BCI2000 Command Line Interface

Jürgen Mellinger

May 16, 2005

Contents

1 Introduction

Command line interfaces are a powerful tool for dynamically combining pieces of software which are typically tiny programs optimized for performing a very limited functionality. With a command line interface, data processing is usually performed *sequentially* on a stream of data. This allows the processing of indefinitely large amounts of data while consuming but a finite, and often very small, amount of memory.

Moreover, combining command line programs to perform complex tasks can be done interactively by entering complex commands directly but also in a "scripting" manner, by entering sequences of commands into text files which allow for iteration and branching. That way, automation of complex tasks can be achieved by rather simple means.

With the BCI2000 command line interface, BCI2000 data files can be

- *converted* into various formats, ready for further processing with available software tools, or for visual inspection in human readable format;
- *processed* off-line with literally the same code as used in the on-line system.

Together, these two concepts provide versatile access to recorded data on all stages of processing. The BCI2000 command line interface may prove useful for

- automated *analysis* of large amounts of recorded data,
- *development* of new filter classes,
- *verification* of external off-line analysis methods.

This document explains the command line tools available for working with BCI2000 files, and how to build BCI2000 filters as single executables that may be combined to form a chain of filters.

2 Command line interfaces

Basically, the interface of a command line tool consists in

- an input stream `stdin`,
- an output stream `stdout`,
- an error stream `stderr`,

- optional arguments (switches) controlling details of its behavior.

The most important feature of a command line interface is its ability to *redirect* streams.

- `stdin` can be redirected to read from a file using the `<` operator;
- `stdout` can be redirected to write to a file using the `>` operator;
- one program's `stdout` can be plugged into another program's `stdin` using the `|` operator ("pipe").

Because redirection is a feature of the program's execution environment ("shell"), and not a feature of the program itself, programs can be very simple, avoiding handling of file names and associated I/O errors.

The following example command line will extract the "sampling rate" parameter from the BCI2000 data file `mydata.dat`, and display the result on the text console:

```
bci_dat2stream < mydata.dat | bci_stream2prm | grep SamplingRate
```

3 Building BCI2000 command line tools

As the command line tools are not automatically built from the main makefile, you need to do this manually. On the Windows NT shell, change to the `BCI2000/Tools/cmdline` directory, and execute `make all` from there.

To use the tools from any directory, you will need to add the full path to the `BCI2000/Tools/cmdline` directory to your system's PATH environment variable. Please consult your operating system's documentation on how to do this.

4 Stream format

The common format used to exchange data between BCI2000 command line tools is called a "BCI2000 binary stream," or just "stream." Such a "stream" transfers parameters, states, and data in exactly the same binary format as it is used for socket communication between the four main modules of the BCI2000 real-time system.

The stream format is different from the BCI2000 `dat` and `prm` file formats, and is not human readable. A number of tools is available to convert data either to or from stream format. As the stream format is only used between tools, and generally not of interest to the user, a typical conversion will use

two of the tools provided, one to translate the original file into a "stream," and one to translate the stream into the desired format.

E.g., to display a `dat` file's contents in a human readable format, one would use a tool called `bci_dat2stream` to convert it into a stream, and then use another tool called `bci_stream2asc` to convert the stream into text. If we want the output to appear in the text window as a sequence of pages, we pipe it into the "more" program:

```
bci_dat2stream < myfile.dat | bci_stream2asc | more
```

4.1 Conversion into stream format

4.1.1 `bci_dat2stream`

Input format `bci_dat2stream` converts a BCI2000 data (`dat`) file into a stream.

Options The `--transmit` option may be used to select states, parameters, and data for transmission, as in
`bci_dat2stream --transmit-sp < myfile.dat`
In the above example, only states and parameters but no data will be contained in the resulting stream. "Data" comprises signal and state vector data. By default, all information contained in the `dat` file will be transmitted.

4.1.2 `bci_prm2stream`

`bci_prm2stream` converts a BCI2000 parameter (`prm`) file into a stream.

4.2 Conversion from stream format

4.2.1 `bci_stream2prm`

`bci_stream2prm` converts a stream into a BCI2000 parameter (`prm`) file. As a parameter file is just a sequence of parameter lines, this is also a text-only, human readable format.

4.2.2 `bci_stream2asc`

`bci_stream2asc` converts a stream into a human readable format. Each object contained in the stream will appear as its C++ type name, followed by an opening brace, its content, and a closing brace. The content will appear as defined by the stream inserter `operator>>` for the object's type.

In the output of BCI2000 state vector information, each state will appear on its own line, thus values of certain states may be easily extracted using the `grep` program.

4.2.3 `bci_stream2table`

`bci_stream2table` converts a stream into a tab-separated table containing state and signal values in ASCII format. Each state, and each entry of the signal, has its own column. The first line of output begins with a `#` comment character, and contains a tab-separated list of column headers. This format is best suited for data import into applications that work on tables.

4.2.4 `bci_stream2mat`

`bci_stream2mat` converts a stream into a matlab binary file. The output `.mat` file contains two matlab variables called `Index` and `Data`. Of these, the `Data` variable is a matrix with each column representing a BCI2000 data block (comprising state information and signal data). `Index` is a matlab structure that contains indices to `Data's` rows, allowing access to BCI2000 states by name, as in:

```
myMatlabVariable = squeeze( Data( Index.TargetCode, : ) );
```

As each BCI2000 data block contains a signal which is a two-dimensional matrix (channels by elements), the signal index is itself a matrix. To copy the first channel's data into a matlab variable, write

```
myChannel1 = squeeze( Data( Index.Signal( 1, : ), : ) );
```

For convenience, there is a matlab function provided that simplifies reading `bci_stream2mat` output files into matlab variables:

```
[ mySignal, myTargetCode ] =
  load_bcimat( 'eegdata.mat', 2, 'TargetCode' );
```

This function takes the file name as its first argument. In a second argument, specify the number of dimensions your output signal will have – typically, this will be 2 for EEG-like data (samples by channels), and 3 for spectral data (blocks by bins by channels). Remaining arguments are treated as state names; the associated state data will be written into the variables specified as remaining output arguments. State variables will always be one-dimensional, with their number of entries matching the first dimension of the signal variable.

5 Applying BCI2000 filters to streams

5.1 Compiling an existing filter as a command line tool

A filter defined in a file `MyFilter.cpp` may be compiled and linked into its own executable by executing
`make MyFilter.exe`
from the Windows command prompt when in `BCI2000/Tools/cmdline`. For this to work, it is necessary that the directory containing the filter's `cpp` file is contained in the makefile's `SIGPROC` variable; you may have to adapt it to fit your needs.

If the filter's code depends on code not contained within its `cpp` file, you will get linker complaints about unresolved externals. To solve the problem, add the missing `cpp` files to the makefile's `BCIOBJ` variable, and execute
`make clean && make MyFilter.exe`
To check whether the executable works, enter
`MyFilter --help`
You should get a message that the program applies the "MyFilter" filter to its input.

5.2 Off-line only filters

For off-line analysis, data must often be partitioned into "segments" before performing statistics. As there is no notion of "segments" in the on-line data and file format, we suggest using the "Running" state to indicate segments in the following way:

When performing segmenting, a filter sets the "Running" state to zero outside segments. A statistics filter will then perform buffering from its `Process()` function, and act on the buffered data from its `StartRun()` and `StopRun()` functions.

Setting the "Running" state to zero will suspend the on-line system, so this kind of segmenting and statistics filtering cannot be used on-line.

6 Examples

The following examples work from the Windows NT command prompt. Nevertheless, we would like to point the reader to the free *cygwin* collection of GNU tools ported to the Win32 API. *Cygwin* provides the power of the `bash` shell, and of programs like the `sed` stream editor. Although the BCI2000 command line tools cannot be compiled with `gcc` they work fine when called from *cygwin*.

6.1 Extracting parameters

To extract parameters from a data file, convert it into a stream using `bci_dat2stream --transmit-p`, and convert the stream into a parameter file using `bci_stream2prm` as in

```
bci_dat2stream < mydata.dat | bci_stream2prm > myprms.prm
```

6.2 Processing data with parameters different from the ones contained in the file

To combine a data file with parameters other than those contained in it, use `bci_dat2stream`'s `--transmit` option to suppress parameters, and combine its output with `bci_prm2stream`'s into a single stream. To affect processing, parameters must precede the data in the stream. Combining the output is effected by the `( ... && ... )` construct.

```
(bci_prm2stream < myparameters.prm &&
  bci_dat2stream --transmit-sd < mydata.dat) |
  MyFilter | bci_stream2table > mytable.txt
```

6.3 Processing data with BCI2000 filters, and importing the results into Matlab

To process data with the filters used in the mu-training on-line system, saving the AR spectrum as matlab file, execute

```
bci_dat2stream < mydata.dat | TransmissionFilter |
  CalibrationFilter | SpatialFilter | ARFilter |
  bci_stream2mat > myspectra.mat
```

Load the data into matlab using

```
[ signal, TargetCode ] =
  load_bcimat( 'myspectra.mat', 3, 'TargetCode' );
```

This requires that the file `load_bcimat.m` is accessible from the matlab search path.

6.4 Exporting BCI2000 data into a table suitable for import into other applications (MS Excel, SPSS)

To process data with the filters used in the mu-training on-line system, saving the AR spectrum as a table in ASCII format, execute

```
bci_dat2stream < mydata.dat | TransmissionFilter |
  CalibrationFilter | SpatialFilter | ARFilter |
  bci_stream2table > mytable.txt
```

6.5 Testing a modified filter on existing data

To verify that changes to a filter's code don't change its behavior with respect to existing data, apply both versions to a stream, convert the output stream into human readable format, and have a file comparison program display any differences. For the following example, we will compare a previous version of the ARFilter, renamed `prev_ARFilter`, to the current one.

1. Create a stream suitable for input to the ARFilter:

   ```
   bci_dat2stream < mydata.dat | TransmissionFilter |
     CalibrationFilter | SpatialFilter > test.bcistream
   ```

2. Apply both filter versions to the stream, and save the results in human readable format:

   ```
   ARFilter < test.bcistream | bci_stream2asc > ARresult.txt &&
     prev_ARFilter < test.bcistream |
     bci_stream2asc > prev_ARresult.txt
   ```

3. Compare the results (using the Windows NT analog to the `diff` program):

   ```
   comp /a /l prev_ARresult.txt ARresult.txt | more
   ```

7 Further information

- A comprehensive and up-to-date description of a number of command line shells, and their scripting, is provided at `http://www.ss64.com/`.
- The *cygwin* Win32 port of GNU tools can be downloaded at `http://www.cygwin.com/`.