# *TclBatchCommands Tutorial*

*Compumedics Neuroscan*
7850 Paseo Del Norte
El Paso, TX 79912
USA

Voice :915-845-5600
Fax: 915-845-2965
e-mail: sales@neuro.com
techsup@neuro.com
web site: www.neuro.com




Document number 2283, Revision B

**For Technical Support with BATCH files, we strongly recommend you send your questions via e-mail to techsup@neuro.com, and that you include your batch file as an attachment to the e-mail. This will greatly facilitate our troubleshooting, it will speed the response, and it will give you an electronic copy to refer to. Tech Support will help with problems specific to SCAN commands, however, you should refer to a Tcl text for more complex issues regarding Tcl commands and functionality. Please see our batch web site for additional information and support options (http://www.neuro.com/neuroscan/batch.htm).**



***Compumedics Neuroscan***
7850 Paseo Del Norte
El Paso, TX 79912
Telephone: 915-845-5600
Fax: 915-845-2965

E-mail: techsup@neuro.com
Web: www.neuro.com



Document number 2283, Revision B

# *TableofContents*





Document number 2283, Revision B

# IntroductiontoTCLBATCHFiles

The purpose of this tutorial is to introduce you to Tcl BATCH files in a quick and easy way. We will start with some basic information that you need to know, and then write some basic BATCH files. The files will become increasingly complex as the tutorial proceeds. By the end, you should have sufficient knowledge to begin writing your own BATCH files.

In the SCAN programs there are actually two ways to do "batch" file processing: Scripts (in EDIT), and Tcl BATCH files. "Scripts" are created and executed entirely within EDIT. Tcl BATCH files are created in a text editor, and are executed in ACQUIRE or EDIT. Scripts are perfectly adequate for many of the basic off-line operations you may wish to perform. The primary advantage of Script files is the easy user interface. You set up the commands using the same dialog boxes you use in Point&Click mode. In BATCH files, the parameters associated with a given command are simply listed on the command line. This requires that you use the BATCH manual, or the Quick Reference Guide at the end of the BATCH manual, to see the parameters that must be included with the commands.

The major advantage with Tcl BATCH files is that you are using a programming language - Tool Command Language. You are limited mainly by the constraints of the language. Normally, Tcl will let you do just about anything you can imagine pertaining to BATCH files. For fairly routine operations, the information in the Tcl BATCH manual, and in this Tutorial will be sufficient. For more complex programming, you should buy a Tcl textbook. A good one is: Tcl and the Tk Toolkit by John K. Ousterhout (the Tcl author). Please see http://www.neuro.com/neuroscan/batch.htm for a link to a site to order the book, as well as for additional support options.

Parts of the information below are also found in the Tcl BATCH manual, where the description is more complete. Highlights from these sections are reproduced here for your convenience.

## UsingthisPDFFile

It is intended that you create the BATCH files manually as you go through this Tutorial. At some points, however, you may find it useful to Copy lines from the PDF file and Paste them into the BATCH file you are creating. You can do this by changing the cursor in the Adobe Reader to the text tool  , and then highlighting the lines that you want to Copy:


Then use the standard Copy (for example, Ctrl+C, or Edit / Copy) and Paste (for example, Ctrl+V, or Edit / Paste) commands to Paste the lines into the BATCH file you are creating. Note that BATCH command text will always appear using the ARIAL font.

Before creating a BATCH file, you need to be familiar with the basic structure of the command line.



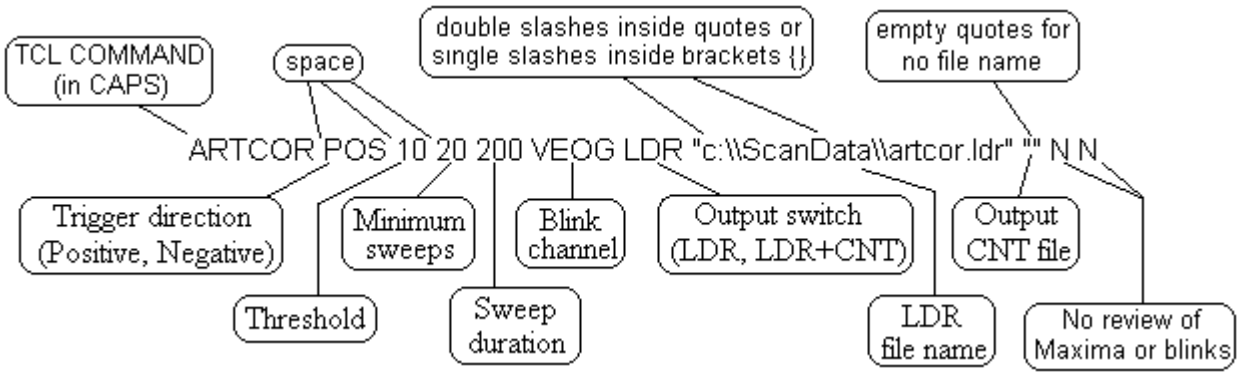
Document number 2283, Revision B

## Tcl BATCH Command Structure

Let's look first at a typical command. Below is how the command for applying the Ocular Artifact Reduction transform might appear.

```
ARTCOR POS 10 20 200 VEOG LDR "c:\\ScanData\\artcor.ldr" "" N N
```

Each element in the command line above has a specific meaning. In this case, each element defines a particular parameter of the Ocular Artifact Reduction transform. Below is the same command line with a description of the elements.




The Tcl command, ARTCOR, appears first (always in CAPS with no spaces). Following that are the arguments, or parameters. The arguments are separated by one or more spaces. Note the use of quotes "" and braces {}. These will be discussed shortly. Some commands use no parameters, such as CLOSEALL, where others may use 10 or more parameters.

## Executing BATCH Files in SCAN

We will assume you have SCAN 4.2 or newer version installed and working, and that you have a basic understanding of the operations of EDIT and ACQUIRE. You can create BATCH files for use in either program: off-line in EDIT, or on-line in ACQUIRE. Generally, most BATCH processing is performed off-line, so we will start there. Go into the EDIT program, and you will see the BATCH file and the Immediate Toolbars.



These may be on the top or bottom of the display (and may be moved like other Toolbars). If you do not see the command bars, enable them from the View option list.



Document number 2283, Revision B


*Batch* - The BATCH controls are used to execute the Tcl files you have created. Use the Browse button >> as needed to select the Tcl file you wish to run. The Batch Editor is a text editor designed especially for Tcl BATCH files (see the *Creating BATCH Files* section of the Tcl BATCH manual for more details). After retrieving a .tcl file, the button (Run) will become active. If you are using the PAUSE or REVIEW commands, the program will stop at that point until you click the button (Resume). The button (Stop) will terminate the BATCH file.

*Immediate* - The Immediate Mode line allows you to enter and execute a single line BATCH command. Enter the line and click the button (Run) to execute it. The pull-down arrow allows you to select from previous commands you have executed.

BATCH files may be created in any text editor, although we strongly recommend that you use the Tcl BATCH Editor accessed from the icon. We will demonstrate its use shortly.

## Some Tcl BATCH Command Conventions

There are some standard conventions used with Tcl. These are somewhat different from other programming languages, so you are encouraged to read them even if you are familiar with programming techniques. Some of the basic conventions will be introduced as we go along; others are summarized here. These are some of the more commonly used conventions.

**$**. The dollar sign causes Tcl to perform a variable substitution; the dollar sign and the variable following it are replaced with the value of the variable you define. For example, you can replace a frequently used path, such as, c:\\scan4.3\\Batch Examples, with a variable you create and substitute: $path.

**Braces{}**. Braces are used to indicate text strings (including empty ones), as in the following example {c:\ScanData\myfile.cnt}. Note that single slashes are used in the path. Conversely, text strings may be enclosed by quotes, as in the following example "c:\\ScanData\\myfile.cnt". Note that double slashes are used within the quotes. Either can be used in most cases. Braces, however, MUST be used in commands that contain a List variable type. Braces are also used as the beginning and ending designators for commands in a loop.

**Capitals**. Commands must be written in CAPITAL letters, such as OPENFILE.



Document number 2283, Revision B

**Comments**. This is used to place informational text in the file that has no effect on the actual BATCH program. Comments are designated by placing "#" or REM at the beginning of a line. If you want to add a comment after the Tcl command and parameters, on the same line, leave at least one space, and then type ";#", followed by your comments. For example:

```
OPENFILE  {c:\ScanData\test.eeg}     ;# open the file to be processed
```

**Double Slashes and Quotes**. Double slashes must be used when defining paths, as in the example "c:\\ScanData\\myfile.cnt", *when the path is enclosed with quotes* (see also, Single Slashes and Braces). Exceptions are the CREATESORT and DELETESORT commands, which use no quotes or braces.

**Parameters**. Following the command in a line in the Tcl file are the values for the parameters. Each command has its own parameters, and a value must be entered into each one, even if that value is a 0 or "". Spaces must separate the parameters.

**Quotes" "**. Quotes are used to indicate text strings (including empty ones), as in the following example "c:\\ScanData\\myfile.cnt". Note that double slashes are used in the path. Conversely, text strings may be enclosed by braces, as in the following example: {c:\ScanData\myfile.cnt}. Note that single slashes are used within the braces.

**Semicolons/Multiple Commands per Line**. You may have more than one command on a line. Leave at least one space at the end of one command, then place a semicolon, and write the next command.

**Single Slashes and Braces**. Single slashes must be used when defining paths, as in the example {c:\ScanData\myfile.cnt}, *when the path is enclosed with braces* (see also, Double Slashes and Quotes).

You need to understand that the arguments, or parameters, come in several forms, referred to as **Variable Types**. There are several different variable types:

**Boolean**. The Boolean variable type will recognize equivalent entries. For example, you may enter Off, No, n, 0, or False for a parameter, and they will all be interpreted the same way. Similarly, On, Yes, y, 1 or True may be used in the parameter field interchangeably. The entries are not case sensitive (NO = no = No, etc.). Any time you see a parameter that is Boolean, you will need to enter one of the above terms.

**defined value**. Defined values are used where there are several options from which to choose, such as, SUM, MEAN, and AREA. Type in the entire word, or as much of the word as is necessary to insure that it will be distinguished from the other possible responses. For example, you could type S, M or A in the above example, since each letter is unique. In the case of BANDPASS and BANDSTOP, you would need to enter either BANDP or BANDS to insure a unique interpretation. It is recommended that you enter enough of the word to make it easy to



Document number 2283, Revision B

recognize in the command line. (It might also avoid confusion in the future if more options are added to the particular parameter).

**double**. Double is, or can be, a very large signed floating point number (5000, -200, 0.5), and may have a decimal point.

**float**. Float is a signed integer larger than an int, but smaller than a double (0.5, 10), and may have a decimal point.

**int**. Int is a signed integer, like 12, -3, and 0, with no decimal points (whole numbers).

**list**. A List is a series of elements, separated by spaces, and enclosed by braces. For example, a list might be {Sterling Virginia El Paso Texas}. All five words are treated as single elements. The elements can be combined by using pairs of braces within the outer pair: {{Sterling Virginia} {El Paso Texas}}. These are treated as two elements. The same result can be accomplished with quotes inside the outer braces: {"Sterling Virginia" "El Paso Texas"}.

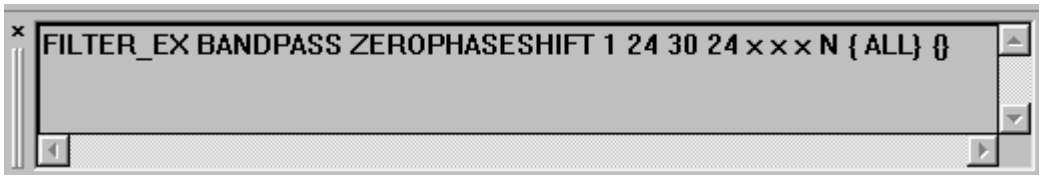
**string**. String is a text string of variable length, such as a file name, a path and a file name, or sorting criteria {1,2, 3-6}. Strings are enclosed by quotes or braces. Empty strings, "" or {}, must be used as place holders if they are not used in the command (with some exceptions, as noted).

Additional conventions will be presented as they are used in the sample BATCH files we will create.

## Auto-write Feature

Beginning with SCAN version 4.3, we have added an "auto-write feature". Whenever you execute a transform in point-and-click (P&C) mode, the Tcl BATCH command will be created automatically, and displayed in the History window. Most commands also appear on the Immediate command line. From either -place you can copy the line into the Tcl BATCH editor in the batch file you are creating.

For example, open an AVG file in EDIT, and apply the Filter transform. After the file has been filtered, you will see the Tcl BATCH command in the History field.




You can use the command by itself with the next data file in the Immediate line (retrieve the new file and click the  button), or, copy and paste it into the BATCH editor.



Document number 2283, Revision B

# Lessons

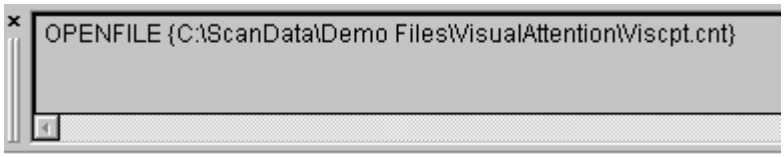
The lessons below are meant to be performed in sequential order. Some of the lessons use commands that were introduced in previous lessons, and some use data files created in previous lessons. To start, you should open EDIT, make sure the BATCH and Immediate Toolbars are showing, and open the Tcl BATCH Editor.

## **Lesson 1**. Creating a Basic BATCH File - Part 1

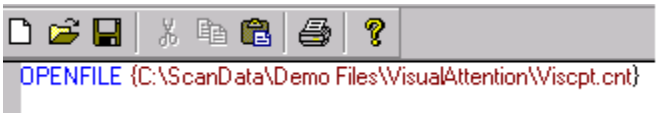
Commands introduced: **OPENFILE, CLOSEALL, REM, #, PAUSE, GETINPUTFILE**

*Purpose*. In this lesson, we will demonstrate how to open and close files, how to insert comments into the BATCH file, and how to pause the BATCH file. We will use the viscpt.cnt file (c:\ScanData\Demo Files\Visual Attention folder).

1. First, we'll just retrieve the file. In EDIT, go to the indicated folder and select the viscpt.cnt file. If you do not have the History window open, you can display it by selecting the icon.




Drag the mouse along the line to highlight it, then click the right mouse button to access the Copy command. Go into the BATCH Editor, and Paste the line into the editor.




The OPENFILE command uses a single "string" parameter. Strings can use either braces or quotes. If you use quotes, you must use double slashes in the path. The same command could be written as:

```
OPENFILE "c:\\ScanData\\Demo Files\\Visual Attention\\viscpt.cnt"
```

With most of the string parameters, you can also use empty quotes (""). This will produce the Select Data File utility display, allowing you to search for and select the file you want to open.

```
OPENFILE  ""
```

For now, just use the first variation.

2. Save the file by clicking the icon. Call it sample batch 1.tcl.

3. In EDIT, if you still have the viscpt.cnt file open, close it. Then use the Browse button to retrieve the BATCH file you just created.


Click the button to execute the BATCH file. You should see the CNT file open.

4. For a useful hint, now type "CLOSEALL" in the Immediate command line, and click the Do button to close the file.


You can store up to 10 commands in the Immediate pull-down list. This lets you use common single-line commands without having to write separate BATCH files for them. Another example might be "SCALE -10 10", which can be used to scale any retrieved data files with a Do button click.

5. Now go back to the Editor, and enter the three variations for the OPENFILE command shown above. Type REM at the beginning of the first one, and # at the beginning of the second one, leaving at least one space before the OPENFILE command. These are two ways to indicate "Remarks" in the BATCH file. The commands following REM or # will not be executed; they provide a way for you to insert comments into the BATCH files for your own purposes.

```
REM OPENFILE {c:\ScanData\Demo Files\Visual Attention\viscpt.cnt}
# OPENFILE "c:\\ScanData\\Demo Files\\Visual Attention\\viscpt.cnt"
OPENFILE  ""
```

To run the file, simply save it from the Editor using the Save option. Then click the button from EDIT. You do not need to retrieve the BATCH file again (as long as the same name appears in the Batch File field). You should see the Select Data File dialog screen appear. Select the viscpt.cnt file (from the Visual Attention folder). Close it using the Immediate command.

6. Now go back to the Editor, and remove the "#" from the second line and add it to the beginning of the third line. Then add CLOSEALL as the last line. If you run the file now, the file will simply open and then close. So, let's add a PAUSE command before the CLOSEALL command. This will pause the BATCH file until you click the button (Resume). Remember to "Save" the BATCH file. The sequence will appear as:

```
REM OPENFILE {c:\ScanData\Demo Files\Visual Attention\viscpt.cnt}
OPENFILE "c:\\ScanData\\Demo Files\\Visual Attention\\viscpt.cnt"
# OPENFILE  ""
```



Document number 2283, Revision B

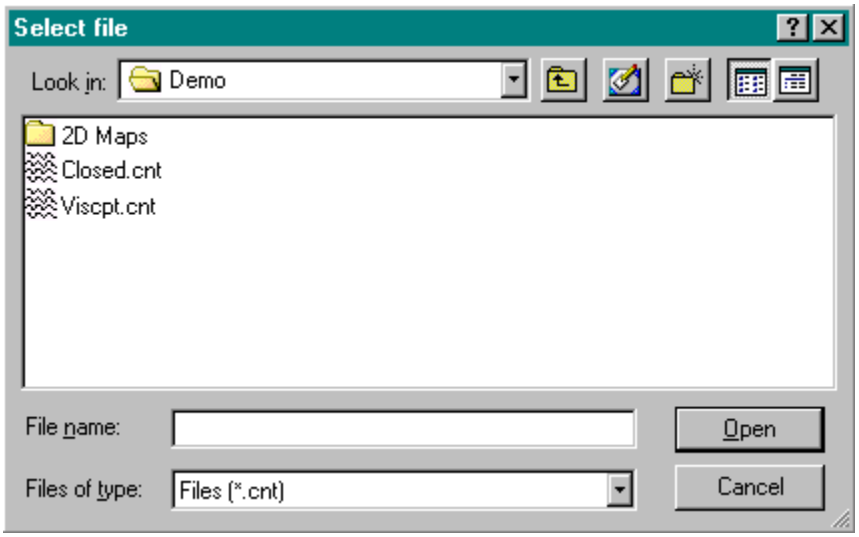
```
PAUSE
CLOSEALL
```

You can delete the first and third lines if you wish; we will not use them in the future. Execute the BATCH file, clicking the Resume button to close the file. Note that while the file is open, you have access to all of the manual controls for scaling, navigating through the file, and so forth.

Incidentally, you can also use the GETINPUTFILE command to open a file using a customized Open File dialog box. The parameters are as follows:

| | | |
|---|---|---|
| 1 | string | String title |
| 2(optional) | string | Extension |
| 3(optional) | string | Initial Path |

*Example*. GETINPUTFILE "Select file" "cnt" "c:\\ScanData\\Demo Files\\". The Select file screen seen below is opened. The first argument (required) is the title of the display. The second argument (optional) will set the extension in the "Files of type" field. The third argument (optional) defines the default path - the folder that is displayed when the window opens. You may then select the file and proceed with the rest of the BATCH commands. The command is particularly useful when you want to open a number of files with the same extension (same file type) in the same folder.




If desired, add:



Document number 2283, Revision B

```
GETINPUTFILE "Select file" "cnt" "c:\\ScanData\\Demo Files\\Visual Attention\\"
```

in place of the OPENFILE line in your BATCH file. Save the BATCH file, and then execute it. Select the file manually, and then click the  button to close it.

## **Lesson 2**. Creating a Basic BATCH File - Part 2

Commands introduced: **VOLTAGETHRESHOLD, SAVEAS, SET, ARTCOR, ENABLEOVERWRITEPROMPT**

*Purpose*. In this lesson, we will add commands to the BATCH file created in Lesson 1 to perform some artifact reduction operations, some variable substitutions, and to disable the overwrite prompt.

1. The viscpt.cnt file has some artifact laden sections in it. You could go through the file manually and reject them (as described in the EDIT tutorial), or you could do it automatically in BATCH. One way is to use the REJECTBLOCK command, assuming you know the start and stop offsets points for each block to be rejected. An easier way is to use the VOLTAGETHRESHOLD command. P3 is the bad electrode causing the artifact, so it is easy to reject blocks following the artifact when it occurs in that channel. The channels return to normal after about 8 seconds, and the voltage of the artifact at P3 is well over 500uVs when it appears. These parameters are included with the VOLTAGETHRESHOLD command.

The VOLTAGETHRESHOLD parameters appear as follows in the body of the Tcl BATCH Manual, or in the Quick Reference Guide.

| | | |
|---|---|---|
| 1 | defined value | Operation type (INSERTEVENTS, REJECTSEGMENTS) |
| 2 | string | Trigger Channel |
| 3 | defined value | Threshold Type (GREATERTHAN, LESSTHAN, ABSOLUTEVALUE) |
| 4 | int | Stim code (ignored if operation is not INSERTEVENTS) |
| 5 | double | Refractory period |
| 6 | double | Threshold |

The command in our example would then read:

```
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
```

This will reject the 8 seconds following any data point greater than 500uVs from the P3 channel. Note that "0" is entered for the Stim code (parameter 4), even though the parameter is ignored. Enter the VOLTAGETHRESHOLD command line in the BATCH file from Lesson 1 just above the PAUSE line.



Document number 2283, Revision B

2. It is necessary to add a SAVEAS command to save the modifications we are making. The command uses a single string to designate the output file name. *As a general rule, you should always retrieve the file after you save it*, if you want to do further operations on the modified file. We therefore added the OPENFILE command. The sequence should appear as follows.

```
OPENFILE "c:\\ScanData\\Demo Files\\Visual Attention\\viscpt.cnt"
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
SAVEAS "c:\\ScanData\\Demo Files\\Visual Attention\\viscpt-REJ.cnt"
OPENFILE "c:\\ScanData\\Demo Files\\Visual Attention\\viscpt-REJ.cnt"
PAUSE
CLOSEALL
```

Save the file in the Editor, and execute it from EDIT. Step through the file to verify that the proper blocks were rejected, then click the  button to complete the BATCH file.

3. You may have noticed the tediousness of inputting the same path information so often. Variable substitutions are easily accomplished in Tcl. Substitutions can be for many reasons; in this example they avoid having to write out the path every time. This is described more thoroughly in the BATCH manual. Briefly stated, use $ and a variable name as a substitute for the text to be replaced. Use the set command to define the variable name and to indicate what text should be replaced (set is a Tcl command, and does not need to be capitalized). The set command is used as follows:

```
set path "c:\\ScanData\\Demo Files\\Visual Attention"
```

In this example, the new variable "path" will contain the information within the quotes. It is used in the BATCH command line with a $ sign: $path. The example below sets the variable "path" and then inserts it in place of the full path information. Make these changes manually in your BATCH file, or Copy them from this file and Paste them into the Editor. Run the file if you wish, overwriting the existing files when prompted.

```
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
SAVEAS "$path\\viscpt-REJ.cnt"
OPENFILE "$path\\viscpt-REJ.cnt"
PAUSE
CLOSEALL
```

4. Now that we have removed the bad blocks, the next step might be to remove the VEOG artifact. This step is added to the existing BATCH file using the ARTCOR command. The parameters from the BATCH manual are:



Document number 2283, Revision B

| | | |
|---|---|---|
| 1 | defined value | Trigger direction (POSITIVE, NEGATIVE) |
| 2 | double | Threshold percentage |
| 3 | int | Minimum sweeps |
| 4 | double | Sweep duration |
| 5 | string | Blink channel |
| 6 | defined value | Output switch (LDR, LDR+CNT) |
| 7 | string | LDR file name |
| 8 | string | Output CNT file (if param 6 is LDR+CNT) |
| 9 | Boolean | Review maxima (CNT files only) |
| 10 | Boolean | Review blinks (CNT files only) |

The command in this example is:

ARTCOR POS 10 30 400 "VEOG" LDR+ "$path\\viscptLDR.ldr" "$path\\viscptVEOG.cnt" N N. The trigger direction is positive, the threshold percent is 10, there must be at least 30 sweeps, the sweeps duration is 400ms, and VEOG is the artifact channel. We are saving the CNT and LDR files, using the file names shown. Normally it is recommended that you review the maxima and individual blinks manually. To save time, we have declined these options (parameters 9 and 10 are "N").

Add the ARTCOR command above the PAUSE command. In that position, it will use the CNT file that has the bad blocks rejected. As above, it is necessary to retrieve the corrected data file (it is saved automatically in the ARTCOR command). Therefore, we need another OPENFILE command after the ARTCOR command. The new BATCH file is as follows:

```
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
SAVEAS "$path\\viscpt-REJ.cnt"
OPENFILE "$path\\viscpt-REJ.cnt"
ARTCOR POS 10 30 400 "VEOG" LDR+CNT "$path\\viscptLDR.ldr"
"$path\\viscptVEOG.cnt" N N
OPENFILE "$path\\viscptVEOG.cnt"
PAUSE
CLOSEALL
```

*Note: if you Copy the lines and Paste them into the Editor, you will likely need to make the ARTCOR command a single line (the return keystroke is needed to conclude the command line, and this is not contained in the text you Copy). Otherwise, you will get an error message saying there is an incorrect number of parameters on the ARTCOR line.*

Save the BATCH file and run it from EDIT. Click "Yes" when the Overwrite command appears. The file you see at the end is the VEOG corrected file. Click Resume  to close the files.

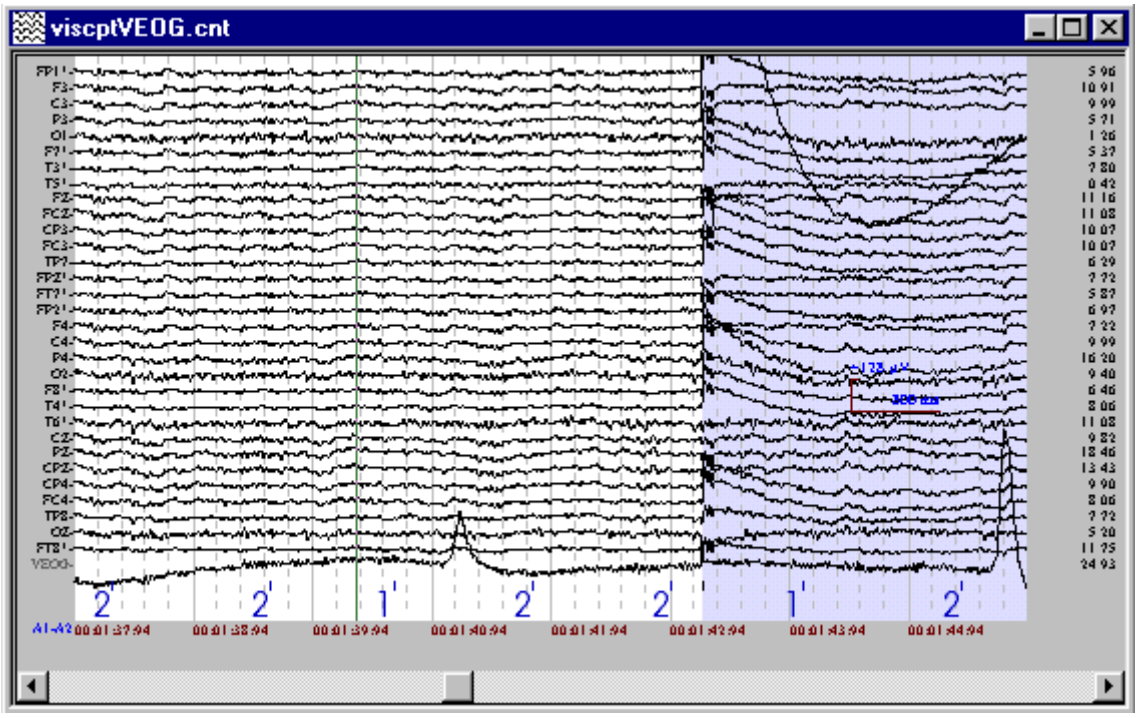


Document number 2283, Revision B

5. Notice that you have to overwrite the existing file manually. This can be a problem if you have a long, time-consuming, BATCH file that you want to run completely in your absence. You can override the prompt using the ENABLEOVERWRITEPROMPT command. If you disable the overwrite message display, the BATCH file will overwrite the existing file automatically. To do this, add ENABLEOVERWRITEPROMPT N to the beginning of the BATCH file. The complete BATCH file is shown below.

```
ENABLEOVERWRITEPROMPT N
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
SAVEAS "$path\\viscpt-REJ.cnt"
OPENFILE "$path\\viscpt-REJ.cnt"
ARTCOR POS 10 30 400 "VEOG" LDR+CNT "$path\\viscptLDR.ldr"
"$path\\viscptVEOG.cnt" N N
OPENFILE "$path\\viscptVEOG.cnt"
PAUSE
CLOSEALL
```

Now Save the BATCH file and execute it from EDIT. At the end you should see the final CNT file. Looking through it, you should see the rejected blocks, as well as the absence of blink artifact in the EEG channels.






Document number 2283, Revision B

## **Lesson 3**. Creating a Basic BATCH File - Part 3

Commands introduced: **EPOCH, SETCHANATTRIBUTE, ARTREJ, CREATESORT, AVERAGE, SELECTFILE, GETCOMP, ZOOMIN**

Now that we have removed the bad blocks and blinks, the next step might be to Epoch the file and create some average files. In the process we will introduce the creation of "Sorts".

1. First, we will add the EPOCH command to create epochs from the artifact-corrected CNT file. The parameters for the EPOCH command are:

| | | |
|---|---|---|
| 1 | defined value | Trigger mode (PORT_INTERNAL, NOTRIGGER, EVENTFILE) |
| 2 | string | Event file name (ignored unless #1 is EVENTFILE) |
| 3 | double | Start latency |
| 4 | double | Stop latency |
| 5 | Boolean | Response locked |
| 6 | Boolean | Reject epochs that overlap rejected blocks |
| 7 | Boolean | Include stimulus events |
| 8 | Boolean | Include keyboard events |
| 9 | Boolean | Include response pad events |
| 10 | string | Sort name |
| 11 | string | Output file |

In this case, we will do "Port Internal" epoching (using the stimulus triggers), with Start and Stop times of -100 and 1000ms, and without any of the remaining options (see the BATCH and EDIT manuals for complete details, if desired). The command is entered into the BATCH file we are creating, with an OPENFILE command afterward to see the new file.

```
ENABLEOVERWRITEPROMPT N
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
SAVEAS "$path\\viscpt-REJ.cnt"
OPENFILE "$path\\viscpt-REJ.cnt"
ARTCOR POS 10 30 400 "VEOG" LDR+CNT "$path\\viscptLDR.ldr"
"$path\\viscptVEOG.cnt" N N
OPENFILE "$path\\viscptVEOG.cnt"
EPOCH PORT "" -100 1000 N N Y N N "" "$path\\viscpt-corr.eeg"
OPENFILE "$path\\viscpt-corr.eeg"
PAUSE
CLOSEALL
```

Notice in the EPOCH command that we used empty quotes ("") for parameters 2 and 10. Even though we were not using an event file or a "sort", these fields still need to be filled with

empty quotes. Execute the file, if desired.

2. Now that we have the epoched file (with individual sweeps) you may have noticed that there are some sweeps that still contain VEOG artifact. We will use the ARTREJ command to delete any sweep that has voltages from the frontal channels in excess of +/- 75uVs.

First, however, the viscpt.cnt file has a number of channels set as artifact rejection channels. We could leave them as they are, but, for illustration purposes, let's set only FP1, FP2, and FPZ as artifact rejection electrodes. We will first set NONE of the channels to be artifact rejection channels, and then set the three that we want. The SETCHANATTRIBUTE command is used; the parameters are as follows.

| | | |
|---|---|---|
| 1 | list | Channel label(s) or "All" |
| 2 | defined value | Attribute Type (-Artifact, -Fsp, -Hide, -Skip, -Bad, -AutoAdd, -AutoAddLast) |
| 3 | Boolean | Set attribute |

In our example, the line should be:

```
SETCHANATTRIBUTE {all} -Artifact N
```

These parameters remove the artifact rejection flag from all channels. We then add:

```
SETCHANATTRIBUTE {"FP1" "FP2" "FPZ"} -Artifact Y
```

These parameters set the three channels as artifact rejection channels. Add the two lines after the last OPENFILE command and before the PAUSE command.

```
.....
OPENFILE "$path\\viscpt-corr.eeg"
SETCHANATTRIBUTE {all} -Artifact N
SETCHANATTRIBUTE {"FP1" "FP2" "FPZ"} -Artifact Y
PAUSE
.....
```

Having set the artifact channels, we will then add the ARTREJ command. The parameters for the ARTREJ command are:

| | | |
|---|---|---|
| 1 | defined value | Operation type (CRITERIA, REJECTALL, ACCEPTALL) |
| 2 | Boolean | Use entire interval |
| 3 | double | Start reject interval (ignored if param 2 is YES) |
| 4 | double | Stop reject interval (ignored if param 2 is YES) |
| 5 | Boolean | Recompute |
| 6 | double | Minimum amplitude |
| 7 | double | Maximum amplitude |



Document number 2283, Revision B

| | | |
|---|---|---|
| 8 | Boolean | Exclude bad channels |
| 9 | Boolean | Exclude skipped channels |

The command appears as follows:

```
ARTREJ CRIT Y 0 0 Y -75 75 Y Y
```

We are rejecting sweeps based on Criteria, using the entire interval, where we re-compute using +/- 75 uVs, with any Bad and Skip channels excluded. Notice that even though parameters 3 and 4 are ignored, we still used 0 as a place holder. Add the command immediately above the PAUSE command. We also added the SAVEAS and a new OPENFILE commands. (The final BATCH file is provided at the end of this lesson if you do not wish to Copy and Paste every line).

```
ENABLEOVERWRITEPROMPT N
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
SAVEAS "$path\\viscpt-REJ.cnt"
OPENFILE "$path\\viscpt-REJ.cnt"
ARTCOR POS 10 30 400 "VEOG" LDR+CNT "$path\\viscptLDR.ldr"
"$path\\viscptVEOG.cnt" N N
OPENFILE "$path\\viscptVEOG.cnt"
EPOCH PORT "" -100 1000 N N Y N N "" "$path\\viscpt-corr.eeg"
OPENFILE "$path\\viscpt-corr.eeg"
SETCHANATTRIBUTE {all} -Artifact N
SETCHANATTRIBUTE {"FP1" "FP2" "FPZ"} -Artifact Y
ARTREJ CRIT Y 0 0 Y -75 75 Y Y
SAVEAS "$path\\viscpt-artrej.eeg"
OPENFILE "$path\\viscpt-artrej.eeg"
PAUSE
CLOSEALL
```

Execute the file, if desired. Remember to click the  button to complete the file.

3. The last step in the example is to create separate average files for the Type 1 and Type 2 stimulus triggers. This requires the creation of a "sort", which means that we first need to use the CREATESORT command. Sorts are used in many different commands. Having a separate command for creating them means that the same sort can be used again in the same BATCH file, and that the number of parameters is reduced for the commands that use sorts. The CREATESORT command is a little different from other commands. The first command creates the "sort".

```
CREATESORT viscpt1
```

Subsequent lines define the parameters of the sort, using the list below. Use only those parameters that are relevant (see the BATCH and EDIT manual for more details, if desired).

| Parameter | type | defined values | default |
|---|---|---|---|
| -TrialEnabled | Boolean | | NO |
| -TrialCriteria | string | | "" |
| -TypeEnabled | Boolean | | NO |
| -TypeCriteria | string | | "" |
| -ResponseEnabled | Boolean | | NO |
| -ResponseCriteria | string | | "" |
| -LatencyEnabled | Boolean | | NO |
| -LatencyMin | double | | 0 |
| -LatencyMax | double | | 0 |
| -CorrectEnabled | Boolean | | NO |
| -CorrectCriteria | defined value | (CORRECT INCORRECT BOTH NORESPONSE) | CORRECT |
| -SortOnEnabled | Boolean | | NO |
| -SortOnCriteria | defined value | (EVEN ODD RANDOM) | EVEN |
| -MaxSweeps | int | | -1 |
| -SeedType | defined value | (CLOCK USERDEFINED) | CLOCK |
| -RandomSeed | int | | 0 |

In our example, we want to perform a simple sorting, where we want to average all trials where the stimulus trigger type is a 1. The commands are as follows:

```
CREATESORT viscpt1
        viscpt1 -TypeEnabled Y
        viscpt1  -TypeCriteria  "1"
```

The default values for the remaining parameters are all appropriate for our needs. We also want to create a sort for the Type 2 triggers, so we will create a second sort.

```
CREATESORT viscpt2
        viscpt2 -TypeEnabled Y
        viscpt2  -TypeCriteria  "2"
```

Add those 6 lines above the PAUSE command (or Copy the entire file presented below).

4. Now that we have the "sorts" created, we can use the AVERAGE command to average the sorted sweeps. The parameters for the command are:



Document number 2283, Revision B

| | | |
|---|---|---|
| 1 | defined value | Domain (TIME, FREQUENCY) |
| 2 | Boolean | Compute standard deviation |
| 3 | Boolean | Compute SNR |
| 4 | string | SNR file name (or "") |
| 5 | defined value | Spectral scaling method (AMPLITUDE, POWER) |
| 6 | int | Spectral window length (Taper%) |
| 7 | defined value | Spectral window type (COSINE, BLACKMAN, HANNING, HAMMING, PARZEN, WELCH) |
| 8 | string | Sort name (or "") |
| 9 | string | Output file name (or "") |

To average the sweeps with trigger type codes of 1, we will use:

```
AVERAGE T N N "" A 0 C "viscpt1" "$path\\viscpt-type1.avg"
```

We are averaging in the Time domain, and we are not computing the SD or SNR. Note the placement of the "sort" we created (viscpt1). The entries for parameters 4, 5, 6 and 7 are ignored because we are doing time-domain averaging, although they still need place holder variables. Last is the output file name. Add this line above the PAUSE command.

5. That creates the average for the sweeps with trigger type codes of 1. Now, we will use the SELECTFILE command to switch the "focus" back to the epoched file so we can AVERAGE the Type 2 sweeps. You do NOT need to use the complete path for SELECTFILE; the command is applied to files that are already open.

```
SELECTFILE "viscpt-artrej.eeg"
```

Add this line immediately above the PAUSE command. Now that the focus is back to the epoched file, we will use the AVERAGE command again, using the second sort and a different output file name. Add it above the PAUSE command.

```
AVERAGE T N N "" A 0 C "viscpt2" "$path\\viscpt-type2.avg"
```

6. We have saved, but not opened the AVG files. The display is getting somewhat cluttered with data files. We could use the CLOSEFILE command several times to close all the unwanted files, although it is easier in our example to add the final OPENFILE command for the resulting AVG files after the CLOSEALL command. Let's open one file and open the second one as a comparison file (using the GETCOMP command). Add the following lines after the CLOSEALL command.

```
OPENFILE "$path\\viscpt-type1.avg"
GETCOMP "$path\\viscpt-type2.avg"
```



Document number 2283, Revision B

7. Lastly, we will zoom in on the PZ channel to see the difference. Add the following command at the end of the BATCH file.

```
ZOOMIN "PZ"
```

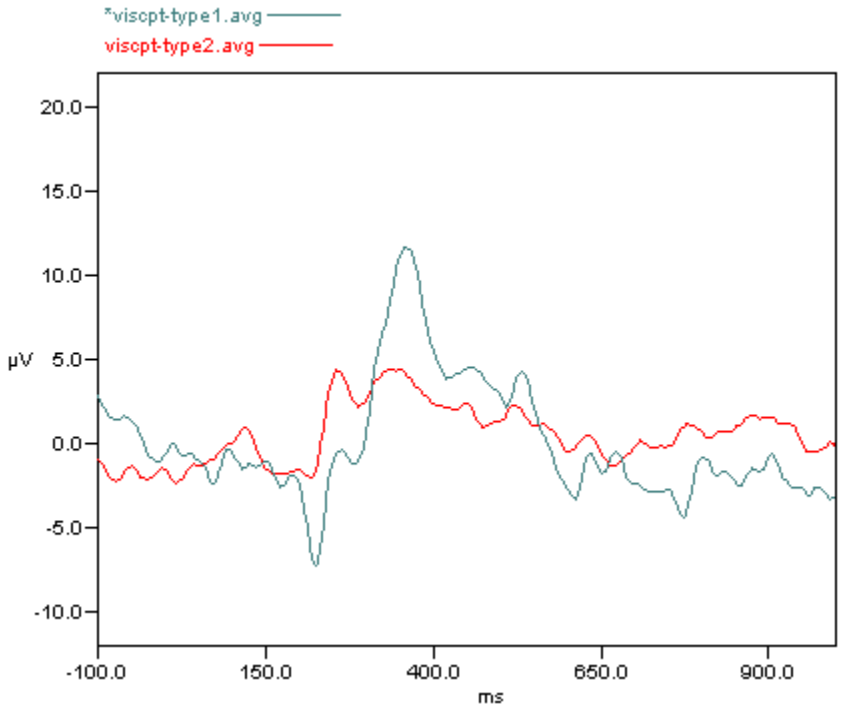
The complete BATCH file now appears as:

```
ENABLEOVERWRITEPROMPT N
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
VOLTAGETHRESHOLD REJ "P3" GREAT 0 8000 500
SAVEAS "$path\\viscpt-REJ.cnt"
OPENFILE "$path\\viscpt-REJ.cnt"
ARTCOR POS 10 30 400 "VEOG" LDR+CNT "$path\\viscptLDR.ldr"
"$path\\viscptVEOG.cnt" N N
OPENFILE "$path\\viscptVEOG.cnt"
EPOCH PORT "" -100 1000 N N Y N N "" "$path\\viscpt-corr.eeg"
OPENFILE "$path\\viscpt-corr.eeg"
SETCHANATTRIBUTE {all} -Artifact N
SETCHANATTRIBUTE {"FP1" "FP2" "FPZ"} -Artifact Y
ARTREJ CRIT Y 0 0 Y -75 75 Y Y
SAVEAS "$path\\viscpt-artrej.eeg"
OPENFILE "$path\\viscpt-artrej.eeg"
CREATESORT viscpt1
        viscpt1 -TypeEnabled Y
        viscpt1 -TypeCriteria "1"
CREATESORT viscpt2
        viscpt2 -TypeEnabled Y
        viscpt2 -TypeCriteria "2"
AVERAGE T N N "" A 0 C "viscpt1" "$path\\viscpt-type1.avg"
SELECTFILE "viscpt-artrej.eeg"
AVERAGE T N N "" A 0 C "viscpt2" "$path\\viscpt-type2.avg"
PAUSE
CLOSEALL
OPENFILE "$path\\viscpt-type1.avg"
GETCOMP "$path\\viscpt-type2.avg"
ZOOMIN "PZ"
```

Remember to click the Resume button  when the BATCH file reaches the PAUSE command (or you can "REM" the PAUSE command out). The final display you see is similar to:



Document number 2283, Revision B




The above BATCH file is more or less typical of the sequences of operations that may be performed automatically. Not only do BATCH files save time, they also insure that the same operations are applied in the same order with every data file. When you archive your data files, you should save a copy of the BATCH file you used with them. This will provide a record of how the files were processed.

Now that we have a complete BATCH file, the next step is to illustrate how BATCH files can be applied automatically to multiple data files, rather than to individual ones. The next two lessons describe how this can be performed. They will focus more on specific techniques, including file naming strategies, looping, conditional branching, etc. The information is not intended to replace what is contained in a Tcl text, but rather to give some basic examples of the applications. You should refer to the Tcl text for more complete details.



Document number 2283, Revision B

## **Lesson 4**. File Naming and Loops

Commands introduced: **FOR, SCALE, CLOSEFILE, ARRANGEWINDOWS**

*Purpose*. In the next two lessons, we will look at some file naming strategies, loops, and some conditional commands. File naming strategies are important in BATCH because it may save a lot of time and trouble if you name files in ways that allow them to be saved or retrieved easily within a loop.

1. Create a BATCH file that opens the P300.eeg demo file (in the \ScanData\Demo Files\P300s folder). Use the variable substitution option, if desired (the "set" command). (Create a new BATCH file called Sample Batch 2.tcl).

```
set path "c:\\ScanData\\Demo Files\\P300s"
OPENFILE "$path\\p300.eeg"
```

Let's say we want to create three AVG files using the p300.eeg demo file, where the files include sequential blocks of 50 sweeps, for the "oddball" stimuli only (trigger type 2). First, we will need to create "sorts", as described in Lesson 3.

```
CREATESORT P3001
        P3001 -TrialEnabled Y
        P3001 -TrialCriteria "1-50"
        P3001 -TypeEnabled Y
        P3001 -TypeCriteria "2"

CREATESORT P3002
        P3002 -TrialEnabled Y
        P3002 -TrialCriteria "51-100"
        P3002 -TypeEnabled Y
        P3002 -TypeCriteria "2"

CREATESORT P3003
        P3003 -TrialEnabled Y
        P3003 -TrialCriteria "101-150"
        P3003 -TypeEnabled Y
        P3003 -TypeCriteria "2"
```

These lines create separate sorts for the blocks of 50 sweeps where the stimulus type code is 2. The sorts are needed for the AVERAGE command below. The naming of the sorts, using sequential numbers at the end of each name, was done for a purpose, as we will see below.

2. You need to understand the "for" command from Tcl to continue. The for command takes four arguments: the first is an initialization script, the second is an expression that determines when to terminate the loop, the third is the reinitialization script (which is evaluated after each



Document number 2283, Revision B

execution of the loop before evaluating the test again), and the fourth argument is the script that forms the body of the loop. Examine the following:

```
for {set index 1} {$index < 4} {incr index} {
        body of script
}
```

The first argument sets "index" to 1. The second will terminate the loop if "index" is 4 or larger. The third argument increments "index" by 1 for each loop. The fourth argument contains the body of the script within the loop. The placement of the braces is a common and useful convention. It clearly delineates the limits of the loop.

We want to use the for command in two ways: 1) to select the sort that we want to use, and 2) to label the AVG files we create. Examine the following script:

```
for {set index 1} {$index < 4} {incr index} {
        SELECTFILE "p300.eeg"
        AVERAGE T N N "" A 0 C "P300$index"
                "$path\\p300$index.avg"
        OPENFILE "$path\\p300$index.avg"
}
```

Within the body of the script, we are using the AVERAGE command, in which we specify the sort file using "P300$index", and then name the output file as "p300$index.avg". The first time through the loop, $index is 1, so the P3001 sort is applied, and the output file is p3001.avg. That file is then opened within the loop. The next time through the loop, we need the SELECTFILE command to return the "focus" to the p300.eeg file for averaging (it does not hurt to have it there in the first loop). The $index value is now 2, the P3002 sort is used, the p3002.avg is created, and so on for the third loop.

3. Now, let's make the body of the script a little more interesting and useful. Let's scale each file as it is created, using SCALE -25 25. Then we will zoom in to the CPZ channel in each average as it is created, using ZOOMIN "CPZ". The complete loop now appears as:

```
for {set index 1} {$index < 4} {incr index} {
        SELECTFILE "p300.eeg"
        AVERAGE T N N "" A 0 C "P300$index"
                "$path\\p300$index.avg"
        OPENFILE "$path\\p300$index.avg"
        SCALE -25 25
        ZOOMIN "CPZ"
}
```

4. Let's make a few more additions to the BATCH file we are creating (complete file shown below). Add ENABLEOVERWRITEPROMPT N to the beginning of the file. Add



Document number 2283, Revision B

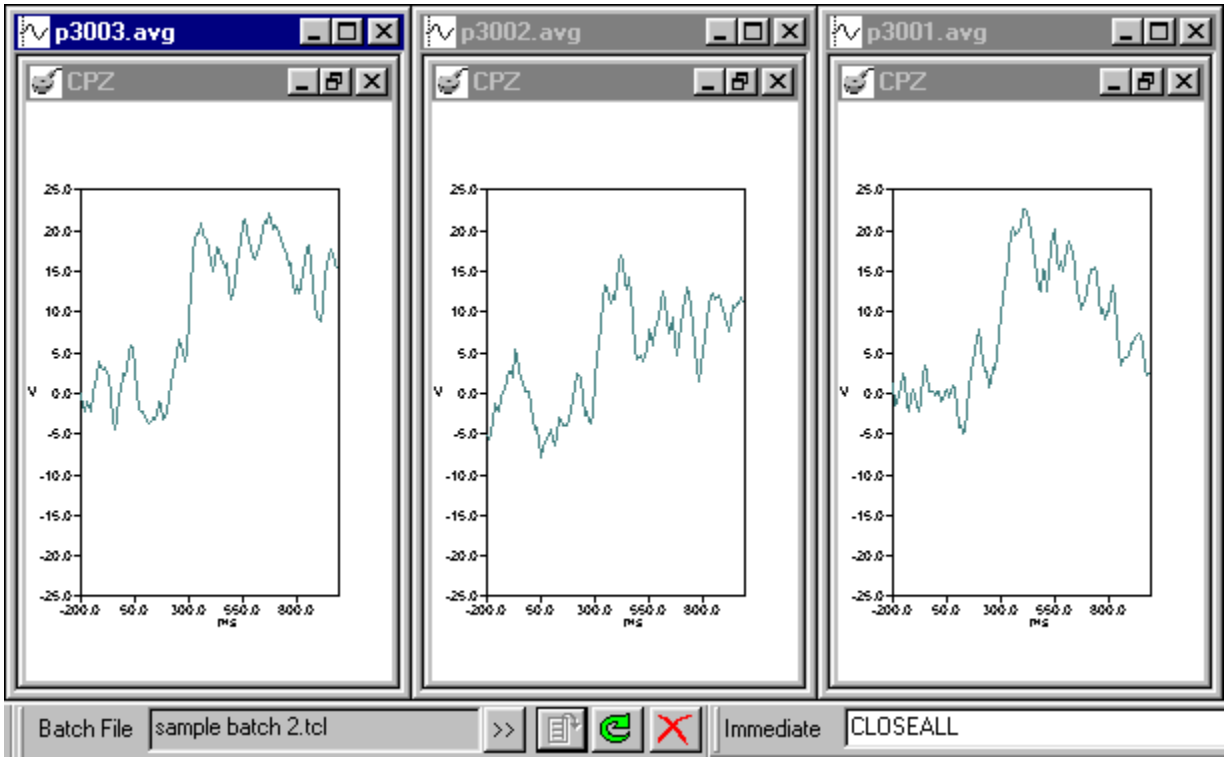
CLOSEFILE "p300.eeg" near the end. Lastly, we will add the ARRANGEWINDOWS TILEV command to arrange the three AVG files in a vertical tile arrangement. Add PAUSE and CLOSEALL at the end, as we did in the previous BATCH file. The complete BATCH file appears as follows.

```
ENABLEOVERWRITEPROMPT N
set path "c:\\ScanData\\Demo Files\\P300s"
OPENFILE "$path\\p300.eeg"

CREATESORT P3001
        P3001 -TrialEnabled Y
        P3001 -TrialCriteria "1-50"
        P3001 -TypeEnabled Y
        P3001 -TypeCriteria "2"

CREATESORT P3002
        P3002 -TrialEnabled Y
        P3002 -TrialCriteria "51-100"
        P3002 -TypeEnabled Y
        P3002 -TypeCriteria "2"

CREATESORT P3003
        P3003 -TrialEnabled Y
        P3003 -TrialCriteria "101-150"
        P3003 -TypeEnabled Y
        P3003 -TypeCriteria "2"

        for {set index 1} {$index < 4} {incr index} {
                SELECTFILE "p300.eeg"
                AVERAGE T N N "" A 0 C "P300$index"
                        "$path\\p300$index.avg"
                OPENFILE "$path\\p300$index.avg"
                SCALE -25 25
                ZOOMIN "CPZ"
        }

CLOSEFILE "p300.eeg"
ARRANGEWINDOWS TILEV
PAUSE
CLOSEALL
```

Save the file and be sure to select "sample batch 2.tcl" before you run the file in EDIT. The final display should appear similar to the following (click the  button to close all the files).



Document number 2283, Revision B




You can see the importance of using a file naming strategy when using BATCH files. In this example, we not only created file names within BATCH and then retrieved them, we also were able to use the sorts within the loop because of the naming strategy we used for them. There are some other file naming strategies that you may find useful (please see the BATCH manual or the Tcl text).



Document number 2283, Revision B

## **Lesson 5**. Loops and Conditional Commands

Commands introduced: **FOREACH, INSTRUCT, IF, ELSEIF**

The example in Lesson 4 illustrated a combination of variable substitutions within a "for" loop. The for command provides one way to do looping. Another is to use the "foreach" Tcl command. This command is useful in cases where you have files that have no naming strategy, that is, nothing in common across files that would allow you to use variable substitutions. The following is an example of the foreach command.

1. First, let's make some files with unrelated names. Create a simple BATCH file as follows (call it Sample Batch 3.tcl). For fun, we will do it within a for loop, and copy the AVG files we created in Lesson 4 using different names. Examine the following loop.

```
set path "c:\\ScanData\\Demo Files\\P300s"
for {set index 1} {$index < 4} {incr index} {
        OPENFILE "$path\\p300$index.avg"
        SAVEAS ""
}
CLOSEALL
```

The for loop first opens each of the three files we created in the previous lesson. The SAVEAS "" command opens the standard Save File utility. As each file opens, you can save a copy using any file name you want. We used File A.avg, File B.avg and File C.avg as the new file names.

2. We will now use the foreach command to open the new data files. The foreach command iterates over all of the elements in a list. It contains three arguments: the first is the name of a variable, the second is a list, and the third is the body of the script file. Examine the lines below:

```
set filelist {{File A.avg} {File B.avg} {File C.avg}}
foreach datafile $filelist {
        set file $path$datafile
        OPENFILE "$file"
}
CLOSEALL
```

The set command creates a variable called "filelist" that contains the three elements (the data files). The foreach command creates a variable called "datafile", then, for each element in "filelist", i.e., for each data file, the body of the script is executed. In the body of the script, the set line creates a variable called "file" that consists of the "path", defined on the first line of the BATCH file, and "datafile", which is each file name in the list. This line also demonstrates how you can use two variable substitutions in the single set command; it is a convenience in this instance. The OPENFILE line then simply calls each of the three data files. You can then add whatever additional operations or transforms you wish to apply within the body of the script.

3. Lastly, this is a good opportunity to illustrate one use of the INSTRUCT command. This allows you, among other things, to display messages during the execution of the BATCH file. For example, let's say that you wrote the file above and that someone else in the lab will be running it. You might want to include instructions/reminders in the file. The INSTRUCT command can be used for this, as follows:

```
INSTRUCT "Enter a new file name for each data file when the Save As
display appears" OK.
```

You can enter any text, and then specify which buttons will appear on the display. In this case, only the "OK" button will appear. Place the command before the for command. The BATCH file to this point appears as follows:

```
set path "c:\\ScanData\\Demo Files\\P300s"
INSTRUCT "Enter a new file name for each data file when the Save As
display appears" OK

for {set index 1} {$index < 4} {incr index}  {
        OPENFILE "$path\\p300$index.avg"
        SAVEAS  ""
}
CLOSEALL

set newpath "c:\\Scan4.3\\Demo Files\\P300s\\"
set filelist {{File A.avg} {File B.avg} {File C.avg}}
foreach datafile $filelist  {
        set file $newpath$datafile
        OPENFILE  "$file"
}
ARRANGEWINDOWS TILEV
PAUSE
CLOSEALL
```

Note that we had to create the "newpath" variable to include the final "\\" in the path. Without it, "$path$datafile" would be "c:\\ScanData\\Demo Files\\P300sFile A.avg", and the "P300sFile A.avg" file does not exist. We also added the ARRANGEWINDOWS and PAUSE commands to help with the display and flow. (The final BATCH file will be presented below).

4. Lastly, we will show how to use the conditional if command. We will include it within the foreach loop. The if command (see a Tcl text for complete details) has the following structure:

**if test1 body1 ?elseif test2 body2 elseif ...? ?else bodyn?**

Tcl evaluates **test1** as an expression. If its value is nonzero, it executes **body1** as a Tcl script and returns its value. Otherwise, it evaluates **test2** as an expression; if its value is nonzero, it



Document number 2283, Revision B

executes **body2** as a script and returns its value. If no test succeeds, it executes **bodyn** as a Tcl script and returns its result. You can have any number of "tests".

Let's say we wish to perform a different set of operations on each of the data files we retrieve. If the first file is retrieved, we want to perform one set of operations; if the second file is retrieved, we want to perform a different set, and so forth. The **if** command might be set up as follows:

```
if {$datafile == "File A.avg"} {
        SCALE -5 5
        } elseif {$datafile == "File B.avg"} {
        SCALE -10 10
        } elseif {$datafile == "File C.avg"} {
        SCALE -15 15
}
```

(Note the use of the **elseif** commands). In other words, we are simply applying different scalings to each file. You could have a much more complicated set of commands. The lines should be placed within the **foreach** loop, as shown:

```
foreach datafile $filelist {
        set file $newpath$datafile
        OPENFILE "$file"
                if {$datafile == "File A.avg"} {
                        SCALE -5 5
                        } elseif {$datafile == "File B.avg"} {
                        SCALE -10 10
                        } elseif {$datafile == "File C.avg"} {
                        SCALE -15 15
                }
}
```

The use of the indents and the placement of the braces help to define the loop and conditional paths. You may use any number of spaces, tabs, and lines to make the script easier to follow. The final BATCH file appears as follows:

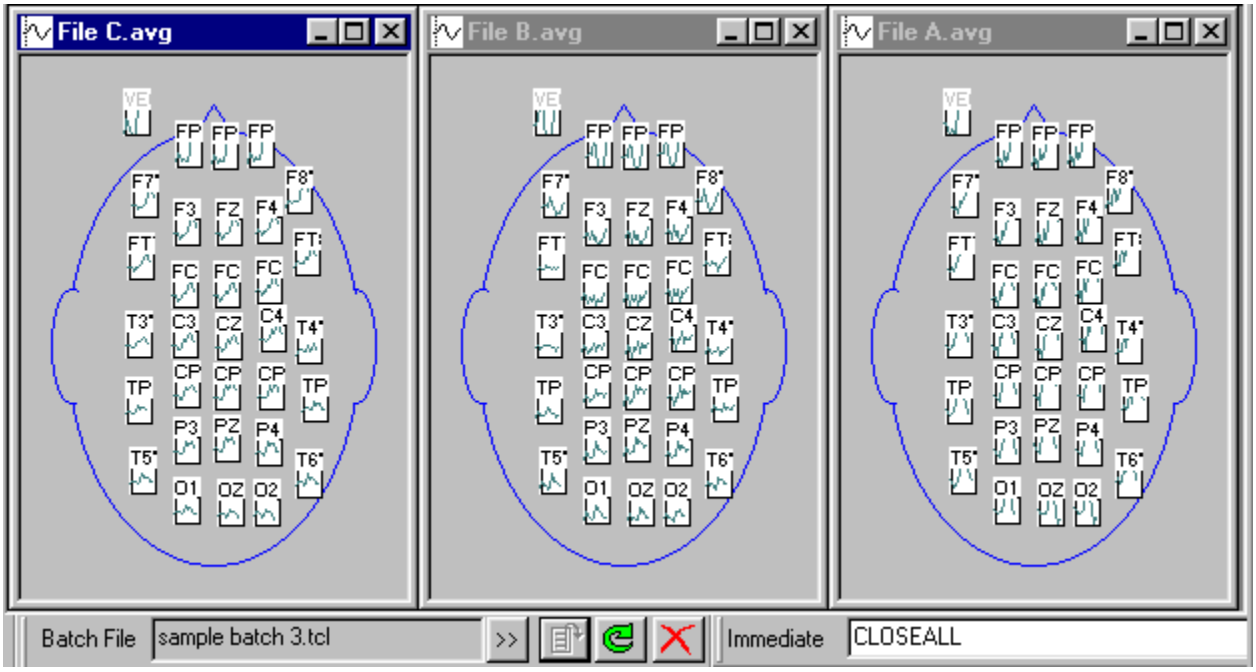
```
set path "c:\\ScanData\\Demo Files\\P300s"
INSTRUCT "Enter a new file name for each data file when the Save As display
appears" OK

for {set index 1} {$index < 4} {incr index} {
        OPENFILE "$path\\p300$index.avg"
        SAVEAS ""
}
CLOSEALL
```



Document number 2283, Revision B

```
set newpath "c:\\ScanData\\Demo Files\\P300s\\"
set filelist {{File A.avg} {File B.avg} {File C.avg}}
foreach datafile $filelist {
        set file $newpath$datafile
        OPENFILE "$file"
                if {$datafile == "File A.avg"} {
                        SCALE -5 5
                        } elseif {$datafile == "File B.avg"} {
                        SCALE -10 10
                        } elseif {$datafile == "File C.avg"} {
                        SCALE -15 15
                }
}
ARRANGEWINDOWS TILEV
PAUSE
CLOSEALL
```

Be sure to save the file as "sample batch 3.tcl", and select that file from EDIT. The final display appears as follows; click the  button to close the files and end the BATCH file.





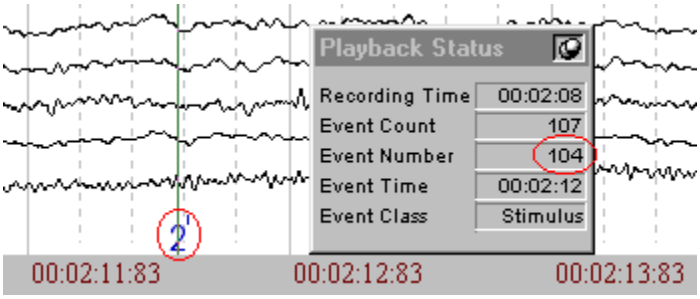

Document number 2283, Revision B

## **Lesson 6**. Modifying the Event Table

Commands introduced: **GETEVENTINFO, SETEVENTINFO**

*Purpose.* This lesson will provide a simple example demonstrating how to modify the event table. Beginning with SCAN 4.3, it is possible to modify information in the event table in BATCH. The event table contains all off the stimulus, response, keyboard, and other event information in CNT files. The GETEVENTINFO and SETEVENTINFO commands are used.

1. Let's say you discover that a stimulus type code was set incorrectly in the Gentask sequence file in STIM, and a type code of 2 was sent where it should have been a 1. We will use the viscpt.cnt demo file. Let's say that we discover that the 104th event should really be a 1.




2. Begin by making a copy of the CNT file. It is always a good idea to work with a copy of your data files, insuring that the originals remain unchanged. This is especially true when you are making changes to the event table. Create a new BATCH program with the following lines (name this BATCH file "sample batch 4.tcl"):

```
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
SAVEAS "$path\\viscpt-copy.cnt"
CLOSEFILE "viscpt.cnt"
OPENFILE "$path\\viscpt-copy.cnt"
```

The script makes a copy of the file, closes the original, and then opens the copy.

3. Before we look at the Tcl command parameters for GETEVENTINFO and SETEVENTINFO, you need to understand what a zero-based index is. A zero-based index is simply a numerical list that starts at zero rather than one. The epochs, or sweeps, seen in the Status boxes in EDIT begin at 1 and continue sequentially to the end of the list: 1, 2, 3, ... N. The same sweeps in a zero-based index are 0, 1, 2, ... N-1. Zero-based indices are used throughout the BATCH commands, and these are noted in the parameter descriptions. The GETEVENTINFO and SETEVENTINFO use zero-based indices. If we want to modify the 104th sweep event, that is the *103rd* event with a zero-based index.



Document number 2283, Revision B

4. First, we will retrieve the event value at the 104th sweep. You do not need to retrieve it in order to modify it; we are retrieving it only to illustrate how you can do it. The GETEVENTINFO command is used. Its parameters are a little different from most of the BATCH commands.

| | | |
|---|---|---|
| 1 | int | Event number (zero-based index) |
| 2 | defined value | Parameter (EventType, Offset, ResponseLatency, Accuracy, KeyboardCode, KeypadCode, StimulusCode) |

Depending upon which Parameter you selected for #2, the program will return differing information. The following translates the meaning of the returned codes.

| | |
|---|---|
| -EventType | Stimulus, Keypad, Rejected sweep, Accepted sweep, Keyboard, DC Correction, and Segment (Stop/Start event) |
| -Offset | Number of points since the beginning of the file. |
| -ResponseLatency | The response latency in ms. |
| -Accuracy | No response, Incorrect, Correct |
| -KeyboardCode | Function key number (2-11) |
| -KeypadCode | Number of response pad button pressed (1-4) |
| -StimulusCode | The stimulus type code number (1-255) |

As with the CREATESORT command used above, you use only the parameters that are needed with the GETEVENTINFO command. In our case, the command would be:

```
GETEVENTINFO 103 -Stim
```

The 103rd stimulus type code number will be returned.

5. If you want to see the value that is retrieved (i.e., returned), you use the INSTRUCT command in a slightly different way than that used in the lesson above. First, we use the set command to substitute the returned value for some other variable, then we display that new variable with the INSTRUCT command.

```
set stimcode [GETEVENTINFO 103 -Stim]
INSTRUCT "The trigger type code is: $stimcode"
```

The complete BATCH sequence thus far is:

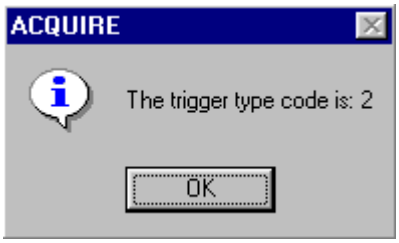
```
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
SAVEAS "$path\\viscpt-copy.cnt"
CLOSEFILE "viscpt.cnt"
OPENFILE "$path\\viscpt-copy.cnt"
```



Document number 2283, Revision B

```
set stimcode [GETEVENTINFO 103 -Stim]
INSTRUCT "The trigger type code is: $stimcode"
```

The returned valued is displayed:




6. Now we will use the SETEVENTINFO command to change the value to a 1. Its parameters and usage are similar to the GETEVENTINFO command:

| | | |
|---|---|---|
| 1 | int | Event number (zero-based index) |
| 2 | defined value | Parameter (EventType, Offset, ResponseLatency, Accuracy, KeyboardCode, KeypadCode, StimulusCode) |
| 3 | varies | Based on list below: |

| | | |
|---|---|---|
| -Offset | int | Number of points since the beginning of the file. |
| -ResponseLatency | double | The response latency in ms. |
| -Accuracy | defined value | No response, Incorrect, Correct |
| -KeyboardCode | int | Function key number (2-11) |
| -KeypadCode | int | Number of response pad button pressed (1-4) |
| -StimulusCode | int | The stimulus type code number (1-255) |

The command in our example is:

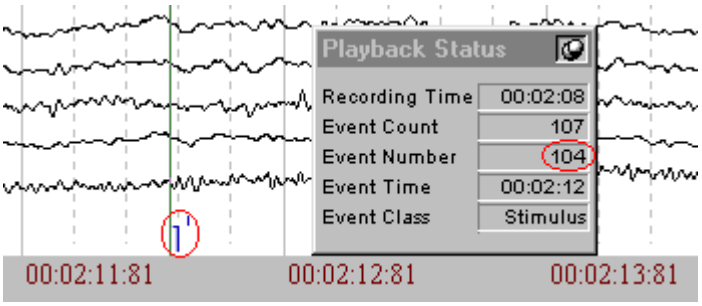
```
SETEVENTINFO 103 -Stim 1
```

This sets the 103rd stimulus event to 1. After that command, we will need to save the new CNT file, close the previous one, retrieve the modified one, and then display the new returned value to show the change. The complete BATCH file is as follows:

```
set path "c:\\ScanData\\Demo Files\\Visual Attention"
OPENFILE "$path\\viscpt.cnt"
SAVEAS "$path\\viscpt-copy.cnt"
CLOSEFILE "viscpt.cnt"
OPENFILE "$path\\viscpt-copy.cnt"

set stimcode [GETEVENTINFO 103 -Stim]
INSTRUCT "The trigger type code is: $stimcode"

SETEVENTINFO 103 -Stim 1

SAVEAS "$path\\viscpt-mod.cnt"
CLOSEFILE "viscpt-copy.cnt"
OPENFILE "$path\\viscpt-mod.cnt"

set newcode [GETEVENTINFO 103 -Stim]
INSTRUCT "The modified trigger type code is: $newcode"
```

The final returned value will appear as:


and the change can be seen in the CNT file:




7. It is also possible to change all of the values in the event file in a single BATCH file. An example is provided in the Advanced Tcl Scripts section at the end of the body of the BATCH manual ("Changing Information in the Event Table"). That BATCH file lets you input a selected value to be added to all of the events in the event table.



Document number 2283, Revision B

## **Lesson 7**. Exporting and Importing Data Files

Commands introduced: **EXPORTAVG, IMPORTAVG, READPOS, SAVEPOS, SUBTRACT**

*Purpose.* This example will demonstrate how to export and import data files from BATCH. In Lesson 4, we created the P3001.avg, P3002.avg and P3003.avg files. These can be exported in a single batch file (call it "sample batch 5.tcl"). The complete BATCH file is presented at the end of the steps.

1. The basic loop for exporting the files is similar to the one used in Lesson 4. The main difference is the EXPORTAVG command line.

```
set path "c:\\ScanData\\Demo Files\\P300s"
for {set index 1} {$index < 4} {incr index} {
        OPENFILE "$path\\p300$index.avg"
        EXPORTAVG "$path\\EXPp300$index.dat" P Y Y N N N Y Y
}
CLOSEALL
```

The parameters for the EXPORTAVG command are:

| | | |
|---|---|---|
| 1 | string | Output file name (or "") |
| 2 | defined value | Method (POINTS, ELECTRODES, BESA) |
| 3 | Boolean | Include header |
| 4 | Boolean | Include electrode labels |
| 5 | Boolean | Include X units |
| 6 | Boolean | Include Y units |
| 7 | Boolean | Include standard deviation |
| 8 | Boolean | Include Bad channels |
| 9 | Boolean | Include Skipped channels |

We are exporting the files where the rows are Points, with the header and electrode labels included, and with Bad and Skip channels included. The loop creates DAT files for each of the AVG files.

2. Now, we will import the files into EDIT. The loop is similar, except for the IMPORTAVG command. Generally speaking, you need to know details about the file including the format it is in (rows = points or electrodes), the number of channels (32), the AD rate (250), the Start time (-200), and the number of points per sweep (300). The parameters are:

| | | |
|---|---|---|
| 1 | string | File name (or "") |
| 2 | defined value | Method (POINTS, ELECTRODES) |
| 3 | integer | Number of channels |
| 4 | double | Acquisition rate |
| 5 | double | Xmin |



Document number 2283, Revision B

| | | |
|---|---|---|
| 6 | int | Number of points |
| 7 | int | Number of sweeps used to create the average |
| 8 | Boolean | Frequency domain |

The seventh parameter sets the number of accepted sweeps in the AVG file. In this case, where we are importing several files based on differing numbers of accepted sweeps for each, we are simply using "1" (we will not be needing the number of sweeps information). The command is added withing the for loop, as well as the SAVEAS line to save the files.

```
for {set index 1} {$index < 4} {incr index} {
        IMPORTAVG "$path\\EXPp300$index.dat" P 32 250 -200 300 1 N
        SAVEAS "$path\\IMPp300$index.avg"
}
CLOSEALL
```

3. You will notice that the electrode positions are lost when you import the data files. This information is not stored with the DAT files. You do not, however, need to reposition the channels manually, as long as you have one data file with the correct position information. We next retrieve one of the original AVG files, and save the electrode position information using the SAVEPOS command. This will be applied to the imported files shortly.

```
OPENFILE "$path\\p3001.avg"
SAVEPOS  "$path\\p300positions.asc"
```

4. When we apply the new positions, we will be retrieving and then re-saving the files using the same names. This means we will have to say Yes each time when asked if we want to overwrite the data files. We can disable the "overwrite" message with the following line:

```
ENABLEOVERWRITEPROMPT N
```

Note: if you run this BATCH file more than once, you may want to move this line to the top of the BATCH file so you do not have to acknowledge overwriting all of the existing files.

5. Now we will apply the position data within a loop. This is also a good opportunity to verify that the imported files are identical to the original data files. To do that we will SUBTRACT each original file from the imported data file, and save the difference files (which should be all zeros).

```
for {set index 1} {$index < 4} {incr index} {
        OPENFILE "$path\\IMPp300$index.avg"
        READPOS  "$path\\p300positions.asc"
        SAVEAS "$path\\IMPp300$index.avg"
        SUBTRACT "$path\\p300$index.avg" "$path\\diff-file$index"
}
CLOSEALL
```



Document number 2283, Revision B

6. We now retrieve the difference files, rescale them, and zoom in to one channel.

```
for {set index 1} {$index < 4} {incr index}  {
        OPENFILE "$path\\diff-file$index"
        SCALE  -1  1
        ZOOMIN "CZ"
}
```

7. Lastly, we will arrange the windows automatically, PAUSE the BATCH file to examine the difference files, and then close all the files by clicking the Resume button  .

```
ARRANGEWINDOWS TILEV
PAUSE
CLOSEALL
```

The complete BATCH file is shown below:

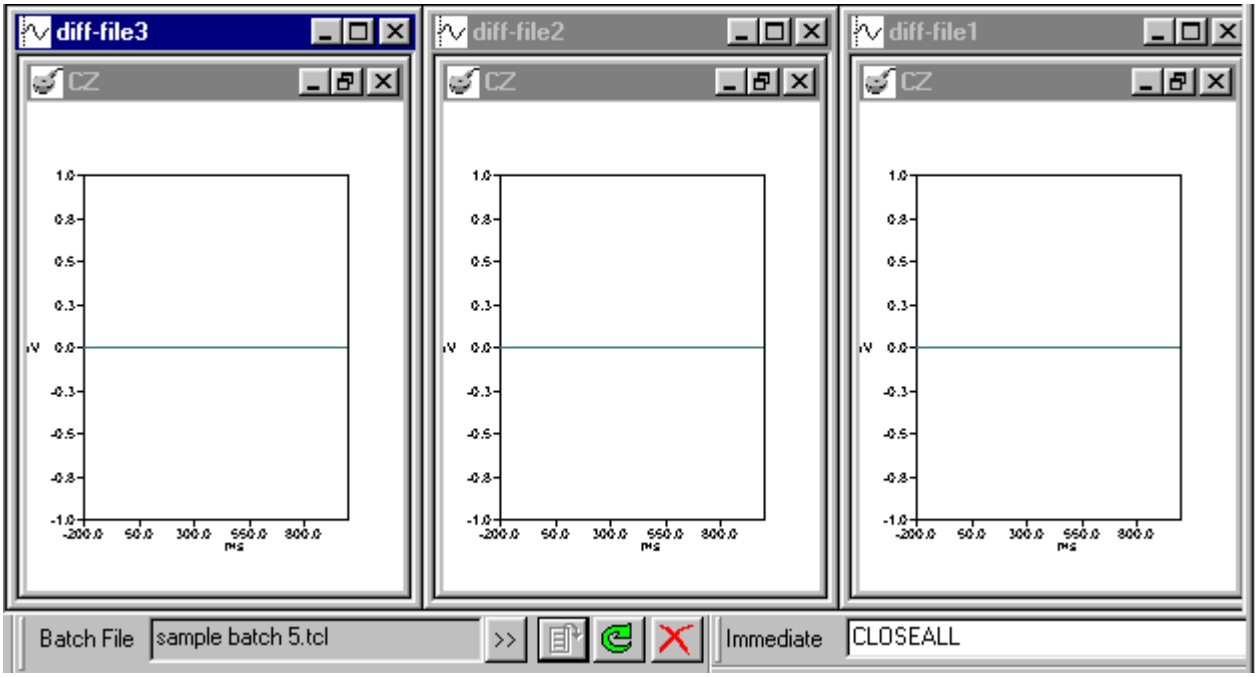
```
set path  "c:\\ScanData\\Demo Files\\P300s"
for {set index 1} {$index < 4} {incr index}  {
        OPENFILE "$path\\p300$index.avg"
        EXPORTAVG "$path\\EXPp300$index.dat"  P  Y  Y  N  N  N  Y  Y
}
CLOSEALL

for {set index 1} {$index < 4} {incr index}  {
        IMPORTAVG "$path\\EXPp300$index.dat" P 32  250  -200  300  1  N
        SAVEAS "$path\\IMPp300$index.avg"
}
CLOSEALL

OPENFILE "$path\\p3001.avg"
SAVEPOS  "$path\\p300positions.asc"

ENABLEOVERWRITEPROMPT N

for {set index 1} {$index < 4} {incr index}  {
        OPENFILE "$path\\IMPp300$index.avg"
        READPOS  "$path\\p300positions.asc"
        SAVEAS "$path\\IMPp300$index.avg"
        SUBTRACT "$path\\p300$index.avg" "$path\\diff-file$index"
}
CLOSEALL
```



Document number 2283, Revision B

```
for {set index 1} {$index < 4} {incr index}  {
        OPENFILE "$path\\diff-file$index"
        SCALE  -1  1
        ZOOMIN "CZ"
}
ARRANGEWINDOWS TILEV
PAUSE
CLOSEALL
```

The final display is shown below, and there are no differences, meaning the imported files are identical to the original files.






Document number 2283,  Revision B

## **Lesson 8**. Acquiring Data with BATCH Files

Commands introduced: **GETAST, CALIB, SUBJECT, READSUB, SAVESUB, DOIMPEDANCE, STARTACQUISITION. STARTRECORDING, STOPRECORDING, STOPACQUISITION**

*Purpose.* To demonstrate the use of BATCH files for acquisition of data. The BATCH files so far have all had to do with the analysis of collected data. It is also possible to control data acquisition in BATCH. A typical sequence might be as follows:

- Retrieve the setup file
- Perform calibration
- Enter subject information
- Perform an impedance check
- Monitor the EEG signals
- Store the EEG signals during stimulation from STIM
- Stop storage
- Stop acquisition of signals

1. Create a new BATCH file called "sample batch 6.tcl". The GETAST command is used to retrieve the setup file. You can specify the setup file in the command, or you can use empty quotes ("") to display the Open File dialog screen to select the file. We will use the QuikCap32.ast file.

```
GETAST "c:\\Program Files\\Neuroscan\\Scan4.3\\Setup
Files\\Synamp1\\QuikCap32.ast"
```

2. The CALIB command opens the initial calibration screen, allowing you to set the parameters as desired. Then perform the calibration as usual. (It is not generally necessary to perform a calibration with every subject; it is included for demonstration purposes). The command uses no parameters.

```
CALIB
```

3. The SUBJECT command is used to enter subject information. For this example, we are using a template file (template.sub) where much of the information has already been entered. We can retrieve the template file with the READSUB command, and then enter only the information unique to a particular subject. You can save the new subject information from the Subject display, or with the SAVESUB command (with the SAVESUB command, just click the OK button on the Subject screen when you are through). We will use the SAVESUB command with empty quotes to allow us to enter whatever subject name we want at the time.

```
READSUB "c:\\Program Files\\Neuroscan\\Scan4.3\\Setup
Files\\Synamp1\\template.sub"
```



Document number 2283, Revision B

```
SUBJECT
SAVESUB  ""
```

4. The DOIMPEDANCE command begins the impedance measuring process. The command has no parameters. Close the screen to continue with the BATCH file.

```
DOIMPEDANCE
```

5. Once the impedances are acceptable, the next step is usually to monitor the signals for a brief period. The STARTACQUISITION command has the same function as clicking the green arrow on the ACQUIRE toolbar. We will put a PAUSE command afterward. This suspends further commands in the BATCH file until the Resume button  is pressed, allowing you to monitor the EEG for an unspecified time span.

```
STARTACQUISITION
PAUSE
```

6. The STARTRECORDING command initiates data storage. You can specify the path and file name in the string parameter for the command, or use empty quotes ("") to display the Save As dialog box.

```
STARTRECORDING ""
PAUSE  5000
```

We placed the PAUSE 5000 command afterwards to allow a 5 second pause in the BATCH file to let the amplifiers adjust, if needed. The next step in our example is to start the STIM program.

7. We will add a reminder to start the STIM program using the INSTRUCT command. The OK and CANCEL buttons will appear on the message display.

```
INSTRUCT  "Start the STIM program now."  OKCANCEL
PAUSE
```

The PAUSE was added to suspend the BATCH file until the recording has been completed. At that point, pressing the Resume button  will go to the next line in the BATCH file.

8. The final lines are the STOPRECORDING and STOPACQUISITION commands.

```
STOPRECORDING
STOPACQUISITION
```



Document number 2283,  Revision B

These will close the data file and then close the acquisition display.

The complete BATCH file is as follows:

```
GETAST "c:\\Program Files\\Neuroscan\\Scan4.3\\Setup
Files\\Synamp1\\QuikCap32.ast"
CALIB
READSUB "c:\\Program Files\\Neuroscan\\Scan4.3\\Setup
Files\\Synamp1\\template.sub"
SUBJECT
SAVESUB  ""
DOIMPEDANCE
STARTACQUISITION
PAUSE
STARTRECORDING ""
PAUSE  5000
INSTRUCT "Start the STIM program now." OKCANCEL
PAUSE
STOPRECORDING
STOPACQUISITION
```

This concludes the BATCH tutorial lessons. We have only scratched the surface of the possible applications using Tcl and SCAN BATCH commands. You are strongly encouraged to read the introductory sections of the BATCH manual, and to examine the various sample BATCH files in the manual. For more complex applications, you will certainly need to get a Tcl text. You should also periodically visit our BATCH web site: http://www.neuro.com/neuroscan/batch.htm.



Document number 2283, Revision B