **interleaved** presents the stimuli in the order given in the Stimulus Grid one at a time, restarting at the beginning of the list until the number of repeats specified have been obtained.
  2. **random interleaved** presents the stimuli from the Stimulus Grid in a randomized order. The order is reshuffled before the next repeat.
  3. **block:** each stimulus in the Stimulus Grid will be presented “repeats” times in a row before the next stimulus is used.
  4. **sequence file:** the order of presentation of stimuli is defined by a separate sequence file. The sequence file must be ASCII, one number per line. A File Open dialog box will be shown to allow the user to choose the sequence file. Unlike for the other sequence conditions, the number of repeats specified in the Sweep Parameters dialog will be ignored when sequence files are used: the sequence in the file will be presented once only and should include any desired repeat presentations. Thus, the sequence ‘1-2-3-2-1’ would lead to the creation of three datasets and recording five sweeps.

**Record** Equivalent to clicking the Record Button. Records data for selected stimulus from Stimulus Grid.

**Batch** Equivalent to clicking the Batch Button. Records data for all stimuli from Stimulus Grid.

**Pause** Equivalent to click the Pause Button. Pauses recording.

## Stimulus Grid



### Stimulus Parameter Table

This grid shows the stimulus parameter values to be used during recording. Select ("highlight") the stimulus you would like to use for searching and initial recording simply by clicking on the relevant line in the grid. BrainWare will record the stimulus parameters along with your data and use them to provide on-line plots of tuning curves etc. in the Dataset Window.

### Configure Stimulus Modules

Clicking this button will reveal a pop-up menu which lets you add or remove stimulus DLL (\*.dll) or SigGen stimuli files to the stimulus cycle, or to set stimulus DLL options.

The menu associated with this button also lets you save the current stimulus configuration. If you save the configuration, the names of the currently loaded stimulus modules, the contents of the stimulus parameter table, as well as the settings of the Sweep parameter dialog are store in an ASCII-readable configuration file (file extension ".s\_c"). You can restore the configuration later by loading the configuration file. You may wish to prepare a number of configuration files before the start of the experiment. You can then retrieve the configurations very quickly and run through a "battery" of tests in a short time.

This "Configure Stimulus Modules" Button is disabled while stimulation or data acquisition is ongoing.

### Edit Grid

Click the Edit button to fill this table with the stimulus parameters you wish to use during recording. These parameters will be sent to your stimulus DLL when Search or Record operations call the DLLs function makeStimulus.

Click the Edit Button again to end editing.

The following key strokes facilitate filling / editing the Stimulus Grid:

**Ctrl + V (Paste)** will paste text data from the windows clipboard. You might, for example, hold the parameter tables for a number of acquisition runs in a spreadsheet or text document, to keep a number of different "paradigms" on file.

**Insert** will insert a line in the table. The inserted line will be a copy of the current line.

**Shift + Delete** will delete the current line.

**Ctrl + Delete** will delete text in the current field, from the cursor position to the end of the field.

**Ctrl+Shift+Delete** will clear the entire Stimulus Grid.

**Ctrl+L** ("Loop fill") opens a dialog that automatically fills the stimulus grid with stimulus parameter ranges.

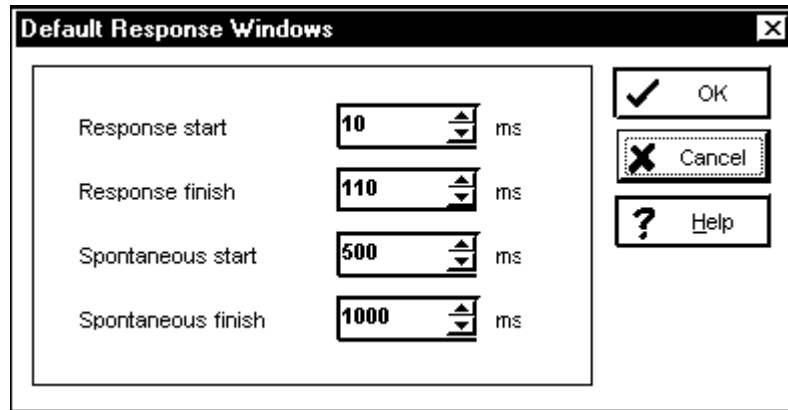
If you are using the Stimulus Grid with SigGen Stimuli, the values for the variable stimulus parameters are imported from the SigGen stimulus file. When the Edit Button is pressed, all of the variables will be changed to Dynamic and may then be edited.

## Options Menu

Various options and customizations are available from this menu.

<b>Artefact Rejection</b>	Determines how action potential candidates are scanned from recorded voltage traces.
<b>Analysis</b>	Set options for the calculation of spike count statistics.
<b>Autosave...</b>	Sets automatic data saving to disk after a set number of sweeps.
<b>Bin size...</b>	Set default bin size for PSTH and CCH windows.
<b>Cluster bounds same in all sets</b>	When checked, this option will keep the cluster boundaries the same for all data points (conditions) in one datafile (i.e. changes in a Featurespace Window affect all feature spaces in the file). This option should be kept checked unless a batch was recorded in a Block sequence and it is apparent that clusters have drifted.
<b>Cluster Colours...</b>	Change colors used to color code different clusters in graphic displays
<b>Font...</b>	Change the font used to label axes in the graphic display windows
<b>Reset Rec. Counter</b>	<p>Reset recording counter. The recording counter is used to automatically generate default file names every time you generate a new data file. Selecting this option will reset the counter to one.</p> <p>You do not have to use the Rest Recording Counter Menu Option to reset the Recording Counter. If you do not accept the default file name that BrainWare offered you in the File Dialog Box, then BrainWare will scan the filename you have entered, and provided the file name you have chosen conforms to the following format it will be used to derive new defaults:</p> <p><i>&lt;FilenameRoot&gt;_&lt;RecordingCounter&gt;e&lt;ElectrodeChannel&gt;.src</i></p> <p>For example, assume that you have rejected default filename <i>r_001e1.src</i> and have instead entered filename <i>rat14_023e1.src</i>, then BrainWare will set the Filename Root to <i>rat14</i> and the Recording Counter to 23, and the next default filename you will be offered will be <i>rat14_024e1.src</i>.</p>
<b>Response Window...</b>	Opens Response window dialog box for response strength quantification. See below.
<b>TDT Hardware...</b>	Run A/D + D/A off a single clock. Runs the D/A converter off the A/D converter clock. This time locks the samples of the stimulus and acquisition (eliminating a one sample period jitter that occurs if this setting is not used). If this option is used the stimulus has to be the same length or shorter than the acquisition time. Stimulus modules must operate at an integer multiple of 25 kHz.

## Response Window Dialog



The Response window dialog allows the user to set values for the beginning and end of “response” and “spontaneous” periods used to quantify spike counts. (See the description of the PSTH Window for further information on “response” and “spontaneous” periods).

When invoked via the Main window’s Option Menu, it will set default values for newly recorded data files. When invoked through the PSTH Window for a particular cluster it will set the response periods for that cluster.

## Window Menu

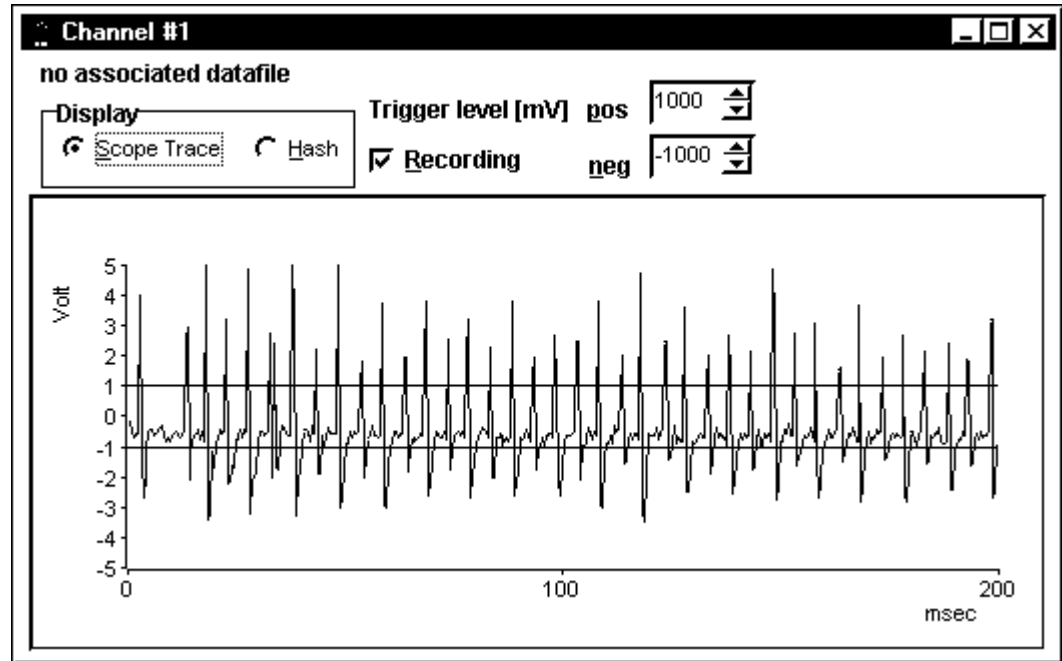
The window menu allows the user to control the layout of child windows on the application desktop.

The commands are Windows standard / self-explanatory.

## Window Overviews

This section describes the main windows in BrainWare. In general, most of the options available for a particular window can be accessed by right-clicking on the window. This will bring up a drop-down menu with choices for plotting, displaying, scaling, and saving data.

### The Electrode Channel Window



The electrode channel windows are created automatically when a hardware channel is opened for recording (Main window Menu Recording | Single Channel or Recording | Tetrode).

By default the channel window will display the voltage trace recorded at the A/D converter, along with two trigger lines.

#### Setting trigger thresholds

Set the trigger levels using the input boxes in the top right corner of the electrode channel window. The trigger levels may be adjusted while acquiring data. How BrainWare uses the trigger levels depends on whether *artefact rejection* is selected in the Options menu.

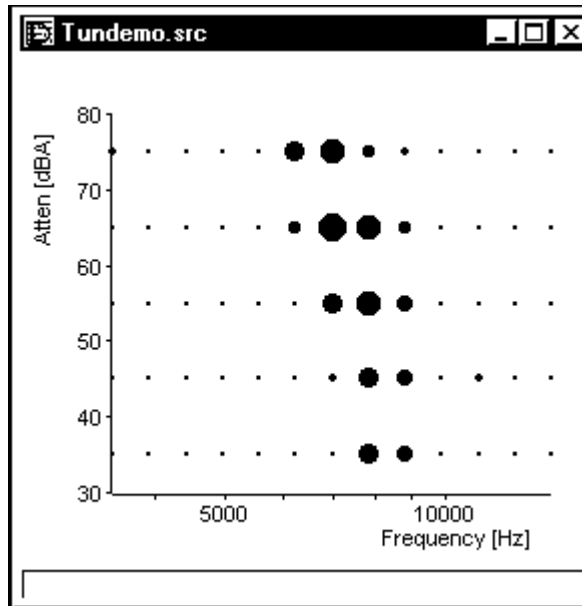
### **Artefact rejection**

The optional artefact rejection determines how the electrode channel window extracts action potential candidates from the digitized input. Without artefact rejection any voltage peak exceeding either trigger level will be recorded. With artefact rejection only *biphasic* action potential candidates with both peaks exceeding respective noise level settings will be recorded, and the first phase of the action potential is not allowed to exceed about 700 $\mu$ s. Artefact rejection is *only* available for single channel recordings, it is ignored by tetrodes.

### **Hash Display**

The hash display will show raster plots of the last seven sweeps. The trigger levels may be changed while either in the Scope Trace or Hash Display.

## The Data View (or Data File, or Data Set) Window



BrainWare organizes recorded spike data in a series of “collections” and allows the user to view, manipulate and analyze these through a hierarchy of “child windows”. The Data View window stands at the root of this hierarchy and provides an overview over one dataset collected through one electrode channel at one electrode position. Typically it will plot the responses for each cluster versus some stimulus parameter.

To delineate clusters or view the clusters making up the data points of the Data View window you must open a Feature Space window, either by clicking on one of the shown data points, or by choosing **Show | Feature Space** on the Data View menu.

The Data View menu is also used to choose which stimulus parameters to plot on the y axis, to set axis parameters or other view options, or to delete clusters or data points. It can be accessed via **Data** on the application main menu when one of the Data View windows is the currently active child windows, or by clicking on the Data View window with the right mouse button.

### The Data View Menu

<b>Plot</b>	choose stimulus parameters to plot on x or y axis
<b>Show</b>	open up Feature space, Shape or PSTH window for the currently selected data point. (Select data points by clicking on them)
<b>Options</b>	sets a variety of view options (see below)
<b>Delete</b>	<ul style="list-style-type: none"><li>• <b>cluster throughout</b> deletes the currently selected cluster at all data points (conditions). The spikes belonging to the deleted cluster are reassigned to other clusters (if they fit) or are returned to the unassigned spikes.</li><li>• <b>data point</b> <i>permanently</i> deletes the data of one recording condition from the data file, including all clusters &amp; unassigned spikes. If in doubt, make a backup copy of the file before using this option.</li></ul>
<b>Copy</b>	<ul style="list-style-type: none"><li>• <b>graph to clipboard</b> copies current graph to clipboard in windows metafile format.</li><li>• <b>data to database table</b> is highly customized for spatial receptive field analysis of SC neurons &amp; therefore probably not generally useful.</li><li>• <b>data to clipboard   averaged</b> creates a table of spike count statistics on the clipboard. If several data points were recorded with the same stimulus parameters their responses are averaged.</li><li>• <b>data to clipboard   raw</b> as above, but without averaging.</li></ul>
<b>Sync Cluster</b>	generates “synchronized spike trains” from the clusters to allow a data analysis similar to that performed in Ghose, G. M., Ohzawa, I. and Freeman, R. D. Receptive-field maps of correlated discharge between pairs of neurons in the cat's visual cortex. <i>J Neurophysiol</i> 71: 330-46,1994 or Meister, M., Lagnado, L. and Baylor, D. A. Concerted signaling by retinal ganglion cells. <i>Science</i> 270: 1207-1210,1995.

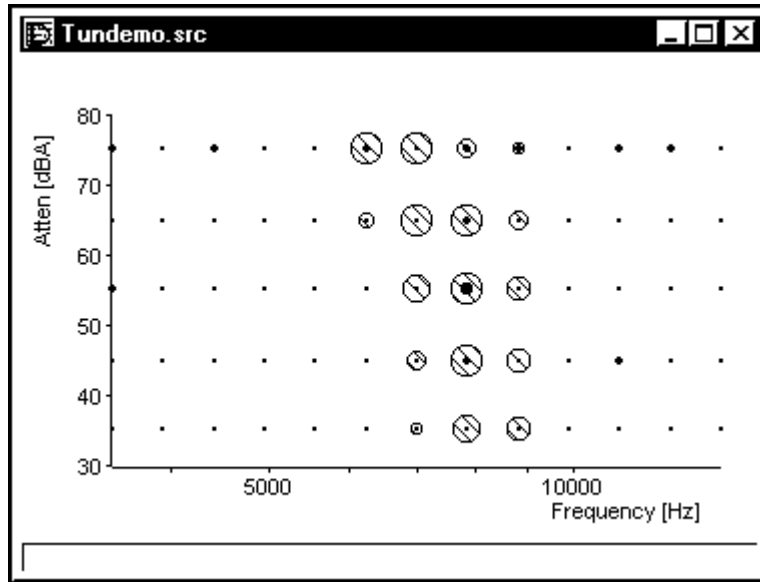
---

## Data View Menu | Options

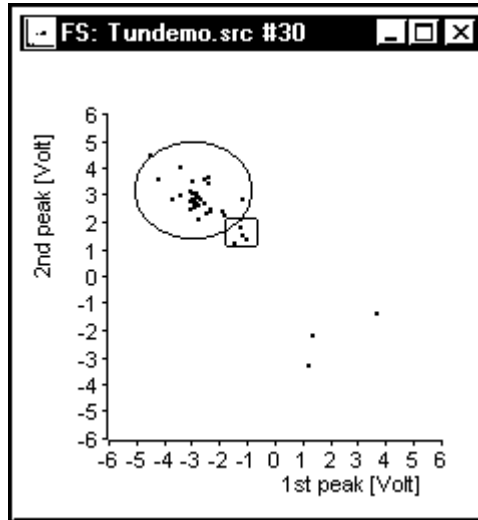
<b>Side of Brain</b>	Changes the datafile's record of which hemisphere the data were recorded from to conform to the <b>Side button</b> (marked "L" or "R") on the main window's toolbar.
<b>Error Bars</b>	Switches the display of error bars on or off. Error bars show standard errors or confidence intervals, depending on the analysis options selected in the Main window's Options menu.
<b>Only one Visible</b>	Shows only the dataset for the currently selected cluster.
<b>Axis Dialog</b>	Allows the user to change the scaling on the current axes.
<b>Rename Cluster</b>	Allows the user to change the ID string of the currently selected cluster.
<b>Inc Cluster Priority</b>	Increases the priority of the currently selected cluster by one (i.e. it will swap place of the cluster just before it in the order of priority. The order of priority decides which cluster takes precedence if a spike fits within the boundaries of several clusters.)
<b>"Surface" Plots</b>	For surface plots (when two parameters are plotted against each other on the x and y axes) the response strength is indicated by varying the size of the data points in the plot. This option allows to customize this type of display.
<b>Set Sweep Range</b>	Response statistics are normally calculated over all recorded sweeps. This option makes it possible to calculate response stats over a limited range of recorded sweeps only. Enter the range of sweeps you wish to include in the analysis (e.g. "1", "2,5", "3-7", "3-4,6,9", "all"). <b>Note:</b> the first set is "Set 1".

### Inhibition

If a cell shows inhibition, this will be shown in the Data View window by open hashed circles as shown below.



## The Feature Space Window



*Click on the Feature Space window with the right mouse button to see the Feature Space Menu.*

The Feature Space window is used to visually identify and delineate (“cut”) clusters of action potentials in order to isolate single unit activity from the action potential candidates recorded by the electrode channel window.

Each action potential candidate is represented by a single point in the feature space plot. The axes of the feature space plot can be set to represent any of a number of possible action potential metrics, or “feature space dimensions”, like the total amplitude of the action potential, the amplitude of the 1<sup>st</sup> or 2<sup>nd</sup> peak, the area under the spike etc. Action potentials from a single unit will have very similar shapes and should cluster together in feature space (provided the signal / noise ratio of the recording is good and the recording conditions are stable).

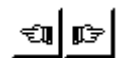
Use the Feature Space Menu to set the desired dimensions for the x or y axis.

The Feature Space window shows the action potential candidates recorded under a single stimulus condition, normally the stimulus condition currently selected in the DataView Window. Clicking on a data point in the DataView Window opens the Feature Space window for the action potentials comprising that data point.

The window is opened automatically at the beginning of recording to a new data file.



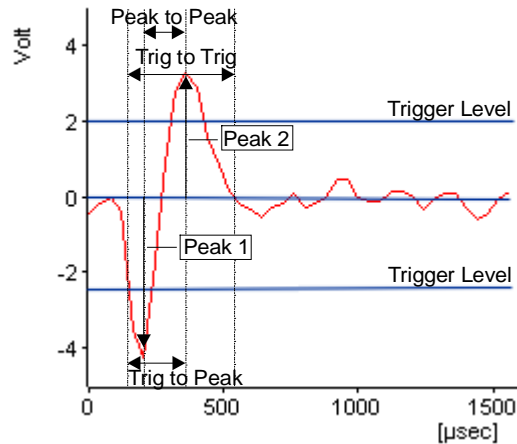
You can move between clusters by using the hands pointing up and down on the toolbar.



You can move between datasets by clicking the right and left pointing hands on the toolbar.

## The Feature Space Menu

Click on the Feature Space window with the right mouse button to see the Feature Space Menu. The X and Y axis features are shown on the plot below.



### X and Y Axis Features

<b>total amplitude</b>	x axis only, total action potential amplitude in Volts recorded at A/D converter
<b>Peak 1</b>	amplitude of the 1 <sup>st</sup> component
<b>Peak 2</b>	amplitude of the 2 <sup>nd</sup> component
<b>Peak to Peak</b>	time in µsec between the 1 <sup>st</sup> and 2 <sup>nd</sup> component
<b>Trig to Peak</b>	time in µsec between the crossing of the 1 <sup>st</sup> trigger point and the 2 <sup>nd</sup> peak
<b>Trig to Trig</b>	total duration of the action potential in µsec, from the crossing of the 1 <sup>st</sup> trigger point until the recrossing into the noise at the 2 <sup>nd</sup> trigger point after the 2 <sup>nd</sup> peak
<b>Area</b>	total absolute area under both the 1 <sup>st</sup> and 2 <sup>nd</sup> peak in arbitrary units
<b>Event time</b>	x axis only, time of action potential occurrence. This feature dimension is for display only; “cutting” clusters in it is not allowed.

**Elliptic X Boundary** Check to make boundary elliptic in current X dimension.

**Elliptic Y Boundary** Check to make boundary elliptic in current Y dimension.

Check this option if you would like the cluster to be elliptic for the feature dimension currently plotted on the x or y axis respectively. You will only see the cluster boundaries plotted as an ellipse if both the current x and y boundaries are elliptic.

Elliptic boundaries tend to capture the spread of action potential clusters better, particularly for amplitude features.

The elliptic menu option is only available if one of the clusters in the Feature Space window is selected. To select a cluster click on it with the left mouse button. The cluster boundary should be stippled. Then reopen the Feature Space Menu by right-clicking *inside* the cluster boundaries (clicking outside might de-select the cluster).

<b>X Axis Channel</b>	Tetrode recordings only: choose channel to plot on X Axis
<b>Y Axis Channel</b>	Tetrode recordings only: choose channel to plot on Y Axis In the submenu choose which of the four electrode channels you would like to plot on the x or y axis respectively.
<b>PSTH Window</b>	Shows a post stimulus time histogram for the currently selected cluster.
<b>Shape Window</b>	Shows spike shapes for the currently selected cluster.
<b>Marker Size</b>	Choose the size of the symbols used for the scatter plot (in pixels).
<b>Axis Dialog</b>	Pop-up a dialog to change axis scaling.
<b>Delete Cluster</b>	Delete currently selected cluster.  <b>Note:</b> this will only delete the cluster for the current data set (condition). To delete a cluster for all sets use the Data View menu.
<b>Copy Boundaries</b>	<b>to other Sets in this File:</b> Use this if “ <i>Cluster bounds same in all sets</i> ” is <b>not</b> selected in the main window’s options menu to copy cluster boundaries from this set to all other sets in this data file.  <b>to Clipboard:</b> Use to copy cluster boundaries from this file to other data files via the Windows clipboard. Useful when reanalyzing several data files that were recorded at the same electrode position.
<b>Paste Boundaries</b>	Paste cluster boundaries from the clipboard to this data file.

## Cutting clusters

### Overview

**Double clicking** on the graph pane in the Feature Space window marks the top-left corner of a *new cluster*. Drag the cursor until a box is drawn around the spikes you wish to cut. A single click will finish the cut and all the spikes will be re-plotted in the color of their respective cluster.

**Boundaries** of existing clusters can be adjusted by “*dragging*” them with the mouse cursor.

By default cluster boundaries are rectangular, but can be set to be elliptic in any of the feature dimensions via the *Feature Space Menu*.

Spikes are assigned to existing clusters according to priority determined by the order of creation (i.e., if a spike fits within the boundaries of two existing clusters, it will be assigned only to the cluster that was created first). The priority of the cluster created later can, however, be increased using an option from the Data View menu.

### In-Depth

The shapes of each recorded action potential are measured automatically to determine metrics (“features”) which are plotted in the coordinate system of the FeatureSpace window. Provided the relative position of the neuron and the electrode remain constant, and that the neuron is not compromised in some way, the shapes of the action potential should remain constant and clusters of dots should become apparent in the feature space window which represent the activity of single units.

Setting the x- and y-axes of the FeatureSpace window to “Peak 1” and “Peak 2” features often provides for a good initial discrimination, as positive going and negative going action potentials will be segregated to opposite corners of the feature space, as shown here:

▪  
▪

└

(Graphs taken from the file rfdemo.src, which is included in directory “data” of the BrainWare installation)

The initial segregation can then be refined by setting one or both of the feature space axes to different features and limiting the extent of the clusters.

Note, however, that while “Peak 1 vs. Peak 2” often gives a good initial segregation, it can lead to erroneous results with certain types of triphasic action potentials..

Spikes are assigned to existing clusters according to priority determined by the order of creation (i.e., if a spike fits within the boundaries of two existing clusters, it will be assigned only to the cluster that was created first). The priority of the cluster created later can, however, be increased using an option from the Data View Menu.

### Triphasic Action Potentials

In the previous section we have seen how the features “Peak 1” and “Peak 2” can be usefully exploited to segregate extracellularly recorded action potentials according to whether they are positive or negative going. However, one occasionally observes a class of triphasic action potentials, where the first peak consists of only a relatively small prepotential, as shown in Figure 1B below.

Problems arise when the first peak is not much larger than the trigger levels set in the electrode channel window. Due to electrical noise, the small first peak may then sometimes exceed the trigger level, sometimes not. However, on those occasions where the small first peak fails to exceed the positive trigger level, the larger, second negative peak is likely to exceed the channel window’s negative threshold, and the resulting discharge will appear to be simply a biphasic negative going action potential (shown in Figure 1C), rather than the triphasic potential with a small first positive peak that it really is. Consequently, the feature space window with axes set to “Peak 1 vs. Peak 2” may give the mistaken impression that we are dealing with two separate, easily discriminated single units (see Figure 1A), whereas in reality we deal with only a single unit that occasionally mistriggers on the first phase, and thereby becomes “split in two”.

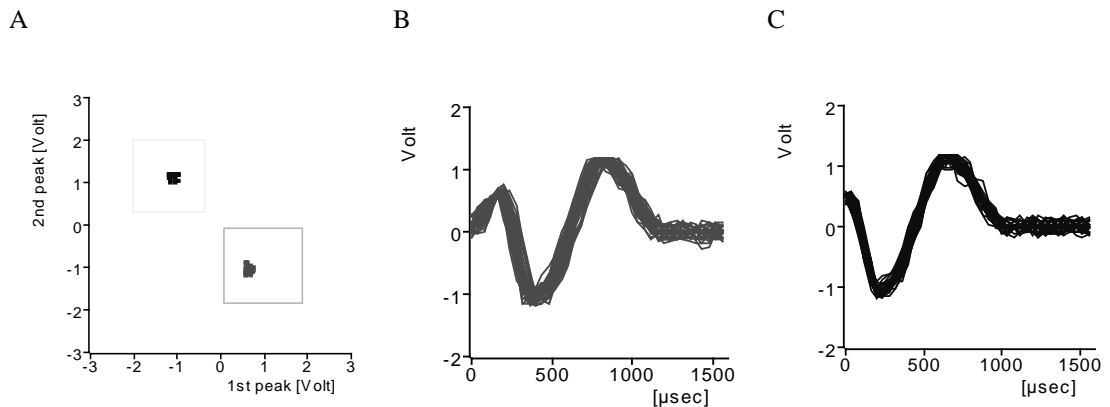


Figure 1

If this problem is recognized immediately it may be possible to avoid this problem by adjusting the trigger levels, but that can not be done in post-hoc reanalysis, and sometimes, when the signal to noise ratio is not very good, it may not be possible to set trigger levels low enough to avoid mistrigging of the first phase. In those cases it is necessary to rely on features other than Peak 1 and Peak 2 to arrive at a sensible clustering of recorded action potential shapes. To do this, open the cluster boundaries for Peak 1 and Peak 2 wide, to enclose both parts of the “split cluster”, as shown in Figure 2A. Then set the feature space axes to “+ve Peak” on the x-axis and “-ve Peak” on the y-axis (or vice versa). Since these features simply take the highest and lowest point of the wave form they are insensitive to mistrigging, and the points in the feature space collapse to a single cluster, as shown in Figure 2B. The shape window for the cluster should show a homogeneous population of spike shapes, (but with a portion of the population time-shifted by a fixed amount), as shown in Figure 2C.

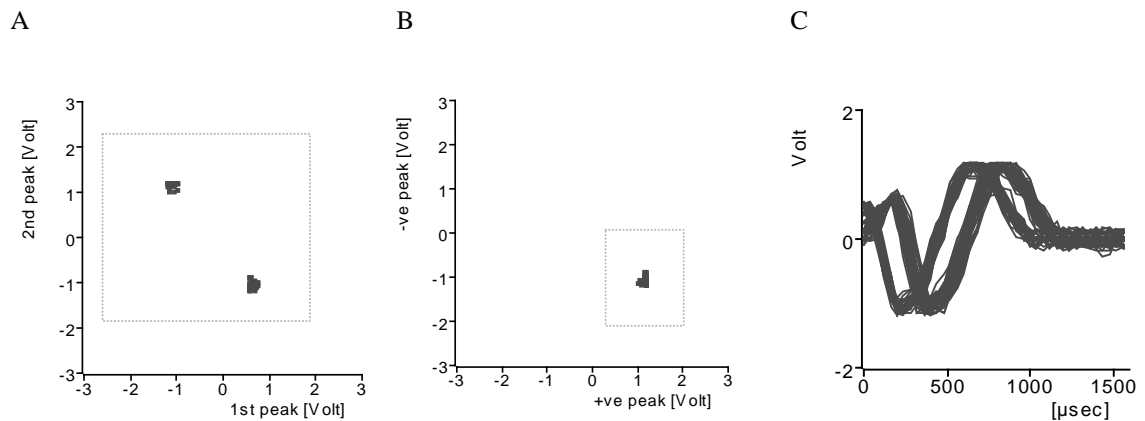
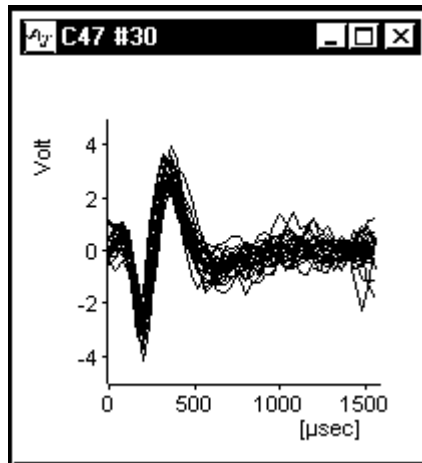


Figure 2

Most of the features available in the current version of BrainWare are sensitive to this type of mistrigging (the only exceptions are “+ve peak”, “-ve peak” and the derivative feature “total amplitude”). This severely limits the discriminability of this type of triphasic action potential with small first peaks, compared to “more conventional biphasics”. Thankfully triphasic action potentials like these are, in our experience, comparatively rare.



## The Spike Shape Window



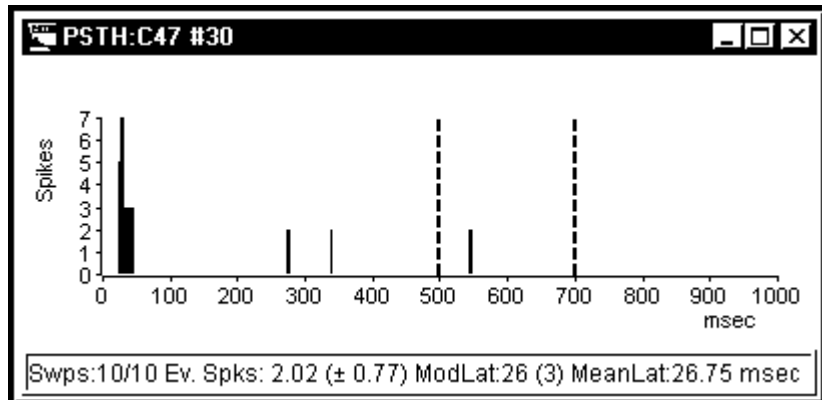
Spike Shape windows plot the shape of all action potential candidates belonging to a cluster superimposed on a time vs. voltage coordinate system. Their main purpose is to help you decide whether the spikes contained within a cluster can reasonably be assumed to represent the activity of a single unit, based on a visual inspection of the similarity of their action potential shapes.

### **Shortcut**

Use **CTRL-H** to bring up a Spike Shape window.

Spike Shape windows can be generated for each cluster either from the DataView or from the Feature Space window. Click on a data point in the DataView window or click inside one of the cluster boundaries in a Feature Space window and then choose “Show Shapes” from the drop-down menus of these windows (accessed via the right mouse button). During recording all Spike Shape windows are updated automatically after the recording of each new sweep. Spike Shape windows are also updated automatically when cluster boundaries are changed in the Feature Space window.

## The Post Stimulus Time Histogram (PSTH) Window



### Shortcut

Use **CTRL-P** to bring up a PSTH window.

Click on the PSTH window with the right mouse button to open the PSTH menu.

PSTH windows can be generated for each cluster either from the DataView or from the Feature Space window. Click on a data point in the DataView window or click inside one of the cluster boundaries in a Feature Space window. Then choose "Show PSTH" from the drop-down menus of these windows (accessed via the right mouse button).

During recording all PSTH windows are updated automatically after the recording of each new sweep.

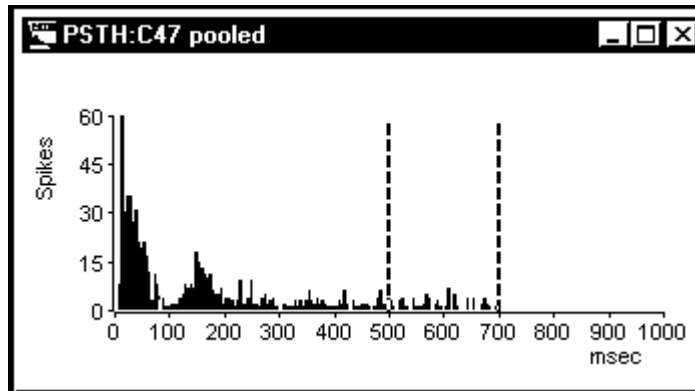
PSTH windows are also updated automatically when cluster boundaries are changed in the Feature Space window.

BrainWare quantifies the evoked response on the basis of spike counts in the "response period" and the "spontaneous period" in the PSTH. These are delineated by the horizontal stippled lines in the PSTH plot, and can be changed either by dragging them with the mouse or by opening the Response window dialog.

The figure shows a dialog box titled "Response windows for C47". It contains four input fields with spinners and "ms" labels: Response start (3 ms), Response finish (83 ms), Spontaneous start (500 ms), and Spontaneous finish (700 ms). On the right side, there are three buttons: OK (with a checkmark), Cancel (with an X), and Help (with a question mark).

The DataView Window is automatically updated when these boundaries are adjusted.

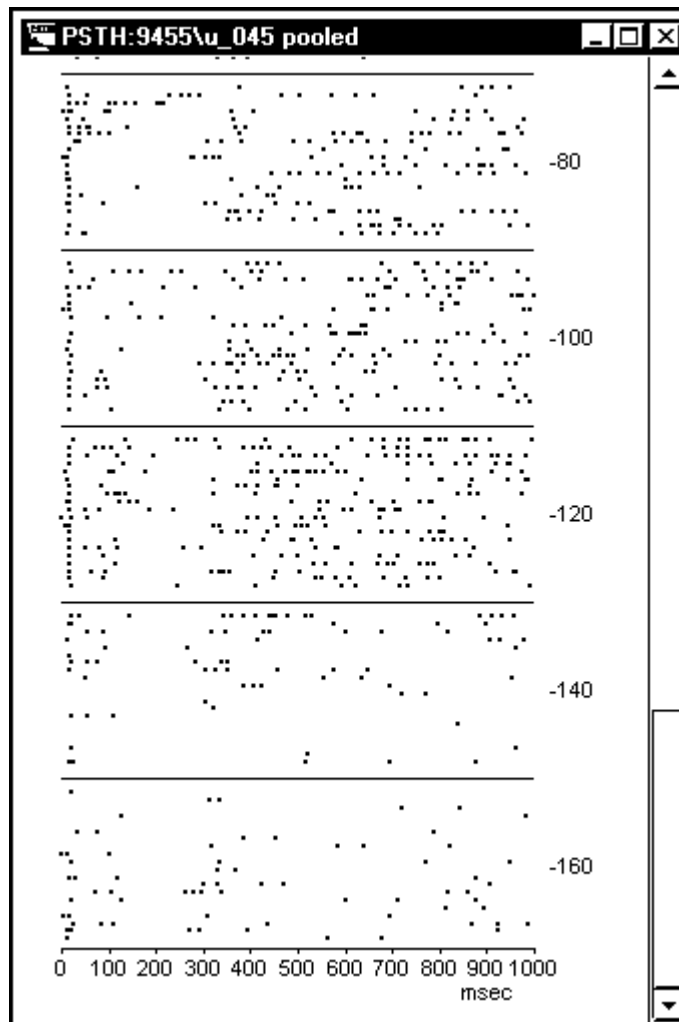
## Viewing all of the data



To pool all or some of the data, right click on the PSTH plot and select pool all records under the Show Records menu.

## Raster Plots

You may also view data as a raster plot by right-clicking in the PSTH plot and choosing Raster Plot from the menu.

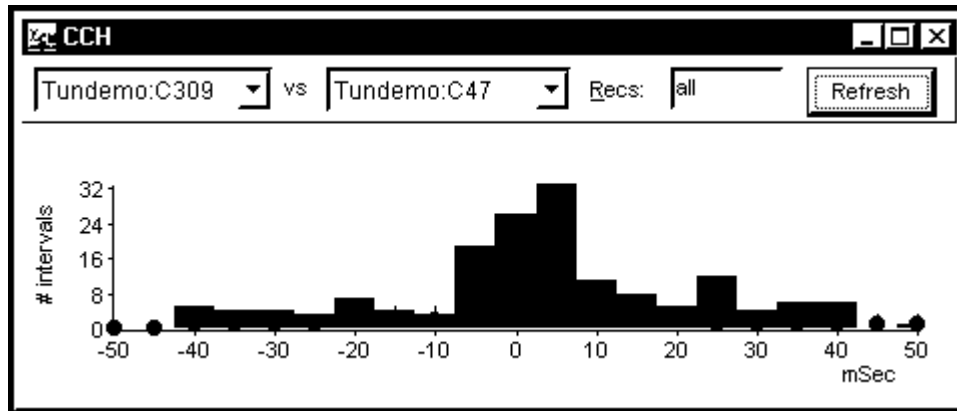


## **Saving Data**

From the PSTH window you can choose to copy the spike times, counts, or graph to the clipboard. Right click in the PSTH plot and choose Copy and select one of the options.

You can also export the data from the File menu by choosing Save As. This is described later in this chapter in the section on Exporting Data.

## Cross-correlation Histogram (CCH) Window



Generate a CCH window by selecting File|Correlogram on the main menu of BrainWare.

This window allows you to generate cross-correlation or auto-correlation histograms on clusters in currently open files.

Select the clusters for which you wish to generate a correlation histogram in the two drop-down list boxes of the CCH window.

Specify the numbers for the datasets which should be included in the correlogram in the "Recs." box (e.g. "1", "2,5", "3-7", "all").

Click on the graph pane with the right mouse button to access a menu of correlogram options.

### Cross-correlation Histogram (CCH) Menu

The CCH window menu offers the following options:

- |                        |                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Binsize</b>         | Sets the Binsize for the correlation histogram in msec. The default binsize is the same as that given for PSTH windows in the Options menu of the Main window.                                                                                                                                                                                                                                  |
| <b>Axis Dialog</b>     | Makes it possible to change the axes of the correlogram                                                                                                                                                                                                                                                                                                                                         |
| <b>Shift Predictor</b> | Correlates sweep N of spike train 1 with sweep N+1 of spike train 2. Used to predict the amount of correlation expected to be due to common stimulation, rather than more or less direct connections. See Perkel, D. H., Gerstein, G. L., and Moore, G. P. (1967). Neural spike trains and stochastic point processes II. Simultaneous spike trains. <i>Neurophys. J.</i> 7, 419-40 for theory. |
| <b>PSTH Predictor</b>  | Similar to Shift predictor, but correlates sweep N of spike train 1 with sweep N+J of spike train 2, where J runs from zero to N. Reported to be more reliable than shift predictors. See: Ghose, G. M., Ohzawa, I., and Freeman, R. D. (1994). Receptive-field maps of correlated discharge between pairs of neurons in the cat's visual cortex. <i>J Neurophysiol</i> 71, 330-46 for theory.  |

### Calculation of cross-correlation histograms

Cross-correlation histograms (CCHs) would more accurately be referred to as "any order cross-interval histograms", and were invented by Perkel, D. H., Gerstein, G. L., and Moore, G. P. (see: "Neural spike trains and stochastic point processes II. Simultaneous spike trains". *Neurophys. J.* 7, p. 419-40, 1967) as a means to study interactions between simultaneously recorded spike trains. The histograms are constructed from all intervals between the occurrence of a spike fired by neuron A and those fired by neuron B for all spikes. If neuron A and neuron B fire independently all intervals should be equally likely and the resulting CCH should appear flat. If neuron A excites neuron B then neuron B should be more likely to fire one synaptic delay (one to a few milliseconds) after neuron A has fired, and the CCH should show a peak centred on the synaptic delay. Similarly, inhibitory interactions should lead to a trough in the CCH. If neuron A and neuron B are both excited from a common source then the CCH should show a peak centred on 0 ms.

Cross-correlation histograms are usually meaningful if the spike trains used to build them have been recorded simultaneously, either from multiple electrode recordings, or from different clusters within the same data file. If you generate CCHs from clusters within the same data file you should be aware of the "sorter artefact": the CCH will then always show a paucity of 0ms intervals. This is due to the fact that two action potentials from different neurons occurring simultaneously and seen through the same electrode will superimpose, generating an action potential shape that is atypical for both neurons and will most likely not be attributed to the clusters for either of the neurons. Sorter artefacts are an inevitable limitation of simultaneous recordings through a single electrode.

### Using auto-correlation histograms

Auto-correlation histograms (ACHs) are calculated in the same way as CCHs, but in this case neuron A = neuron B. Obviously ACHs will have a large peak at 0 ms, the height of this peak being equal to the total number of spikes in the spike train, for every spike occurred within a 0 ms interval of itself. Furthermore ACHs are symmetric in time around 0 ms, since if spike X occurred N ms before spike Y then spike Y occurred N ms after (i.e. -N ms before) spike X. This may appear silly, but ACHs have uses, either to identify periodic firing of a neuron (in which case the ACH should show repeated peaks at intervals corresponding to the neurons firing period) or to test for the "purity" of a cluster. Due to the refractory period in the firing of a neuron the ACH should show a paucity of 1 ms intervals. If there is no sharp dip at the 1ms interval the cluster probably does not reflect single unit activity, but contains spikes from several neurons (To see this you may have to use the CCH window options to set bin size to one and to limit the height of the y axis appropriately). Note, however, that the refractory period of some neurons has been reported to be less than 1 ms, so even if the 1 ms bin is not completely empty this does not necessarily mean that the cluster is multi-unit. The current version of BrainWare does not allow the generation of ACHs with less than 1ms bins, so refractory periods less than 1 ms can not be revealed. But as spike times are recorded internally at 25 kHz "clock ticks" and it is planned that future versions of BrainWare make this higher temporal resolution available for finer grained CCH and ACH analysis.

The ACH is related to the "inter-spike interval histogram" or ISIH which is often used in studies of cochlear nucleus, the only difference being that the ISIH only compiles intervals for each spike to its nearest neighbor (it is said to be a "first order interval histogram") whereas the ACH shows all intervals for all spikes (it is said to be "any order").

## **BrainWare Stimulus Modules**

A unique feature of BrainWare is that it will produce plots of neural responses versus stimulus parameter on-line, so that you can get immediate feedback about the responsiveness of your cells. However, the Neuroscience community requires an ever increasing number of different and more sophisticated of stimuli. Also, neurophysiologists often have a need for their stimulation regimes to “evolve”, and to be easily reconfigured, ideally in a few seconds, in the middle of an experiment. BrainWare stimulus “modules” provide an elegant way to meet these needs.

BrainWare stimulus modules can be of two types: “TDT SigGen files” (of interest mainly to the auditory physiologist) and “Dynamic Link Libraries” (DLLs) (for everyone). There is no limitation to the number of stimulus modules you can load, and you can mix SigGen files and DLLs. DLLs and SigGen files are, of course, simply computer files like any other, so in what sense do they constitute “modules”?

To grasp the concept, consider the following scenario: Say you are an auditory neurophysiologist, and for your research you require, or at least would like to have, a large arsenal of acoustic, stimuli, some very simple, like pure tones and noise bursts, some quite complex, perhaps artificial vowels which allow you to set arbitrary formant combinations. Many of these stimuli you may be able to generate quite easily with TDTs SigGen, for others you may have to compile your own software libraries.

Of course, to each of these base stimuli you probably need to add the possibility to control sound intensity, perhaps through a set of programmable attenuators. You may also decide you would like to have the option of band limiting the spectrum of these stimuli, so you acquire a programmable filter module. Then your collaborator, who is interested in spatial hearing, would like to be able to present your stimuli from any of a large number of loudspeakers in a spatial array, and acquires a multiplexer to switch the different speakers on or off. Yet another collaborator is interested in multimodal stimulation, and wants to combine the presentation of your auditory stimuli with visual stimuli.

You could, of course, change each of your original stimulus routines in order to add software control for all these additional hardware components, but to your originally quite small and elegant stimulus software routines will then quickly grow into bloated, impenetrable monoliths. In most cases it would be a better strategy to have separate “software modules” to build the sounds, different ones to control the attenuators, programmable filters, trigger pulse timing generators, multiplexers, visual stimulus generator interfaces, e.t.c., of your laboratory’s arsenal, and to hook them into their respective places in the stimulus cycle as and when required, on the fly, at the click of a button. Each of these software modules can then be small, sometimes only a few lines of code, and can be updated and debugged independently of the remaining software. And if at some stage you start to do work in awake, behaving animals, you can add further software modules, perhaps to read the animals eye position from magnetic search coils to ensure that stimuli are only triggered when the animal fixates, or modules that control manipulanda to control the animals behaviour, without having to change any other aspect of your stimulation software. BrainWare 6.1 makes it easy for you to implement this strategy.

## **SigGen Files as BrainWare Stimulus Modules**

From version 6.0 of BrainWare you can use TDT SigGen (\*.sig) files to specify the stimuli used for recording.

BrainWare 6.1 “embeds” SigGen files into a Stimulus Module framework, which makes it possible to configure stimuli from arbitrary combinations of SigGen files and Stimulus DLLs.

For auditory neurophysiologists using SigGen is probably the quickest way to build even quite complex stimuli for their experiments. SigGen offers a great deal of flexibility in the design of auditory stimuli while remaining (relatively) easy to use. In the following we assume that you are familiar with SigGen. (Documentation for SigGen can be downloaded from the TDT Web site <<http://www.tdt-quikki.com>>).

### **Signal Files**

To load SigGen files, simply click the “Configure Stimulus Modules” button on the Stimulus Grid, select “Add Module”, choose the file and click “OK”. The SigGen stimuli will be played through the DA channel specified in the file (you can set this using the “D / A channel” list box in SigGen’s “Signal Parameters” window.

You can load two SigGen files if you require two-channel (e.g. binaural) stimulation. Simply add the second file as another module. However, bear in mind that two signal files must specify different D/A channels and set the same Sampling Period for DA conversion, otherwise they can not be played concurrently through the same DA module. (In that event BrainWare would produce an error message and abort stimulation. Future versions of BrainWare could mix signals from several stimulus modules that are directed to the same channel digitally. Let us know if this is a feature that you require).

## Triggering

If the SigGen Stimuli control triggering BrainWare will check whether there is a TG6 module in your rack. If a TG6 module is detected, the TG6 outputs 2 and 3 will generate the trigger pulses for your D/A and A/D modules, respectively. In this case, make sure that these outputs are connected (with a BNC lead) to the “TRIG” inputs of the D/A and A/D modules.

If there is no TG6 module in your rack, BrainWare will send a global XBus trigger command (xb1gtrig) to your System II rack when it is time to trigger the stimulus. In that case you should leave the “TRIG” ports of your AD and DA modules unconnected.

You can synchronize further hardware with the AD or DA module by using pulses sent through the “SYNC” ports of these modules when these are triggered.

If you do not provide the appropriate connections to the trigger inputs BrainWare will appear to “hang” as it is waiting for a signal to stop playing that has never started, and data acquisition will not continue (in that case the “ARM” LED on you AD module will remain on). BrainWare may detect triggering problems and give an error message.

## SigGen Variables and Stimulus Parameters

Upon loading the SigGen files BrainWare will add rows to the parameter table of the Stimulus Grid, and fill these with the SigGen variable names and values specified in your SigGen files.

With the new 32-bit drivers for TDT System II it is now possible to have both SigGen and BrainWare open at the same time during the course of an experiment. You can switch between the applications to modify your SigGen Signals, and then reload them quickly using the “Configure Stimulus Modules” and choosing “Reload Modules”

## Peripheral Devices

Any control of programmable attenuators (PA4s) that you specified in your signal design with SigGen will be implemented in BrainWare, but *don't forget to specify two different PA4 device IDs in the SigGen files when creating files for dual channel stimulation.*

Other devices, like digital switches, programmable filters and wave form generators are also supported, but there is no TG6 support in the current version of BrainWare. As a consequence, any “onset delay” specified for a SigGen signal will be ignored by BrainWare. You can nevertheless easily implement fixed, and even variable, delays, simply by making the total signal duration longer (so as to create stretches of “silent signal”) and setting the onset of the “signal segments” to values corresponding to the desired delays.

## Using SigGen Variables with BrainWare

BrainWare will read the values specified in your SigGen signal files to the Stimulus Grid table. The values will be stored along with the acquired data and are used to produce online plots of neural response strength statistics vs. stimulus parameter in the Dataset Window.

You should use the facilities of SigGen to create stimulus parameter (variable) lists that cover the full range of parameters you wish to test during your recording. For example, if you wish to record a pure tone response area you should make sure that your SigGen variables run through the desired frequency / intensity combinations. As BrainWare will use the variable values to produce online plots of response versus stimulus parameter as the data are acquired, **BrainWare requires that at least one of the SigGen files loaded defines at least one variable.** This variable should also not be defined as using a "Random Method" (see below).

## SigGen Variable Types

SigGen offers a variety of variable "methods", i.e. ways of determining what values your stimulus parameters should take. Most of them (like *Linear step*, *Log step*, *Constant*, *Value list*...) are simply a means of generating fixed lists of parameter values over the range of interest, and you should use these to generate the parameter lists that you want to import into the stimulus table.

*Consult the SigGen Manual chapter 3 for more information on SigGen variable termination.*

It is highly recommended that you make sure that your variables will **reach a “termination” (or “boundary”) condition.** If you use two SigGen files for dual channel stimulation you should also match the number of stimulus conditions in each file, but that is not an absolute requirement: you could, for example, present stimuli of variable frequency and intensity through one channel and constant stimuli (e.g. a “masker”) through the other.

SigGen also offers variable "methods" that generate new values at runtime, which require comment:

**Prompted variables** are supposed to be set as a function of a response obtained in the course of the experiment. BrainWare will *not* prompt, and simply use the default value of the variable.

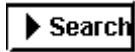
**Random variables:** these are given a new random value on every presentation of the signal. BrainWare supports (i.e. updates) random variables, but does *not* keep track of the values these variables took at particular presentations, or save these values with the data. It is therefore not possible to plot responses against these random values. The values of the random variables are therefore also not shown on the stimulus grid.

**Dynamic variables** may be set by an application at run-time. Upon loading, BrainWare will *not* alter the values for these variables, but simply use the defaults. However, when you click the “Edit” Button on BrainWare’s Stimulus Grid, BrainWare will make *all* non-random variables dynamic, so that they can take the values that you enter into the stimulus grid.

## Data Collection

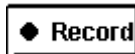
Before you can collect data you have to make sure that the Stimulus Grid is set up to provide appropriate stimuli.

Data collection then generally involves three stages. Searching for units, recording a sample to verify trigger levels and clusters cut, and finally recording a sequence, or batch, of samples to characterize the units.



### The Search Mode

Click the Search button on the tool bar to present “search” stimuli. When the Search button on the toolbar is depressed the system will continuously present the stimulus currently selected in the Stimulus Grid window. Electrode channels can be selected for recording and trigger levels set appropriately and the electrodes should be advanced until a responsive electrode position is encountered.



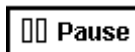
### Record

Once a promising electrode position has been identified in the search mode the user can use the Record button to collect data sweeps with the stimulus parameters specified in the Stimulus Grid window. The length and number of sweeps to be recorded is specified in the Sweep Parameters dialog (Menu Recording | Sweep Params).



### Batch Recording

Click the Batch button on the toolbar to record data for ALL the stimuli currently in the Stimulus Grid. To determine the order in which the stimuli in the grid will be presented use Menu Recording | Sweep Params.



### Pause

Clicking the Pause button will interrupt recording (when the Record or the Batch buttons are pressed) or repeat stimulation (when the Search button is pressed).

To resume recording or stimulation simply click the Pause button again.

## Analysis Options

This menu offers a number of sub-menus where the user can set his preferences for the calculation of spike count statistics. The available options are:

- Use Spon Window** If checked, BrainWare will subtract the averaged spike counts obtained in the “spontaneous” period from those counted in the “response” period to calculate the number of evoked spikes. If spontaneous and response periods are not of the same length then counts are normalized to compensate for this. (See the description of the PSTH window for further information on “response” and “spontaneous” periods).
- Calc Spike Rate** If checked, BrainWare will give spike count statistics as rates in Hz by dividing the averaged number of evoked spikes by the duration of the response period.
- Error Intervals** Allows the user to choose whether **standard errors of the mean** or **95% confidence intervals** should be used as error interval estimates. Note that the 95% confidence interval calculation uses a formula that assumes a normal distribution of the data, an assumption that may not hold as spike counts are often closer to Poisson random variables. (Error intervals can be displayed as error bars in Data View windows).

## Analysis of a previous recording

All the analysis functions can be applied to data collected in a previous session. Load the spike file you wish to analyze by clicking on the File|Open menu item or the file open icon on the toolbar.

BrainWare will open the data file and present an overview of the data in a Data View window. Click on the window with the right mouse button for a menu of view options. Click on data points to focus on a point and view the feature space representation of the underlying data.

## Exporting Data

To export data to ASCII or Binary formats, use the File|Save As command, then choose one of the several possible export formats in the “Save as type” box. The data will be saved in a file with extension “.txt” or “.bin”, depending on the chosen format.

### Spike Counts as ASCII Text (\*.txt)

This will produce a table of the spike counts for all of the recorded spikes. The table will be written to the specified file (default extension “\*.txt”).

### Spike Times as ASCII Text (\*.txt)

This will produce a table of the spike times in milliseconds for all spikes recorded. The table will be written to the specified file (default extension “\*.txt”).

For “*Spike Times as Text*” the table has following format:

<pre> Set 1 Sweep 1 StimParam -160.0 10      0 36      0 497     0 497     0 354     1 612     1 43      2 55      2 62      2 618     2 </pre>	<p>← <b>Header line</b> one per sweep, indicates which sweep to which set (stimulus condition) the following data points belong. “StimParam” gives the value of the stimulus parameter currently plotted on the x Axis of the Data View window.</p>
<pre> 354     1 612     1 43      2 55      2 62      2 618     2 </pre>	<p><b>Spike Times</b> in milliseconds, one spike per line. After the spike time, separated by a tab character (char(9)), follows a number which indicates which <b>cluster</b> the spike is currently assigned to. “0” means the spike is unassigned, the remaining clusters are numbered through in order of creation.</p>
<pre> &lt;----- Set 1 Sweep 2 StimParam -160.0 334     0 115     1 379     1 502     1 ... </pre>	<p>Data from subsequent sweeps is separated by an extra carriage return.</p>

### Spike Times and Shapes as ASCII Text (\*.txt)

For “*Spike Times and Shapes as Text*” the format is identical, except that after the cluster number the voltage waveform for the spike is given as a series of integer numbers separated by tabs. The voltages are encoded as eight bit integers, such that 127 corresponds to 5 Volts.

### Spike Counts as Binary (\*.bin)

This will write the response statistics **only for the cluster currently selected in the Data View window** to the specified file (default extension (\*.bin)). If no cluster is selected, an error message will be shown.

BrainWare will write three or four numbers for each data point for the selected cluster in sixteen bit integer format.

The first number corresponds to the response statistics \* 100 (mean number of evoked spikes or rate in each data set).

The second is the error statistic \* 100 (confidence interval, standard error), depending on the currently set analysis options.

The third will be 100 times the value of the stimulus parameter currently plotted on the x axis in the Data View window.

The fourth (if present) will be 100 times the value of the stimulus parameter currently plotted on the y axis in the Data View window. (This is written only for “surface” type plots where the y axis does not show the response).

The order will correspond simply to the order of the datasets in the file.

Integer format was chosen because importing binary floating point variables written with Delphi Pascal appeared not to be readable by other applications, such as Matlab.

To import data exported with this option into Matlab you can use the commands:

```
>> f=fopen('myfile.bin', 'r'); resps=fread(f,[3, inf], 'int16');
resps=resps/100; fclose(f);
```

or

```
>> f=fopen('myfile.bin', 'r'); resps=fread(f,[4, inf], 'int16');
resps=resps/100; fclose(f);
```

if two stimulus parameters are to be output.

This will read data from *myfile.bin* and give you three columns in *resps*, the first containing the response, the second the error statistics, the third (and fourth, if applicable) the stimulus parameter(s).

### Spike Times as Binary (\*.bin)

This will write the response statistics only for the cluster currently selected in the Data View window to the specified file (default extension “\*.bin”). If no cluster is selected, an error message will be shown.

BrainWare will write the spike times for each spike of the current cluster as 32-bit integer (longint). Spike times are written in clock ticks. Since spikes are sampled at 25kHz there are 25 clock ticks per millisecond. All spike times are positive integers. Negative integers are therefore used as “special characters” signaling:

- 1 new sweep follows
- 2 stimulus parameters for new dataset followed by the sweep length (in milliseconds) and the stimulus parameters for the dataset.

Stimulus parameters are also written as 32-bit longint (multiplied by 100 to export first two decimals). Prior to the list of actual stimulus parameters a longint indicating the number of stimulus parameters following is output.

**Note that this export format has changed as of BrainWare Versions 6.03! Previous versions did not write the sweep length to the binary file. However, having the sweep length available is advantageous if spike trains are to be represented in array or matrix format.**

Example:

a data file containing 3 data points (conditions) for the current cluster with the stimulus parameters 15.4, 17 and 19.45 currently plotted on the X axis, with 2 sweeps for each data point, might produce the following list of numbers:

- 2 new (first) dataset
- 1000 sweep length is 1000 ms = 1 second
- 4 four stimulus parameters will follow
- 1540 first stimulus parameter for first set x 100 (i.e. parameter was 15.40)
- 100 2nd stimulus parameter for first set x 100 (parameter was 1)
- 0 3rd stimulus parameter (parameter was 0)
- 0 4th stimulus parameter (parameter was 0)
- 1 new (first) sweep
- 2905 time of 1st spike (divide by 25 to get ms)
- 4035 time of 2nd spike
- ... ..
- 1 new sweep (2nd sweep of condition one)
- 294 time of spike 1
- 3024 time of spike 2
- ... ..
- 2 start of 2nd dataset (condition)
- 4 four stimulus parameters will follow
- 1700 first stimulus parameter for 2nd set x 100

```

15      2nd stimulus parameter
...      ...
-1      new (first) sweep for dataset 2
2905    time of 1st spike
4035    time of 2nd spike ...
...      ...

```

### Importing binary spike times into Matlab

You can use the following Matlab function to import spike data exported with File | Save As | Spike Times as Binary into Matlab 5.

```

function data=spikedat(fname);
% reads binary spike data file "fname" generated with
% BrainWare 6.1 "File | Save As | Spike Times as Binary
% (c) Jan Schnupp, June 1998
%
f=fopen(fname, 'r');

numsets=0;
numsweeps=0;
totalsweeps=0;
totalspikes=0;
i=fread(f,1,'int32');
while ~isempty(i);

    switch i
    case (-2) % new dataset
        numsets=numsets+1;
        numsweeps=0;
        % read sweeplength
        data(numsets).sweeplength=fread(f,1,'int32');
        % read stimulus parameters
        numparams=fread(f,1,'int32');
        data(numsets).stim=[fread(f,numparams,'int32')/100];
        % divide by 100 to restore decimals

    case (-1) % new sweep
        numsweeps=numsweeps+1;
        totalsweeps=totalsweeps+1;
        data(numsets).sweep(numsweeps).spikes=[];

```

```
    otherwise % read spike time for next spike in current sweep
        data(numsets).sweep(numsweeps).spikes=...
            [data(numsets).sweep(numsweeps).spikes i/25];
        % divide by 25 to get times in ms
        totalspikes=totalspikes+1;
    end;

    i=fread(f,1,'int32');
end;
fclose(f);
disp(sprintf('read %d sets, %d sweeps, %d spikes',...
    numsets,totalsweeps, totalspikes))
```

This function will return a Matlab “structure” containing the spike data. Refer to your Matlab manual to learn about data structures. To list the spike times of the spikes in the 3<sup>rd</sup> sweep of the 2<sup>nd</sup> data set (condition) in milliseconds you might type at the Matlab command prompt:

```
>> bindata=spikedat('myfile.bin');
>> bindata(2).sweep(3).spikes
```

## Timing Issues

The A/D converter acquires spike data with a 25 kHz sampling rate. This means that there is a 40  $\mu$ s sample period.

If you are using a software trigger (and not the TG6) the D/A triggers the beginning of spike acquisition. The time between when the A/D receives a trigger and when it starts to convert can vary up to one sample period. In BrainWare this results in a 40  $\mu$ s jitter between sound presentation and spike acquisition. The jitter is the same for all A/D channels (i.e. the A/D samples simultaneously on all channels).

If you use the TG6 for triggering, both the D/A and the A/D will have a jitter associated with their sample rate.

## Running the D/A and A/D on the Same Clock (microsecond spike timing)

The purpose of this option is to make it possible to investigate phase locking in auditory neurons with a temporal resolution that is much higher than the 40 microsecond resolution normally afforded by the 25 kHz digitization of the spike train.

The D/A and A/D modules in TDT System II are each equipped with their own clocks, and by default BrainWare will set up each module to run off its own clock. This allows a great deal of flexibility: D/A and A/D conversion can start and stop independently and run at completely independent sampling rates. However, the independence of these clocks limits the temporal resolution of action potential time stamps that can be achieved. At the moment a trigger pulse arrives at the D/A and A/D modules the clocks in these modules will be ticking “randomly out of phase”. Therefore the onset of the A/D and the D/A conversion will have an “onset jitter” that can be as large as longer of the two sample periods. The signal from the recording electrode is digitized at 25 kHz, which means that the action potential time stamps are typically accurate “only” to +/- 40 microseconds.

A 40 microsecond accuracy is ample for most applications, but for some experiments a higher accuracy is required. In these cases, and in these cases only, you should select “Options | TDT Hardware | run A/D + D/A off a single clock”. If this option is selected, both the A/D conversion and the D/A conversion will run off the clock of the A/D module. (The A/D module will “export” its clock via an internal line in the TDT System II XBUS.) With this option it is possible to achieve action potential time stamps that are accurate to within a few microseconds, depending on the conditions. In order to achieve this accuracy BrainWare interpolates linearly between the samples just below and just above the electrode channel window's trigger levels.

(Linear interpolation will give accurate results in an absolute sense only when the slope of the spikes to be time stamped is approximately a straight line. As a consequence, for optimal timing accuracy you need a good signal to noise ratio in your electrode signal which allows you to set the trigger levels well below the peaks and troughs of the spike. If the spikes cross the trigger level at a point close to the cusps of the action potential the linear interpolation will be off, but if the action potential shape does not vary much from spike to spike then the error introduced will be the same throughout, so that the relative timing information should remain accurate.)

### **Limitations of locking clocks**

As mentioned before, having independent clocks for A/D and D/A conversion brings a great deal of flexibility, and when you select “run A/D + D/A off a single clock” you sacrifice some of this flexibility to achieve higher timing accuracy.

### **Sample rates must be a multiple of 25 kHz**

BrainWare requires electrode voltage traces sampled at 25 kHz. However, a 25 kHz D/A conversion rate is inadequate for most auditory experiments. Accordingly, if “run A/D + D/A off a single clock” is selected, your stimulus modules must operate at a sampling rate that is an integer multiple of 25 kHz (i.e. 25, 50, 75, 100, 125, 150, 175, 200, ... kHz. Note that AD1 and DA1 modules will typically fall over at sampling rates higher than 170-175 kHz. They will generate random noise when driven faster). The spike train will be digitized at the higher rate, and then downsampled before analysis. Therefore, this option not only limits your choice of D/A conversion rates, but also considerably increases the demands on DSP memory on your System II AP/2 card. Also, the maximum aggregate bandwidth of the fiber optic connection is 700 kHz using play and record. Oversampling of the A/D conversion in order to run A/D and D/A from a single clock can therefore limit the number of electrode channels that can be recorded simultaneously.

### **D/A length must be the same or less than the A/D length**

Furthermore, because both A/D and D/A conversion run off the clock of the A/D module you must ensure that the duration of the D/A conversion (i.e. the sound stimulus) does not exceed the A/D conversion (i.e. the data sweep), otherwise the D/A conversion will not be completed and the system will “hang”. Nor can you trigger the sound stimulus before the sweep acquisition.

Because of these limitations it is recommended that you select “run A/D + D/A off a single clock” only if a spike timing resolution better than 40 microseconds is absolutely necessary.

When “run A/D + D/A off a single clock” is selected the color of the “scope traces” shown in the Electrode Channel Windows is blue (instead of the default green) to remind you that this option is selected.

# Chapter 3 Stimulus DLLs

One of the beauties of BrainWare is that it will produce plots of neural responses versus stimulus parameters on-line, so that you can get immediate feedback about the responsiveness of your cells. However, the Auditory Neuroscience community requires an ever increasing number of different and more sophisticated stimuli. The stimulus “Dynamic Link Library” (DLL) Interface makes it possible for you to exploit all signal processing capabilities of TDT system II, but you have to do some programming to do so.

BrainWare 6.1 requires either stimulus DLLs and/or TDT SigGen files to work. DLLs generating very simple stimuli (tones and noise bursts) are provided with BrainWare. These are provided with full source code and described in the following, and are mainly intended as examples and starting points from which you can develop DLLs that suit your own specific needs.

Once you have a DLL to generate your stimuli, you load the DLL from BrainWare’s Stimulus Grid, fill the Stimulus Grid with stimulus parameters, and you can start searching for units and recording.

## Writing Stimulus DLLs for BrainWare

DLLs are a feature of the Windows operating system. Their purpose is to bundle together a number of routines in a library to make them accessible from any number of different applications. If you want to use a novel stimulus for your recordings which you cannot obtain from SigGen then you have to write a DLL that will generate this stimulus, and load the DLL from BrainWare's Stimulus Grid.

On loading, the Stimulus Grid will then "ask" your DLL how many variable parameters your stimulus has, and what names shall be given to these parameters. (An amplitude modulated tone, for example, might have four variable parameters, called "carrier freq", "modulation freq", "duration" and "intensity".)

BrainWare then sets up the parameter table in the Stimulus Grid accordingly, and you can fill the table with values to pass to your DLL during recording.

### What you need:

In order to develop stimulus DLLs for BrainWare 6.1 you must have the software development libraries for the TDT 32-bit drivers and either a Borland Delphi Version II or higher or Microsoft Visual C++ Version 6 or higher. BrainWare 6.1 comes with three example DLLs with full source code. Two have been written with Borland Delphi (Version II or above) and, these are found in directories "DLLs\noise\noise.dpr" and "DLLs\caltone\calibton.dprcalibton\_dll" above the BrainWare install directory (by default directory "c:\TDT\Siggen\Brainware"). The third one has been written in Microsoft Visual C++, and can be found in "DLLs\c\_noise\noise.dsp".

Those who have previous experience of TDT System II APOS and XBDRV commands and programming in either C or Pascal will (hopefully) find it easy to adapt these sample DLLs to build their own, more complex, stimulus DLLs.

### Do's and Don'ts:

BrainWare will set up and arm your AD module 1 and allocate DAMA buffers for stimulus acquisition. Therefore: do not assume that particular DAMA id's are available, BrainWare may already have earmarked it as a spike data buffer. To avoid problems, use "\_allotX", rather than "allotX" APOS routines to obtain DAMA id's that are free.

Your DLL must also provide a trigger pulse to your AD module to trigger spike data acquisition (see function triggerDLLStimulus).

### Updating DLLs written for BrainWare 6.0:

BrainWare 6.0 ran on the now obsolete 16-bit drivers for TDT System II, and had to hook up to these drivers through another dynamic link library (BW\_TDT.DLL). BrainWare 6.1 runs via TDT's new 32 bit multiprocessing "S232" drivers. A consequence of this change is that writing Stimulus DLLs for BrainWare has actually become a little easier, particularly if you use C-compilers. But it also means that any DLLs written for BrainWare 6.0 have to be recompiled with s232.pas replacing bw\_tdt.pas or s232.h and s2drv32c.lib replacing bw\_tdt.h respectively. To avoid DLLs using the 16 bit drivers being linked to BrainWare 6.1 accidentally (which would lead to a crash) the required DLL function "makeStimulus()" has been renamed makeDLLstimulus()makeStimulus and it now receives stimulus parameters directly as a pointer to an array of floating point variables, rather than wrapped in the cumbersome memory objects that were required under Windows 3.1. If you need to update stimulus DLLs look through the examples below and the required changes should become obvious.

## The Stimulus Cycle

If you only ever use SigGen files as stimulus modules and you are familiar with SigGen you may not have to worry about the details of BrainWare's stimulus cycle, as these will behave largely as you would expect. However, if you wish to develop and add your own DLL stimulus modules you should familiarize yourself with this.

During data acquisition and stimulation BrainWare's stimulus modules loop through four "states" or stages, as illustrated here:

### Stage 1: "Ready to build next stimulus?"

At the start of the cycle, BrainWare queries each of the loaded stimulus modules whether they are ready to receive new stimulus parameter values from the Stimulus Grid. Note that BrainWare may return to this stage very soon after triggering the previous stimulus, and while the presentation of the previous stimulus is still ongoing. A module should signal "

not ready" when computing the next stimulus might interfere with the currently still active stimulus or the acquisition of the current data sweep. SigGen Modules signal not ready to build next stimulus until the D to A conversion of the previous stimulus is complete, to avoid overwriting the digital waveform (DAMA) buffer currently playing.

DLL Modules can export a boolean function

DLLreadyToBuildNextStimulus() to signal their readiness. DLLs which do not export this function are assumed to be always ready.

Once a module has signaled that it is ready it proceeds immediately to stage 2, it does not have to wait for other modules.

### Stage 2: Compute Stimulus

At this stage of the cycle BrainWare passes the stimulus parameter values from the stimulus grid to all the loaded stimulus modules which require them. The modules should use these parameters to make ready for the presentation of the next stimulus. For all loaded SigGen Modules BrainWare will update the values of all SigGen Variables with the values specified in the Stimulus Grid. DLL Modules can export the function computeStimulus to receive and process stimulus parameter values at this stage.

### Stage 3: “Ready for next sweep?”

At this stage BrainWare queries each of the loaded stimulus modules whether the next stimulus / sweep acquisition can be triggered. Only once all loaded stimulus modules have signalled readiness will BrainWare proceed to the next stage. A module might signal “not ready” if it has not finished computing the stimulus, or if it requires access to a resource (a D/A converter or visual stimulus generator) which has not yet completed presenting the previous stimulus. If recordings are made from awake animals, a module might signal “not ready” if the animal is not focusing or in some other way indicating that it is distracted.

DLL Modules can export a boolean function `DLLreadyForNextSweep()` to signal their readiness. It is assumed that a DLL which does not export this function is always ready.

SigGen Modules are ready for triggering soon as stage 2 is completed.

### Stage 4: Trigger stimulus

This stage follows immediately after stage 3. Unlike for the previous three stages, BrainWare will not call the triggering functions for all the loaded stimulus modules. Such sequential calls would not ensure accurate synchrony of potentially time critical processes. BrainWare therefore only calls the triggering functions associated with the first Module in the list of loaded modules which provides triggering. If no module defines a trigger function BrainWare will send a global XBUS trigger (see function `xb1gtrig()` in the TDT XBDRV handbook. Note that the global XBUS trigger may not work if you connect something to the trigger line inputs of your A/D or D/A modules, so make sure these are not connected if you wish to use default triggering behaviour.

DLL Modules can export a procedure `triggerDLLstimulus` to define a triggering procedure. SigGen Modules automatically define a triggering procedure that implements the settings of the TG6 timing generator specified in the SigGen file (provided a TG6 module is detected).

Once the trigger has been sent, BrainWare proceeds through to stage 1 again almost immediately, but stage 3 will not be reached again until an interval equal to or greater than that specified as “Repeat Rate” in the sweep parameter dialog has elapsed.

## Stimulus DLL Functions

These are the functions that your stimulus DLL can export to take part in the stimulus cycle:

```
computeStimulus  
DLLreadyToBuildNextStimulus  
DLLreadyForNextSweep  
triggerDLLStimulus
```

You can use these functions to configure your DLL at start up, runtime or on exit:

```
initializeStimulusDLL  
setDlloptions  
tidyUpStimulusDLL
```

By exporting these functions your DLL can set up the stimulus parameter table:

```
defaultStimParam  
numDefaultConditions  
numStimParams  
parameterName
```

If your DLL wants to present acoustic stimuli through one or both D/A channels it should export

```
DParameters
```

If you want your DLL to be able to send messages to the user via BrainWare's "Messages" window you can access this via function

```
getMessageFunc
```

To prevent stack errors make sure that all these functions are declared with "cdecl" (C-Declaration) Calling Conventions (see your compiler's documentation, if you are unfamiliar with this).

Note that DLL exports are case sensitive, even if procedure declarations in Pascal aren't.

## Writing stimulus DLLs with Delphi

The easiest way to write a stimulus DLL in Delphi is to copy the code of one of the existing demo DLLs and adapt it for your purposes.

Delphi 4 can run DLLs from the IDE Desktop, a great help for debugging: once your DLL compiles correctly you can set Run|Parameters in Delphi 4 to Brainware32.exe, then use Run to launch BrainWare, load your new DLL to the Stimulus Grid, and then set breakpoints in your functions and step through your code.

The following is a simple step-by-step example. Let us assume that you want a stimulus DLL to generate binaural tone stimuli. Let's assume that you wish to keep the rise-fall time fixed at 5 ms but want to be able to vary the duration and set the frequency and in each ear (channel) separately. We therefore have three variable parameters, "Duration", "Freq1" and "Freq2".

To generate this DLL do the following:

1. Start Delphi, and open the project "modules\source\delphi\noise\noise.dpr".
2. Generate a copy of that project with menu File|Save Project As. Call the new project "bitone.dpr".
3. To be able to access the TDT APOS and XBDRV functions, your DLL should link to the TDT s232 driver. In order to achieve this, all you have to do is make sure that your project source has a "uses" statement that includes s232 and you must ensure that the directory with file "s232.pas" (typically c:\tdt\s232\Delphi") is in the compiler's search path (accessed via Project|Options|Directories). If you create a new DLL by copying and editing an old one, as described here, then this should be automatically the case.
4. Edit the constants "riseFall" and "DAperiod" to set the desired ramping, and sampling period. Add a constant "duration" to give the desired stimulus duration in ms.
5. The global variables DamaBufID and DamaBufSize are used to handle the DAMA Buffers that will hold your stimuli. We will need a second DAMA buffer, so replace  
DamaBufId : integer; with  
DamaBufId1, DamaBufID2 : integer;
6. Function initializeStimulusDLL can stay as it is, as can DLLreadyToBuildNextStimulus and DAparameters.
7. The return value for function numStimParams has to become 3 (instead of 2)
8. Change the case statement in function parameterName to read case i of  
1 : result:='Duration1 [ms]';  
2 : result:='Freq1 [Hz]';  
3 : result:='Freq2 [Hz]';  
else result:='Error';  
end; { case }

9. Of course we also want our new DLL to provide appropriate default stimulus parameter values. If we change the case statement in function `defaultStimParam` to read

```
case paramIdx of
  1 : result:=2000 {ms};
  2 : result:=700 {Hz};
  3 : result:=700+paramIdx/10 {Hz};
else result:=0;
end;
and add a function numDefaultConditions as follows:
function numDefaultConditions : integer; cdecl;
begin
  result:=5;
end;
```

We can get BrainWare to set up the Stimulus Grid for us to produce "binaural beat stimuli", which, if presented over headphones, will give an illusion of slow auditory motion from the midline towards channel 2. The motion should be slowest for stimulus 1 and increasingly faster for subsequent stimuli. (Don't forget that the new function `numDefaultConditions` must also be listed under "exports", near the end of the file!)

Function `computeStimulus` has to undergo a number of changes. To see what `makeDLLStimulus` should look like to produce binaural tone stimulation click [here](#).

10. Given that we now wish to play sounds on two channels we must also change function

`DParameters`, to read as follows:

```
procedure DParameters(var DAChan, damaID1, damaID2:smallint;
  var npts:longint; var speriod: single ); cdecl;
begin
  DAChan:=DUALDAC;
  damaID1:=DAMABufID1;
  damaID2:=DAMABufID2;
  npts:=DAMABufSize;
  speriod:=DAperiod;
end;
```

11. Finally we have to make sure the two DAMA buffers we are used are released when we're done: in function `tidyUpStimulusDLL` replace

```
deallot(DamaBufId);
with
begin
  deallot(DamaBufId1);
  deallot(DamaBufId2);
end;
```

12. Compile, run Brainware, load the new "bitone.dll" using `Configure Stimulus Modules | Add Module` from the StimulusGrid and click on "Search" to test your stimuli.

---

13. Remember, if you want to add intensity control and you have PA4 programmable attenuator modules, you do not have to alter your code. Simply add modules “

PA4\_1.dll” and “PA4\_2.dll” from directory “Modules”. That’s the beauty of the modular approach.

## Writing stimulus DLLs with MSVC++

In order to write any software in C for the new TDT 32-bit drivers you must have Version 5 or higher of Microsoft Visual C.

You can use the debugger built into the MSVC++ Editor to debug stimulus DLLs you are writing in MSVC. To do that, in version 6 you select Project | Settings from the MSVC++ Window Menu, select the Debug Tab, click on the little arrow to the right of the input box labelled Executable for debug session and locate BrainWare32.exe (typically in c:\TDT\Siggen32\BrainWare32\BrainWare32.exe). Set breakpoints at the beginning of the functions you wish to debug, then choose Build | Start Debug | Go. A warning will appear that BrainWare.exe does not contain any debug info. Ignore that and click "OK". BrainWare will be launched and you can load and use your DLL in the usual manner, but the execution of BrainWare will be interrupted when one of your breakpoints is reached, and you can then step through your code and watch the values variables take, etc. Very useful!

The following is a simple step-by-step example. Let us assume that you want a stimulus DLL to generate binaural tone stimuli. Let's assume that you wish to keep the rise-fall time fixed at 5 ms but want to be able to vary the duration and set the frequency and in each ear (channel) separately. We therefore have three variable parameters, "Duration", "Freq1" and "Freq2". The DLL is to be called "bitone.dll".

To generate this DLL do the following:

1. Start Microsoft Visual C and create a workspace for the new DLL. (In MSVC++ Version 5 this is done by choosing File | New | Projects | Win32 Dynamic Link Library. Set Project Name to "bitone" and choose a suitable new directory for Location. In MSVC++ you will be asked whether you'd like an "empty library" or a host of other options. Choose the "empty library")
2. Copy the files "Noisedll.cpp", "Noisedll.def" from "c:\TDT\SigGen32\BrainWare32\Modules\Source\VC\noise" to your new project location. Rename the "noisedll" files to "Bitone.cpp" and "Bitone.def".
3. "Insert" the files "bitone.cpp" and "bitone.def" to your new project (In MSVC++ Versions 5 and 6 this is done with Project | Add to Project | Files). Also add files "s232.h" and "s2drv32c.lib" from "c:\TDT\s232\VC" to your project. These two files are part of the software development libraries for TDT's 32-bit driver, and provide access to the APOS and XBDRV functions.
4. File "bitone.def" simply lists the DLL functions your DLL is going to export. Change "LIBRARY noise" to "LIBRARY bitone". Also add numDefaultConditions to the end of the list of exported functions (see below).
5. If you wish you can change the definitions for "riseFall" and "DAperiod" to set the desired ramping, and sampling period.
6. Change the definition of numVarStimParams from 2 to 3.

7. The global variables DamaBufID and DamaBufSize are used to handle the DAMA Buffers that will hold your stimuli. We will need a second DAMA buffer, so replace  
`int DamaBufID;` with  
`int DamaBufID1, DamaBufID2;`

8. Function `initializeStimulusDLL` can stay as it is, as can `DLLreadyToBuildNextStimulus` and `triggerDLLStimulus`.

9. The return value for function `numStimParams` has to become 4 (instead of 2)

10. Change the "switch" statement in function `parameterName` to read

```
switch(i) {
case 1:
    sprintf(paramName,"Duration [ms]"); break;
case 2:
    sprintf(paramName,"Freq1 [Hz]"); break;
case 3:
    sprintf(paramName,"Freq2 [Hz]"); break;
default:
    sprintf(paramName,"Error, no param #%"d",i);
}
```

11. Function `computeStimulus` has to undergo a number of changes. To see what `computeStimulus` should look like to produce binaural tone stimulation [click here](#).

12. We have to make sure the two DAMA buffers we are used are released when we're done: in function

`tidyUpStimulusDLL` replace  
`deallot(DamaBufId);`

with

```
{
    deallot(DamaBufID1);
    deallot(DamaBufID2);
}
```

13. We must change function `DParameters` to inform BrainWare that we now require two-channel output. Change the function to read:

```
void cdecl DParameters(short* DAChan, short* damaID1,
    short* damaID2, long* npts, float* speriod)
{
    *DAChan =DUALDAC;
    *damaID1 =DamaBufID1;
    *damaID2 =DamaBufID2;
    *npts =DamaBufSize;
    *speriod =DAperiod;
}
```

You may also wish to delete the declaration of global variable "myChannel", as that is no longer required.

14. Finally let us make sure that the DLL sends some sensible default values for our stimulus parameters to the stimulus grid. By changing the switch block in function `defaultStimParam` to

```
switch (paramIdx)
{
    case 1 : return 2000;
    case 2 : return 700;
    case 3 : return 700+float(rowIdx)/10;
    default : return 0;
}
```

and adding a function `numDefaultConditions` as follows:

```
short cdecl numDefaultConditions()
{
    return 5;
}
```

we can get BrainWare to set up the Stimulus Grid for us to produce "binaural beat stimuli", which, if presented over headphone, will give an illusion of slow auditory motion from the midline towards channel 2. The motion should be slowest for stimulus 1 and increasingly faster for subsequent stimuli.

15. Build the DLL, run Brainware, load the new "bitone.dll" using Configure Stimulus Modules | Add Module from the StimulusGrid and click on "Search" to test your stimuli.

16. Remember, if you want to add intensity control and you have PA4 programmable attenuator modules, you do not have to alter your code. Simply add modules "

PA4\_1.dll" and "PA4\_2.dll" from directory "Modules". That's the beauty of the modular approach.

## DLL Function Definitions

### computeStimulus

#### Prototypes

```
function computeStimulus(params :
PParameters):boolean; cdecl;
(PParameters being type of pointer to an array
of single, see below)
BOOL cdecl computeStimulus(float * params)
```

Note: In previous versions of BrainWare this function was called “makeStimulus” and received the parameter array wrapped in a “global memory object”.

Brainware will call this function every time it wants to inform your stimulus DLL of the stimulus parameters for the next stimulus. As argument function computeStimulus() receives an array of floating point numbers (type single, i.e 32-bit float) containing the numerical values of the required stimulus parameters. Note that when you write in C this array will be zero-based, in Pascal it is base one.

Your DLL should export function numStimParams to indicate how many stimulus parameter values it needs (i.e. how big the “params” array should be). If your DLL does not export a numStimParams() function it is assumed that it requires no numerical stimulus parameters, and “params” will be an invalid pointer.

The function makeDLLStimulus should return true if successful, false on error (e.g. if illegal parameter values were passed).

Remember that this function may be called while the presentation of the previous stimulus is still ongoing. This is done so that computeStimulus() can take advantage of the time available during the acquisition of the previous sweep to perform time consuming computations, should this be necessary. Consider the following example, from a DLL that controls a PA4 programmable attenuator:

```
type
  TParameters = array [1..100] of single;
  PParameters = ^TParameters;
function computeStimulus(params : PParameters):boolean; cdecl;
begin
  PA4atten(1,params^[1]); // WRONG!!
  result:=true;
end;
or, in C
BOOL cdecl computeStimulus(float * params)
```

```
{
  PA4atten(1,params[0]); // WRONG!!
  return TRUE;
}
```

This is wrong, since the attenuator settings may change as the previous stimulus is still being presented. A correct solution is implemented in the PA4.DLL modules supplied with BrainWare, (Delphi source code in Modules\Source\Delphi\PA4) which use the function DLLreadyForNextSweep to circumvent this problem as follows:

```
var
  myatten : single;
function computeStimulus(params : PParameters);boolean; cdecl;
begin
  myatten:=params^1]; // copy parameter value to use later
  result:=true;
end;
function DLLreadyForNextSweep : boolean; cdecl;
begin
  // Don't change attenuator until previous DA conversion is complete
  result := (DAstatus(1) <> 0);
  if result then
    PA4atten(1,myatten);
end;

or in C:
float myatten;
BOOL cdecl computeStimulus(float * params)
{
  myatten=params[0]; // copy parameter value to use later
  return TRUE;
}
BOOL cdecl DLLreadyForNextSweep()
{
  // Don't change attenuator until previous DA conversion is complete
  if (DAstatus(1) != 0)
  {
    PA4atten(1,myatten);
    return TRUE;
  }
  else
    return FALSE;
}
```

Note that `computeStimulus` must copy the values of the parameters you may wish to process later into variables declared in your DLL. Just remembering the address pointed to by “params” may seem like a clever idea, but it will get you into trouble because the pointer “params” is no longer valid once `computeStimulus` returns. So don’t do this:

```
var
  myParamPtr : PParameters;
function computeStimulus(params : PParameters):boolean; cdecl;
begin
  myParamPtr:=params; // NOT USEFUL
  result:=true;
end;
function DLLreadyForNextSweep : boolean; cdecl;
begin
  ...
  PA4atten(1,myParamPtr^[1]); // WRONG !!!
end;
```

```
or in C:
float * myParamsPtr;
BOOL cdecl computeStimulus(float * params)
{
  myParamsPtr = params; // NOT USEFUL!
  return TRUE;
}
BOOL cdecl DLLreadyForNextSweep()
{
  ...
  PA4atten(1,myParamsPtr[0]); // WRONG!!!
  ...
}
```

If it is unavoidable that your `computeStimulus` function will interfere with the previous stimulus, use function `DLLreadyToBuildNextStimulus` to block premature calls of `computeStimulus()`.

## DLLReadyToBuildNextStimulus

### Prototypes

**function DLLReadyToBuildNextStimulus : boolean; cdecl;**

BOOL cdecl DLLreadyToBuildNextStimulus()

If exported, this function signals whether the DLL is ready to proceed from stage 1 to stage 2 in the stimulus cycle.

To work efficiently, Brainware will try to instruct your stimulus DLL to start building the next stimulus while the recording of the previous sweep of data is still ongoing. However, your stimulus generator may not yet be ready to receive new commands. In this case you should provide a DLLReadyToBuildStimulus function, which should return false if your DLL can not yet build the next stimulus.

A typical example: you might be writing a DLL in Delphi for TDT System II, and the system has to wait for D/A conversion to finish before the D/A converter can be set up for the next stimulus. In that case your function might look like this:

```
function DLLReadyToBuildNextStimulus:boolean;  
begin  
    result := (DAstatus(1)=0);  
end;
```

In some cases it is possible and desirable to split the setting up of the next stimulus into two parts, one which does not interfere with DA conversion, and a later part which would interfere. In these cases it is preferable NOT to define a function DLLReadyToBuildNextStimulus, but use the function makeStimulus to execute only the early parts of stimulus generation, and finish stimulus preparation (e.g. transfer of the stimulus to buffer) in function DLLReadyForNextSweep or triggerDLLStimulus. But note: it is bad practice to leave data on the TDT System II stack between functions makeStimulus and triggerStimulus.

## DLLReadyForNextSweep

### Prototypes

```
function DLLReadyForNextSweep : boolean; cdecl;  
BOOL cdecl DLLReadyForNextSweep()
```

If exported, this function signals whether the DLL is ready to proceed from stage 3 to stage 4 in the stimulus cycle. It will be called after `computeStimulus`, and before `triggerDLLstimulus`.

If the function returns "false" (i.e. "not ready") it will be called again, approximately once every 100 ms, until it returns true or the user aborts stimulation.

## triggerDLLStimulus

### Protopes

```
procedure triggerDLLStimulus; cdecl;  
void cdecl triggerDLLStimulus()
```

This procedure will be called after computeStimulus, and should provide any code required to set off your stimulus and data acquisition. Note that Brainware will set your systems ADx module up for data acquisition, and that that module will be "armed". In the simplest case your trigger routine might simply set off your DA module, and you would connect the "SYNC" port on your DA module to the "TRIG" port on your AD module.

Example:

```
procedure triggerDLLStimulus;  
begin  
    DAgo(1);  
end;
```

## initializeStimulusDLL

### Prototypes:

```
function initializeStimulusDLL: boolean; cdecl;  
BOOL cdecl initializeStimulusDLL()
```

If present, this function will be called when the stimulus DLL is loaded. You can use this function to initialize DAMA buffers, read calibration tables, open windows that provide feedback, or whatever. The function should return true if successful. If the function returns false BrainWare will assume that your DLL will not be able to operate correctly, and will stop loading the DLL.

## setDLLOptions

### Prototypes

**procedure setDLLOptions; cdecl;**

void cdecl setDLLOptions()

If present, BrainWare will create an additional menu item in its main menu under StimulusGrid | DLL Options. When the user selects this menu item, the DLL function setDLLOptions will be called. You can create your own menu dialogs or popup menus which will then be shown in response to this menu selection, and allow the user to set some optional configurations.

### *Example:*

Suppose you were writing a stimulus DLL in Delphi and wanted to give the user the option to load calibration files which calibrate the output for a particular transducer. You might then include the following code in your DLL:

...

```
procedure setDLLOptions; cdecl;
var
  filename : string;
begin
  filename :=inputBox('Calibration file',
    'Calibrate output according to
file', 'default.nrm');
  loadCalibrationFile(filename);
end;
...
exports
  ...
  setDLLOptions;
```

## tidyUpStimulusDLL

### Prototypes

```
procedure tidyUpStimulusDLL; cdecl;  
void cdecl tidyUpStimulusDLL()
```

If present, this function will be called when the stimulus DLL is "unloaded". You should use this function to free up any memory or DAMA buffers that your DLL may have allocated, or close any windows or files your DLL may have opened.

## numDefaultConditions

## defaultStimParam

### Prototypes

```
function numDefaultConditions : integer; cdecl;
```

```
function defaultStimParam(paramIdx, rowIdx : integer): single;  
cdecl;
```

```
short cdecl numDefaultConditions()
```

```
float cdecl defaultStimParam(int paramIdx, int rowIdx)
```

These functions can be used to instruct BrainWare to fill the Stimulus Grid with suitable default parameter values when a Stimulus Module is loaded.

For example, assume that your Stimulus DLL defines two variable stimulus parameters, and that the first parameter should by default assume the value 5.75, and the second should run through the values 10, 20, 30, ... 100.

You can make sure that the Stimulus Grid will be filled with these default values when your DLL gets loaded by exporting the following functions:

### In Pascal:

```
function defaultStimParam(paramIdx, rowIdx : integer):  
single; cdecl;
```

```
begin
```

```
    case paramIdx of
```

```
        1 : result:=5.75;
```

```
        2 : result:=rowIdx*10;
```

```
    end;
```

```
end;
```

```
function numDefaultConditions : integer; cdecl;
```

```
begin
```

```
    result:=10;
```

```
end;
```

### Or in C:

```
float cdecl defaultStimParam(int paramIdx, int rowIdx)
```

```
{
```

```
    switch (paramIdx)
```

```
    {
```

```
        case 1 : return 5.75;
```

```
        case 2 : return rowIdx*10;
```

```
        default : return 0;
    }
}
short cdecl numDefaultConditions()
{
    return 10;
}
```

Bear in mind that your DLL might be loaded together with, and "behind" another stimulus module which specifies a higher value for numDefaultConditions. In that case of our example here this might mean that function defaultStimParam could be called with a value for rowIdx greater than ten, and return a default parameter outside the range 10..100 we had planned, which may, or may not, be a problem when computeStimulus is called. Always make sure that computeStimulus traps illegal values!

## numStimParams

## parameterName

### Prototypes

```
function numStimParams : integer; cdecl;
function parameterName(i:integer):pchar; cdecl;

short int cdecl numStimParams()
char * cdecl parameterName(short i)
```

Use these function to inform the Stimulus Grid of the number and names of variable stimulus parameters for your DLL.

If, for example, you were to write a stimulus DLL to generate tones of varied frequencies, durations and intensities these functions might look like this:

```
function numStimParams : integer;
begin
    result:=3;
end;

function parameterName(i:integer):tParameterName;
begin
    case i of
        1 : result:='Freq [Hz]';
        2 : result:='SPL [dB]';
        3 : result:='Duration [ms]';
    else
        result:='Error';
    end; { case }
end;
```

## DParameters

### Prototypes

```

procedure DParameters(var DAChan, damaID1,
damaID2:smallint;
    var npts:longint; var speriod: single ); cdecl;
void cdecl DParameters(short * DAChan, damaID1, damaID2;
    long int * npts; float * speriod)

```

If your stimulus DLL should generate sound stimuli to be played on the DA module of your TDT system II rack, then your DLL must export a DParameter function to pass the stimuli to be played to BrainWare. Your DLL should not set up the DA module itself through calls to the S232 driver, because this would make it impossible to reconcile the requirements of your DLL with those other stimulus modules might have which may be loaded concurrently. (However, if you have overriding technical reasons why your DLL should take direct control of the D/A module then your DLL can "monopolize" the module. See below.)

To have a sound played on the DA module, simply pop the sound into a DAMA buffer, then pass the ID of the DAMA buffer to BrainWare by setting damaID1 or damaID2 respectively when function DParameters is called. For example, if your DLL wants to play a 50000 point long digital stimulus at a sampling rate of 100 kHz (sampling period 10ms) from DAMA buffer "myDamaBuf" on channel one, then simply export the following function:

```

procedure DParameters(var DAChan, damaID1,
damaID2:smallint;
    var npts:longint; var speriod: single );
cdecl;
begin
    DAChan:=1;
    damaID1:=myDamaBuf;
    npts:=50000;
    speriod:=10;
end;

```

### or in C:

```

void cdecl DParameters(short* DAChan, short* damaID1,
short* damaID2, long* npts, float* speriod)
{
    *DAChan =1;
    *damaID1 =myDamaBuf;
    *npts =50000;
    *speriod =10;
}

```

```
}
```

If you wish to output to channel 2, assign damaID2 (instead of damaID1) to "myDamaBuf" and set DAChan to 2. If you want dual-channel, stereo sound, set DAChan to "DUALDAC" (this is a constant defined in s232.h or s232.pas), and assign damaID1 and damaID2 to the respective DAMA buffer IDs. (But consider that it may be possible, and in some cases preferable, to achieve two channel stimulation through two separate stimulus DLL modules).

Function DAparameters is called at the end of the 3rd stage of the stimulus cycle, after function DLLreadyForNextSweep and before function triggerDLLStimulus (if your DLL exports these functions). DAparameters is called every time BrainWare loops through the stimulus cycle. Your DLL can therefore request different output channels or DA conversion rates on a stimulus by stimulus basis, for example as a function of the stimulus parameters that were passed to computeStimulus.

In some very specialised cases it may be necessary for your DLL to take direct control of the D/A module, rather than leaving the scheduling of D/A to BrainWare. An example would be very high frequency stereophonic sounds for research in bat hearing which have to be digitized from two AP2 cards in order to achieve sufficiently high sampling rates. In cases like this your DLL can "monopolize" the D/A module by setting the variable DAChan to -1, as in the following example:

```
const
    MONOPOLIZE = -1;

procedure DAparameters(var DAChan, damaID1, damaID2:smallint;
    var npts:longint; var speriod: single ); cdecl;
begin
    DAChan:=MONOPOLIZE;
end;
```

**or in C:**

```
#define MONOPOLIZE -1

void cdecl DAparameters(short* DAChan, short* damaID1, short*
damaID2, long* npts, float* speriod)
{
    *DAChan = MONOPOLIZE;
```

```
}
```

In these cases your DLL should issue its own `DAmode()`, `DAnpts()`, `DAsrate()` and `DAarm()` commands at appropriate points in the stimulus cycle.

NOTE: some stimulus modules, including SigGen files, may signal NOT ready to build next stimulus if the DA module is armed. If you load such a module together with a DLL that issues a `DAarm` command before the function `triggerDLLStimulus` is called, then the modules may not be able to progress through the stimulus cycle.

## getMessageFunc

### Prototypes

```
type TPointerProc = procedure(aPtr: Pointer); cdecl;
procedure getMessageFunc(aMsgFunc : TPointerProc); cdecl;
```

```
typedef void (__cdecl * voidPCharFunc) (char *);
void cdecl getMessageFunc(aMsgFunc : voidPCharFunc)
```

More than likely the time will come when you want your stimulus DLLs to be able to generate warnings, debugging messages or other feedback for the user. Your DLL can, of course, create it's own message boxes. Alternatively your DLL can get access to BrainWare's "Messages" window and post messages there. Accessing the Messages Window is somewhat indirect. The functions writing to that Window are defined in BrainWare, but DLLs usually only export functions, they don't import them. To get around that "one way street" your DLL can export a function "getMessageFunc", which, if defined, will be called when the DLL is first loaded, and will receive as argument a pointer to a function which can write C-strings (type Pchar) to the Messages Window. All you have to do is save that pointer in a global variable of type "procedure of pointer", and then use that variable as your output procedure. The following Pascal code fragments illustrate this.

```
type
  TPointerProc = procedure(aPtr: Pointer); cdecl;
...
var
  BWmessage : TPointerProc;
...
procedure getMessageFunc(aMsgFunc : TPointerProc); cdecl;
begin
  BWmessage:=TPcharProc(aMsgFunc);
end;
...

function computeStimulus(params:PParameters):boolean; cdecl;
begin
  BWmessage(pchar(
    format('now computing stimulus with params %g, %g',
      [params^[1],params^[2]])));
  ...
end;

exports
```

```
computeStimulus,  
...  
getMessageFunc;
```

# DDE (Dynamic Data Exchange) with BrainWare

Dynamic Data Exchange, abbreviated DDE, is a means by which different programs concurrently running under Microsoft Windows can "hold a conversation" and exchange data.

Through DDE you can write programs which send instructions to BrainWare and receive information back. Thereby you can automate some tedious, repetitive tasks, i.e. setting up a particular stimulus configuration. DDE is supported by a large number of Windows tools and programming languages, including all Borland and Microsoft Compilers (Delphi, Visual C, Visual Basic), Matlab, even Microsoft Word and Excel, so hopefully DDE will allow you to customize your control of BrainWare with the software tools of your choice.

At present the DDE capabilities of BrainWare are still somewhat limited, but further capabilities are continuously added to make BrainWare more flexible. So if you require DDE access to some parameters of BrainWare that are not currently supported, contact TDT ([tdt@tdt-quikki.com](mailto:tdt@tdt-quikki.com)).

To use DDE with BrainWare, make sure you are familiar with DDE fundamentals, then look at the tables of defined DDE "topics" and "items" and peruse some of the examples referenced there.

## ***DDE Basics***

### **Client and Server**

The following explains the key concepts of "client", "server", "conversation", "topic" and "item" which you must be familiar with in order to make use of DDE.

In a typical DDE scenario one Windows application acts as a "client" and requests information or issues commands, another application acts as "server" and responds to these requests. In order to send it's requests the client has to link up to the server, i.e. it has to "establish a conversation". In the cases discussed here, BrainWare will always act as DDE server. In the following you will find examples of how to set up DDE Clients that can interact with BrainWare.

## Using DDE with BrainWare

### Initializing DDE

The client tries to establish a communications link to the server. This is also referred to as "opening a conversation". Programming languages that support DDE will provide functions called something like "DDEinitiate" or "DDEinit" for this purpose.

The DDEinit command typically takes two parameters. The first is the name of the executable of the server application (for BrainWare this is "BrainWar" without "e" at the end). The second parameter is a string that identifies the "topic of the conversation". BrainWare currently supports two topics, "Settings", and "StimulusTable". The former provides access to recording parameters and defaults, the second gives access to BrainWare's stimulus grid. All DDE conversations with BrainWare therefore begin with a command like

```
myDDEchan=ddeinit('BrainWar','Settings')
```

or

```
myDDEchan=ddeinit('BrainWar','StimulusTable')
```

or equivalent.

Future versions of BrainWare will include DDE topics that allow access to open data files. The opening of a DDE conversation may fail if the server application is not active or cannot be found or started or is in some way busy or blocked, or if the specified "topic" is not known. A client may need to provide appropriate error handling for these cases. If successful, DDEinit functions will typically return a number that specifies the established DDE channel.

### Macros

If the conversation opens successfully the client can then send command strings (also called "macros") or requests for information through the open channel to the server. For that purpose software tools provide commands called "executeMacro" or "DDEexecute" or "DDEexec". Macro commands are essentially a one-way street: the client tells the server what to do, but will not receive any information back. A successful execute command typically only indicates that the macro was dispatched successfully, not that it was carried out successfully. This lack of feedback being rather unappealing, BrainWare does not make much use of macros.

### Items

In order to have an exchange that can go both ways DDE servers can define so called "items". For example, BrainWare defines item "PresOrder" under the topic "Settings", which allows you to access the current setting for the stimulus presentation order as defined in the Sweep Parameter Dialog.

### Requesting Data

You can read the current value of an item using a “DDErequest” command.

### Changing or “Poking” Data

You can attempt to change the value of an item by “Poking” a new value to that item. Performing a DDErequest after a DDEpoke you can check whether your attempt of changing the settings was successful.

### Terminating the DDE Conversation

At the end of the DDE conversation the client application should close open communication channels using “DDEterminate” commands.

## *Supported DDE Topics and Items*

The following DDE “Items” are defined under the DDE topic “BrainWare”

### DDE topic “Settings”

<b>SweepLen</b>	integer, returns / sets the value of "Sweep Length" in the Sweep Params Dialog.
<b>StimPeriod</b>	integer, returns / sets the value of "Repeat Rate" in the Sweep Params Dialog.
<b>StimRepeats</b>	integer, returns / sets the value of "# of repeats" in the Sweep Params Dialog.
<b>PresOrder</b>	integer, returns / sets the status of "Batch Presentation Order" radio button box in the Sweep Params Dialog. Value 0 corresponds to "Interleaved" 1 corresponds to "Random Interleaved" 2 corresponds to "Block" 3 corresponds to "Sequence File".

### DDE topic “StimulusTable”

<b>Rows</b>	integer, returns the number of rows in the stimulus grid (i.e. the number of currently defined stimulus conditions).
<b>Columns</b>	integer, returns the number of columns in the stimulus grid (i.e. the number of variable parameters in a stimulus).
<b>Row</b>	integer, returns / sets the current ("selected") row in the stimulus grid.

<b>Column</b>	integer, returns / sets the current ("selected") column in the stimulus grid.
<b>Cell</b>	floating point number, returns / sets the value of the stimulus parameter in the table at the current cursor position (which can be accessed / set through Items "Row" and "Column" above). To change a value by poking data to this Item the Stimulus Grid must be in edit mode (see Item "Editing" below).
<b>Contents</b>	comma separated list of floating point numbers corresponding to the contents of the stimulus parameter table. Consecutive rows are separated by newline characters. BrainWare only updates this item when editing is completed, so you cannot use requests of this item to monitor editing that is still being performed. You can "poke" lists to this item to set up a new table of parameters. The stimulus grid must be in edit mode to accept the poked data (see Item "Editing" below).
<b>Editing</b>	integer, returns / sets edit status of the Stimulus Parameter table. Values greater than zero indicate that the table can be / is being edited. It's use is also illustrated in this example.

## Using DDE to set up BrainWare's Stimulus Grid

### An Example using Matlab

The following Matlab function illustrates the use of DDE to fill the stimulus grid with numerical parameters. The stimulus grid is set to edit mode by poking '1' to the DDE Item "Editing". A comma-separated list of values is built in variable "text" using a loop and Matlab's "sprintf" function. Finally the list is poked to the DDE Item "Contents".

```
function fillSin;
% function fillSin;
%   example showing how to use DDE to fill BrainWare's
%   Stimulus Table
%   with parameter values calculated by Matlab
%   The first column of the grid will be filled with
%   the values 0-360 in steps of 20,
%   the second column will contain the sine of the value in
%   the first column.
%   The third and fourth columns will be filled with zeros
%   (c) Jan Schnupp, Jan 1999

% Establish DDE conversation

chan=ddeinit('BrainWar','StimulusTable');
if chan == 0
    error('Cannot open DDE channel to BrainWare');
end;

% To set Stimulus Grid to Edit Mode we poke '1' to Item
'Editing'
ddepoke(chan,'Editing','1');
% Verify that we can now edit the stimulus grid
%   by reading back the value of Item 'Editing'
if ddereq(chan,'Editing',[1,1]) ~= '1'
    % if the value returned is not the one we poked

    % this indicates that Stimulus Grid cannot be edited,
    % perhaps because recording is active. => abort.
    ddeterm(chan); % close DDE channel
    error('Cannot edit stimulus grid!');
end;

% Build the text to transfer
```

```
text=''; % start off with empty string
for i=0:20:360,
    if ~ isempty(text)
        % add new line char (ASCII 10) to separate lines
        text=[text, char(10)];

    end;
    % fill each line with "<value>, <sin(value)>, 0, 0"
    text=[text, ...
        sprintf('%d,%3.3f,0,0',i,sin(i*pi/180))];
end;

disp(text); % show text to transfer to BrainWare
ddepoke(chan,'Contents',text); % poke new contents to
Stimulus Grid
ddepoke(chan,'Editing','0'); % finish editing Stimulus Grid

ddeterm(chan); % close DDE channel
```