

## EEG Tutorial 1: EEG Data Reading using C# and DotNetEmotivSDK.dll

This example demonstrates how to extract live EEG data using the EmoEngine™ in c#. Data is read from the headset and sent to an output file for later analysis.

### Project set-up:

EEG\_Logger is a c# console project. To enable access to the EmoEngine™ via managed code, it is necessary to reference the .NET dll - DotNetEmotivSDK.dll, which can be found in the installation folder of the EDK. This dll exposes the EmoEngine™ class, which is explained further in the 'API Reference for dot NET' help file.

### Listing 1: Initialization

```
EmoEngine engine;  
int userID = -1;  
...  
  
// create the EmoEngine  
engine = EmoEngine.Instance;  
engine.UserAdded += new EmoEngine.UserAddedEventHandler(engine_UserAdded_Event);  
  
// connect to Emoengine.  
engine.Connect();
```

We assign our engine reference to the global EmoEngine™ via the 'Instance' property. This instance property is used to access all EmoEngine functions:

```
engine = EmoEngine.Instance;
```

We add a new event handler to the UserAdded event, which our application will process. This is necessary to ensure we are logging data from the correct headset.

```
engine.UserAdded += new EmoEngine.UserAddedEventHandler(engine_UserAdded_Event);
```

We commence access to the engine by a call to Connect():

```
engine.Connect();
```

And finally we create a pre-defined column header for the CSV file:

```
// create a header for our output file  
WriteHeader();
```

### Listing 2: Main loop

```
static void Main(string[] args)
{
    Console.WriteLine("EEG Data Reader Example");

    EEG_Logger p = new EEG_Logger();

    for (int i = 0; i < 10; i++)
    {
        p.Run();
        Thread.Sleep(100);
    }
}
```

The main loop of the application simply creates the logging class, loops 10 times and exits. The total amount of data collected is approximately 1 second's worth.

### Listing 3: User connection

```
void engine_UserAdded_Event(object sender, EmoEngineEventArgs e)
{
    Console.WriteLine("User Added Event has occurred");

    // record the user
    userID = (int)e.userId;

    // enable data acquisition for this user.
    engine.DataAcquisitionEnable((uint)userID, true);

    // ask for up to 1 second of buffered data
    engine.EE_DataSetBufferSizeInSec(1);
}
```

To enable data collection via EmoEngine, it is necessary to ensure a valid user has connected. Once connected (and the event thrown), data acquisition is enabled with a call registering the connected user as follows:

```
engine.DataAcquisitionEnable((uint)userID, true);
```

The EmoEngine maintains an internal buffer of sampled data. To adjust the size of this data, make a call as follows:

```
engine.EE_DataSetBufferSizeInSec(1);
```

At this point, you are ready to collect data.

#### Listing 4: Data collection

```
// Handle any waiting events
engine.ProcessEvents();

// If the user has not yet connected, do not proceed
if ((int)userID == -1)
    return;

Dictionary<EdkDll.EE_DataChannel_t, double[]> data =
engine.GetData((uint)userID);

...

int _bufferSize = data[EdkDll.EE_DataChannel_t.TIMESTAMP].Length;

...

// Write the data to a file
TextWriter file = new StreamWriter(filename, true);

for (int i = 0; i < _bufferSize; i++)
{
    // now write the data
    foreach (EdkDll.EE_DataChannel_t channel in data.Keys)
        file.Write(data[channel][i] + ",");
    file.WriteLine("");
}
file.Close();
```

We first process any waiting engine events with a call as follows. This is necessary to ensure the user has connected and our user-setup code is called.

```
engine.ProcessEvents();
```

We confirm that a valid user has connected by checking the ID (Note - the first user to connect has an ID of 0)

```
if ((int)userID == -1)
    return;
```

We then request our data as shown. This call will return all available EEG data currently held in the EmoEngine buffers. For each data channel, we receive an array of type double, whose length is the number of samples collected since the last time GetData was called. Note that if GetData is called too infrequently the buffers may overflow.

```
Dictionary<EdkDll.EE_DataChannel_t, double[]> data =
engine.GetData((uint)userID);
```

We can easily establish the number of samples captured as follows:

```
int _bufferSize = data[EdkDll.EE_DataChannel_t.TIMESTAMP].Length;
```

To access the data (in this case, we output directly to a file), it is simply a matter of looping over all data held in the 'data' structure as follows:

```
for (int i = 0; i < _bufferSize; i++)
{
    // now write the data
    foreach (EdkDll.EE_DataChannel_t channel in data.Keys)
        file.Write(data[channel][i] + ",");
    file.WriteLine("");
}
```