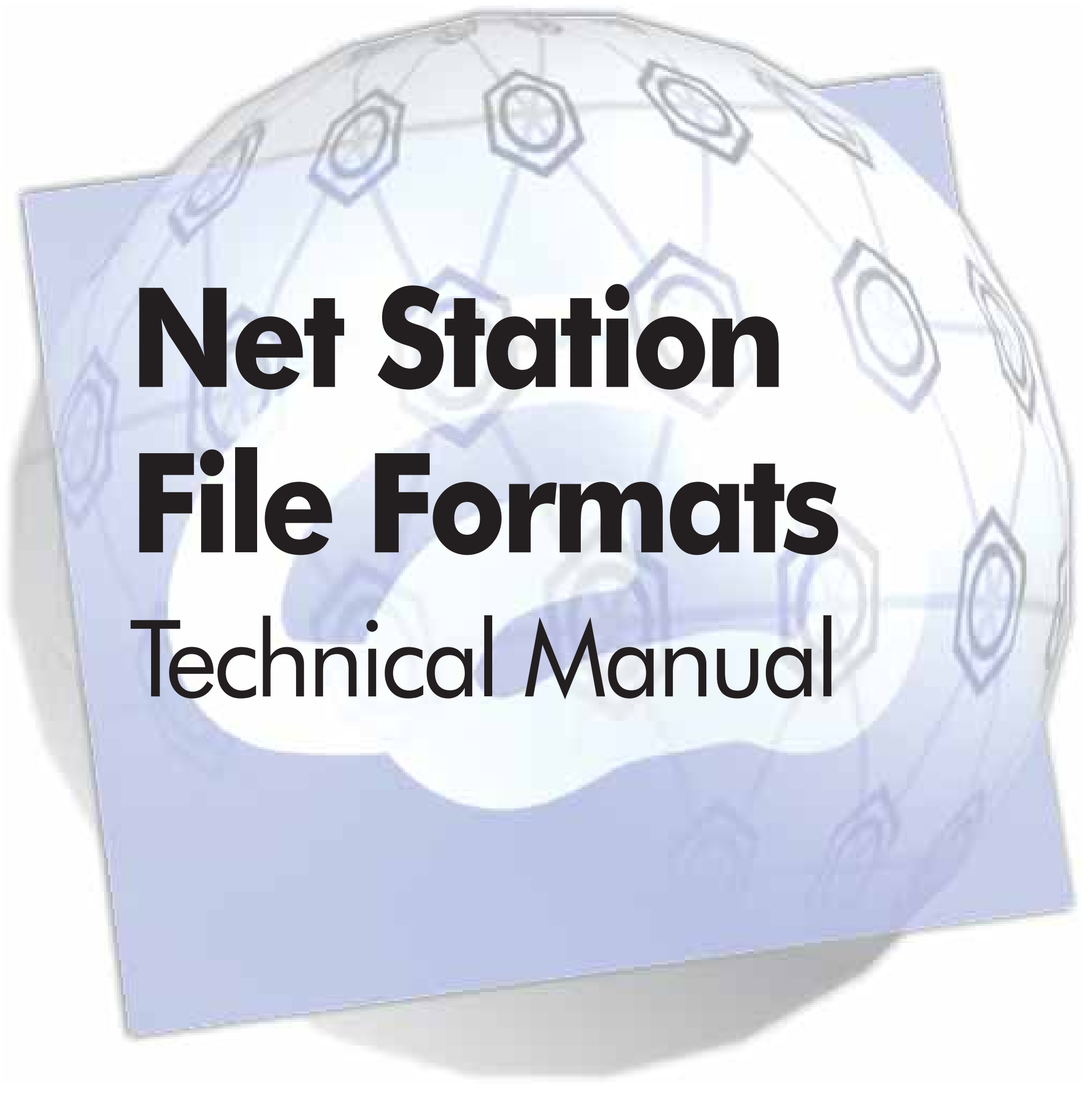


Net Station File Formats

Technical Manual

EGI
dense-array EEG

Electrical Geodesics, Inc.
Riverfront Research Park
1600 Millrace Drive, Suite 307
Eugene, OR 97403
info@egi.com
www.egi.com

Net Station File Formats

Technical Manual

S-MAN-200-FFTR-001
September 30, 2004

CONTENTS

LIST OF FIGURES

List of Tables

PREFACE

Net Station icon

Net Station from Electrical Geodesics, Inc. (EGI) is a complete software package for working with electroencephalography (EEG) and event-related potential (ERP) data. With Net Station, you can:

- acquire EEG, in conjunction with EGI's Net Amps and Neurotravel amplifiers, and dense-array Geodesic Sensor Nets
- perform various operations on your data, primarily for basic ERP derivation and analysis
- view and navigate EEG and ERP data

A variety of publications is available (Figure P-1) as PDF files at www.egi.com/documentation.html.

Acquisition

- Net Station Acquisition Technical Manual
- Geodesic Sensor Net Technical Manual
- EGI System Technical Manual

Review

- Net Station Viewer Technical Manual
- Net Station Viewer and Waveform Tools Tutorial

Data manipulation

- Net Station Waveform Tools Technical Manual
- Net Station Viewer and Waveform Tools Tutorial
- Net Station File Formats Technical Manual

Figure P-1. EGI manuals for EEG acquisition, review, and data manipulation

Net Station publications

- This manual, the ***Net Station File Formats Technical Manual***, documents the objects contained in a native Net Station file, the formats of the export files, and other files associated with Net Station.
- The Net Station Acquisition is the component of the Net Station software for acquiring EEG, in conjunction with the dense-array Geodesic Sensor Nets. The ***Net Station Acquisition Technical Manual*** describes Acquisition features and functions.
- The Net Station Viewer is the component of Net Station for viewing and navigating EEG and ERP data. The ***Net Station Viewer Technical Manual*** describes Viewer features and functions.
- The Net Station Waveform Tools is the component of Net Station for performing various operations on EEG data, primarily for basic ERP derivation. The ***Net Station Waveform Tools Technical Manual*** describes Waveform Tools features and functions.
- The ***Net Station Viewer and Waveform Tools Tutorial*** instructs you in the use of Net Station Viewer and Waveform Tools by guiding you through the analysis of a sample data set. It is not intended to be a comprehensive guide to these components, but it is a good place to start when learning about the software.

Other EGI publications

- The Geodesic Sensor Net is EGI's patented device for acquiring electrical signals from the human scalp. The ***Geodesic Sensor Net Technical Manual*** describes GSN features and functions.
- EGI manufactures and distributes complete EEG and ERP systems. The ***EGI System Technical Manual*** describes the hardware components and features of EGI Systems.

To get the most out of these books, you should have some background in EEG methods. These manuals assume you are familiar with the Macintosh computer, the platform for Net Station software.

About This Manual

The *Net Station File Formats Technical Manual* describes the different file formats used or exported by Net Station. The objects contained in a native Net Station file, the formats of the export files, and other files associated with Net Station are documented.

The purpose of the publication is to enable the sharing of Net Station data with other applications. EGI strongly recommends exporting to the ***epoch-marked simple-binary*** file format (which is simpler and more streamlined than other, older EGI formats), unless you must export to a specific format, such as MATLAB MAT-file or Persyst.

This manual is targeted toward researchers, clinicians, and graduate students in fields involved with collecting and processing EEG data, such as psychology, epilepsy, developmental research, presurgical planning, and neonatal monitoring. If you have an application that imports or works with Net Station EEG data, you need to understand the material in this book.

Manual Organization

The manual consists of four main topics:

- file formats overview
- Net Station native files
- Net Station export files
- other export files

Each section contains chapters that elaborate on the topic.

- The file formats overview describes formats in general, differentiates between Net Station native file formats and export formats, and briefly mentions the characteristics of each format discussed in the manual. (See Chapter 1.)
- The Net Station native files section describes Recording and Session files (the two main Net Station native files) and discusses their common underlying data structure and architecture. Net Station native files are object oriented, so no detailed discussion of data format structure is included. However, excerpts from the Net Station Waveform Recording application programming interface (API) are provided for reference. (See Chapter 2.)

- The Net Station export files section describes the format pros and cons, file headers and data structure (where possible), available resources, and export procedures of epoch-marked simple-binary, simple-binary, tab-delimited text, EGIS, MATLAB MAT-file, and Persyst files. Tables of the data formats are included. (See Chapters 3, 4, 5, 6, 7, and 8.)
- The last section, about other files, describes non-EEG data such as event export and history export, and discusses the Net Station File Exporter application. (See Chapter 9.)

Appendix B provides example code for Net Station files.

Manual Conventions

Data Tables for Export Formats

This manual treats an export file as a structured container for data. The file contains a finite sequence of bytes, with the first byte located at offset 0 (the beginning of the file) and the last, or Nth, byte at offset N-1. The structural rules that govern a file identify the offsets where various pieces of data begin and end.

Where possible, we place these structural rules in tables that indicate what types of data can be found at various offsets. (Data tables for formats such as MATLAB MAT-file are available from their manufacturers.) For brevity, the tables refer to metadata, events, and EEG data all simply as *data*, but they also provide additional descriptions that help differentiate among the data types (Figure P-2).

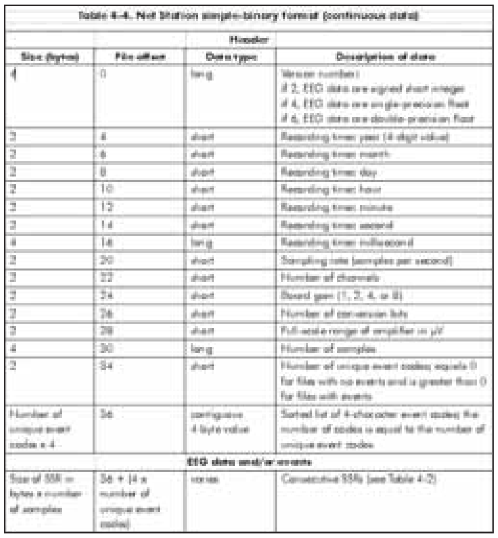


Table 4-4. Net Station simple-binary format (continuous data)

		Header	
Size (bytes)	File offset	Data type	Description of data
4	0	long	Version number: if 2, EEG data are signed short integer if 4, EEG data are single-precision float if 6, EEG data are double-precision float
2	4	short	Recording time: year (4 digit value)
2	6	short	Recording time: month
2	8	short	Recording time: day
2	10	short	Recording time: hour
2	12	short	Recording time: minute
2	14	short	Recording time: second
4	16	long	Recording time: millisecond
2	20	short	Sampling rate (samples per second)
2	22	short	Number of channels
2	24	short	Board gain (1, 2, 4, or 8)
2	26	short	Number of conversion bits
2	28	short	Full-scale range of amplifier in μV
4	30	long	Number of samples
2	34	short	Number of unique event codes; equals 0 for files with no events and is greater than 0 for files with events
Number of unique event codes x 4	36	contiguous 4 byte value	Sorted list of 4-character event codes; the number of codes is equal to the number of unique event codes
		EEG data and/or events	
Size of SSR = bytes x number of samples	36 + (4 x number of unique event codes)	varies	Consecutive SSRs (see Table 4-2)

Figure P-2. An example data table, showing byte size, file offset, data type, and data description

Terminology

Net Station is an application that can read, write, and display the contents of a variety of different EEG data files, both native and export. Following are definitions of some of the main terms used in this book.

- ***Native and Export Formats***. This manual defines a *native* format as one that is created only in Net Station and not in any other application. The term *non-native* typically refers to formats that are created and read by other applications. However, because Net Station can also read and, in some cases, write these same formats, this book uses the term *export* format instead.
- ***Data File, Continuous or Segmented***. A *data file* is any file that contains EEG data. Some data files can contain event data as well. This manual further classifies data files as *continuous* (unsegmented) and *segmented*. Continuous files can store only one block of data, and segmented files can store from one to many blocks of data.
- ***Sample***. In Net Station, as in other analog-to-digital (A/D) data-acquisition systems, analog signals (voltages) are sampled at discrete moments in time. When examined consecutively, the samples form a series of values that approximate the original analog signals.

  Because Net Station is part of a dense-array EEG system, it collects many of these signals simultaneously. So, a *sample* actually is a *collection of samples*, one for each channel. A Net Station sample fundamentally is a vector containing as many values as channels (for example, 64, 128, or 256 values if using a 64-, 128-, or 256-channel Geodesic Sensor Net).
- ***Reference Channel***. Internally, Net Station stores an additional element in a sample vector: the reference value for the sample. Referred to as the *reference channel*, it becomes the 65th, 129th, or 257th value in the sample. This manual describes the reference value for some of the export file formats.
- ***Source File***. A *source file* is defined as a file that contains the data to be exported from Net Station into one of the export formats. In most cases, the source file will be a Net Station Session or Recording file. In others, the source file can itself be an export file such as EGIS.

Typography

In general, a minimal amount of special fonts are used: *italics* for definitions or newly introduced terms, ***boldface italics*** for important concepts, **boldface** for command paths (for example, **File > Save**), and `Courier New` for computer code. The computer code is indented to set it off from the regular chapter text. Ellipses (`. . .`) indicate missing sections of example code.

```
d =  '01-Mar-1999'
d =
      01-Mar-1999
. . .
datestr(d, 17)
ans =
      Q1-99
```

Additional Information

This book uses two methods to convey additional information: notes and cautions.

Note: This indicates information that is helpful in understanding Net Station operations.

⚠ ***Caution!:*** *This denotes important information that, if unheeded, could hinder Net Station use or result in data damage.*

Support Resources

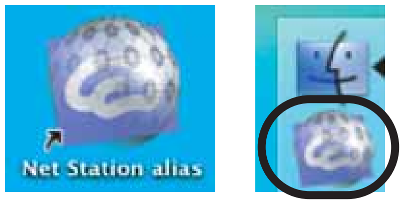
EGI Documentation page at www.egi.com/ documentation.html

- For online updates to this book, check EGI's Documentation page at www.egi.com/ documentation.html.
- For Net Station EEG and ERP software support, see Appendix A, "Software Technical Support," on page 135.

Starting Net Station

To start Net Station, double-click on its icon on the Finder desktop. If the Net Station icon is in the Dock in Macintosh OS X, only a single click is needed. A triangle pointing to the icon will appear, indicating the application's active status.

The Net Station icon



desktop Dock

chapter 1

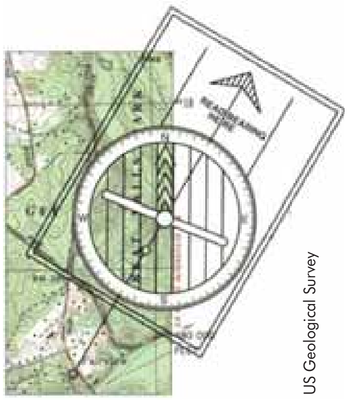
FILE FORMATS OVERVIEW

Introduction

A *file format* is the way a particular application lays out or organizes the data in a file. For another application to use or access data from that application, it must be able to recognize the format and know where specific information is stored within that format.

The purpose of this publication is to describe Net Station native, export, and other file formats so that you can write applications that can use Net Station data. Net Station produces or exports a handful of formats: Net Station Session or Recording, epoch-marked simple binary, simple binary, tab-delimited text, EGIS, MATLAB MAT-file, Persyst, and other data (for example, event export, history export).

These formats are categorized as *native*, *export*, and *non-EEG*.



File formats are like maps of a document's structure, allowing you to come up with your own routes.

Byte-Swapping

All the file formats, except Persyst (see Chapter 8), discussed in this manual use big-endian byte order to represent numeric data. This is the native byte order for Apple and Unix computers. Applications that run on Intel hardware or other little-endian machines must perform byte-swapping to correctly read the numeric data of these formats.

⚠ ***Caution!:*** *Some header fields in simple-binary format files contain character data, notably the sorted list of event codes and category name array. This data type is "platform neutral," and you should not perform byte-swapping on these fields.*

Macintosh OS File-Type Codes

Nearly every file type discussed in this manual has a Macintosh OS file-type code. The term *file type* is part of the operating system and refers to the case-sensitive, four-character code that the Finder uses to identify a file's type. This method for identifying file types is available only on the Macintosh OS because the file-type code is not a component of the file's data fork.

Data Type Precision

Each format allows you to save EEG with various levels of precision. Some formats offer integer, floating-point single precision, and floating-point double precision; others offer only the first two options.

The advantages of saving with integer precision, rather than floating point, are a faster export and a smaller file (about half the size). The disadvantage is that the individual gains and zeros are not applied to the data; however, if your amplifiers are in spec, then the loss of accuracy should be negligible.

The advantage and disadvantages of saving with floating-point precision, rather than integer, are the reverse: a much more precise file, but a slower export and a larger file (about twice the size).

Net Station Native Files

Native file formats

- Net Station Recording
- Net Station Session

Net Station creates two kinds of native files: *Recording* and *Session*. These files have a number of different forms, depending on whether a file has been segmented, edited, or both. (Session files have two more variants: combined, and edited and combined.) By "edited," we include such operations as bad channel replacement, averaging, average referencing, baseline correction, grand averaging, and difference wave.

The internal data format of the Recording and Session files is the same. The only differences between the two are that Session files contain descriptive information (that is, fields) and they can be instantiated using the New command. This enables Session files to reduce the complexity of running an acquisition session by combining a preconfigured *Acquisition Setup* (a data-acquisition template) and a waveform recording into one storage container. To design and save an Acquisition Setup, you must create a Recording file using the Net Station Workbench (see the *Net Station Acquisition Technical Manual* for more information about the Workbench).

Net Station Export Files

Export file formats

- Epoch-marked simple binary
- Simple binary
- Tab-delimited text
- EGIS
- MATLAB MAT-file
- Persyst

Net Station export files are created and read by other applications and, once created, do not require Net Station or its API to access the EEG data. This manual describes six export formats: epoch-marked simple binary, simple binary, tab-delimited text, EGIS, MATLAB MAT-file, and Persyst.

The manual describes the structures of the export file, placing them in tables that indicate the types of data found at different offsets. This will enable you to use various software applications to open, read, and extract the EEG data on any platform.

Epoch-Marked Simple Binary

Epoch-marked simple binary (also known as *epoch-marked raw*) is EGI's preferred export file format. This format contains metadata and EEG data and, optionally, events. The metadata are stored in the header and include information such as recording time, recording date, sampling rate, number of channels, board gain, and number of conversion bits. The metadata also indicate where the header ends and whether the file contains events.

Epoch-marked simple binary can represent a variety of data: continuous recordings, recordings with epoch breaks, segmented files, and averaged files. Epoch marked is an extension of simple binary, but it features two additional codes: `epoc` for epoch breaks and `tim0` for timepoint indicators. This allows it to generate only two output files: one for the EEG data, and the other for the epoch breaks and timepoint indicators.

Simple Binary

Simple binary (also known as *raw*) contains metadata and EEG data, and can contain events. The header contains the metadata, which indicate where the header ends and whether the file contains events. Simple binary is easy to use and well known. Only minor modifications are needed to allow a simple-binary reader to read epoch-marked simple binary.

Tab-Delimited Text

The tab-delimited text (also known as *ASCII*) file is ideal for importing into spreadsheets or databases. This format is a single, unsegmented block of EEG data that is saved with one level of precision and one type of unit (microvolts). In other words, a tab-delimited text file has no metadata, events, or header; the data begin at the start of the file. This file format is the most common format for text files in computers and on the Internet. It can be read by any text editor and imported into a variety of applications, including Microsoft Excel.

EGIS

The EGIS export file format is best for users of early versions of EGI software and hardware who may have many files written in this format, or for customers who simply prefer EGIS. The advantage of this format is that it allows customers to continue using a familiar format, dating back many years.

MATLAB MAT-File

This file format is for developers who want to use Net Station data in MATLAB programs. MATLAB, by The MathWorks, is well known for it predictive mathematical models and application-specific algorithms.

Persyst

This file format is for developers who want to use Net Station data in Persyst programs designed for clinical digital EEG applications such as epilepsy monitoring.

Non-EEG Files and the File Exporter

Net Station also exports to a variety of non-EEG data formats. This manual describes the two main types of data exported: event export and history export. It also discusses the Net Station File Exporter application.

Event Export

Net Station allows you to export some or all events from your source data into a tab-delimited text file. You can use the event file with custom programs or applications.

History Export

You can also export a log of operations performed on your source data into a tab-delimited text file. The resulting history file provides useful background information about your file that can allow you to replicate or troubleshoot an experiment.

Net Station File Exporter

The File Exporter is a drag-and-drop utility for transferring data from Net Station files. It extracts EEG data into a simple-binary data file suitable for cross-platform analysis; an event history into a tab-delimited text file; and gain, zero, and impedance histories into separate tab-delimited text files.

Net Station Miscellaneous Files

Net Station also has several miscellaneous noncustomizable files that support the application. Your work with Net Station native and export files will not require direct interaction with these files, and consequently, this manual does not describe them in detail. However, it is good to be aware of them and their functions.

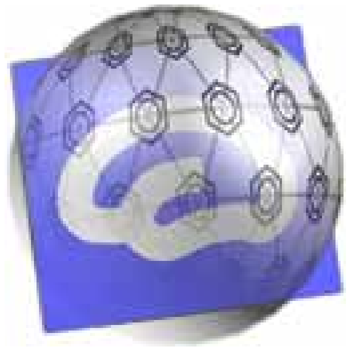
Table 1-1. Net Station miscellaneous noncustomizable files

Icon	File	Description	Type code
	Preferences	Resides in the <user>/Library/Preferences/ folder. If Net Station fails to find this file in its default location, it will create a new one.	`'pref'`
	Shared Library	Binary file containing code resources needed by Net Station. Examples include Net Station Interface and Net Station Services. These files are located in the <user>/Library/CFmSupport/ folder.	`'shlb'`
	Resource Database	Stays in the <user>/Documents/Net Station User Data/Support/ folder; not customizable by users. If Net Station fails to find this file, it will create a new one in the Support folder.	`'SPRT'`
	Tool Database	Stays in the <user>/Documents/Net Station User Data/Support/ folder; not customizable by users. If Net Station fails to find this file, it will create a new one in the Support folder.	`'SPRT'`

NET STATION NATIVE FILES

Introduction

Net Station native files are object-oriented databases. This chapter describes the objects in a Net Station file. Also, to provide a sense of the API's architecture, we describe some of its methods, declarations, and initializations in the "Introduction to the Waveform API" section at the end of this chapter.



More information about the Net Station Waveform Recording API is available from support@egi.com; please note, however, that EGI does not officially provide support for the API.

Net Station Background

Net Station creates two kinds of native files: *Session* and *Recording*. These files have a number of variants, depending on whether a file has been segmented, edited, or both. (Session files have two more variants: combined, and edited and combined.) By "edited," we include such operations as bad channel replacement, averaging, average referencing, baseline correction, grand averaging, and difference wave.

The only differences between the two are that Session files contain descriptive information (in *fields*) and can be instantiated using the **New** command in a Session Template. Recording files are created using the Workbench, which allows you to design Acquisition Setups (for details about the Workbench, see the *Net Station Acquisition Technical Manual*).

To create an Acquisition Setup, you can drag the needed devices from the Devices palette and place them in the cells on the Workbench, or you can click in a Workbench cell to select the cell and choose a device from the Acq menu. You can then save your setups for later use.

Net Station Recording Files

After creating an Acquisition Setup using the Workbench (note that the Workbench should be in the "off" mode), you save the setup, turn the Workbench on, and begin recording EEG waveforms.

Net Station Session Files

Net Station Session files are generated by starting a session, which is always initiated using a Session Template.

The internal data format of Session and Recording files is the same and will be discussed as one topic in the following sections.

File Type Variations

Net Station offers 10 variants of its native format: Session, Combination Session, Edited Session, Edited Combination Session, Segmented Session, Edited Segmented Session, Recording, Edited Recording, Segmented Recording, and Edited Segmented Recording. Table 2-1 lists file icons, descriptions, and Mac OS file type codes (see "Macintosh OS File-Type Codes" on page 22 for more information).

Table 2-1. Net Station native formats

Session files			
Icon	**File**	**Description**	**Mac OS type code**
	Session	Created using a Session Template, which contains an embedded Acquisition Setup. Acquired EEG data will be saved in this file along with metadata.	'dSES'
	Combination Session	Net Station's Combine methods produce new files that are combinations of segmented EEG data. Such files are called Combination files.	'aSES'
	Edited Session	A Session file containing edited EEG data. Editing generally means the addition of user or auto event markers.	'dSE!'
	Edited Combination Session	A Combination Session file that has been edited manually using the Viewer. Editing generally means the addition of user or auto event markers.	'aSE!'
	Segmented Session	A Session file that has been segmented using one of Net Station's Segmentation techniques.	'eSES'
	Edited Segmented Session	A Segmented Session file that has been edited manually using the Viewer. Editing generally means the addition of user or auto event markers.	'eSE!'
Recording files			
	Recording	Original waveform recording created by means of a Recorder device on the Workbench.	'rEEG'
	Edited Recording	A Recording file that has been edited manually using the Viewer. Editing generally means the addition of user or auto event markers.	'rEE!'
	Segmented Recording	A Recording file that has been segmented using one of Net Station's Segmentation techniques.	'eEEG'
	Edited Segmented Recording	A Segmented Recording file that has been edited manually using the Viewer. Editing generally means the addition of user or auto event markers.	'eEE!'

In addition, Net Station features a number of ancillary files. Table 2-2 lists file icons, descriptions, and type codes.

Table 2-2. Net Station support files

Icon	File	Description	Mac OS type code
	Acquisition Setup	User-generated file. Contains a particular arrangement of interconnected Workbench devices.	'dACQ'
	Session Template	User-generated file. Contains an embedded Acquisition Setup. Created by clicking the Session Template button in the Net Station start-up window or by clicking the Create New Template button in the Select Session Template window.	'sSES'
	Combination Tool clipping	User-generated file. Created by dragging to the Desktop a Combination Tool icon from the Combine panel of the Waveform Tools window. Saves the tool for use by Net Station users other than the originator of the tool. The originator can drag the tool into his or her own Waveform Tools window, where it will persist and become part of that person's collection of tools.	'AOBJ'
	Segmentation Tool clipping	User-generated file. Created by dragging to the Desktop a Segmentation Tool icon from the Segment panel of the Waveform Tools window. Saves the tool for use by Net Station users other than the originator of the tool. The originator can drag the tool into his or her own Waveform Tools window, where it will persist and become part of that person's collection of tools.	'AOBJ'

ERP Experiment Structure

This section lays the groundwork for the main discussion about Net Station's data structure and architecture, by providing an overview of ERP experiments.

A Sequence of Trials

Virtually all ERP experiments consist of a sequence of trials (Figure 2-1).

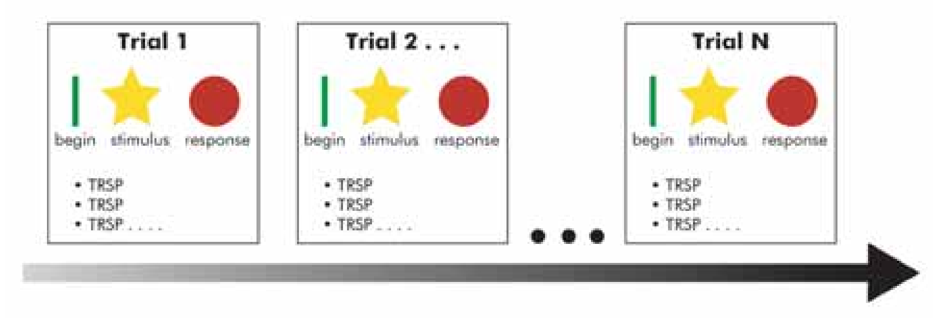


Figure 2-1. A sequence of trials in an ERP experiment

Trials usually consist of the following:

- beginning event
- stimulus event
- response event
- trial specifications or TRSPs (that is, an annotation of the trial), which include:
  - *rsp#*: which key was pressed
  - *rtim*: reaction time
  - *eval*: evaluation of response (correct or incorrect)
  - what stimulus was used (optional; can be added by the user when coding the experiment)
  - percentage of correct responses so far (optional; can be added by the user when coding the experiment)

Classifying Trials

Cells

Cells provide a means of logically grouping the trials of an experiment into sets. Trials are grouped into cells on the basis of stimulus characteristics, not responses. Therefore, the process of grouping trials into cells typically is not affected by the subject's input. Cells are assigned by giving all similar events in the trial the same cell number.

Let's take an auditory target detection experiment as an example. Here, trials are divided into two cells: *standard* and *target*. Figure 2-2 shows all the standards grouped into Cell 1, and all the targets grouped into Cell 2.

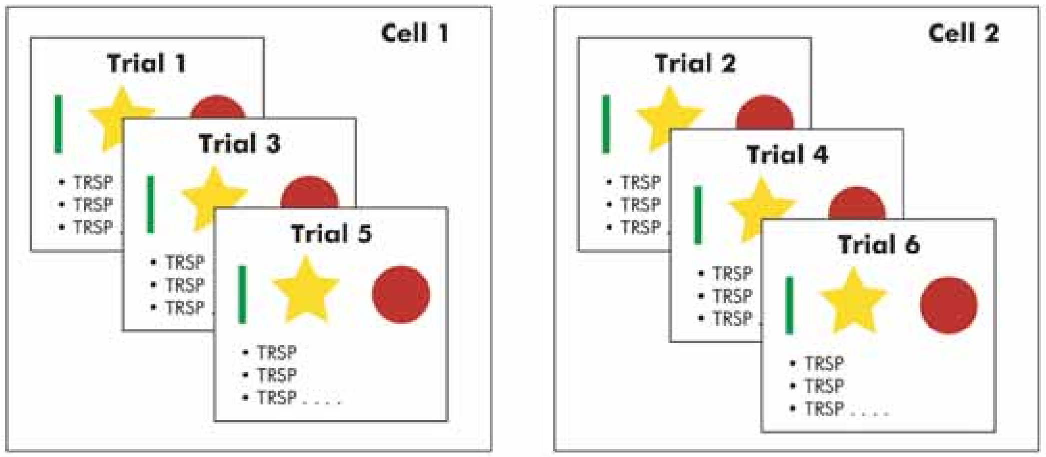


Figure 2-2. Grouping trials into cells

Categories

Categories provide another means of logically grouping the trials of an experiment into sets (Figure 2-3). Categories differ from cells in several important ways:

- Categories are applied to the data after acquisition, during processing.
- The subject's responses can be factored in.
- Categories are more flexible than cells. With cells, each trial can belong to only one cell, and all the events in the trial must have the same cell number. With categories, a trial can belong to multiple categories or no categories, and the events in a trial can belong to different categories (Figure 2-3).

You define categories by specifying *rules* for determining if an event belongs to a certain category. Rules can refer to cells or they can specify other factors.

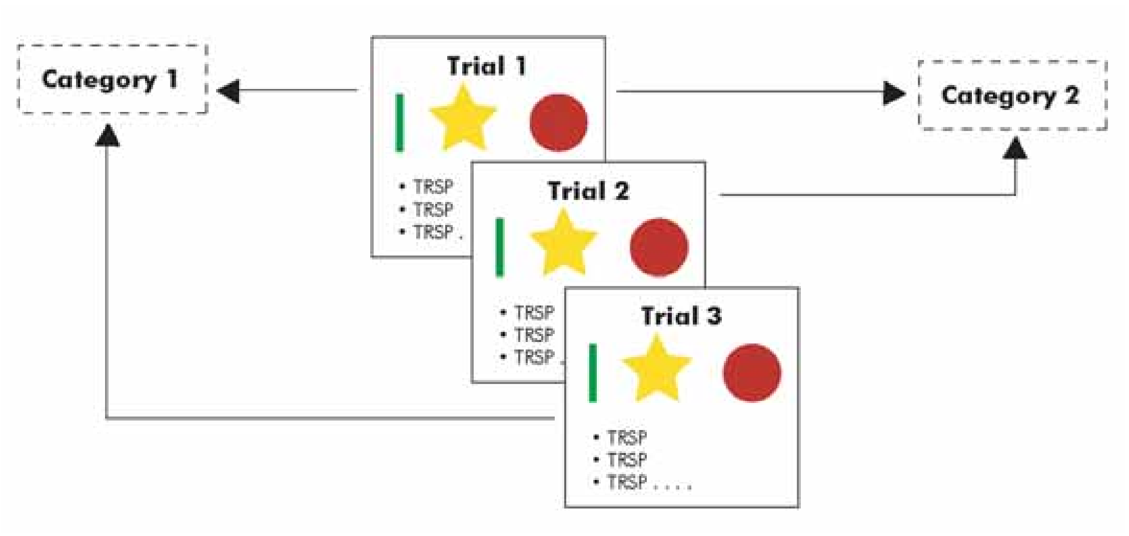


Figure 2-3. Grouping trials by categories

Net Station Data Structure

Elements, Tracks, and Epochs

As the basic building block of a Net Station recording file, the *object* is the interface between the user and the application. All the details of the recording are hidden.

The lowest level of objects is the *element*. A sample is an element and is defined as one datapoint for each channel of a recording. All elements of the same type are found on a *track,* which is described further on page 37. Each element has a time variable, and all the elements together provide a start time and an end time for the track. All the tracks from a specific start time to a specific end time make up an *epoch*. A number of epochs then make up a *recording* (Figure 2-4).

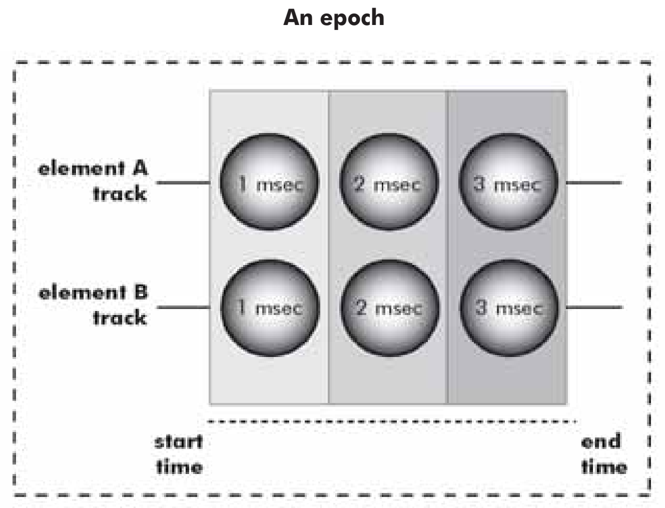


Figure 2-4. In Net Station, elements make up a track, tracks make up an epoch, and epochs make up a recording

Note: Tracks cross epoch boundaries, but elements cannot be positioned in these boundaries.

In short, elements are the smallest objects in a recording, tracks string similar elements together like beads on a string, and epochs contain all tracks within a timespan given by the elements' time variables. All of this is based on synchronization with Net Station's time base.

Time Base and Synchronization

Net Station uses a *millisecond* time base (that is, one-thousandth of a second). The time base can be conceptualized by using the metaphor of *data bins*.

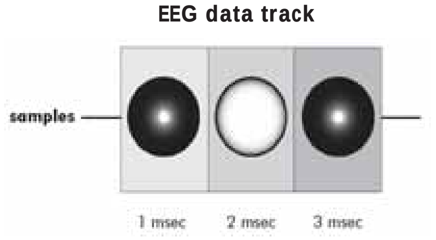


Figure 2-5. An example EEG data track, sampled every millisecond (black = with data; white = empty)

For each track (sample, sampling rate, event, and so on) of a Net Station recording, a row of bins exists. The length of the row depends on the length of the recording. One bin is assigned to each millisecond of time in the recording. As each new millisecond of time passes, Net Station fills the current bin of each track. The bin is either empty or it contains a datapoint, if data are available for that track (Figure 2-5).

For example, at 500 samples per second, a sample is processed every other millisecond. Because a bin exists for every millisecond, the sample track's bins alternate between containing a sample and containing nothing.

Time Modes

Because Net Station collects data as they are being created and also stores them for later viewing and editing, two different time references exist for Net Station files: *real time* and *review.*

In real time, a Net Station recording always begins with time set to 0, which corresponds to the *Relative Time* mode (relative to the time the recording began). The recording is also represented by *Absolute Time*, the "real time" as set by the computer's clock.

During review (that is, when you review the recorded data after acquisition), Relative Time means *relative to the time you recorded the first sample.* Any breaks caused by stopping and resuming the recording of the data will be indicated as *epochs,* which are defined as continuous periods of data acquisition. This introduces another time mode—*Epoch Time*—which consists of the time since the beginning of the epoch and the number of the epoch. Finally, if you segment the file, a fourth time mode is introduced: *Segment Time*. This is the time with respect to the sample to which the segmentation is temporally referenced (Figure 2-6).

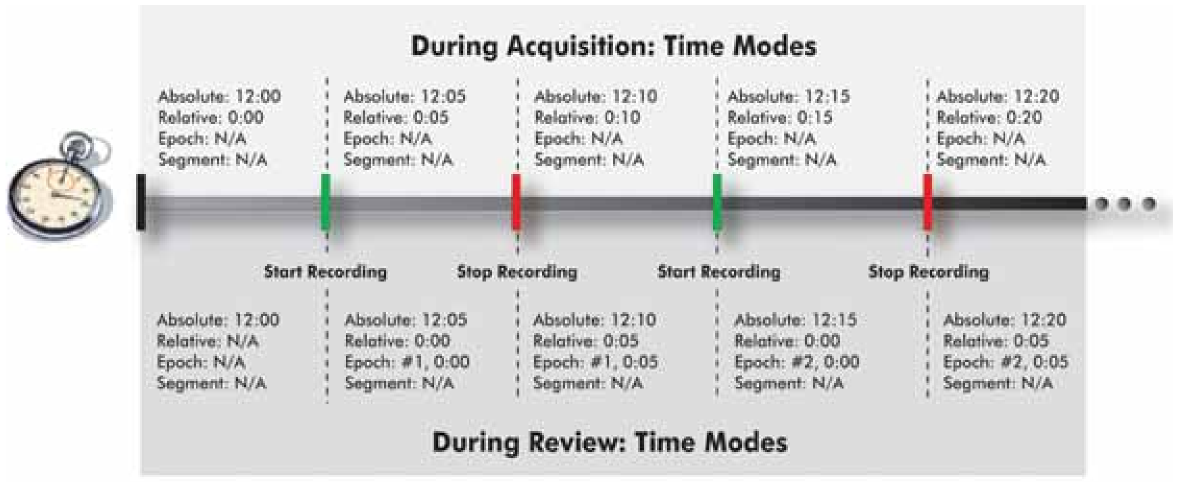


Figure 2-6. Net Station's real-time and review time modes

Synchronizing Objects

Net Station synchronizes all its time-based objects by declaring them to be members of a *real-time element class*. Each member of this class has an identifier and a time variable, in addition to any features of its class hierarchy. Net Station sets the time variable in relation to the time when the file was created. The real-time element superclass allows any object declared as a real-time object to have a specific value at a specific point in time.

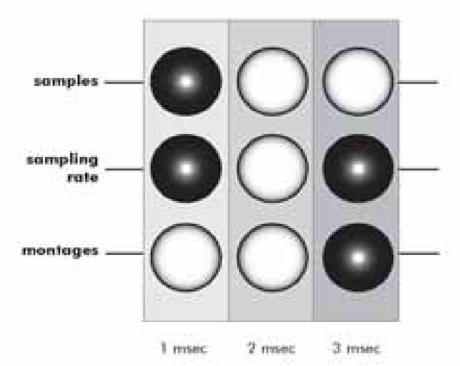
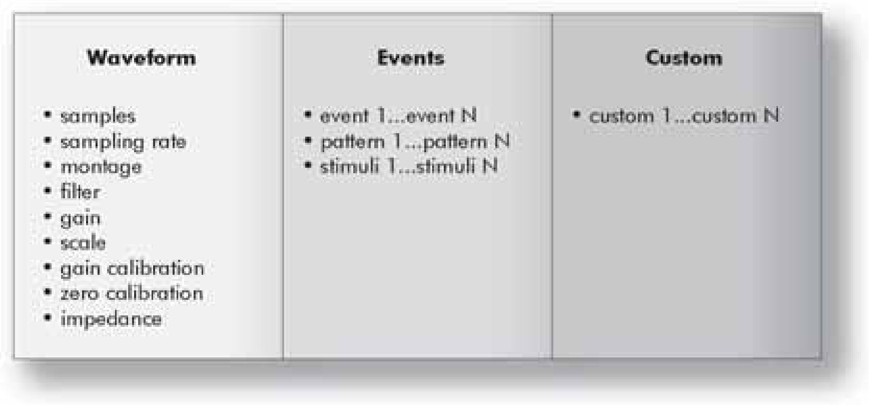


Figure 2-7. Each time-based Net Station object is synchronized by its time variable

This synchronization allows us to choose a point in time and view the data for samples, sampling rates, montages, and so on (Figure 2-7).

Tracks

A track is a collection of chronologically arranged elements superimposed on the data. Tracks are divided into three general classes: *waveform*, *event*, and *custom* (Figure 2-8).



Waveform	Events	Custom
• samples • sampling rate • montage • filter • gain • scale • gain calibration • zero calibration • impedance	• event 1...event N • pattern 1...pattern N • stimuli 1...stimuli N	• custom 1...custom N

Figure 2-8. Net Station classes of tracks: waveform, events, and custom

- *Waveform tracks*: Contain all the information needed to describe the waveform. There are a total of nine waveform tracks: samples, sampling rate, montage, filter, gain, scale, gain calibration, zero calibration, and electrode impedance. (See the next section for details.)
- *Event tracks*: Come in three different types: regular event, automatic pattern, and stimulus. Net Station can accommodate any number of these three types of tracks. (See page 39 for details.)
- *Custom tracks*: Are currently unavailable. They are intended to support future Net Station functionality and are not described further in this manual.

Waveform Tracks

Following are brief descriptions of the nine waveform tracks:

- *Samples*: Contains both simulated data and digital information created by the A/D converter. The data on this track require the other tracks to convert the A/D values into the microvolt values of a waveform (Figure 2-9).
- *Sampling rate*: Carries the sampling rate, which is chosen by the user and can be changed at any time. The user has a choice of 50, 100, 125, 200, 250, 500, or 1,000 samples per second.

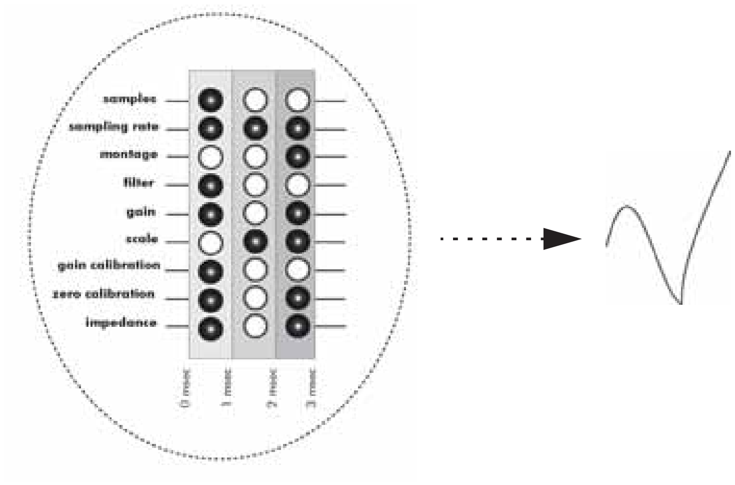


Figure 2-9. Nine different waveform tracks are needed to create a typical waveform.

- *Montage*: Contains a listing of the channels or groups of channels being displayed.
- *Filter*: Notes hardware filters being used, if any.
- *Gain*: Contains the record of the gain as set on a source device. For USB amplifiers, such as the Net Amps 200, gain is always 1. (For the Net Amps 100, the amplification factors available were 1x, 2x, 4x, and 8x for a machine with a NuBus card and 1x, 2x, 5x, and 10x for a machine with a PCI-1200 card.)
- *Scale*: Contains two variables. One is the maximum microvolt measurement representable; the other is the number of bits.

  Net Amps 200 has a range of ±2,500 μV and uses 16 bits. The 16-bit A/D converter changes measurements of –2,500 to +2,500 μV into numbers in the range that are the digital equivalent of the microvolt measurements. (Amplifiers with serial numbers one through three had a 1,600 μV range.)

  The scale is the conversion factor that changes the digital values back into microvolts for the display unit. The scaling factor is known as the *nominal gain* and is the full-range scale of microvolts divided by the full-range bit scale.

- *Gain calibration*: Contains the values from the last gain calibration measurement (see the *Net Station Acquisition Technical Manual*). A value is saved for each channel. If these calibrations are done multiple times during a session, a record known as a *calibration history* is kept. A user can track the performance of Net Station over time by consulting these histories. The gain calibration record is available through the Net Amps Controller.
- *Zero calibration*: Describes the zero value for each channel of an amplifier. These values are used to convert A/D units to microvolts.
- *Electrode impedance*: Describes the scalp impedance of each electrode, if measured.

Event Tracks

We define an *event* as a particular section of a waveform that is distinguishable from the overall waveform for some reason. Multiple devices of the same class can be used, so a file can have multiple event tracks, one for each device capable of creating events. Following are brief descriptions of the main event tracks.

- *ECI track*: Is the only track to have cells. All events on the Experiment Control Interface (ECI) track have only the following elements (which are described further in the next section):
  ° Code
  ° Label
  ° Type
  ° Track
  ° Onset
  ° Duration
  ° Key lists
- *DIN track*: Represents digital input (DIN) from an external source. The events on the DIN track have the same elements as the ECI track. The key list depends on the settings in the Digital Input Controls panel.

- *Markup track*: Contains events created for export. Other applications might not support key lists, so markup creates new events, usually using information from the key lists. The events on the Markup track have the same elements as the ECI track except the key lists are empty.

Event Data Structure

An event can be a stimulus, response, or other marker. All events have the following components:

- *Code*: a unique, user-defined, four-character descriptor that identifies the event type (*atg_* for auditory targets or *resp* for responses, for instance)
- *Label*: an easily readable description of the event code that is not limited to four characters (for example, *ATDTarget* for targets, which are infrequent tones, in an auditory target detection experiment); can be empty
- *Type*: a descriptor that indicates the nature of the event or how it was generated (such as Stimulus Event or Event Response)
- *Track*: a descriptor that indicates the source of the event (for example, if the event were created automatically by the experiment control computer, the track would be ECI)
- *Onset*: the event start time in Absolute Time and in milliseconds (18:30:30.500 represents 6:30 p.m.)
- *Duration*: the time span between the start (onset) and the end (offset) of the event, in milliseconds
- *Key lists*: metadata about the event (subject information, number of cells, and so on)

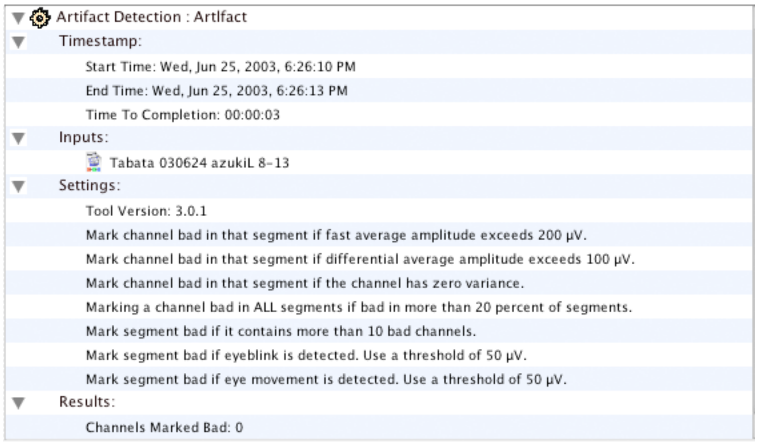
Non-Time Based Objects

A Net Station recording can also contain three types of objects that have no time variable: blobs, persistent objects, and fields. You can use these objects to add information or structures to a file.

- *Blobs*: Are used to add custom structures to a recording. You can access them through their type, ID, index, or handle.
- *Persistent objects*: Function as a base class that allows you to create an object that can read and write itself. You can access them through their type, ID, or index.
- *Fields*: Allow you to enter information into a file. You can access them through their index or field name.

History

For each native file, Net Station automatically records every Waveform Tool used and its timestamp, inputs, settings, and results, if applicable (Figure 2-10). This history log of Waveform Tools operations can be used later for replicating an experiment or diagnosing a problem. Chapter 9, "Non-EEG Data and the File Exporter," describes this feature in more detail.



```
Artifact Detection : Artifact
Timestamp:
    Start Time: Wed, Jun 25, 2003, 6:26:10 PM
    End Time: Wed, Jun 25, 2003, 6:26:13 PM
    Time To Completion: 00:00:03
Inputs:
    Tabata 030624 azukiL 8-13
Settings:
    Tool Version: 3.0.1
    Mark channel bad in that segment if fast average amplitude exceeds 200 µV.
    Mark channel bad in that segment if differential average amplitude exceeds 100 µV.
    Mark channel bad in that segment if the channel has zero variance.
    Marking a channel bad in ALL segments if bad in more than 20 percent of segments.
    Mark segment bad if it contains more than 10 bad channels.
    Mark segment bad if eyeblink is detected. Use a threshold of 50 µV.
    Mark segment bad if eye movement is detected. Use a threshold of 50 µV.
Results:
    Channels Marked Bad: 0
```

Figure 2-10. A history of the artifact detection operations performed on a file

Introduction to the Waveform API

The following paragraphs include the code for five methods plus the needed declarations and initializations for the API. The methods are examples of what a typical user might want to do with a recording. The example code is C++.

Using the sample code in your application, you can open a file, retrieve a section of a recording, list the events in a recording, and write out a section of a file as a matrix while referencing the data. Finally, the method needed to close a file is shown.

The code contains comments that explain unusual features; however, each method called is not explained. (For descriptions of individual methods, see the “Waveform Recording API Reference” section in the API.)

Declarations

These are the declarations needed in an application that is to use the API. Note that your application object must inherit from CWaveformRecordingUser.

```
class MyApplication : public LApplication,
                      public CWaveformRecordingUser {
public:
                  MyApplication(void);
      virtual     ~MyApplication(void);


protected:
                  CWaveformRecording* recording;

      Boolean     OpenRecording(const FSSpec & inRe-
                  cordingFileSpec);
      float       *GetSamples(Time startTime,
                  Time endTime);
      void        ListEvents(const DescType eventI-
                  dentifier);
      void        WriteReferencedData(const FSSpec
                  &outReferencedFileSpec);
      void        CloseRecording(void);
};
```

Initialization

Following is the code to initialize the application and the Waveform Recording API. This example program explicitly assigns values for the default values. Code to initialize the toolbox and any framework-related variables should also go here. The code to initialize the API should go as close to the beginning of the application as possible, after toolbox and heap initialization.

During initialization, the API might create temporary files; thus, you should assign your own creator and file type for these kinds of files.

```
int main(void)

{

 //put your application creator here

const OSType      applicationCreator = 'ABCD';

  //put your temp file type here

const OSType      tempFileType = 'TEMP';

 /*The API performs memory management using the "grow zone" function.
 Determine the amount of memory for emergency grow-zone padding.*/

const Size        growZoneSize = 1000000L;

 /*Determine the size of the object cache. The cache needs to be somewhat large so
 that several seconds of data, as well as event information, can be kept in memory.
 This example begins with a minimum size for the cache and then adjusts it upward if
 more RAM is allocated to the program.*/

const Size        zoneSize = FreeMem();//what we got

 //stdCacheSize is the minimum size for the cache

const SizestdCacheSize = 0x600000L;
const SizeadditionalK = (zoneSize - 20L * 1024L *
                  1024L) / 1024L;
```

```
const Size        cacheSize = (additionalK >= 1 ?
                  stdCacheSize + additionalK * 1024 :
                  stdCacheSize);
```

*/ *'purgeQuantum' is the amount of the cache that is purged when the cache limit is exceeded. This example purges an amount equal to 2,048 samples of 128 channels each.* /*

```
const Size        purgeQuantum = 2048L * 128L *
                  sizeof(short);
```

//If your application has threads other than the main thread, then pass
//true for the last parameter. This example doesn't use threads.

```
CWaveformRecording::Initialize(applicationCreator,
tempFileType,
growZoneSize, cacheSize, purgeQuantum, false);
```

//put other initialization code here

```
} /* main */
```

Opening a File

This function asks for read/write permission initially and then opens read-only, if failure. Returns true if the recording is successfully opened.

```
Boolean MyApplication::OpenRecording(const FSSpec
                &inRecordingFileSpec)
   {
   recording = CWaveformRecording::Open(inRecording-
                FileSpec, readWrite, nil, false, true,
                nil);
   if (CWaveformRecording::Error() || !recording)
      return false;
   else
      return true;

   } /* End, MyApplication::OpenRecording */
```

Retrieving a Block of Data from a Recording

This function retrieves a selection of data and stores it in a freshly allocated block of memory. Regardless of the precision with which the data were stored, this function retrieves the data as low-precision (4-byte) floating-point values.

```
float *MyApplication::GetSamples(Time startTime,
                      Time endTime)
{
   //Check to see if the selection range is valid
   if (!recording->ValidSelection(startTime, endTime))
                  return false;

   /*Check to see if samples in the range are consistent, which means
   that there is no epoch boundary, sampling-rate change, or montage
   change in the range.*/

   if (!recording->SamplesConsistent(startTime, end-
      Time)) return false;

   /*We can create the block now—note that no memory-checking code is in this
   example. We need to know the number of samples and the number of channels
   to determine how much memory will need to be allocated. Because the
   selection is consistent in the number of channels, we need to get only the
   channel count at the start time.*/

   const long      nSamples = recording->CountSamp-
                   ples(startTime, endTime)
   const long      nChannels = (long)recording->Coun-
                   tChannels(startTime);
   long            blockSize = nSamples * nChannels *
                   sizeof(float);
   float           *samples = (float *)malloc(block-
                   Size);
      /*Get the samples. The samples will be converted to microvolts before being
      placed in the sample buffer.*/

      recording->GetSamples(startTime, endTime, samples,
                        &blockSize, true);

      return samples;
   }  /*End, MyApplication::Get Samples */
```

Retrieving Events of a Particular Type

This method retrieves a list of events with a particular identifier and writes their onset times to standard output.

```
   void MyApplication::ListEvents(const DescType eventI-
      dentifier)
   {

      //Get the list of events. It does not matter what track the events are on or
      what//event type (user, auto, stimulus) they are.

      CObjectList        *events;

      events = recording->GetEvents(nil, &eventIdentifier,
                        "\p", nil, "\p", "\p");
      events->Claim(this); //CObjectList is a shared object; claim it.

      /*Count the events and then loop through them and write out their labels*/

      const longnEvents = events->GetCount();
      long              e;

      for (e = 1; e <= nEvents; e++) {
         CStimulus  *anEvent = (CStimulus *)events->Get (e);
```

```
        //write the event time
        printf("%ld\r", anEvent->GetTime());
    }/* end for */

        //release the event list
    events->Release(this);
    }/* End, MyApplication::ListEvents */
```

Reading and Writing Matrices

This method copies the data from one file and writes them to another file. The output file is created from the FSSpec passed in. The data that are written out in the destination file will be average referenced. This function demonstrates the matrix-handling functions of the API.

Note: Using matrices to create output files causes the output files to have only a minimum set of tracks. Supplementary information from the source file is ignored. This example assumes that the sampling rate and channel count do not change.

```
void MyApplication::WriteReferencedData(const FSSpec
                    &outReferencedFileSpec)

{

    //create the destination recording
    CWaveformRecording* destRecording;

    destRecording = CWaveformRecording::New(outRefere-
                    encedFileSpec, GetTimeWorld(),
                    this);

    //Get the channel count and montage information.
    //As noted above, ignore any other track information.

    CSamplingRate* rate;
    CMontage* montage;
```

```
rate = (CSamplingRate *)recording->GetElement-
                  FromTrack(nsTypeSamplingRate, 0);
rate->Claim(this);
montage = (CMontage *)recording->GetElement-
                  FromTrack(nsTypeMontage, 0);
montage->Claim(this);

//we have to loop through the recording epochs
const TimemaxChunkSize = 5000L; //get 5 secs of data at a time
const longnEpochs = recording->CountRecordingEp-
                  ochs();
long              e;
BooleanfirstWrite = true;

for (e = 1; e <= nEpochs; e++) {

    //get information on the start time and duration of the epoch
    RecordingEpochInfo epochInfo;
    recording->GetRecordingEpochInfo(e, epochInfo);

    Time          startTime = epochInfo.startTime;
    const Time    endTime = startTime +
                  epochInfo.duration;

    //get data from the epoch
    while (startTime < endTime) {
        TimechunkSize = endTime - startTime;
        if (chunkSize > maxChunkSize) chunkSize =
          maxChunkSize;

    //the samples are retrieved as a matrix
    CShortMatrix* samples;

    samples = (CShortMatrix *)recording->GetSam-
     ples(startTime, startTime + chunkSize,
     CSample::integer, false);
```

```
        //average reference the samples
        CShortMatrix&m = *samples;
        const longnSamples = m.CountRows();
        long      s;
        const longnChannels = m.CountColumns();
        long      c;
        long      average = 0;

        for (s = 1; s <= nSamples; s++) {
         for (c = 1; c <= nChannels; c++)
                  average += m[ s][ c] ;
         average /= (nSamples + 1);
         for (c = 1; c <= nChannels; c++)
                  m[ s][ c]  -= average;
        }

        //write out the samples
        destRecording->RecordSamples(samples, start-
                  Time, (firstWrite ? rate : nil),
        (firstWrite ? montage : nil));
        firstWrite = false;

        //delete the sample matrix
        delete samples;

        //calculate the start time of the next chunk
        startTime += chunkSize;
    }   /* while */
}    /* for */

//release the shared objects
rate->Release(this);
montage->Release(this);

//close the destination recording
destRecording->Close(this);
} /* MyApplication::WriteReferencedData */
```

Close the Recording

```
void MyApplication::CloseRecording()

{
   if (recording) {
      recording->Close(this);
      recording = nil;
   }
}/* MyApplication::CloseRecording */
```

Examples and Available Resources

Example code for the EGI Net Station file format is provided in Appendix B, "Net Station Example Code," on page 137 of this manual.

Available resources for the Recording file format can be found at www.egi.com/documentation.html. These include Chapter 2, “Introducing Net Station,” and Chapter 4, “Workbench: Controls/Displays,” in the *Net Station Acquisition Technical Manual*, and Chapter 2, “Files,” and Chapter 4, “Acquisition Setup,” in the legacy document *Net Station 2.0 Technical Manual* (alpha).

Available resources for the Session file format can be found at www.egi.com/documentation.html. These include Chapter 7, “Sessions and Session Templates,” in the *Net Station Acquisition Technical Manual*, and Chapter 2, “Files,” and Chapter 5, “Session Template,” in the legacy document *Net Station 2.0 Technical Manual* (alpha).

The Waveform Recording API is available at ftp://ftp.egi.com/pub/support/.

chapter 3

EPOCH-MARKED SIMPLE BINARY

Introduction

This chapter describes EGI's epoch-marked simple-binary (or *epoch-marked raw*) format. EGI recommends the epoch-marked simple-binary format for third-party application developers and laboratories interested in developing their own analysis software.

Net Station 3.0 was the first version of the software to offer this export format. At this time, the epoch-marked format is available *only* with the File Export tool of the Waveform Tools. It is unavailable by choosing **File > Save Selection** or **File > Save a Copy As**.

Epoch-marked simple-binary files can represent continuous recordings (with or without recording breaks) and *categorized* data (segmented or averaged files).

File Type Variations

The epoch-marked format has two file type variations:

- data saved with integer precision
- data saved with floating-point single precision

(See "Data Type Precision" on page 22 for a brief outline of the difference between integer and floating-point precision.)

Interpreting File Types

To identify the precision of the EEG data contained in an epoch-marked simple-binary file, Net Station writes into each file a single-digit version code that can be read from the first four bytes of the file:

- If the version number is 2, then the EEG data have integer precision.
- If the version number is 4, then the EEG data have floating-point single precision.

To tell whether the data contain events, Net Station has a "number of unique event codes" field, beginning 34 bytes into the file:

- If the value in this field equals 0, then the EEG data contain no events.
- If the value in this field is greater than 0, then the EEG data contain events.

To determine whether the data are continuous or categorized, Net Station has a field that lists any event codes, beginning 36 bytes into the file:

- If this field contains no `epoc` or `tim0` events, then the data are continuous.
- If this field contains `epoc` events but no `tim0` events, then the data are continuous with recording breaks.
- If this field contains both `epoc` and `tim0` events, then the data are categorized.

The four-character Mac OS file type code for all epoch-marked simple-binary files is `UGLY`. (See "Macintosh OS File-Type Codes" on page 22 for more information.)

Table 3-1 encapsulates the preceding discussion for quick reference or comparison with other formats.

Table 3-1. Net Station epoch-marked simple-binary files

Icon	File	Description	Version number	Mac OS type code
	Continuous, integer precision, with no epoch breaks	An unsegmented EEG recording containing 0 or 1 (at first sample) `epoc` or `tim0` events, with integer precision (2 bytes per value)	2	'UGLY'
	Continuous, integer precision, with recording breaks	An unsegmented EEG recording containing 2 or more `epoc` events but no `tim0` events, with integer precision (2 bytes per value)	2	'UGLY'
	Categorized (segmented or averaged), integer precision	A categorized recording containing 1 or more `epoc` and `tim0` events, with integer precision (2 bytes per value)	2	'UGLY'

Table 3-1. Net Station epoch-marked simple-binary files

📄	Continuous, floating-point precision, with no epoch breaks	An unsegmented EEG recording containing 0 `epoc` or `tim0` events, with floating-point single precision (real, 4 bytes per value)	4	'UGLY'
📄	Continuous, floating-point precision, with recording breaks	An unsegmented EEG recording containing 2 or more `epoc` events but no `tim0` events, with floating-point single precision (real, 4 bytes per value)	4	'UGLY'
📄	Categorized (segmented or averaged), floating-point precision	A categorized recording containing 1 or more `epoc` and `tim0` events, with floating-point single precision (real, 4 bytes per value)	4	'UGLY'

File Data Structures

Epoch-marked simple binary is an extension of simple binary. EGI developed this format to overcome some of the limitations of the simple-binary format. EGI's epoch-marked format, consequently, features two additional codes missing in simple binary: `epoc` for epoch breaks and `tim0` for timepoint indicators. These additional codes allow the format to output only two files (instead of the multiple files sometimes generated by simple binary): one for the EEG data, and one with labels for the epochs.

This section describes the structures of a single sample record and of continuous and categorized epoch-marked simple-binary formats. Also in this section is a data table that displays the structure of the file header, followed by the structure of its data (EEG data and events, or EEG segments), and indicates file offsets and how to calculate them, if necessary.

File Headers

Each epoch-marked simple-binary file has a header, a series of bytes having its starting point at the beginning of the file. The headers contain metadata, some values of which are essential to calculating or otherwise determining the file's relevant offsets. In the sections that follow, we refer to values that can be found in these headers by giving their offsets.

The length of the header will vary, but you can calculate it based on certain values found at fixed offsets in the file. The event and EEG data will alternate according to a pattern that can be determined using the header information. (See Table 3-2.)

EEG Data

A *single sample record (SSR)* is a container for the data of a single sample. An SSR contains fields for EEG data and events.

The event states for a given sample occur *after* the EEG data for that sample, with each event state having the same precision as its associated EEG data. Hence, event states associated with *integer data* will be valued at 0 or 1; event states associated with *floating-point data* will be valued at 0.0 or 1.0. If there are no event states for a given sample, then the EEG data for the next sample, rather than an event state, will follow the EEG data for that sample. The EEG and events of a simple data file occur in the file as consecutive SSRs, one for each sample (see Table 3-3).

Table 3-2. Net Station epoch-marked simple-binary format

Header			
Size (bytes)	**File offset**	**Data type**	**Description of data**
4	0	long	Version number: if 2, EEG data are signed short integer if 4, EEG data are single-precision float
2	4	short	Recording time: year (4-digit value)
2	6	short	Recording time: month
2	8	short	Recording time: day
2	10	short	Recording time: hour
2	12	short	Recording time: minute
2	14	short	Recording time: second
4	16	long	Recording time: millisecond
2	20	short	Sampling rate (samples per second)
2	22	short	Number of channels
2	24	short	Board gain (1, 2, 4, or 8)
2	26	short	Number of conversion bits
2	28	short	Full-scale range of amplifier in μV
4	30	long	Number of samples
2	34	short	Number of unique event codes; equals 0 for files with no events and is greater than 0 for files with events
# of unique event codes x 4	36	contiguous 4-byte value	Sorted list of four-character event codes; the number of codes is equal to the number of unique event codes
EEG data and events			
Size of SSR in bytes x # of samples	36 + (4 x # of unique event codes)	varies	Consecutive SSRs (see Table 3-3)

Table 3-3. Structure of a Single Sample Record
(Nc = number of channels; Ne = number of events)

Data type	Structure of record	Number of bytes per record
Integer	Consecutive values for channel 1 through channel Nc, followed by event states from state of event 1 through state of event Ne	(2 x Nc) + (2 x Ne)
Floating point, single precision	"	(4 x Nc) + (4 x Ne)

For epoch-marked simple binary files, the EEG data and events part of a file consist of consecutive SSRs. If the value in the events field is more than 0, then the end of the header is where the identities of the events are stored. The events are identified by four-character codes. The event states in each SSR are in the same order as the codes.

Converting to Microvolts

Net Station gives you the option of saving EEG data in either A/D units (raw values that come from the amplifier A/D converter that are technically unitless) or microvolts. This option is represented by the Calibrate Data box (which converts data from A/D units to microvolts) in the Waveform Tools export.

If the source file data are in the form of microvolts, Net Station cannot convert them to A/D units; the sole output choice is microvolts.

The only way to determine the units of measure of the stored EEG data is by looking at the bits field (at offset 26) and the range field (at offset 28) in the header:

- If the values in those two fields are 0, then the data are in microvolts.
- If the values in those two fields are not 0, then the data are in A/D units.

If desired, you can export the data as A/D units and later programmatically convert them to microvolts. There are two different conversion formulae, depending on whether the *channel gains* and *zeros* are available. (*Channel gain* is the amount a channel amplifies a known calibration signal; the gain is expressed as an amplitude. *Channel zero* is the amount a channel is offset from digital zero, in response to a grounded input.)

If channel gains and zeros are ***not*** available, then use the following formula to convert A/D units to microvolts:

$$\text{microvolt value} = (\text{range} \;/\; 2^{\text{bits}}) \times \text{A/D value}$$

where *bits* and *range* are read from the header of the file. This formula uses approximate values for the gains and zeros and will not produce as accurate a microvolt value as using the actual gains and zeros (for more information, see the “Gains and Zeros” section in Chapter 2, “Net Amps 200,” of the *EGI System Technical Manual*).

If channel gains and zeros ***are*** available, then use the following formula:

$$\text{microvolt value} = [(\text{A/D value}) - (\text{channel zero})] \times (\text{calibration signal p–p value}) \;/\; (\text{channel gain})$$

where the current *calibration signal p–p value* is 400 microvolts.

In general, if the data do not need to be in A/D units, select the Calibrate Data checkbox for the greatest microvolt conversion accuracy.

Events

- `epoc`: Indicates epoch breaks. Each occurrence of this event indicates that the corresponding sample is the beginning of a new epoch.
- `tim0`: Indicates that the timepoint at this event is to be considered time 0. Within the epoch, timepoints before this event have negative values. The period before this event is sometimes referred to as the *prestimulus interval* or the *baseline interval*. The presence of this event in a file indicates that the epochs are *categorized* (segmented or averaged).
- The following events lose their meaning outside of the context of Net Station and can be ignored in exported files: CELL, SESS, and TRSP.
- If an event occurs in consecutive samples, it indicates the *duration* of the event, not multiple occurrences of the event. Ignore all but the first one.
- Because of Net Station’s segmentation markup feature, be aware that different events can occur simultaneously.

Epoc Files

When Net Station generates a simple-binary file, it appends ".raw" to the file name. With epoch marked, if the data have been categorized then Net Station generates an additional file, appending ".epoc" to the file name. This second file is a text file that contains labels for all the segments in the file separated by new lines. Each label describes the condition associated with each segment. You can use a text editor to change the labels in the .epoc file, if desired.

For segmented files, in most cases the .epoc file will use the same label for more than one segment. For averaged/grand-averaged files, each segment has a unique label in the .epoc file.

- An .epoc file should have as many lines as there are epochs. If the file has been edited, the new lines may be in Mac, Unix, or PC style. The last label may or may not be followed by a new line.
- User error may lead to more or fewer lines than epochs. If there are more lines than epochs, you should develop a reader that ignores the extra lines. If there are fewer lines than epochs, your reader should handle this situation gracefully, as well.

Programming Tips

This section provides some tips and warnings for programmers writing code to access and use EGI's epoch-marked simple-binary file format.

The epoch-marked format is based on EGI's simple-binary format (types 2 and 4). If you have developed a reader for these two formats, then only small modifications will be necessary to accommodate epoch marked's codes `epoc` and `tim0`.

Epoch Breaks

- Many programs indicate epoch breaks graphically with a vertical line.
- Segmentation routines should *reject* segments where the event on which the segmentation is temporally based is too close to the epoch break with respect to the segment size (for example, if an event is 50 ms after an epoch break, and the Segmentation specification indicates that the segment includes 100 ms before the event).

Tim0 Indicators

- If *no* `tim0` events are present, then time in the new epoch continues where the previous epoch left off.
- If a file contains both `epoc` and `tim0` events, then each epoch should have exactly one `tim0` event. This event may appear simultaneously with the `epoc` event, or on any other sample in the epoch. Other events may appear at any time in the epoch (even simultaneously with other events), including the `epoc` event and the `tim0` event. Other events may appear between the `epoc` event and the `tim0` event.

***Caution!:** If no `tim0` events occur in the epoch, assume that the first sample (simultaneous with the first `epoc` event) is time 0.*

***Caution!:** If an epoch has more than one `tim0` event, ignore all but the first `tim0` event.*

Available Resources

Contact EGI Technical Support (Appendix A) for resources for the epoch-marked simple-binary file format, including documentation and example files.

Export Instructions

The instructions for exporting EEG data to epoch-marked simple binary vary, depending on whether the data are unprocessed or ERP. In either case, exporting data is available only through the File Export tool of the Waveform Tools.

Exporting Unprocessed Data

Note: The following discussion assumes a familiarity with Net Station segmentation and the structure of a simple standard/target experiment. For more information about those topics, see the Net Station Viewer and Waveform Tools Tutorial.

This section discusses the two-step process of exporting EEG data into the epoch-marked simple-binary format for segmenting in other programs. First, before exporting, use the Net Station segmentation markup feature to add all the events you might want to use in the other programs. Second, export the data.

Segmentation Markup

Net Station events contain *key lists* (mini-databases). In most programs, events have only names; they do not have key lists. Consequently, valuable information may be lost during export.

For example, in a simple standard/target experiment, all stimuli events might be named "stim" in Net Station. You cannot determine the distinction between standards and targets from the name of the event. You can determine it only from the key list in the events.

Net Station's segmentation markup feature can help you make the jump from key-list events to non-key-list events. Segmentation markup adds events corresponding to the key-list information to the recording to be exported.

1. Open the Waveform Tools window either by clicking the Waveform Tools button in the Net Station start-up window or by choosing **Tools > Waveform Tools**.

2. In the Specifications panel of the Waveform Tools window, select Segmentation from the Create pop-up menu to open a Segmentation specification editor.

3 In the specification editor, name the specification, select a destination, and set the tool-specific parameters. (See the *Net Station Viewer and Waveform Tools Tutorial* for more information about how to do this.)

4 In the specification editor, select the Mark Up File checkbox before clicking the close button in the top-left corner of the window and saving the changes, which closes the editor and reveals the Waveform Tools window (see Figure 3-1).

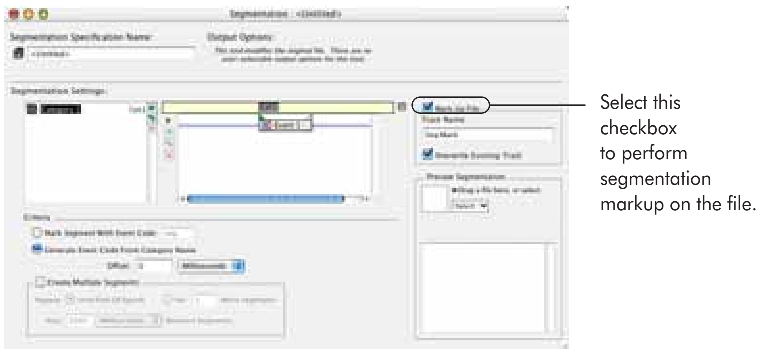

Select this checkbox to perform segmentation markup on the file.

Figure 3-1. Segmentation markup

5 In the Inputs pane of the Waveform Tools window, select your Net Station file by clicking the Add button, which displays the Add Inputs dialog.

6 Navigate to the file and add it by either selecting it and clicking the Add button or by double-clicking on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.

7 In the Specifications pane of the Waveform Tools window, choose Segmentation from the Filter List pop-up menu to make the Segmentation specification editor appear in the panel.

8 Highlight the specification and click Run.

Instead of segmenting the file, Net Station will add new events without the key lists. So, in reference to the earlier example of all the standard and target stimuli being named “stim,” you would create a segmentation specification for marking up standard and target categories. After running the tool, you will have a file containing new events called “stnd” and “targ.”

Note: Do not use spaces in the names of the markup events. Spaces may cause unpredictable results when the data are exported and used by other programs.

The Export Process

1. In the Specifications pane of the Waveform Tools window, choose File Export from the Create pop-up menu to open the File Export specification editor.

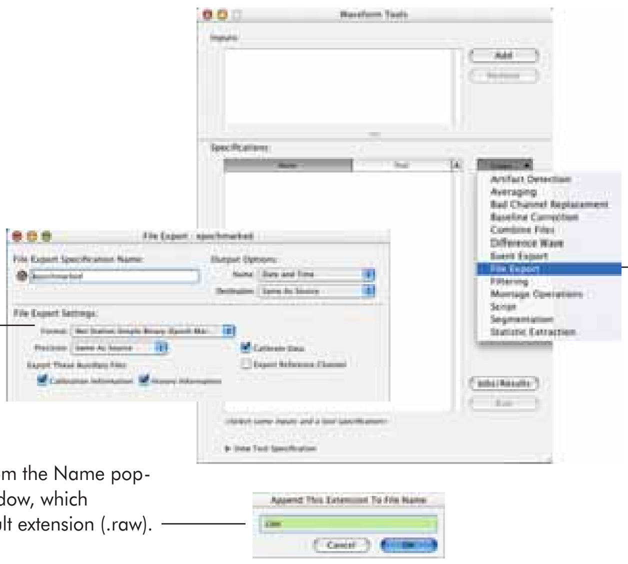
2. In the specification editor, type a name for the specification and select a location from the Destination pop-up menu.

3. In the File Export Settings pane, choose Simple Binary (Epoch Marked) from the Format pop-up menu and make sure the Export Reference Channel checkbox is unselected (because the data have not been rereferenced).

4. In the File Export Settings pane, choose the precision and whether to calibrate the data (Figure 3-2).
   - If you choose integer precision (for a faster export and a smaller resulting file, but a less accurate result), then select the Calibrate Data checkbox.
   - If you choose floating-point precision (for greater precision, but a slower export and a larger resulting file), then select the Calibrate Data checkbox.

5. In the File Export Settings pane, choose whether to export auxiliary text files containing calibration and history information. The files are identical to those produced using the history export feature and the File Exporter, both described in Chapter 9, "Non-EEG Data and the File Exporter.”

6. From the Output Name pop-up menu, choose to append the extension “.raw” (the default).



File Export Specification Name: epochmarked
Output Options:
Name: Date and Time
Destination: Same As Source
File Export Settings:
Format: Net Station Simple Binary (Epoch Ma...
Precision: Same As Source
✓ Calibrate Data
☐ Export Reference Channel
Export These Auxiliary Files:
✓ Calibration Information ✓ History Information

1 Select File Export from the Create pop-up menu to open the File Export specification editor.

2 Choose Net Station Simple Binary (Epoch Marked) from the Format pop-up menu and fill in the rest of the information, where applicable, for the File Export specification.

3 Choosing Append from the Name pop-up menu opens this window, which suggests using the default extension (.raw).

Figure 3-2. Exporting unprocessed data to epoch marked

7 Click the close button in the top-left corner of the specification editor and save the changes, which closes the editor and reveals the Waveform Tools window.

8 In the Inputs pane of the Waveform Tools window, click the Add button, which displays the Add Inputs dialog.

9 Navigate to the desired file and add it by either selecting it and clicking the Add button or by double-clicking on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.

10 In the Specifications pane of the Waveform Tools window, choose File Export from the Filter List pop-up menu to make the File Export specification appear in the pane.

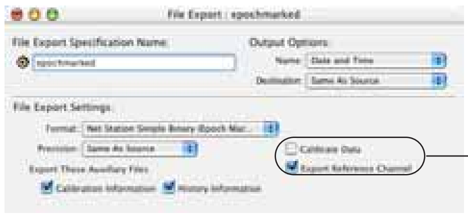
11 Highlight the specification and click Run.

Note: A simple standard/target experiment was used as an example in this section. By combining the full power of Net Station's segmentation with the segmentation markup feature, you can apply these procedures to more sophisticated experiments.

Exporting Averaged ERP Data

Note: The following explanation assumes that the data have been rereferenced during the ERP derivation process. For more information about rereferencing, see the Net Station Viewer and Waveform Tools Tutorial.

1. Open the Waveform Tools window by either clicking the Waveform Tools button in the Net Station start-up window or choosing **Tools > Waveform Tools**.
2. In the Specifications pane of the Waveform Tools window, select File Export from the Create pop-up menu to open a File Export specification editor.
3. In the specification editor, type a name for the specification and select a location from the Destination pop-up menu.
4. In the File Export Settings pane of the File Export specification editor, choose Simple Binary File (Epoch Marked) from the Format pop-up menu.
5. Select Floating Point from the Precision pop-up menu.
6. Make sure the Calibrate Data box is unselected (remember that the data have been calibrated during the ERP process).
7. Select the Export Reference Channel checkbox (see Figure 3-3).


File Export : epochmarked

File Export Specification Name: epochmarked

Output Options
Name: Date and Time
Destination: Same As Source

File Export Settings
Format: Net Station Simple Binary (Epoch Mar...
Precision: Same As Source
[ ] Calibrate Data
[✓] Export Reference Channel
Export These Auxiliary Files
[✓] Calibration Information [✓] History Information

Because deriving ERPs involves calibrating the data, do not select the Calibrate Data checkbox; do, however, select the Export Reference Channel checkbox.

Figure 3-3. When exporting ERP data, do not select the Calibrate Data checkbox

8. In the File Export Settings pane, choose whether to export auxiliary text files containing calibration and history information. The files are identical to those produced using the history export feature and the File Exporter, which are both described in Chapter 9, "Non-EEG Data and the File Exporter."
9. From the Output Name pop-up menu, choose to append the extension ".raw" (the default).

10. Click the close button at the top-left corner of the window and save the changes, which closes the editor and reveals the Waveform Tools window.
11. In the Inputs pane of the Waveform Tools window, click the Add button, which displays the Add Inputs dialog.
12. Navigate to the desired file and add it by either selecting it and clicking the Add button or by double-clicking on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.
13. In the Specifications pane of the Waveform Tools window, choose File Export from the Filter List pop-up menu to make the File Export specification appear in the pane.
14. Highlight the specification and click Run.

Your EEG data will be exported to a simple-binary file and the names of the conditions for each epoch will be saved in a text file, in the designated location.

chapter 4

SIMPLE BINARY

Introduction

This chapter describes EGI's simple-binary (or *raw*) format. Net Station enables you to store EEG data and events in a simple-binary data file, either unsegmented or segmented. Unsegmented source files and unsegmented Viewer window selections are saved to continuous simple-binary format files. Segmented source files are automatically saved as a segmented simple-binary file, one for each segment The structure of each segment includes a *miniheader* that identifies the segment's category and a *timestamp*, which is explained later in the chapter.

File Type Variations

The simple-binary format has six file type variations:

- continuous, unsegmented data saved with integer precision
- continuous, unsegmented data saved with floating-point single precision
- continuous, unsegmented data saved with floating-point double precision
- segmented or averaged data saved with integer precision
- segmented or averaged data saved with floating-point single precision
- segmented or averaged data saved with floating-point double precision

(See "Data Type Precision" on page 22 for a brief outline of the difference between integer and floating-point precision.)

Net Station writes into each file a single-digit version code that can be read from the first four bytes of the file (see Table 4-1). The code identifies how the EEG values are stored in the file (integers, single precision, or double precision) and whether the file is structured as continuous or segmented data. The header structures for continuous and segmented files are similar but do have some important differences, as shown by the simple-binary format tables beginning on page 71.

Interpreting File Types

To identify whether the EEG data contained in a simple-binary file are continuous or segmented and their precision, consult the single-digit version code in the file's first four bytes. Continuous data have even version numbers; segmented data have odd numbers:

- If the version number is 2, then the EEG data are continuous and have integer precision.
- If the version number is 4, then the EEG data are continuous and have floating-point single precision.
- If the version number is 6, then the EEG data are continuous and have floating-point double precision.
- If the version number is 3, then the EEG data are segmented and have integer precision.
- If the version number is 5, then the EEG data are segmented and have floating-point single precision.
- If the version number is 7, then the EEG data are segmented and have floating-point double precision.

To tell whether the data contain events, Net Station has a "number of unique event codes" field, beginning 34 bytes into the file in continuous files and at various offsets in segmented files:

- If the value in this field equals 0, then the EEG data contain no events.
- If the value in this field is greater than 0, then the EEG data contain events.

For the continuous simple-binary formats, the four-character Mac OS file type code is `UGLY`. For the segmented formats, the code is `eGLY`. (See "Macintosh OS File-Type Codes" on page 22 for more information.)

Table 4-1 encapsulates the preceding discussion for quick reference or comparison with other formats.

Table 4-1. Net Station simple-binary variants

Icon	File	Description	Version number	Mac OS type code
	Continuous, integer precision	An unsegmented EEG recording, with integer precision (2 bytes per value)	2	'UGLY'
	Continuous, floating-point single precision	An unsegmented EEG recording, with floating-point single precision (real, 4 bytes per value)	4	'UGLY'
	Continuous, floating-point double precision	An unsegmented EEG recording, with floating-point double precision (real, 8 bytes per value)	6	'UGLY'
	Segmented, integer precision	A segmented EEG recording, with integer precision (2 bytes per value)	3	'eGLY'
	Segmented, floating-point single precision	A segmented EEG recording, with floating-point single precision (real, 4 bytes per value)	5	'eGLY'
	Segmented, floating-point double precision	A segmented EEG recording, with floating-point double precision (real, 8 bytes per value)	7	'eGLY'

File Data Structures

This section describes the structures of a single sample record and of unsegmented and segmented simple-binary formats (with or without events). It also includes tables of the structures of unsegmented and segmented simple-binary formats. Each table displays the structure of the file header, followed by the structure of its data (EEG data, EEG data and events, or EEG segments), and indicates file offsets and how to calculate them, if necessary.

File Headers

Each simple-binary file has a header, a series of bytes having its starting point at the beginning of the file. The headers contain metadata, some values of which are essential to calculating or otherwise determining the file's relevant offsets. In the sections that follow, we refer to values that can be found in these headers by giving their offsets. The file structures of the different variations of Net Station simple binary are provided in tables on pages 71 and 72.

The length of the header will vary, but you can calculate it based on certain values found at fixed offsets in the file. The event and EEG data will alternate according to a pattern that can be determined using the header information. (This differs from exporting data *without* events, where the length of the header remains a fixed number of bytes.)

EEG Data

A *single sample record (SSR)* is a container for the data of a single sample. An SSR contains fields for EEG data and events.

The event states for a given sample occur *after* the EEG data for that sample, with each event state having the same precision as its associated EEG data. Hence, event states associated with *integer data* will be valued at 0 or 1; event states associated with *floating-point data* will be valued at 0.0 or 1.0 at single or double precision. If a given sample has no event states, then the EEG data for the next sample, rather than an event state, will follow the EEG data for that sample.

The EEG and events of a simple data file occur in the file as consecutive SSRs, one for each sample (see Table 4-2).

Table 4-2. Structure of a Single Sample Record
(Nc = number of channels; Ne = number of events)

Data type	Structure of record	Number of bytes per record
Integer	Consecutive values for channel 1 through channel Nc, followed by event states from state of event 1 through state of event Ne	(2 x Nc) + (2 x Ne)
Floating-point single precision	"	(4 x Nc) + (4 x Ne)
Floating-point double precision	"	(8 x Nc) + (8 x Ne)

Unsegmented Files

For unsegmented files (versions 2, 4, and 6), the EEG data and/or events part of the file consists of consecutive SSRs.

If events are included, then the end of the header is where the identities of the events are stored. These identities are in the form of four-character codes. The event states in each SSR are in the same order as these codes.

Segmented Files

For segmented files (versions 3, 5, and 7), the EEG segment part of the file consists of consecutive segments.

Each segment has a miniheader of its own that takes up its first six bytes. Following this miniheader are the SSRs; there are as many SSRs as there are samples (which can be read from the "number of samples" field).

Each miniheader contains a *category index* (short integer) and *timestamp* (long integer):

- The 1-based category index identifies the category with which the segment is associated. If the value in the index field is 1, then the segment is associated with the first category of the *categories array*. If it is 2, then it is associated with the second category, and so forth.

  The categories array, which starts at the header of these segmented files, holds the names of the categories to which the segments can belong as a series of Pascal strings (pstrings). A *pstring* is a one-dimensional character array with an added byte at the beginning that holds the length of the array.
- The four-byte timestamp holds the start time of the segment in milliseconds.

Net Station's segmented file format prohibits variable-length segments. As a result, you must first consult Table 4-3 to determine the segment size. Then, check the "number of segments" field in Table 4-5 to determine the offset to the beginning of each segment.

Table 4-3. Structure of a segment
(number of channels = Nc; number of events = Ne; number of samples = Ns)

Data type	Structure of segment	Number of bytes per segment
Integer	Index (short), followed by TimeStamp (long), followed by consecutive SSRs	2 + 4 + [(2 x Nc) + [2 x Ne)] x Ns
Single precision	"	2 + 4 + [(4 x Nc) + (4 x Ne)] x Ns
Double precision	"	2 + 4 + [(8 x Nc) + (8 x Ne)] x Ns

Table 4-4. Net Station simple-binary format (continuous data)

Header			
Size (bytes)	**File offset**	**Data type**	**Description of data**

4	0	long	Version number:
			if 2, EEG data are signed short integer
			if 4, EEG data are single-precision float
			if 6, EEG data are double-precision float

2	4	short	Recording time: year (4-digit value)
2	6	short	Recording time: month
2	8	short	Recording time: day
2	10	short	Recording time: hour
2	12	short	Recording time: minute
2	14	short	Recording time: second
4	16	long	Recording time: millisecond
2	20	short	Sampling rate (samples per second)
2	22	short	Number of channels
2	24	short	Board gain (1, 2, 4, or 8)
2	26	short	Number of conversion bits
2	28	short	Full-scale range of amplifier in μV
4	30	long	Number of samples
2	34	short	Number of unique event codes; equals 0 for files with no events and is greater than 0 for files with events
# of unique event codes x 4	36	contiguous 4-byte value	Sorted list of 4-character event codes; the number of codes is equal to the number of unique event codes
EEG data and/or events			
Size of SSR in bytes x # of samples	36 + (4 x # of unique event codes)	varies	Consecutive SSRs (see Table 4-2)

Table 4-5. Net Station segmented simple-binary format

Header			
Size (bytes)	**File offset**	**Data type**	**Description of data**

4	0	long	Version number:
			if 3, EEG data are signed short integer
			if 5, EEG data are single-precision float
			if 7, EEG data are double-precision float

2	4	short	Recording time: year (4-digit value)
2	6	short	Recording time: month
2	8	short	Recording time: day
2	10	short	Recording time: hour
2	12	short	Recording time: minute
2	14	short	Recording time: second
4	16	long	Recording time: millisecond
2	20	short	Sampling rate (samples per second)
2	22	short	Number of channels
2	24	short	Board gain (1, 2, 4, or 8)
2	26	short	Number of conversion bits
2	28	short	Full-scale range of amplifier in μV
2	30	short	Number of category names
Size in bytes of packed array of category names	32	pstring array	Packed array of category names; each name encoded as a pstring
2	varies	short	Number of segments
4	varies	long	Number of samples per segment
2	varies	short	Number of unique event codes; greater than 0 for files with events
# of unique event codes x 4	varies	contiguous 4-byte values	Sorted list of 4-character event codes; the number of codes is equal to the value in the number of unique event codes field
EEG data and/or events			
Size of a segment in bytes x # of segments	varies	varies	Consecutive segments (see Table 4-3)

Converting to Microvolts

Net Station gives you the option of saving EEG data in either A/D units (raw values that come from the amplifier A/D converter that are technically unitless) or microvolts. This option is represented by the Calibrate Data box (which converts data from A/D units to microvolts) in the Waveform Tools export. Data saved in A/D units are half the size as those saved in microvolts.

If the source file data are in the form of microvolts, Net Station cannot convert them to A/D units; the sole output choice is microvolts.

The only way to determine the units of measure of the stored EEG data is by looking at the bits field (at offset 26) and the range field (at offset 28) in the header:

- If the values in those two fields are 0, then the data are in microvolts.
- If the values in those two fields are not 0, then the data are in A/D units.

If desired, you can export the data as A/D units and later programmatically convert them to microvolts. There are two different conversion formulae, depending on whether the *channel gains* and *zeros* are available. (*Channel gain* is the amount a channel amplifies a known calibration signal; the gain is expressed as an amplitude. *Channel zero* is the amount a channel is offset from digital zero, in response to a grounded input.)

If channel gains and zeros are ***not*** available, then use the following formula to convert A/D units to microvolts:

$$\text{microvolt value} = (\text{range} \ / \ 2^{\text{bits}}) \times \text{A/D value}$$

where *bits* and *range* are read from the header of the file. This formula uses approximate values for the gains and zeros and will not produce as accurate a microvolt value as using the actual gains and zeros (for more information, see the "Gains and Zeros" section in Chapter 2, "Net Amps 200," of the *EGI System Technical Manual*).

If channel gains and zeros ***are*** available, then use the following formula:

$$\text{microvolt value} = [(\text{A/D value}) - (\text{channel zero})] \times (\text{calibration signal p–p value}) \ / \ (\text{channel gain})$$

where the current *calibration signal p–p value* is 400 microvolts.

In general, if the data do not need to be in A/D units, select the Calibrate Data checkbox for the greatest microvolt conversion accuracy.

Events

Following are some notes about exporting EEG data containing events to the simple-binary format:

- The following events lose their meaning outside of the context of Net Station and can be ignored in exported files: CELL, SESS, and TRSP.
- If an event occurs in consecutive samples, it indicates the *duration* of the event, not multiple occurrences of the event. Ignore all but the first one.
- Because of Net Station's segmentation markup feature, events can occur simultaneously.

Programming Tip

- When Net Station exports to simple binary, it saves all the events. However, if you export the simple-binary file back to Net Station, the events will not be visible unless you write a reader in another program (for example, BESA) to view the events.

Available Resources

Available resources for the simple-binary file format can be found at www.egi.com/documentation.html. These include Chapter 2, "Files," and Appendix A, "Files," in the legacy document *Net Station 2.0 Technical Manual* (alpha).

Export Instructions

To save Net Station EEG data into a simple-binary file, you have two choices:

- choose **File > Save Selection** or **File > Save A Copy As**
- use the File Export tool from the Waveform Tools window

The File Menu Export

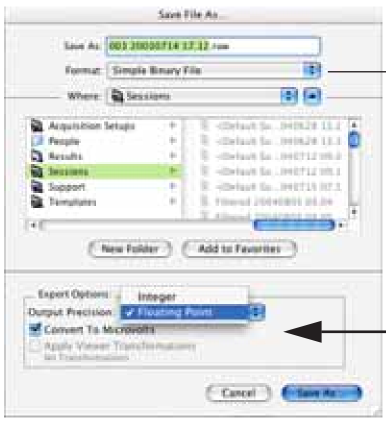
1. If necessary, open the Net Station data file.

2. Choose **File > Save Selection** or **File > Save A Copy As**, which causes a Save dialog to appear.

3. In the dialog, choose Simple Binary File or Simple Binary File (Ignore Events) from the Format pop-up menu.

4. Select the units (analog-to-digital [A/D] or microvolt [μV]) and precision (integer or floating point) of the output file (Figure 4-1).

   Note: Integer precision provides a faster export with a smaller but less accurate resulting file. Floating point yields a more accurate but larger file, with a slower export.



After choosing Simple Binary File from the Format pop-up menu, select the precision and units of the data.

Figure 4-1. Exporting to simple binary using the File menu

5 Type a file name in the name box and select a location from the Destination pop-up menu.

6 Click the Save As button.

Your EEG data will be saved in a simple-binary file in the designated location.

The Waveform Tools Export

1 Open the Waveform Tools window by either clicking the Waveform Tools button in the Net Station start-up window or choosing **Tools > Waveform Tools**.

2 In the Specifications pane of the Waveform Tools window, select File Export from the Create pop-up menu to open a File Export specification editor.

3 In the specification editor, type a name for the specification and select a location from the Destination pop-up menu.

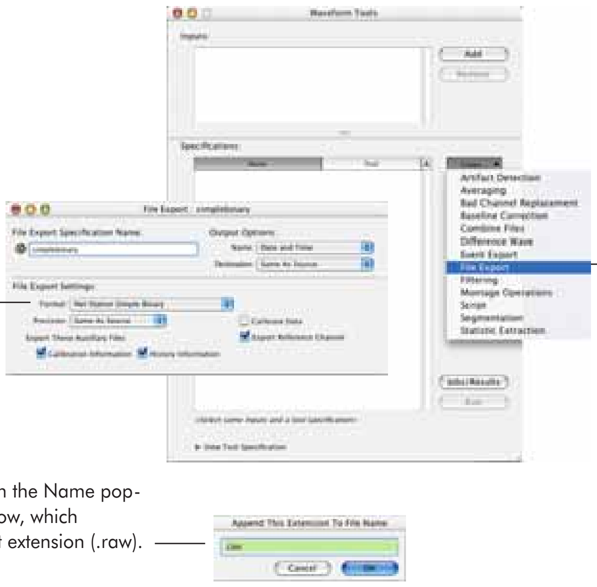
4 In the File Export Settings pane of the File Export specification editor, choose either Simple Binary File or Simple Binary File (Ignore Events) from the Format pop-up menu.

5 Select the output precision (same as source, integer, or floating point), choose whether to calibrate the data, and select whether to export the reference channel (see Figure 4-2):

- If the data have been rereferenced, select the Export Reference Channel checkbox. Otherwise, leave it unselected.
- If you choose same as source, then ignore the Calibrate Data checkbox.
- If you choose integer precision (for a faster export and a smaller resulting file, but a less accurate result), then leave the Calibrate Data checkbox unselected.
- If you choose floating-point precision (for greater precision, but a slower export and a larger resulting file), then select the Calibrate Data checkbox.

6 In the File Export Settings pane, choose whether to export auxiliary text files containing calibration and history information. The files are identical to those produced using the history export feature and the File Exporter, which are both described in Chapter 9, "Non-EEG Data and the File Exporter.”

7 From the Output Name pop-up menu, choose to append the extension ".raw" (the default).



1 Select File Export from the Create pop-up menu to open the File Export specification editor.

2 Choose Net Station Simple Binary [or Simple Binary (Ignore Events)] from the Format pop-up menu and fill in the rest of the information, where applicable, for the File Export specification.

3 Choosing Append from the Name pop-up menu opens this window, which suggests using the default extension (.raw).

Figure 4-2. Exporting to simple binary using Waveform Tools

8 Click the close button at the top-left corner of the window and save the changes, which closes the editor and reveals the Waveform Tools window.

9 In the Inputs pane of the Waveform Tools window, click the Add button, which displays the Add Inputs dialog.

10 Navigate to the desired file and add it by either selecting it and clicking the Add button or by double-clicking on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.

11 In the Specifications pane of the Waveform Tools window, choose File Export from the Filter List pop-up menu to make the File Export specification appear in the pane.

12 Highlight the specification and click Run.

Your EEG data will be saved in a simple-binary file in the designated location.

chapter 5

TAB-DELIMITED TEXT

Introduction

This chapter describes EGI’s tab-delimited text (also known as *ASCII*) file format. This format is a single, unsegmented block of EEG data, saved with one level of precision and one type of unit (microvolts). In other words, it contains no metadata, events, or header; the microvolt data are in the form of readable, tab-delimited text, and begin at the start of the file.

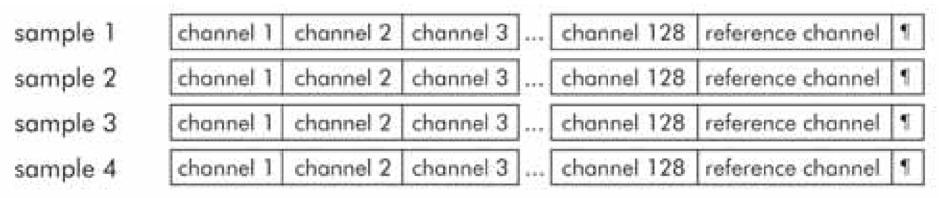
The advantages of this file format are that it is the most common format for text files in computers and on the Internet, and it can be read by any text editor. Net Station’s default is Microsoft Excel, which allows you to perform calculations on the exported data easily, such as determining zeros or baseline correction values.

The disadvantages are that the export process is time-consuming and it generates large files. Small files or selections of files are best for exporting to the tab-delimited text format.

In a tab-delimited text file, each line contains the EEG data for one point in time (one sample). A *sample*, as mentioned in the Preface, is actually a *collection of samples*, one for each recording channel, plus the reference channel.

The data are arranged as “channels by samples,” where the values in the file are read as a pure sequence. The values read first will run across successive channel values until all the channels have been read, and this same sequence will repeat consecutively for all the samples stored in the file.

So, the first line in a tab-delimited text file starts with the value for channel 1 and continues left to right to the last channel (sensor) used to record the original EEG. This value is followed by the value for the reference channel, and the end of the first sample is signified by a return character. The next line begins with the values from the second sample, starting with recording channel 1 (see Figure 5-1).



sample 1	channel 1	channel 2	channel 3	...	channel 128	reference channel	¶
sample 2	channel 1	channel 2	channel 3	...	channel 128	reference channel	¶
sample 3	channel 1	channel 2	channel 3	...	channel 128	reference channel	¶
sample 4	channel 1	channel 2	channel 3	...	channel 128	reference channel	¶

Figure 5-1. In the tab-delimited text format, data are arranged as “channels by samples”

File Type Variation

The tab-delimited text file format is the most basic data file format available. It has no header, no events, and no file type variations (see Table 5-1).

Table 5-1. Tab-delimited text format

Icon	File	File description	Mac OS type code
📄	Tab-delimited text	EEG microvolt data in tab-delimited text format.	‘TEXT’

File Data Structure

The data format of a Net Station tab-delimited text file is shown in Table 5-2.

Table 5-2. Net Station tab-delimited text data format

Header — none			
EEG data			
Size (bytes)	**File offset**	**Data type**	**Description of data**
Varies	0	µV data represented as tab-delimited text	EEG channels x samples array

The size of a tab-delimited text file depends on three factors:

- the number of text characters needed to represent each data value
- the number of EEG samples written to the file
- the number of channels used to record the original EEG

Net Station renders a microvolt value in this format using a variable number of numeric characters, a decimal point, and an optional minus (–) sign. The maximum number of numeric characters to the right of the decimal point is 15.

The number of samples stored in a tab-delimited text file is equivalent to the number of lines in the file. The number of channels is equivalent to the number of tab-delimited values on each line, plus the value of the reference value. So, the number of values per line will be 65, 129, or 257, and each line will contain the same quantity of tab-delimited values. The channel data will be listed from lowest sensor number to highest, and from left to right.

Converting to Microvolts

Net Station always exports data to this format in microvolts, so it is unnecessary to convert.

Available Resources

Available resources for the tab-delimited text file format can be found at www.egi.com/documentation.html. These include Chapter 2, “Files,” and Appendix A, “Files,” in the legacy document *Net Station 2.0 Technical Manual* (alpha).

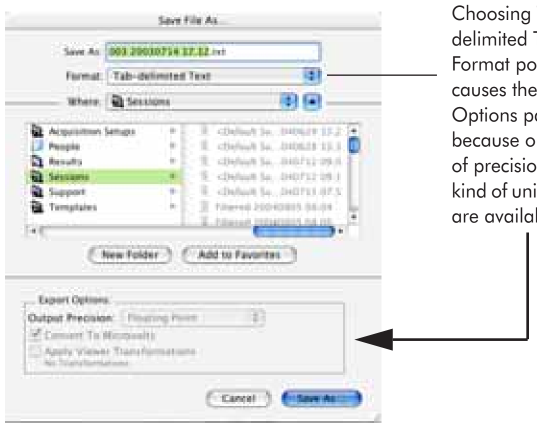
Export Instructions

To save Net Station EEG data into a tab-delimited file, you have two choices:

- choose **File > Save Selection** or **File > Save A Copy As**
- use the File Export tool from the Waveform Tools window

The File Menu Export

1. If necessary, open the Net Station data file.
2. Choose **File > Save Selection** or **File > Save A Copy As**, which causes a Save dialog to appear.
3. In the dialog, choose Tab-delimited Text from the Format pop-up menu (Figure 5-2).
4. The Export Options panel is dimmed because tab-delimited text offers no options; the data can be saved with only one level of precision and in microvolts.



Choosing Tab-delimited Text from the Format pop-up menu causes the Export Options panel to dim because only one level of precision and one kind of unit (microvolts) are available.

Figure 5-2. Exporting to tab-delimited text using the File menu

5 Type a file name in the name box and select a location from the Destination pop-up menu.

6 Click the Save As button.

Your EEG data will be saved in a tab-delimited text file in the designated location.

The Waveform Tools Export

1 Open the Waveform Tools window by either clicking the Waveform Tools button in the Net Station start-up window or choosing **Tools > Waveform Tools**.

2 In the Specifications pane of the Waveform Tools window, select File Export from the Create pop-up menu to open a File Export specification editor.

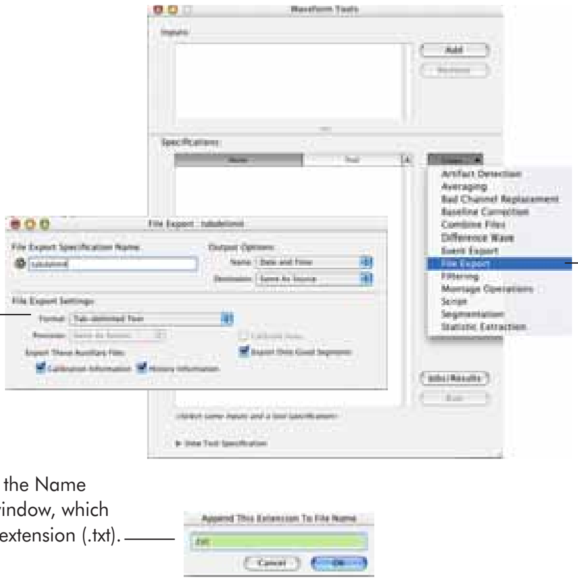
3 In the specification editor, type a name for the specification and select a location from the Destination pop-up menu.

4 In the File Export Settings pane of the File Export specification editor, choose Tab-delimited Text from the Format pop-up menu.

5 From the Output Name pop-up menu, choose to append the extension "txt" (the default).

6 Decide whether to select the Export Only Good Segments checkbox (Figure 5-3).

7 In the File Export Settings pane, choose whether to export auxiliary text files containing calibration and history information. The files are identical to those produced using the history export feature and the File Exporter, which are both described in Chapter 9, "Non-EEG Data and the File Exporter."



1 Select File Export from the Create pop-up menu to open the File Export specification editor.

2 Choose Tab-delimited Text from the Format pop-up menu and decide whether to export only good segments.

3 Choosing Append from the Name pop-up menu opens this window, which suggests using the default extension (.txt).

Figure 5-3. Exporting to tab-delimited text using the Waveform Tools

8 Click the close button at the top-left corner of the window and save the changes, which closes the editor and reveals the Waveform Tools window.

9 In the Inputs pane of the Waveform Tools window, click the Add button, which displays the Add Inputs dialog.

10 Navigate to the desired file and add it by either selecting it and clicking the Add button or by double-clicking on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.

11 In the Specifications panel of the Waveform Tools window, choose File Export from the Filter List pop-pane menu to make the File Export specification appear in the pane.

12 Highlight the specification and click Run.

Your EEG data will be saved in a tab-delimited text file in the designated location.

chapter 6

EGIS

Introduction

This chapter describes the EGIS file format. EGIS is also the name of a discontinued experiment control package. This chapter will discuss the file format only.

As an EGI legacy format, EGIS is typically used by owners of early versions of Net Station. Although exporting to epoch-marked simple binary is best, if you have many files in the EGIS format or simply prefer EGIS, the information in this chapter will prove useful.

In this chapter, you will find descriptions of the organizational structure of each type of data header and conventions for standard use of these headers. Each field is codified as containing a certain sort of information. This codified information consists of important descriptors common to most ERP experiments.

A typical electrophysiological experiment is composed of several kinds of trials that correspond to the different conditions being tested. In EGIS, trial conditions are defined as *cells*.

File Type Variations

Unlike most of the other formats discussed in this manual, EGIS contains only segmented data files. Thus, you must segment Net Station data before exporting them to EGIS.

For example, you can use a set of edited session files as input to one or more *averaged* files containing averages for each subject. You can use an averaged file as input for creating a grand-averaged file that contains an average of subject averages.

The EGIS format has three main file type variations:

- session data saved with integer precision and in A/D units
- averaged data saved with integer precision and in scaled microvolts
- grand-averaged data saved with integer precision and in scaled microvolts

As indicated in the preceding list, EGIS session data can be saved only in A/D units, and EGIS averaged data in microvolts. Also, only integer precision is available for the EGIS file format. (See “Data Type Precision” on page 22 for a brief outline of the difference between integer and floating-point single precision.)

Table 6-1 encapsulates the preceding discussion for quick reference or comparison with other formats.

Table 6-1. EGIS formats

Icon	File	Description	Mac OS type code
EGIS	EGIS session	Legacy format of the EGIS application. The EGIS session file contains raw data from a single subject run. Net Station can read and write this format. Do not confuse this with the Net Station Session file.	‘EGIS’
EGIA	EGIS average	Legacy format of the EGIS application. The EGIS averaged file contains averaged data, with one observation per input session file and direct correspondence between the session and averaged-file cells.	‘EGIA’
EGIA	EGIS grand average	Legacy format of the EGIS application. The EGIS grand-averaged file is the same as averaged file, but with one observation per cell (i.e., with all the session averages in each cell of an averaged file combined into one grand average).	‘EGIA’

File Data Structure

Overview of File Structure

A file contains data organized into one or more analysis cells. In a session file, the observations in a cell are *trials*.

Cells in a session file are based on different levels in a factorial design, which can include components such as stimulus parameters or the behavioral task being performed by the subject. In an averaged file, the observations in a cell are subject averages. You can derive averaged data directly from a corresponding cell in a session file or create them from combinations of input cells.

The information in both session and averaged files is divided into two parts: a ***header*** and a ***data region***. The header can be further divided into three parts: *file information*, *cell information*, and *miscellaneous information*.

File Header

The first part of the header, the file information, appears once at the very beginning of the file. It can be broken down into three subsections, as shown in Table 6-2. Each subsection will be described in further detail on the following pages.

Table 6-2. File information

Size (bytes)	Type of information
92	File information (see Table 6-5)
26	General information (see Table 6-6)
12 + (2 * # of cells) + (4 * # of channels)	Data description (see Table 6-7)
Total size of file information = 130 + (2 * # of cells) + (4 * # of channels)	

After the file information, there is one block of cell information for every cell in the file. Thus, the structure shown in Table 6-3 can be repeated from 1 to 256 times in an EGIS file.

The number of trial specifics in a cell information block is often the same for every cell in the file. However, this does not always occur, so the size of the cell information may vary for different cells.

Table 6-3. Cell information

Size (bytes)	Type of information
90	Cell header
2 * total number of trial specifics (see note following this table)	Trial specifics (subject specifics in averaged file)
Total size of cell information = 90 + (2 * # of trial specifics)	

Note: The total number of trial specifics is the number of specifics per trial x *the number of trials (observations) in the current cell. In an averaged file, trial specifics are replaced by subject specifics, which are constant sizes.*

After the cell information, the miscellaneous information section (Table 6-4) contains experimenter comments about the session, a text section for inserting special instructions for analysis programs, and an area of padding designed to make the size of the file header evenly divisible by 512. The size of this area is highly variable and can even be 0 (this area of the EGIS file may be completely missing).

Table 6-4. Miscellaneous information

Size (bytes)	Type of information
Variable	Experimenter comments
Variable	Other text
Variable	Padding

Session and Averaged Data

Raw data collected using the Net Amps 200 amplifier consist of signed integers in the range of –32,768 to +32,767, corresponding to 16-bit precision. Raw data collected with Net Amps units preceding the 200 Series are in the range of ±4,095, corresponding to 12-bit precision.

The data in both EGIS session and averaged files are ordered by channel, sample, observation, and cell, with channel varying fastest and cell varying slowest. In other words, the first data point in the data region is for channel 1, sample 1, observation 1, cell 1; the second data point is for channel 2, sample 1, observation 1, cell 1; and so on. The following sections describe both EGIS session and averaged-data files.

Header: File Information Section

File Description

The first five fields of the header, the file description (Table 6-5), are the same for both session and averaged files.

Table 6-5. EGIS file description format

Header (first part)				
Size (bytes)	**File offset**	**Description of data**	**C declaration**	**Pascal declaration**
4	0	Byte-order indicator. Must always be set to hexadecimal 01020304. If the byte order is swapped, then this number will become hexadecimal 02010403.	int byteOrderIndicator	BytOrd: longInt
2	4	Header version: if session files, then 3 if averaged file, then –1	short headerVersion	HdrVer: integer
2	6	Header length: 65,535 bytes	ushort headerLength	LHeader: unsignedWord
4	8	Data length: 4,294,967,295 bytes	int dataLength	LData: longInt
80	12	Experiment name: up to 80 characters long	char experimentName [80]	ExptNam: packed array[1..80] of char
Total size of file description = 92				

General Information

The second part of the file header is used quite differently for session files and averaged files. This is shown in Table 6-6, where field usage in session files is shown in plain text and usage in *averaged* files is shown in *italics*.

Table 6-6. EGIS general information format (session files = roman type; averaged files = italics)

Header (second part)				
Size (bytes)	**File offset**	**Description of data**	**C declaration**	**Pascal declaration**
6	92	Recording time: month, day, year (3 integers)	struct CollectionDateType {short month, short day, short year} *unused*	RunDate: array[1..3] of integer *unused*
6	98	Recording time: hour, minute, second (3 integers)	struct CollectionTimeType {short hour, short minute, short second} *unused*	RunTime: array[1..3] of integer *unused*
2	104	Subject ID *Last subject (observation)*	short subjectID *short lastDone*	SubjID: integer *LastDone: integer*

2	106	Subject handedness:
		if left, then 1
		if right, then 2
		if left-handed relative, add 10
		if no left-handed relative, 20
		Relatives are limited to birth parents and natural siblings. (e.g., a right-handed subject with a left-handed relative would be coded as 12.)
		Data bins per microvolt measured in scalebins
			short Handedness
			short scaleBins
				Handed: integer
				ScaleBins: integer

2	108	Subject sex:
		if male, then 1
		if female, then 2
		Maximum microvolts represented, estimated or measured
			short sex
			short scaleCal
				Sex: integer
				ScaleCal: integer

Table 6-6. EGIS general information format (session files = roman type; averaged files = italics)

Size (bytes)	File offset	Description of data	C declaration	Pascal declaration
2	110	Subject age *Duration of baseline in milliseconds*	short age *short baseDur*	Age: integer *BaseDur: integer*
2	112	Experimenter ID	short experimenterID *unused*	ExperID: integer *unused*
2	114	Edit version of scheme: used to code the flags identifying artifacts in individual epochs. Currently this value is 1	short editVersion *unused*	EdVer: integer *unused*

2	116	Calibration flag*: indicates which type, if any, of calibration information is included in the session file:
		if 0, no calibration done
		if 1, gain done
		if 2, zero done
		if 3, both gain and zero done
			short calibrationFlag
			unused
				CalFlag: integer
				unused

*The information coded in the calibration flag is used to scale the A/D units stored in an EGIS session file into microvolts. The formula uses a voltage value that is always equal to 400 and is as follows: microvolt value = (A/D value – channelZero) x (voltage value / channelGain) If you are using 1.0 system amplifiers, you can find more information about calibration in the *Calibration Technote* that accompanied your system. If you are using the System 200, consult Chapter 2, "Net Amps 200," in the *EGI System Technical Manual*. Note that each channel has an individual gain and zero value. A nominal gain value can be used with a loss in accuracy. Amplifier versions have different nominal gain values; the number needs to be determined for each amplifier.				
Total size of general information = 26				
Note: Total size of file up to this point is 118 (i.e., 92 + 26)				

Data Description

The data description section (Table 6-7) of the header is nearly the same for both session and averaged files. The only difference is that averaged files do not use the calibration arrays; this is indicated in the table by *italics*.

Table 6-7. EGIS data fields

Header (third part)				
Size (bytes)	**File offset**	**Description of data**	**C declaration**	**Pascal declaration**
2	118	Number of cells: can range from 1 to 256	short numOfCells	NCells: integer
2	120	Maximum number of channels: 129	short numOfChannels	NChan: integer
2	122	Comment length (refer to the miscellaneous information description on page 98)	ushort commentLen	LComment: integer
2	124	Text length (refer to the miscellaneous information description on page 98)	ushort textLen	LText: integer
2	126	Padding length (refer to the miscellaneous information description on page 98)	ushort paddingLen	LPad: integer
2	128	Board gain (1, 2, 4, or 8)	short boardGain	BrdGain: integer
2 to 512	130	Cell header length array	ushort *cellHeaderLengths	LCellHdr: array[1..NCells] of unsignedWord
2 to 258	varies	Zero array	short *zeroCalibrations *unused for averaged files*	CalZero: array[1..NChan] of integer *unused for averaged files*
2 to 258	varies	Gain array	short *gainCalibrations *unused for averaged files*	CalGain: array[1..NChan] of integer *unused for averaged files*
Total size of data fields = 12 + (2 * NCells) + (4 * NChan)				
Note: Total size of file up to this point = 130 + (2 * NCells) + (4 * NChan)				

Header: Cell Information Section

Cell Headers

After the data description in the session file header, there follows one repetition of the structure shown in Table 6-8 for each cell (NCells). This information specifies the data organization and provides for the accumulation of dependent measures (for example, response time) and condition codes (for example, target present/absent).

The information is similar for both session and averaged files, except for the "number of observations in cell" section (see the italics in the table for information for averaged files) and the trial specs (see "Session File Trial Specifics" on page 96 and "Averaged-File Subject Specifics Array" on page 97 for details).

Table 6-8. EGIS cell header

Size (bytes)	File offset	Description of data	C declaration	Pascal declaration
2	0	Cell ID: can be any integer but is usually set to the cell number	short cellID	CellID: integer
80	2	Cell name: up to 80 characters long	char cellName[80]	CellName: packed array[1..80] of char
2	82	Number of observations in cell: for session files, this is the number of trials *for averaged files, this is the number of subjects*	short numOfObs	Ntrials: integer NSubjects: integer
2	84	Number of samples per observation: must be a multiple of the next power of 2 larger than the sampling rate (e.g., if the sampling rate is 125 samples per second, the number of samples per observation must be 128 or 256 or 384, etc.)	short numOfSamplePointsPerOb	Npoints: integer
2	86	Sampling rate (50, 100, 125, 200, 250, 500, or 1,000 samples per second)	short numOfSampsPerSecond	SampRate: integer
2	88	Length of trial specifics (must be an even number greater than or equal to 2)	ushort lengthOfObEntry	LSpec: integer

Table 6-8. EGIS cell header

Size (bytes)	File offset	Description of data	C declaration	Pascal declaration
varies	90	Trial specifics: can be thought of as an array of integers with LSpec/2 rows and Ntrials columns. The information is the same for each observation in a cell, but it is different for session and averaged files.	see Table 6-9 and Table 6-10	
Total size of cell header = 90 + (2 * total trial specs)				
Note: Total trial specs = number of specs per trial x number of trials in cell				

Session File Trial Specifics

Session trial specifics (Table 6-9) contain information such as response number, reaction time, and trial number for each trial in the experiment. The exact information saved here depends on how the experiment program was coded.

EGI strongly encourages you to follow the coding conventions shown in Table 6-9 for the first six fields (that is, all but the last row in the table) of the trial specifics. In particular, the first field (the edit code field) must ***not*** be written into by the experiment program. You can devise your own conventions for any additional trial specifics.

Table 6-9. Session file trial specifics

Data	Description of data	Coding convention
Edit code	When the data are edited, this value changes from –1 to either 0 for good trials or to some combination of 1 = eye movement, 2 = motion artifact, 4 = EMG signal, and 8 = electrical noise (e.g., 9 = electrical noise + eye movement)	–1 = unedited; 0 = good; >0 = bad
Response number	Depending on the coding of the experiment, the response number may not be the same as the response key on the keypad (e.g., if the allowed responses are keys 1 and 4, they will be encoded as response numbers 1 and 2, respectively. Consult the zz unit for the exact meaning of the response number)	Left to right, 1 to N
Response evaluation	Whether the response is correct or incorrect	1 = correct; 2 = incorrect
Response time	The time it took for the subject to respond to the stimulus, in milliseconds	in milliseconds
Trial number	The position of the observations in the sequence of trials making up a session	–1 = unedited; 0 = good; >0 = bad

Table 6-9. Session file trial specifics

Data	Description of data	Coding convention
Stimulus number	Can be the position of the stimulus in the stimulus set, or anything you decides It can be used as an index, or it can be left unused.	Any short integer, programmer defined
Additional trial specs	Other trial specs you wish to add	Any number of short integers, programmer defined

Note: You can use the Event Export tool to create a tab-delimited text file of the trial specs. The first column in the file is the cell number, the second column is the observation number, the third column is the edit code, the fourth column is the response number, and so on. (See Chapter 9, "Non-EEG Data and the File Exporter," for more information.)

Averaged-File Subject Specifics Array

The first field of the averaged subject specifics list (Table 6-10) shows the number of observations from the session file that were included in the average. The remaining fields are an exact copy of the information contained in the general information section of the session file shown in Table 6-6; see that table for additional details.

There is one set of subject specs for each cell. The only number that varies is the first field. Averaged-file subject specs are always the same size: 28 bytes.

Table 6-10. Averaged-file subject specifics

C declaration	Pascal declaration
NAvg	NAvg: integer
struct CollectionDateType {short month, short day, short year}	RunDate: array[1..3] of integer
struct CollectionTimeType {short hour, short minute, short second}	RunTime: array[1..3] of integer
short subjectID	SubjID: integer
short Handedness	Handed: integer
short sex	Sex: integer
short age	Age: integer
short experimenterID	ExperID: integer
short editVersion	EdVer: integer
short calibrationFlag	CalFlag: integer

Header: Miscellaneous Section

Session File Flexible Area

The text section (Table 6-11) allows the encoding of variables not otherwise defined in the header description.

Table 6-11. Additional cell header variables

Variable	Coding	Type	Description
LComment	string	Comment	Experimenter comments and descriptions of unusual or noteworthy circumstances related to the session, etc.
LText	string	Text	Text (see the paragraph following the table)
LPad	0's	Pad	Zero fill to make header fit evenly in disk block structure

The text section consists of an unlimited (except by storage) number of text segments, which may appear in any order in the text section. Each text segment has the following format:

Label(Length){delimiter}String{delimiter} {String{delimiter} . . .}

Labels must be unique. A Header Text Labels log will be established to track labels and to define the structures to which they refer.

Length specifies the number of bytes representing the portion of the text beginning with the first character following the closing parenthesis and ending with the last character in this text entry (including the final delimiter). The strings and delimiters must extend as far as is indicated by the length specification.

The use of *delimiters* is recommended but optional, as indicated by braces (curly brackets). If delimiters are not used, it is assumed that special-purpose routines will be used to read text associated with that particular label. Such routines would need to be coded with knowledge of that text segment's structure.

Example format for channel names:

ChanNames(51)|Fp1|Fp2|F3|F4|C3|C4|P3|P4|O1|O2|F7|F8|T3|T4|T5|T6|

Averaged-File Flexible Area

These required segments specify the derivation of the averaged file in terms of

- input files (InputFiles)
- source cells in session/averaged files (CellNsource)
- the criteria applied for inclusion in the average (CellNCrit)

This specification enables a flexible transformation of cells during computation of the averaged data set. Simple collapsing or merging of previously established cells is allowed. The cells that serve as sources for the averaged file's output cells can all derive from a single source data set, or from multiple session files with identical structure and encoding conventions. Cells can also be constructed based on trial-specific information not represented in the cell organization of the session file.

Input file(s) should be listed in this format:

```
InputFiles(length)|File1|{File2|...}
```

Defining the inputs to an averaged-file cell requires one list of all source cells for each output cell. Each source-output specification is of the form:

```
CellNsource(length)|source cell 1|{source cell 2|. . .source cell n|}
```

Example:

```
Cell1source(21)|1|2|7|8|13|14|19|20|
```

This would result in cells 1, 2, 7, 8, 13, 14, 19, and 20 from the source file being combined into cell 1 of the averaged file. Additional specifications would define the sources for each of the Ncells output cells in the averaged file; input cells can be reused and combined in any manner.

Specification of criteria from source file trial/subject specifics requires an explicit acceptance criterion unless all observations should be used in computing the average. Each criterion specification is of the form:

```
CellNCrit(length)|source|index|relation|parameter|{source|index|relation|
parameter|...}
```

where *source* indicates to which source file cell the criterion applies (0 indicates all sources), *index* is the position of the datum of interest in the source file cell's trial/

subject specifics array, *parameter* is some limiting value of analytic interest, and *relation* defines the desired relationship between the datum and the limiting value.

Table 6-12 lists the values for relations.

Table 6-12. Cell criteria relations

Relations	Values
EQ	equals
NE	does not equal
LT	less than
GT	greater than
LE	less than or equal to
GE	greater than or equal to
B0	bit specified by parameters is off or 0
B1	bit specified by parameters is on or 1

Example:

```
Cell1Crit(31)|1|9|EQ|2|0|13|EQ|0|1|0|GT|35|
```

Assuming a subject averaged source file, this would define the contents of output cell 1 as averaged data from previously specified source cells:

- *if* the contents of datum 9 in the specifics array of cell 1 was equal to 2
- *and* for all selected sources the contents of datum 13 was equal to 1
- *and* for all selected sources the contents of datum 0 was greater than 35

For conjunctions, additional criteria may be specified for the same cell. Any input from a specified cell that meets all of one criteria will be included in the output cell.

Grand-Averaged Files

A grand-averaged file collapses data from all subjects into a single average. All applicable information is global to the grand average; subject specific information is not retained. The header field labeled *LastDone* contains 1 when the grand average has been computed. The number of subject averages included in the grand average is contained in NAvg, stored at index 0 of the cell particulars. This is the only datum contained in the particulars of each cell.

EEG Data

In the data array, the actual EEG (or ERP in an averaged file) is stored as 16-bit signed integers. The data are stored as a flattened array. The dimensions are cells, trials, and samples. The number of trials can vary by cell. The number of cells in a file will also vary. The array is arranged using the following formula:

1 to NCells by 1 to NTrial by 1 to NSamples

In pseudo-code, use embedded loops to access the data:

```
for n := 1 to NCells
      for m := 1 to NTrials (varies by cell)
            for p := 1 to NSamples
                  for g := 1 to NChannels
```

Converting to Microvolts

The following formula converts A/D units in the EGIS file format to microvolts:

microvolt value = (bits – channel zero) x [(400 μV x board gain) / channel gain]

where zero = 0 and gain = 16,000 (gains and zeros are technically unitless).

Programming Tip

- Some differences in amplitude may occur when you export to the EGIS file format. Net Station uncalibrates and rounds the data to the nearest tenth of a microvolt and then recalibrates them for display in the EGIS format.

Available Resources

An available resource for the EGIS file format is the *EGIS 2.5 Programmer's Manual*, posted at www.egi.com/documentation.html.

Export Instructions

To save Net Station EEG data into an EGIS format file, you have two choices:

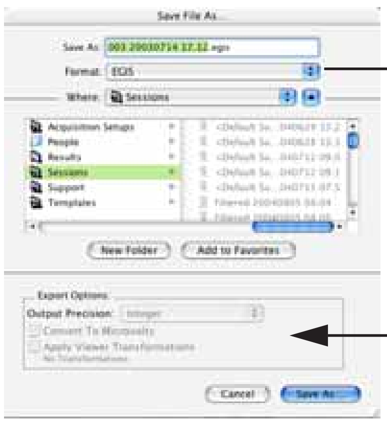
- choose **File > Save Selection** or **File > Save A Copy As**
- use the File Export tool from the Waveform Tools window

The File Menu Export

1. If necessary, open the Net Station data file.

2. Choose **File > Save Selection** or **File > Save A Copy As**, which causes a Save dialog to appear.

3. In the dialog, type a file name in the name box and select a location from the Destination pop-up menu.



Save As: 003 20030714 17.12.egis

Format: EGIS

Where: Sessions

Output Precision: Integer

Choosing EGIS from the Format pop-up menu causes the Export Options panel to dim because only integer precision and one kind of unit per file type (A/D units for session, microvolts for averaged) are possible.

Figure 6-1. Exporting to the EGIS format using the File menu

4 Choose EGIS from the Format pop-up menu, which dims all the export options because the only choices are A/D units for session data, microvolts for averaged data, and integer precision for both (see Figure 6-1).

Note: Calibrated files will be uncalibrated using nominal gain values.

5 Click the Save As button.

Your EEG data will be saved in an EGIS format file in the designated location.

The Waveform Tools Export

1 Open the Waveform Tools window by either clicking the Waveform Tools button in the Net Station start-up window or choosing **Tools > Waveform Tools**.

2 In the Specifications pane of the Waveform Tools window, select File Export from the Create pop-up menu to open a File Export specification editor.

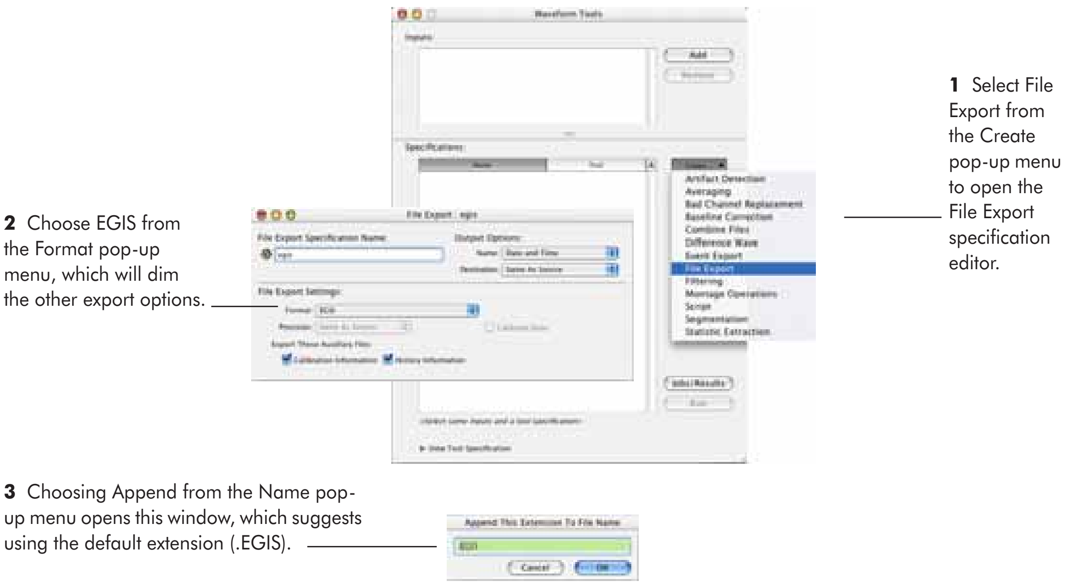
3 In the specification editor, type a name for the specification and select a location from the Destination pop-up menu.

4 In the File Export Settings pane of the File Export specification editor, choose EGIS from the Format pop-up menu, which dims all the export options because

the only choices are A/D units for session data, microvolts for averaged data, and integer precision for both (see Figure 6-2).

5 From the Output Name pop-up menu, choose to append the extension ".EGIS" (the default).

6 In the File Export Settings pane, choose whether to export auxiliary text files containing calibration and history information. The files are identical to those produced using the history export feature and the File Exporter, which are both described in Chapter 9, "Non-EEG Data and the File Exporter."



1 Select File Export from the Create pop-up menu to open the File Export specification editor.

2 Choose EGIS from the Format pop-up menu, which will dim the other export options.

3 Choosing Append from the Name pop-up menu opens this window, which suggests using the default extension (.EGIS).

Figure 6-2. Exporting to the EGIS format using the Waveform Tools

7 Click the close button at the top-left corner of the window and save the changes, which closes the editor and reveals the Waveform Tools window.

8 In the Inputs pane of the Waveform Tools window, click the Add button, which displays the Add Inputs dialog.

9 Navigate to the desired file and add it by either selecting it and clicking the Add button or by double-clicking on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.

10 In the Specifications pane of the Waveform Tools window, choose File Export from the Filter List pop-up menu to make the File Export specification appear in the pane.

11 Highlight the specification and click Run.

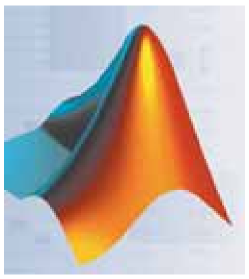
Your EEG data will be saved in an EGIS format file in the designated location.

chapter 7

MATLAB

Introduction

This chapter describes the MATLAB MAT-file format. MATLAB language is well known among designers of predictive mathematical models and application-specific algorithms. This is because MATLAB contains built-in algorithms that provide access to elementary and advanced numeric computing. MATLAB uses the same language for both structured and object-oriented programming.


courtesy of The MathWorks, Inc.

You can export continuous, segmented, or combination Net Station data to the MAT-file format. Currently, the only output options for data exported from Net Station to the MAT-file format are microvolt data, no events, and floating-point, double-precision accuracy.

Net Station offers three export options:

- individual arrays
- cell array
- 3D array

The individual arrays contain epochs, each of which has a variable called <EGIdataN>, which is a matrix of the data. For segmented or combination data, each *segment* has a variable called <Category><SegmentN> (also a data matrix). The data are arranged as "channels by samples" (see Chapter 5, "Tab-Delimited Text").

The cell array is structured as 2 x nSegs. One column contains the data, and the other contains the segment name. Each of the elements in the first column of the cell array is the matrix nChan x nSamples.

The 3D array is structured as channel x time x segments, for each category. This option is useful if you wish to preserve the distinction between the categories and the segments.

File Type Variation

Net Station exports to MAT-file format, Version 5 (Table 7-1).

Table 7-1. MATLAB file type

Icon	File	File description	Mac OS type code
📄	MATLAB MAT-file	EEG microvolt data in MATLAB data array format. Default Net Station file extension is '.MAT'.	'MATW'

File Data Structures

The data format of the MAT-file consists of a 128-byte header followed by one or more data elements. Each data element is composed of an 8-byte tag followed by the data in the element. The tag specifies the number of bytes in the data element and how these bytes should be interpreted (that is, as floating point, double-precision values).

For details about the MAT-file format, see the *MAT-File Format, Version 5*, available at www.mathworks.com.

Converting to Microvolts

Net Station always exports data to this format in microvolts, so it is unnecessary to convert.

Events

The current format of MAT-file exported by Net Station does not include events.

Programming Tip

- When you load a MAT-file into MATLAB, it automatically enters the MATLAB Workspace where other MAT-files may be stored. A MAT-file with the same name as another file in the Workspace will overwrite that file. Because such an incident may not be flagged by MATLAB, you may want to rename MAT-files, once you import them into MATLAB.

Available Resources

Available resources for the MAT-file format can be found at the developer's website at www.mathworks.com.

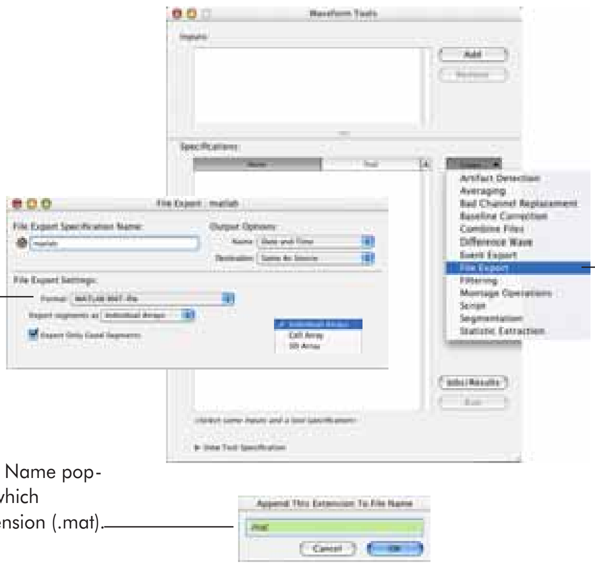
Export Instructions

To save Net Station EEG data into the MAT-file format, you must use the File Export tool from the Waveform Tools window.

The Waveform Tools Export

1 Open the Waveform Tools window by either clicking the Waveform Tools button in the Net Station start-up window or choosing **Tools > Waveform Tools**.

2 In the Specifications pane of the Waveform Tools window, select File Export from the Create pop-up menu to open a File Export specification editor.



1 Select File Export from the Create pop-up menu to open the File Export specification editor.

2 Choose MATLAB MAT-file from the Format pop-up menu, choose export array option, and decide whether to select the Export Only Good Segments checkbox to the right.

3 Choosing Append from the Name pop-up menu opens this window, which suggests using the default extension (.mat).

Figure 7-1. Exporting to the MATLAB MAT-file format using the Waveform Tools

3 In the specification editor, type a name for the specification and select a location from the Destination pop-up menu (see Figure 7-1).

4 In the File Export Settings pane of the File Export specification editor, choose MATLAB MAT-file from the Format pop-up menu.

5 Decide whether to select the Export Only good Segments checkbox, which excludes segments that were marked “bad” by Net Station’s Artifact Detection tool.

6 From the Output Name pop-up menu, choose to append the extension “mat” (the default)

7 Click the close button at the top-left corner of the window and save the changes, which closes the editor and reveals the Waveform Tools window.

8 In the Inputs pane of the Waveform Tools window, click the Add button, which displays the Add Inputs dialog.

9 Navigate to the desired file and add it by either selecting it and clicking the Add button or by double-click on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.

10 In the Specifications pane of the Waveform Tools window, choose File Export from the Filter List pop-up menu to make the File Export specification appear in the pane.

11 Highlight the specification and click Run.

Your EEG data will be saved in the MAT-file format in the designated location.

chapter 8

PERSYST

Introduction

This chapter describes the Persyst file format as exported by Net Station. This format is for users who want to export Net Station data into Persyst programs, which are designed for clinical digital EEG applications such as epilepsy monitoring, brain mapping, and neurosurgical applications.

www.eeg-persyst.com

File Data Structures

When you export to Persyst, Net Station generates two files:

- a layout file (with a ".lay" extension)
- a data file (with a ".dat" extension)

The layout file is a text file that contains header information (events) necessary to interpret the data file. The header information is divided into sections that describe the file type, channel labels, sheets, events, patient's personal and medical information, and epochs. This file is described in the next section.

The .dat file contains the actual EEG data. This file format is described in "The Data File" on page 118.

The Layout File

The Persyst layout file contains the header information that allows Persyst (or you) to interpret the data file. The layout file also allows you to edit in Persyst a recording (marking spikes or adding comments) and save the edits without having to modify the data file.

The layout file contains the header information, in this order: file information, channel map, sheets, comments, patient information, and sample times.

Note: This chapter describes the format as exported by Net Station. When you open the file in Persyst, Persyst may add sections or modify the file in other ways.

File Information

This section of the header contains information about the file type and will vary depending on the type of file opened.

Following is an example of a file information section from a Net Station Session file. In this example:

- `File` is the <filename>.dat.
- `FileType` is always interleaved.
- `SamplingRate` is the number of samples per second.
- `HeaderLength` is always 0.
- `Calibration` is always 1 and is used to convert the data to microvolts.
- `WaveformCount` is the number of channels.
- `DataType` is always 0.

```
[ FileInfo]
File=00.a.ar.bl.dat
FileType=Interleaved
SamplingRate=250
HeaderLength=0
Calibration=1.00000000
WaveformCount=129
DataType=0
```

Channel Map

This section of the header contains information about the channel map, which retains the recording system's channel labels with the exported data. Following is an example of exported data from a 128-channel Net Station recording (the information has been placed into four columns on this page, to save space; in the file itself, the information is listed in a single column):

```
[ChannelMap]   33=33          66=66          99=99
1=1            34=34          67=67          100=100
2=2            35=35          68=68          101=101
3=3            36=36          69=69          102=102
4=4            37=37          70=70          103=103
5=5            38=38          71=71          104=104
6=6            39=39          72=72          105=105
7=7            40=40          73=73          106=106
8=8            41=41          74=74          107=107
9=9            42=42          75=75          108=108
10=10          43=43          76=76          109=109
11=11          44=44          77=77          110=110
12=12          45=45          78=78          111=111
13=13          46=46          79=79          112=112
14=14          47=47          80=80          113=113
15=15          48=48          81=81          114=114
16=16          49=49          82=82          115=115
17=17          50=50          83=83          116=116
18=18          51=51          84=84          117=117
19=19          52=52          85=85          118=118
20=20          53=53          86=86          119=119
21=21          54=54          87=87          120=120
22=22          55=55          88=88          121=121
23=23          56=56          89=89          122=122
24=24          57=57          90=90          123=123
25=25          58=58          91=91          124=124
26=26          59=59          92=92          125=125
27=27          60=60          93=93          126=126
28=28          61=61          94=94          127=127
29=29          62=62          95=95          128=128
30=30          63=63          96=96          Cz=129
31=31          64=64          97=97
32=32          65=65          98=98
```

Sheets

This section of the header contains information about sheets, which refer to viewing modes in a Persyst source file. Net Station does not contain this kind of information but must export placeholders in this field for integration with Persyst. Net Station exports the following for this section:

```
[Sheets]
Patient=
Review=
```

Comments

This section of the header lists events, which are called “comments” in the Persyst format. The fields of each comment line are:

- time (in seconds)
- duration (in seconds)
- 0 (unused)
- 100 (comment color)
- comment name

```
[Comments]
0.201,0.1,0,100,stim
1.403,0.1,0,100,stim
2.603,0.1,0,100,stim
3.800,0.1,0,100,stim
5.1,0.1,0,100,stim
6.202,0.1,0,100,stim
7.401,0.1,0,100,stim
8.601,0.1,0,100,stim
9.802,0.1,0,100,stim
11.1,0.1,0,100,stim
12.201,0.1,0,100,stim
13.401,0.1,0,100,stim
14.602,0.1,0,100,stim
15.801,0.1,0,100,stim
17.3,0.1,0,100,stim
18.203,0.1,0,100,stim
19.400,0.1,0,100,stim
20.601,0.1,0,100,stim
```

So, in the preceding example, the first event occurred 0.201 seconds into the recording, lasted for 0.1 seconds, and was a stimulus event.

You can also add comments to a layout file. Following is an example of additional comments that help synchronize the EEG data with associated video recordings. The following example shows a comment with information on delta from start and Iframe offset (in milliseconds):

```
0, 0, 0, 100, MPEG File Start: clip.mpg DeltaStartMs: 0.0
IframeOffsetMs: 0.0
```

Patient Information

This section of the header includes personal and medical information about the patient, and allows you to open Persyst layout files from the patient database.

```
[ Patient]
First=Jane
MI=M.
Last=Doe
Sex=F
Hand=R
BirthDate=10/22/66
ID=123-45-6789
TestDate=07/15/03
TestTime=19:58:20
Physician=Yuen Chan, MD
Technician=Bob Smith
Medications=
History=
Comments1=
Comments2=
```

Sample Times

This section of the header lists sample times, with each line representing a different epoch. The number preceding the equals symbol represents the number of the first sample in that segment, and the number following it represents the start time of the segment, in seconds (after midnight). So, in the example that follows, the first line indicates that the first segment occurred at the first sample (`0`) and at 7:58 p.m. (`71900.000`)

(The example information has been placed into two columns on this page, to save space; in the file itself, the information is listed in a single column):

```
[SampleTimes]
0=71900.000
300=71901.000
600=71902.000
900=71903.000
1200=71904.000
1500=71906.000
1800=71907.000
2100=71908.000
2400=71909.000
2700=71910.000
3000=71912.000
3300=71913.000
3600=71914.000
3900=71915.000
4200=71916.000
4500=71918.000
4800=71919.000
5100=71920.000
```

The Data File

The Persyst data file contains data from the raw recording (that is, the recorded montage). The recordings are saved as 16-bit signed binary data files, with a .dat extension, and in Intel byte-order (little-endian).

The data are arranged as “channels by samples” (see Chapter 5, "Tab-Delimited Text,” for more information). If you are using a Macintosh computer, you must perform byte-swapping to read the numeric data in this format.

Converting to Microvolts

Net Station always exports data to this format in microvolts, so it is unnecessary to convert. Thus, the `Calibration` field of the file information section of the .lay file is always 1.

Available Resources

Available resources for the Persyst file format can be found at the developer's website at www.eeg-persyst.com.

Export Instructions

To save Net Station EEG data into a Persyst file, you have two choices:

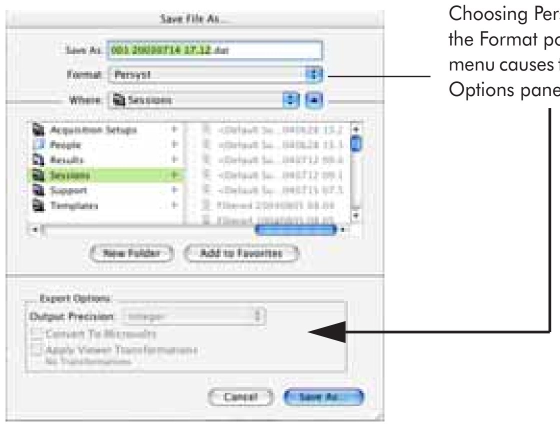
- choose **File > Save Selection** or **File > Save A Copy As**
- use the File Export fool from the Waveform Tools window

The File Menu Export

1 If necessary, open the Net Station data file.

2 Choose **File > Save Selection** or **File > Save A Copy As**, which causes a Save dialog to appear.

3 In the dialog, choose Persyst from the Format pop-up menu (see Figure 8-1).



Save As: 001 20030714 17.12.dat
Format: Persyst
Where: Sessions
Output Precision: Integer

Choosing Persyst from the Format pop-up menu causes the Export Options panel to dim.

Figure 8-1. Exporting to the Persyst file format using the File menu

4 Type in a file name in the name box and select a location from the Destination pop-up menu.

5 Click the Save As button.

Your EEG data will be saved in a Persyst format file in the designated location.

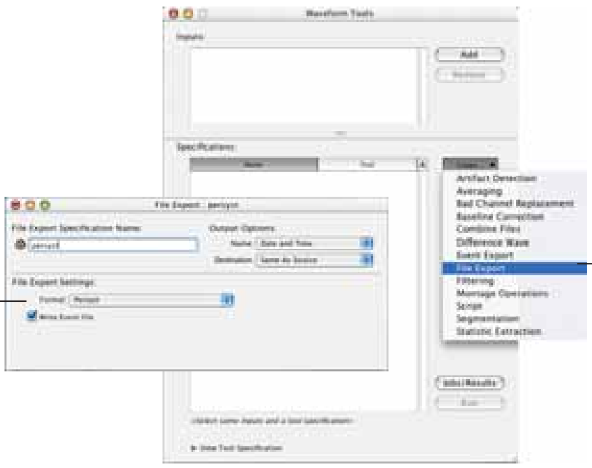
The Waveform Tools Export

1 Open the Waveform Tools window by either clicking the Waveform Tools button in the Net Station start-up window or choosing **Tools > Waveform Tools**.

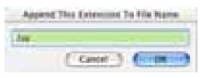
2 In the Specifications pane of the Waveform Tools window, select File Export from the Create pop-up menu to open a File Export specification editor.

3 In the specification editor, type a name for the specification and select a location from the Destination pop-up menu.

4 In the File Export Settings pane of the File Export specification editor, choose Persyst from the Format pop-up menu (see Figure 8-2).



File Export Specification Name: persyst
Name: Date and Time
Destination: Same As Source
Format: Persyst
✓ Write Event File


Append This Extension To File Name
Cancel
OK
3 Choosing Append from the Name pop-up menu opens this window, which suggests using the default extension (.lay).

.lay

Figure 8-2. Exporting to the Persyst file format using the Waveform Tools

5. From the Output Name pop-up menu, choose to append the extension ".lay" (the default).
6. Select the Write Event File checkbox.
7. Click the close button at the top-left corner of the window and save the changes, which closes the editor and reveals the Waveform Tools window.
8. In the Inputs pane of the Waveform Tools window, click the Add button, which displays the Add Inputs dialog.
9. Navigate to the desired file and add it by either selecting it and clicking the Add button or by double-click on it, which closes the Add Inputs dialog, reveals the Waveform Tools window, and places the file in the Inputs pane.
10. In the Specifications pane of the Waveform Tools window, choose File Export from the Filter List pop-up menu to make the File Export specification appear in the pane.
11. Highlight the specification and click Run.

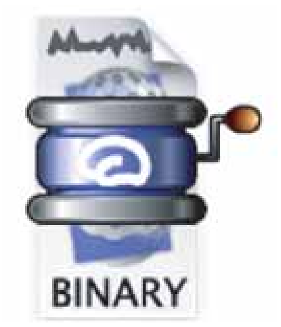
Your EEG data will be saved in a Persyst format file in the designated location.

chapter 9

NON-EEG DATA AND THE FILE EXPORTER

Introduction

Net Station exports to a variety of non-EEG data formats. It can export events from an EEG data file into text files and extract history logs about the operations performed on your data (timestamp, inputs, settings, results, and so on).



Net Station also contains a File Exporter application, a drag-and-drop utility for extracting data from Net Station files. File Exporter extracts EEG data into a simple-binary file format and also extracts event, gain, zero, and impedance histories into separate Simple Text files.

This chapter describes the two main types of data exported (events and histories), the export processes, and the data uses and structures. The end of the chapter discusses the Net Station File Exporter application.

Note: Several of the Waveform Tools File Export options—epoch-marked simple binary, simple binary, tab-delimited, and EGIS—allow you to output calibration (gains and zeros) and history information during the export process, by selecting the Export These Auxiliary Files checkboxes in the specification editor. These options produce Simple Text files that are identical to the ones produced using the history export feature and the File Exporter described in this chapter.

Event Export

Net Station allows you to export all or just select events from your source data into a text file. You can use the resulting event file with custom programs or applications.

You can export events in two ways:

- using the Event Export tool
- saving event information from the Event List

The Event Export tool allows you to export events from more than one file at a time through a multistep process. The default text file format is Microsoft Excel.

Saving event information from the Event List is a straightforward procedure, but it can be used on only one file at a time. The default text file format is tab-delimited text.

Event Export Tool

Before you can use Event Export, you must first create a tool specification for it that defines the tool's parameters. You do this using the Event Export specification editor, which allows you to edit fields to specific the operation parameters. In the editor, you will also specify the output options and the name and location of the output files.

Defining the Parameters

1 Display the Waveform Tools window either by clicking the Waveform Tools button in the Net Station start-up window or by choosing **Tools > Waveform Tools**.

2 In the Waveform Tools window, click the Create button. A pop-up menu of specifications will appear (see Figure 9-1).

3 Choose Event Export from the menu. The Event Export specification editor will appear.

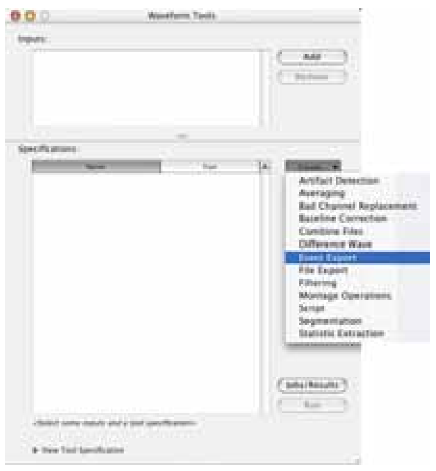
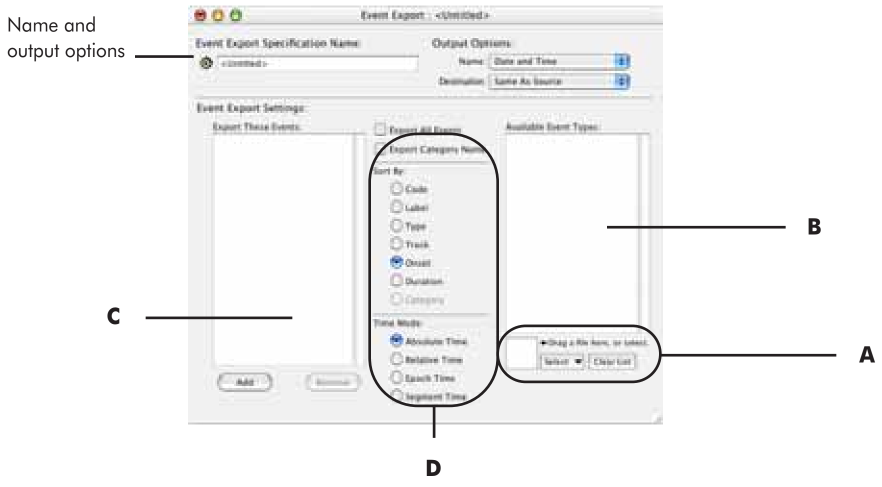


Figure 9-1.
Create an Event Export specification

4 Name the specification and define the output options (top of the window; see Figure 9-2).

5 Several specification options are available:

- *To export all events*: Select the Export All Events checkbox, and choose the sort order, time mode, and whether to include category names.
- *To export selected events and you know the event type names*: Click the Add button in the Export These Events pane and enter the event type, to populate that pane.
- *To export selected events and you do* ***not*** *know the event type names* (again see Figure 9-2): (A) Select the files that contain the events you wish to extract using the file selector; (B) notice that the event types from those files appear in the Available Event Types pane; and (C) drag specific events to export from the Available Event Types pane to the Export These Events pane. In either case, (D) remember to go to the middle and choose the sort order, time mode, and whether to include category names.



Name: Date and Time
Destination: Same As Source
Sort By: Onset
Time Mode: Absolute Time

Figure 9-2. Specify the Event Export parameters using the specification editor

6 When ready to save the specification, click the close box in the top-left corner of the window, causing the Close dialog to appear.

7 Click Save in the Close dialog.

Running the Tool

The Close dialog will disappear, revealing the earlier Waveform Tools window. To export events from the file (Figure 9-3):

1 In the Inputs pane of the Waveform Tools window, click the Add button. The Add Inputs dialog will appear.

2 Navigate to and select the desired file and click Add. The Add Inputs dialog will disappear.

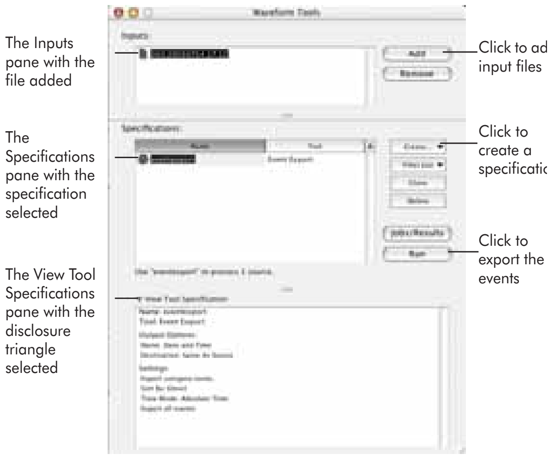

The Inputs pane with the file added
The Specifications pane with the specification selected
The View Tool Specifications pane with the disclosure triangle selected
Click to ad input files
Click to create a specificatic
Click to export the events
Waveform Tools
Add
Remove
Run

Figure 9-3. Event Export in the Waveform Tools

3 In the Specifications pane of the Waveform Tools window, select the Export Events specification.

4 Select the disclosure triangle in the View Tool Specification pane to review the specification.

5 If you wish to edit the specification, double-click on the specification name in the Specifications pane. The specification editor will appear.

6 If OK, click the Run button.

A Microsoft Excel text file of the specified events will appear in the designated folder.

The Event List

Saving events from the Event List is similar to taking a snapshot of the window's contents: The events displayed are saved in the order in which they appear in the window. If a disclosure triangle is closed and its events are hidden, they will not be exported to the output file.

To export events from the Event List window, first open the desiredNet Station data file. Then display the Event List by:

- choosing **Events > Event List** or
- clicking the Event List icon in the Events control strip

Event List Contents

The Event List displays all the events in that Net Station file. Directly above the list are the event element toggles that allow you to sort the events by code, label, type, track, onset, and duration. The arrow at the far right is also a toggle, changing the sort order between ascending and descending. Directly above the toggles are a row of buttons (from left to right): the Event List icon, Go To Event button, Edit Event button, Filter Events button, and the Time Mode buttons. (See Figure 9-4.)

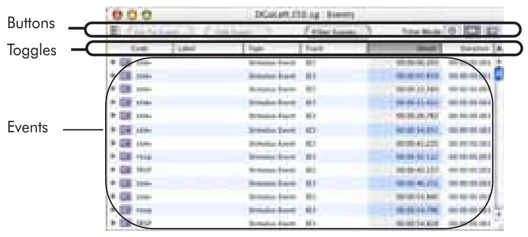


Figure 9-4. Event List window

Configuring and Exporting Events

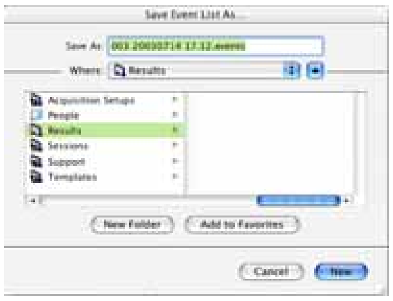
Now, configure the Event List until it displays the information you wish to export. You can configure the list by

- filtering (click the Filter Events button)
- opening or closing key lists (selecting disclosure triangles)
- sorting (selecting an event element toggle)
- changing time modes

Then, to export the events in the desired order:

1 Choose **File > Save Events**. The Save Event List As dialog will appear (Figure 9-5).

2 Provide a file name and destination in the Save Event List As dialog and click the New button. (The default destination is the Results folder.)



Save As: 003 20030714 17.12.events
Where: Results

Figure 9-5. Save Event List As dialog

History Export

You can also export a log of operations performed on your source data into a tab-delimited text file. The resulting history file provides useful background information about your file and can also serve as a diagnostic tool.

The History Pane

Each Net Station file contains a log of the Waveform Tools operations performed on it. You can find this information using two methods:

- open the desired Net Station file and choose **Edit > File Info**, or
- open the desired Net Station file and press Command-I

The File Info dialog will appear, with its six tabs. Click on the History tab to display the History pane.

History Pane Contents

At the top of the pane are four toggles and three buttons. Below the buttons are the contents of the pane, which are the names of the tools used to create that file (Combine Files, Baseline Correction, Difference Wave). Next to each name is a disclosure triangle and a tool icon. (See Figure 9-6.) In the default state, no toggles or disclosure triangles are selected.

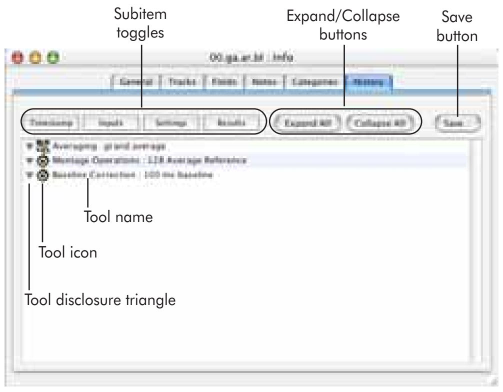


Figure 9-6. History pane components

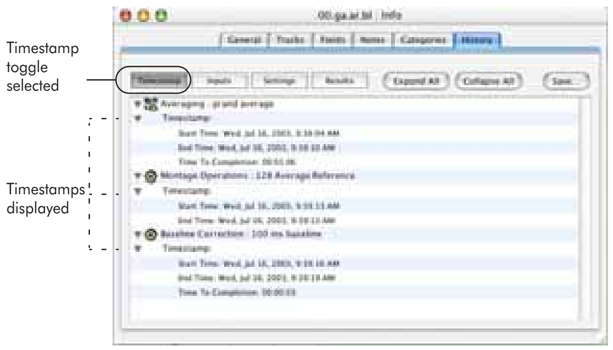
The four toggles display the timestamp, inputs, settings, and results of the tools (note that not all tools create results). To the right of the toggles are the Expand All and Collapse All buttons, which open and close all the disclosure triangles.

Configuring and Exporting History Subitems

To view the tools' subitems (for example, timestamp), it is not enough to click on the disclosure triangle next to a tool or to click the Expand All button. The disclosure triangle(s) will change direction and point downward, but no contents will be displayed.

To display the history information, you must first select the toggle(s) of interest and then click a tool's disclosure triangle or the Expand All button.

For example, selecting the Timestamp toggle and clicking Expand All will reveal the timestamp for all the tools (see Figure 9-7). Or selecting the Timestamp toggle and clicking the disclosure triangle next to Artifact Detection will display the timestamp of only Artifact Detection, if so desired.



Averaging : grand average
Timestamp:
Start Time: Wed, Jul 16, 2003, 9:59:04 AM
End Time: Wed, Jul 16, 2003, 9:59:10 AM
Time To Completion: 00:01:06
Montage Operations : 128 Average Reference
Timestamp:
Start Time: Wed, Jul 16, 2003, 9:59:11 AM
End Time: Wed, Jul 16, 2003, 9:59:13 AM
Baseline Correction : 100 ms baseline
Timestamp:
Start Time: Wed, Jul 16, 2003, 9:59:16 AM
End Time: Wed, Jul 16, 2003, 9:59:19 AM
Time To Completion: 00:00:03

Figure 9-7. History pane, with the Timestamp toggle selected

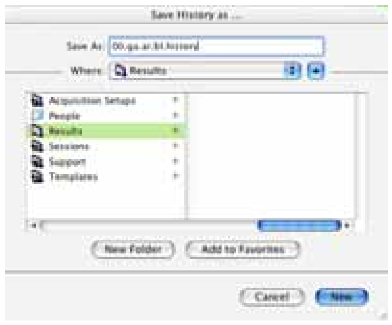
Here, as with the Event List, only the displayed subitems are saved. If subitems are hidden, they will not be exported to the output file. Unlike the Event List, however, no sorting of the History subitems is available.

So, to save History information:

1. Set up the contents of the window so that only the information you want is displayed.

2. Click the Save button at the far right. The Save History As dialog will appear (see Figure 9-8).

3. Provide a file name and destination in the Save History As dialog and click the New button. (The default destination is the Results folder.)



Save As: 00.ga.ar.bl.history
Where: Results

Figure 9-8. Save History As dialog

Net Station File Exporter

The Net Station File Exporter application is a drag-and-drop utility for extracting data from Net Station files (see Table 9-1). It saves EEG data into a simple-binary data format suitable for cross-platform analysis. In addition, it can also extract real-time histories of event, gain, zero, and impedance and place them into separate Simple Text files.

Table 9-1. Net Station File Exporter

Icon	File	Description	Type code
	Net Station File Exporter	A droplet application, installed along with Net Station. Data from Net Station Session and Recording files can be exported to other formats by dropping them on this application. Double-clicking the file icon accomplishes nothing; the application quits the moment it launches.	`'APPL'`

Using the File Exporter

Simply drag one or more Net Station data files onto the File Exporter icon. Simple-binary files will be extracted from each Net Station data file and placed in the same location as the source Net Station file.

To extract real-time histories (event, gain, and so forth) with the raw data, drag one or more Net Station data files onto the application icon while holding down the Command key. A beep will notify you that Simple Text and simple-binary files of the individual histories and the EEG data, respectively, have been created.

Raw Data Epoch

Raw data file epochs are determined by changes in real-time information such as gain, zero, impedance, filter, and sampling rate. Consequently, real-time changes within a single Net Station file or epoch will cause multiple raw data files to be exported from a single Net Station data file.

The File Exporter exports to a format that contains only one epoch of EEG (see Table 9-2).

Table 9-2. Net Station File Exporter output format

Name	**Length**	**Description**
Version	Long	Current version number for file (should be 2)
Year	Short	Recording date (year)
Month	Short	Recording date (month)
Day	Short	Recording date (day)
Hour	Short	Recording time (hour)
Minute	Short	Recording time (minute)
Second	Short	Recording time (second)
Millisecond	Long	Recording time (millisecond)
Rate	Short	Sampling rate
Channels	Short	Number of channels per sample
Gain	Short	Board gain (1, 2, 4, or 8)
Bits	Short	Number of conversion bits
Range	Short	Full-scale range (r) of amplifier in microvolts. A data value (v) for a channel can be converted to microvolts by the following formula: μV = v * r / 2**Bits
Samples	Long	Number of samples (a sample is a list of values for each channel)
Events	Short	Number of unique event codes (note that this is not the number of event tracks created by devices on the Workbench, because an event track can contain events with different codes and the Net Station Recording file records only nonempty tracks)
EventsCodes	Events * 4	Sorted list of event codes. An event code is a four-character code.
RawData	(Channels + Events) * 2 * Samples	An array of raw data for each channel plus the event states for each event code in the order given in the EventCodes list. Each value is a short signed integer. Event states are either 0 or 1.

Sample 1: Ch 1, Ch 2, . . . Ch n, Event 1, Event 2, Event m
Sample 2: Ch 1, Ch 2, . . . Ch n, Event 1, Event 2, . . . and so forth.

For 128 channels and four used event codes:

- RawData[0] = channel 1, EEG sample 1
- RawData[1] = channel 2, EEG sample 1

- RawData[127] = channel 128, EEG sample 1
- RawData[128] = state of Event 1 during sample 1

- RawData[131] = state of Event 4 during sample 1
- RawData[132] = channel 1, EEG sample 2
- RawData[1] = channel 2, EEG sample 2

. . . so forth.

To peel off a particular channel from the raw data format, extract every (Channels + Events)th data point of the RawData array, starting at the (channel number – 1)th data point.

To peel off a particular event track from the raw data format, extract every (Channels + Events)th data point of the RawData array, starting at the (Channels + event number – 1)th data point.

appendix A

SOFTWARE TECHNICAL SUPPORT

Before Contacting EGI

Please check the Contents on page v or the Index on page 147 for coverage of your issue or question. You can also perform an electronic search using Find or Search in the PDF version of this manual posted on the Documents page of the EGI website (www.egi.com/ documentation.html).

In addition, the Support page of the EGI website (www.egi.com/support.html) may have the information you need.

If you need more help, EGI recommends the following:

• ***Try to isolate the problem***. Is your problem well defined and repeatable?

• ***Document the problem.*** Carefully record and organize the details gleaned from the above step and report the problem to EGI.

Contacting EGI

EGI Support web page	www.egi.com/ support.html
Email support	support@egi.com
Sales information	info@egi.com
Telephone	+541-687-7962
Fax	+541-687-7963
Address	Electrical Geodesics, Inc. 1600 Millrace Drive Suite 307 Eugene, OR 97403 USA

appendix B

Net Station Example Code

The example code in this appendix is available electronically from EGI.

Example 1

The following code traverses through a file, asks for a Net Station Recording file, opens it, and prints to a file information including start time and duration of each epoch, total number of samples, event information, and every sample.

```
#include <NetStationServices.h>
#include <CWaveformRecording.h>
#include <CEventKey.h>
#include <stdio.h>
#include <sioux.h>

typedef struct SampleHandlerData {
        FILE                           *stream;
        long                           lastEpoch;
        CWaveformRecording      *recording;
}  SampleHandlerData;

int main(void);

static void MessageProc(short messageID,
                                               void *param);

static bool SampleHandler(CSample *sample,
                                             long epochIndex,
                                                bool
*outChanged,
                                                void *param);

int main(void)
```

```
{
        SFTypeList              typeList = { nsftEEGRecording,

nsftSegEEGRecording,

nsftEdEEGRecording,

nsftEdSegEEGRecording };
FILE* stream;
charname[10];

SIOUXSettings.rows = 70;

// Initialize the Mac appliation and toolbox
InitializeHeap(1);
UQDGlobals::InitializeToolbox(&qd);

// Initialize the Net Station Services library
InitNetStationServices('NETs', '????', MessageProc, true);

// Present Standard Get File to the user
StandardFileReplyreply;

StandardGetFile(nil, 4, typeList, &reply);
if (reply.sfGood) {
//********** C.T. - opens output file
strcpy(name, "cetout.dat");
stream = fopen(name, "w");
fprintf(stream, "File: %#s  ", reply.sfFile.name);

// Attempt to open the file as read-only
// Usually you would have error checking here, but I left it
out
CWaveformRecording* recording;

recording = CWaveformRecording::Open(reply.sfFile, readOnly,
nil,
false, false, nil);

// print the file name
printf("File: %#s\n\n", reply.sfFile.name);
```

```
// Count the number of recording epochs
// Print out information for each epoch
const longnRecordingEpochs = recording-
>CountRecordingEpochs();
longep;
//********** C.T.
floatmicrovoltsPerBit = recording->GetMicrovoltsPerBit();
longsamp;
longchan;
longe = 1;
longtSamples = 0;
longtDuration = 0;

fprintf(stream,"  micovolts per bit=%10g", microvoltsPerBit);
fprintf(stream,"   Epochs = %10ld", nRecordingEpochs);

for (ep = 1; ep <= nRecordingEpochs; ep++) {
RecordingEpochInfoepochInfo;

recording->GetRecordingEpochInfo(ep, epochInfo);
tDuration += epochInfo.duration;
tSamples += epochInfo.numberOfSamples;
fprintf(stream,"       [epoch %3ld:] %8ld to %8ld = %8ld
Samples",
ep, epochInfo.startTime, epochInfo.startTime +
epochInfo.duration,
epochInfo.numberOfSamples);
}
fprintf(stream,"       Total: %8ld to %8ld - %8ld Samples\n",
0L, tDuration,
tSamples);

// Count the number of tracks
// Print information about each track
const shortnTracks = recording->CountTracks();
shortt;

// Count the number of events
const longnEvents = recording->CountEvents();
CObjectList*events = recording->GetEvents();
chareventClassIDString[5] = { 0, 0, 0, 0, 0 };
chareventCodeString[5] = { 0, 0, 0, 0, 0 };
chareventKeyString[5] = { 0, 0, 0, 0, 0 };
```

```
fprintf(stream, "%10ld events total\n", nEvents);

for (e = 1; e <= nEvents; e++) {
CStimulus*anEvent = (CStimulus *)events->Get(e);
DescTypeeventClassID = anEvent->GetClassID();
DescTypeeventCode = anEvent->GetCode();
TimeeventStartTime = anEvent->GetStartTime();
TimeeventDuration = anEvent->GetDuration();

BlockMoveData(&eventClassID, eventClassIDString, 4);
BlockMoveData(&eventCode, eventCodeString, 4);
fprintf(stream, "event %-10d class=%.4s code=%.4s start=%10ld
duration=%10ld",
e, eventClassIDString, eventCodeString, eventStartTime,
eventDuration);

const longnKeys = anEvent->CountKeys();
longk;

fprintf(stream, "        %ld keys: ", nKeys);
for (k = 1; k <= nKeys; k++) {
DescTypeeventKeyCode = anEvent->GetKey(k)->GetKey();

BlockMoveData(&eventKeyCode, eventKeyString, 4);
fprintf(stream, "(%1d)%.4s ", k, eventKeyString);
}
fprintf(stream, "\n");
}/* for */

delete events;

//**********************************
// Traverse through the entire file and print out each
sample.
// This is different from the previous example that just
extracted a couple of
// samples in matrix form. This example uses the sample
traversal proc, which
// operates on each sample in order.
// Samples will be calibrated and scaled to microvolts.
SampleHandlerDatahandlerData = { stream, 0 , recording };
```

```
recording->TraverseSamples(0, recording->GetDuration(),
SampleHandler,
&handlerData, true);

// Don't forget to close the recording
recording->Close(nil);

fclose(stream);
}
}/* main */

static void MessageProc(short messageID,
void *param)

{
// This message proc does absolutely nothing
}/* CNetStationApp::NSMessageProc */

static bool SampleHandler(CSample *sample,
  long epochIndex,
  bool *outChanged,
  void *param)

{
SampleHandlerData* data = (SampleHandlerData *)param;

if (epochIndex != data->lastEpoch) {
RecordingEpochInfoepochInfo;

data->lastEpoch = epochIndex;
data->recording->GetRecordingEpochInfo(epochIndex,
epochInfo);
fprintf(data->stream, "epoch number %ld ", epochIndex);
fprintf(data->stream, "   samples in epoch=%20ld\n",
epochInfo.numberOfSamples);
}

const shortnChannels = sample->CountChannels();
shortc;

for (c = 1; c <= nChannels; c++)
 fprintf(data->stream, "%6d", sample->GetInteger(c));
fprintf(data->stream, "%6d\n");
```

```
*outChanged = false;// we didn't modify the sample - just
looked at it
return false;// don't cancel traversal
}/* SampleHandler */
```

Example 2

The following code performs the same tasks as the code in Example 1, except: (1) it pulls out the file information in the form of a matrix, and (2) it prints only the first four samples.

```
#include <NetStationServices.h>
#include <CWaveformRecording.h>
#include <stdio.h>
#include <sioux.h>

extern int main(void);

static void MessageProc(short messageID,
void *param);

int main(void)

{

SFTypeListtypeList = { nsftEEGRecording,

nsftSegEEGRecording,

nsftEdEEGRecording,

nsftEdSegEEGRecording };

SIOUXSettings.rows = 70;

// Initialize the Mac appliation and toolbox
InitializeHeap(1);
UQDGlobals::InitializeToolbox(&qd);
```

```
// Initialize the Net Station Services library
InitNetStationServices('NETs', '????', MessageProc, true);

// Present Standard Get File to the user
StandardFileReplyreply;

StandardGetFile(nil, 4, typeList, &reply);
if (reply.sfGood) {
// Attempt to open the file as read-only
// Usually you would have error checking here, but I left it
out
CWaveformRecording* recording;

recording = CWaveformRecording::Open(reply.sfFile, readOnly,
nil,
false, false, nil);

// print the file name
printf("File: %#s\n\n", reply.sfFile.name);

// Count the number of recording epochs
// Print out information for each epoch
const longnRecordingEpochs = recording-
>CountRecordingEpochs();
longep;

printf("Recording Epochs = %ld\n", nRecordingEpochs);
for (ep = 1; ep <= nRecordingEpochs; ep++) {
RecordingEpochInfoepochInfo;

recording->GetRecordingEpochInfo(ep, epochInfo);
printf("%3ld: %8ld to %8ld - %8ld Samples\n", ep,
epochInfo.startTime,
epochInfo.startTime + epochInfo.duration,
epochInfo.numberOfSamples);
}
printf("\n");

// Count the number of tracks
// Print information about each track
const shortnTracks = recording->CountTracks();
shortt;
```

```
printf("Tracks = %d\n", nTracks);
for (t = 1; t <= nTracks; t++) {
// get a track reference
// this is a shared object, so it needs to be claimed and
released
CTrackRef* ref = recording->GetTrackRef(t);

ref->Claim(nil);

// get information about the track
const LStr255&trackName = recording->GetTrackName(ref);
const longnElements = recording->CountElementsInTrack(ref);

printf("%3d: \"%#s\" has %ld elements.\n", t,
(ConstStringPtr)trackName,

nElements);

ref->Release(nil);
}
printf("\n");

// Count the number of events
const longnEvents = recording->CountEvents();

printf("This file contains %ld events\n\n", nEvents);

// Get the first 4 samples as a matrix object
// Samples are requested to be calibrated and scaled to
microvolts
const longnSamplesToGet = 4;
longsamplingRate = recording->GetSamplingRate(0L);
longchannelCount = recording->CountChannels(0L);
CShortMatrix* first4Samples = (CShortMatrix *)recording-
>GetSamples(0,
nSamplesToGet * 1000L / samplingRate,
CSample::integer, true);
CShortMatrix&m = *first4Samples;
longsamp;
longchan;

first4Samples->Claim(nil);
printf("----------------------------------------\n");
printf("SamplingRate = %ld\nChannel Count = %ld\n\n",
samplingRate, channelCount);
for (chan = 1; chan <= channelCount; chan++) {
for (samp = 1; samp <= nSamplesToGet; samp++)
printf("%6d", m[ samp][ chan] );
printf("\n");
}
first4Samples->Release(nil);

//
************************************************************
*******************
// Code from page 16 of WaveformRecordingAPI


//
************************************************************
*******************
const DescTypeeventIdentifier = 'DIN3';// retrieve all 'DIN3'
events
CObjectList*events;
longne;
longe;

events = recording->GetEvents(nil, &eventIdentifier, "\p",
nil, "\p", "\p");
events->Claim(nil);
ne = events->GetCount();
for (e = 1; e <= ne; e++) {
CStimulus*anEvent = (CStimulus *)events->Get(e);

printf("%ld\n", anEvent->GetTime());
}
events->Release(nil);

// Don't forget to close the recording
recording->Close(nil);
}
} /* main */

static void MessageProc(short messageID,
void *param)
{
 // This message proc does absolutely nothing
}/* CNetStationApp::NSMessageProc */
```

INDEX