EGI's Matlab MFF API

4/23/2014

1. Introduction

EGI's Matlab MFF API consists of a set of functions that allow you to use Matlab to read and write Net Station EEG files that are in EGI's native MFF (meta-file format) version 3 (the current version).

If you have MFF files whose versions are older than version 3, you can convert your data to version 3 using EGI's MFF File Converter.

The data-reading API is a high-level API that returns EEG data, event information and meta-data in Field Trip (http://fieldtrip.fcdonders.nl/) format in order to work seamlessly with Field Trip and programs like EEGLAB that work with Field Trip.

The data-writing functions have some limitations. If you want to write code for writing MFF data, please read about those limitations carefully. Future releases of this API will allow more flexible data-writing.

The Matlab MFF API accesses the low-level MFF API through a Java layer (.jar file).

Because the Matlab MFF API is based on the Field Trip event structure, it does not give you access to key lists and TRSPS. If you need to use key-lists or TRSPS, we recommend you use the Net Station's Segmentation Markup feature before reading your event data with the API.

Net Station 5 allows the user to create MATLAB functions that can be called as tools in Net Station. You can use this API to write Matlab functions that can be called from Net Station.

This document is supplemented be demo code that demonstrates how to use the API for most things you might want to do. It has the following sections:

2. What's New In This Release

Compared to the first release of EGI's Matlab MFF API, this version:

- Can read and write PIB channels.
- Can add an entry to the history structure of the MFF file so a record of the processing you have done can be seen in Net Station's File-History pane.
- Can read bad-segment information.
- Can read subject data from multi-subject files.
- Supports (and only supports) the most recent version of the MFF format (Version 3).
- Reads large files much more efficiently in terms of memory requirements and speed.
- Has improved error handling.
- Has a changed calling syntax for write_mff_event.

Any code that works with previous versions of the Matlab MFF API should work unchanged with this version, with the following exceptions:

- Any calls to write_mff_event need to be modified to use the new calling syntax. See the section on write_mff_event below for the current syntax.
- In the previous version, calls to read_mff_data, with no channels specified, return all the EEG channels, and no PIB channels. In the current version, they return all the channels, i.e. both EEG and PIB channels. If none of your data contain PIB channels, your code does not need to change. If your data do contain PIB channels, and you want read_mff_data to return the PIB channels along with the EEG, then your code doesn't need to change. If your data do contain PIB channels, but you only want to read the EEG, then your code does need to change. See the section on read_mff_data below for more details.

3. Installation

To install EGI's Matlab MFF API, copy it to a folder on your computer, and make sure it's on your Matlab path.

This release comes with a set of demo code that shows how to use the Matlab MFF API's functions. To install the demo code,

1. Copy it to a folder on your computer, and make sure it's on your Matlab path.
2. Copy the demo data to a folder on your computer.
3. Edit the file demoMFF3Release so that the line that sets the "dataPath" variable points to the folder that has the demo data.

If you are reading large files, you might need to increase the heap space that Matlab allocates to Java. To do this, create a file called java.opts and put it in your Matlab startup directory. Include the following line:

-Xmx1024m

This last step is only necessary if you get an error message like the following when reading MFF files:

??? Java exception occurred:
java.lang.OutOfMemoryError: Java heap space

4. The Functions

4.1. Initialization Function

function mff_setup

It is necessary to call this function, which takes no parameters, at least one time in your program or Matlab session before calling any of the other functions. It initializes access to the Java library that underlies this API.

4.2. Data-Reading Functions

The data-reading functions are designed to work seamlessly with FieldTrip. They consist of the following routines:

- **read_mff_header:** returns a header structure described at http://fieldtrip.fcdonders.nl/reference/ft_read_header.
- **read_mff_data:** returns a 2-D matrix of size Nchans*Nsamples as described at http://fieldtrip.fcdonders.nl/reference/ft_read_data.
- **read_mff_event:** returns an array of event structures described at http://fieldtrip.fcdonders.nl/reference/ft_read_event.

Note that these routines are not structured to use variable length argument lists, like the Field Trip routines typically do. For example, the read_data routine uses the indType parameter to clarify the meaning of beginInd and endInd. Each of these routines are described in more detail below.

function header = read_mff_header(filePath)

Takes the path to the data and returns the header in the structure described at http://fieldtrip.fcdonders.nl/reference/ft_read_header.

filePath – The path to the .mff file.

function data = read_mff_data(filePath, indType, beginInd, endInd, chanInds, hdr)

Takes the path to the data and returns the 2-D matrix of size Nchans*Nsamples as described at http://fieldtrip.fcdonders.nl/reference/ft_read_data.

filePath – The path to the .mff file.

indType – Indicates how to interpret the next two parameters. There are two possible values:

- 'epoch', which interprets the following two parameters as epoch numbers.
- 'sample', which interprets the following two parameters as sample numbers.

chanInds – Indicates which channels to include. If you pass in [] (MATLAB's equivalent of NULL), you get all the channels. You can pass in lists of channels in MATLAB format, such as [1 2 3] or [1:3], and so on.

hdr – FieldTrip header. You have the option of passing in the header, or []. If you pass in the header, it pulls data out of it, rather than recomputing them.

In the previous version, calls to read_mff_data, with no channels specified, return all the EEG channels, and no PIB channels. In the current version, they return all the channels, i.e. both EEG and PIB channels. If none of your data contain PIB channels, your code does not need to change. If your data do contain PIB channels, and you want read_mff_data to return the PIB channels along with the EEG, then your code doesn't need to change.

If your data do contain PIB channels, but you only want to read the EEG, then your code does need to change. To read either EEG only, or PIB channels only, you need to provide the proper values for the chanInds parameter. You can determine these values by looking at the following fields in the header structure:

- header.nChans – the total number of channels in the data. Includes both EEG and PIB channels.
- header.orig.nChans – the total number of EEG channels.
- header.orig.pibNChans – the total number of PIB channels.

So:

To read all channels, set the chanInds parameter to [] or 1:header.nChans.
To read EEG only, set the chanInds parameter to 1:header.orig.nChans.
To read PIB channels only, set the chanInds parameter to header.orig.nChans+1:header.nChans or header.orig.nChans+1:header.orig.nChans+ header.orig.pibNChans.

filePath – The path to the .mff file.

hdr – FieldTrip header. You have the option of passing in the header, or []. If you pass in the header, it pulls data out of it, rather than recomputing them.

function events = read_mff_event(filePath, hdr)

Takes the path to the data and returns the events in the structure described at http://fieldtrip.fcdonders.nl/reference/ft_read_event.

filePath – The path to the .mff file.

hdr – FieldTrip header. You have the option of passing in the header, or []. If you pass in the header, it pulls data out of it, rather than recomputing them.

This function returns two types of events:

- Normal Events associated with actual events that occurred during the data collection - typically stimuli or responses, but possibly other types of events such as blinks or seizure onsets that have been added manuallly or automatically.
- Metadata events that contain information about epoch breaks.

You can tell these events apart because the normal events never contain anything in the event's 'value' field, wherease the metadata events do. For more information about the metadata events, see the section "Epochs in Continuous Recordings and Segmented Data" below.

Epochs in Continuous Recordings and Segmented Data

Information about epochs is contained in metadata events. You can tell metadata events from normal events because only metadata events contain information in the event's 'value' field.

Files that are broken into epochs are handled as one of three cases.

1. Continuous recordings that have zero or more recording breaks in the middle.
2. Files that are cleanly segmented into trials.
3. Files that are segmented into trials, but the trials have variable lengths, or variable time zeros.

Events are used in each of these cases is discussed below.

1. Continuous recordings that have zero or more recording breaks in the middle.

In these types of recordings, the first sample at the beginning of an epoch is marked by an event as follows:

type: 'break cnt' (continuous)

value: 'epoch'

2. Files that are cleanly segmented into trials.

In these types of recordings, the first sample at the beginning of an epoch is marked by an event as follows:

type: 'break seg' (segmented)
value: the name of the condition of the trial, e.g., 'standard' or 'target'.

3. Files that are segmented into trials, but the trials have variable lengths, or variable time zeros.

type: 'break var' (variable)
value: the name of the condition of the trial, e.g., 'standard' or 'target'.

In this case, there are Time 0 events, as discussed below.

For the above three 'break' events, the duration of the epoch is put in the 'duration' field of the event in terms of the number of samples.

Time 0

For cases 2 and 3, where the data has been segmented into trials, it is necessary to indicate where the 'time 0' is, which is the sample with the event that was segmented on (usually interpreted as the baseline period). For case 2, the 'time 0' is indicated in the nSamplesPre field of the header data. Case 3 uses an event whose 'type' and 'value' fields are both set to 't0'.

Header

In case 1, the header gets 1 for nTrials, and nSamples is the total number of samples in the file.

In case 2, the header gets the actual number of trials for nTrials, and nSamples is the number of samples per trial. So, the total number of samples is always nTrials x nSamples.

Case 3 is treated like case 1.

Summary

1. For continuous recordings that have recording breaks in the middle:

- Epoch breaks are all indicated with events with 'break cnt' for the type, and 'epoch' for the value, and duration of the epoch in samples for the 'duration' field.

- The header gets 1 for nTrials, and nSamples is the total number of samples in the file.

2. For files that are cleanly segmented into trials:

- Epoch breaks are all indicated with events with 'break seg' for the type, and the name of the condition for the value, and duration of the epoch in samples for the 'duration' field.
- The header gets the actual number of trials for nTrials, and nSamples is the number of samples per trial.

3. For files that are segmented into trials, but the trials have different lengths, or different time zeros:

- Epoch breaks are all indicated with events with 'break var' for the type, and the name of the condition for the value, and duration of the epoch in samples for the 'duration' field.
- The header gets 1 for nTrials, and nSamples is the total number of samples in the file.
- For each trial, the time 0 is indicated with an event with 't0' for both the 'type' and 'value' fields.

4.3. Data-Writing Functions

The data-writing functions are designed to use he FieldTrip data structures (event, data and header), although they aren't necessarily callable from FieldTrip. They consist of the following routines:

- **write_mff_data:** writes a 2-D matrix of size Nchans*Nsamples as described at http://fieldtrip.fcdonders.nl/reference/ft_read_data.
- **write_mff_event:** writes an array of event structures described at http://fieldtrip.fcdonders.nl/reference/ft_read_event.
- **write_mff_history:** adds an entry to the history structure of the MFF file so a record of the processing you have done is embedded in the file, can be seen using Net Station's File-History pane.

Each of these routines are described in more detail below.

function write_mff_data(srcMFFPath, dstMFFPath, newData, hdr)

Writes channel data (newData) in newData to an MFF file. NewData is a 2-D matrix of size Nchans*Nsamples as described at http://fieldtrip.fcdonders.nl/reference/ft_read_data.

If dstMFFPath is empty, or the same as srcMFFPath, then the data gets overwritten, otherwise, the srcMFFPath is copied to dstMFFPath, and the data is written to dstMFFPath. In the later case, hdr is ignored.

hdr – FieldTrip header. You have the option of passing in the header, or [ ]. If you pass in the header, it pulls data out of it, rather than recomputing them.

This function has the following limitations: It requires a source data file (srcMFFPath), in other words, it doesn't allow you to create synthetic data and create a new MFF file. Also, it assumes that the data is in the same structure as the srcFile in terms of number of channels, number of samples, number of epochs and sizes of epochs. So, it doesn't support processes that change any of those items, for example, channel downsampling, or segmentation. Future releases of this API will allow more flexible data-writing.

This function doesn't modify the MFF file's history data. To do that, call mff_write_history, described below.

function write_mff_event(filePath, trackName, events, replace, hdr)

Note that the calling syntax for this function has changed from the previous version.

Writes an events structure (as described at http://fieldtrip.fcdonders.nl/reference/ft_read_event) to an event-track file in an existing MFF file (filePath).

filePath – The path to the .mff file.

trackName – The name of the track as it will appear in Net Station.

replace – a boolean that indicates what the code should do if the track already exists. If set to 'true', the existing track will get overwritten. If set to 'false', the code will end with an error message.

hdr – FieldTrip header. You have the option of passing in the header, or [ ]. If you pass in the header, it pulls data out of it, rather than recomputing them.

This function doesn't modify the MFF file's history data. To do that, call mff_write_history, described below.

function write_mff_history(filePath, entryStruct)

This function adds an item to the list of history items. It is intended to be run after running code that either a) creates a new mff file based on an existing one, or b) modifies an existing one. The history item can be seen in Net Station through the File-Info History feature.

filePath – The path to the .mff file.

The entryStruct has the following elements:

- name - string that corresponds to the name of the tool specification in NetStation. For example, if you create a segmentation specification for a VTD experiment, and call it "VTD Seg", then the string "VTD Seg" would go here.
- method - The name of the tool (not the tool specification.) In the above example, the string "Segmentation" would go here.
- version - The version of the tool, eg "1.0".
- beginTime - The time the tool started running. The following matlab command creates a time in the proper format:

```
sprintf('%d-%02d-%02dT%02d:%02d:%02.5f',clock);
```

- endTime - The time the tool finished running. See beginTime, above, for the matlab command that creates a time in the proper format.
- sourceFileList - A cell array of zero or more strings for the files that were processed by the tool to generate the resulting file. For example, a tool that filters a single file to create a filtered version of the data would have just one item in this array. A tool that creates a grand average by averaging multiple single-subject files would have multiple items in this array.
- settingList - A cell array of zero or more strings that express the settings/parameters for the tool. These are optional, and are intended to be read by humans, not machines.
- resultList - A cell array of zero or more strings that express the results of the tool. These are optional, and are intended to be read by humans, not machines.

5. The Demo Code

We have provided some example code files to serve as tutorials for using the Matlab MFF API. The example files are briefly described below.

These three functions provide simple examples of Matlab code that read and write MFF files, and do simple processing:

- negTheFile.m – multiplies the data by -1, and writes out the results to a new file.
- addHelloWorldEvents.m – adds a couple of events to an existing file.
- filterWithEEGLAB.m – filters the data using EEGLAB's eegfilt function. This function assumes EEGLAB is on the Matlab path.

These three functions wrap the above three functions so they can be called with the Net Station Matlab Tool described in Section 7 below:

- negTheFileNS.m
- addHelloWorldEventsNS.m
- filterWithEEGLABNS.m

These three functions are for converting between MFF and EEGLAB's .set formats:

- eeglab2mff.m
- mff2eeglab.m – given a path to an MFF file and an output filename, this function converts the MFF file to an EEGLAB .set file whose name is the output filename. It calls mff2eeglabStruct below.
- mff2eeglabStruct.m – given a path to an MFF file, this function returns an EEGLAB EEG structure.

These functions demonstrate additional elements about the Matlab MFF API:

- displayMFFFileInfo.m – displays information about the MFF file from the header structure and plots some data.
- demoMFFRelease.m – cycles through the sample data provided with the Matlab MFF API, and calls displayMFFFileInfo, negTheFile, filterWithEEGLAB, and addHelloWorldEvents for each file.

6. Segmentation Markup

This section assumes you are familiar with Net Station Segmentation, and the structure of a simple standard/target experiment.

Net Station events contain key lists, in other words, mini databases. Using a simple standard/target experiment as an example, in Net Station all stimuli events might be named "stim." The distinction between standards and targets can't be determined from the name of the event. It can only be determined from the key list in the events. In most systems, including FieldTrip, events only have names. They don't have key lists.

To make the jump from key-list events to non-keylist events, you must use Net Station's segmentation markup. Segmentation markup adds events to the recording that you can use to distinguish different conditions. To use segmentation markup, create a segmentation specification, and check the "Mark Up File" checkbox in the segmentation specification editor. When you run segmentation using this specification, instead of segmenting the file, new events will be added.

For example, if all your standard and target stimuli are named "stim", you would create a segmentation specification just as you would for segmenting this file into standard and target categories. Then, if you check the "Mark Up File" checkbox, the specification will cause new events to be added to the file instead of segmentation. Then, you can add events called "stnd" for standard, and "targ" for target.

Note: Although Net Station allows you to add markup events with spaces in the names, it might cause unpredictable results in other programs. Also, Net Station generates the event names automatically, but you can modify them. Sometimes the automatically generated ones contain spaces. Simply edit them, for example, replace the spaces with underscore characters.

The objective is to be able to do distinguish different conditions based on event names. So, before reading events with the Matlab MFF API, use segmentation markup to add all the events you might want to use. Although a simple standard/target experiment was used in this example, you can combine the full power of Net Station's segmentation with the segmentation markup feature.

7. Net Station Matlab Tool

Net Station 5 allows the user to create MATLAB functions that can be called as tools in Net Station.

In order for a Matlab function to work as a Net Station Matlab Tool, it must be structured as follows:

- If takes one MFF file as an input, and creates a new MFF file as an output.
- It takes exactly two parameters: the path to the input file and the path to the output file.
- If the output file is created with the write_mff_data function, then it must have the same structure as the input file (see the discussion above on the write_mff_data function).
- Any functions that are called by your function (e.g. Matlab MFF API, EEGLAB functions, functions you wrote) must be on Matlab's path.
- It initializes the Matlab MFF API Java library with a call to mff_setup.
- The function must end by dismissing Matlab with the exit command.

Future versions of this tool will have more flexibility.

If you have Matlab code that does not conform to the above structure, you can use it by writing a wrapper function that does conforms, and that calls your code. The demo code provided with the Matlab MFF API includes examples of this.

Because of the above limitations, a good candidate function for use with the Net Station Matlab Tool is a function that adds events to a file or a function that filters or cleans your data.

Below are the steps to create and use the MATLAB tools.

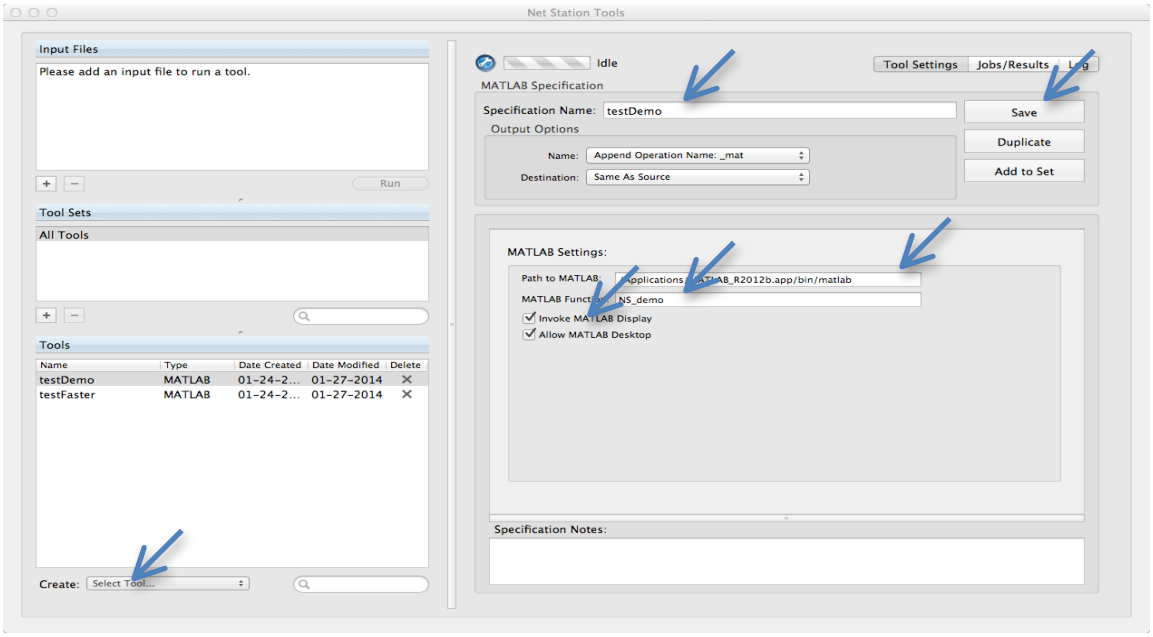
1. Write the Matlab function.

2. Create the MATLAB tool in Net Station

- Launch Net Station Tools application.
- From the "Create" drop-down list, click "Matlab".
- Enter a name for the tool.
- Enter the path to the MATLAB executable. (Note that in recent versions of Matlab, Matlab.app is a package file (a directory hidden as a file) and the

executable is bin/matlab relative to the package file. So “bin/matlab” must be appended to the path to Matlab.app.)

- Enter the full path to the MATLAB tool function. The filename, rather than the entire path may be specified if the tool function is in the default Matlab directory.
- Click “Save”. Check “Invoke MATLAB Display”.
- Check “Allow MATLAB Desktop” to allow MATLAB desktop display.

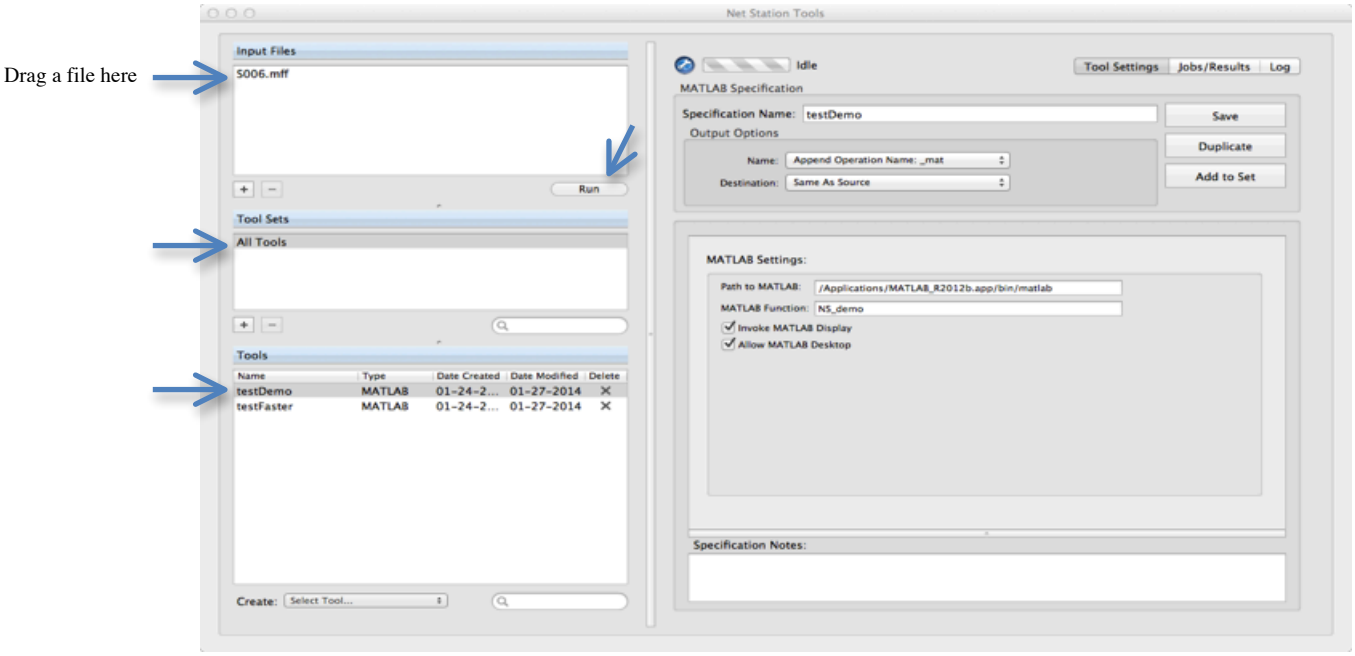


3. Use the MATLAB tool

- Launch Net Station Tools application.
- Click “All Tools”.
- Select the MATLAB tool to be used.
- Drag or add a MFF file to the “Input Files” box.
- Click “Run”.

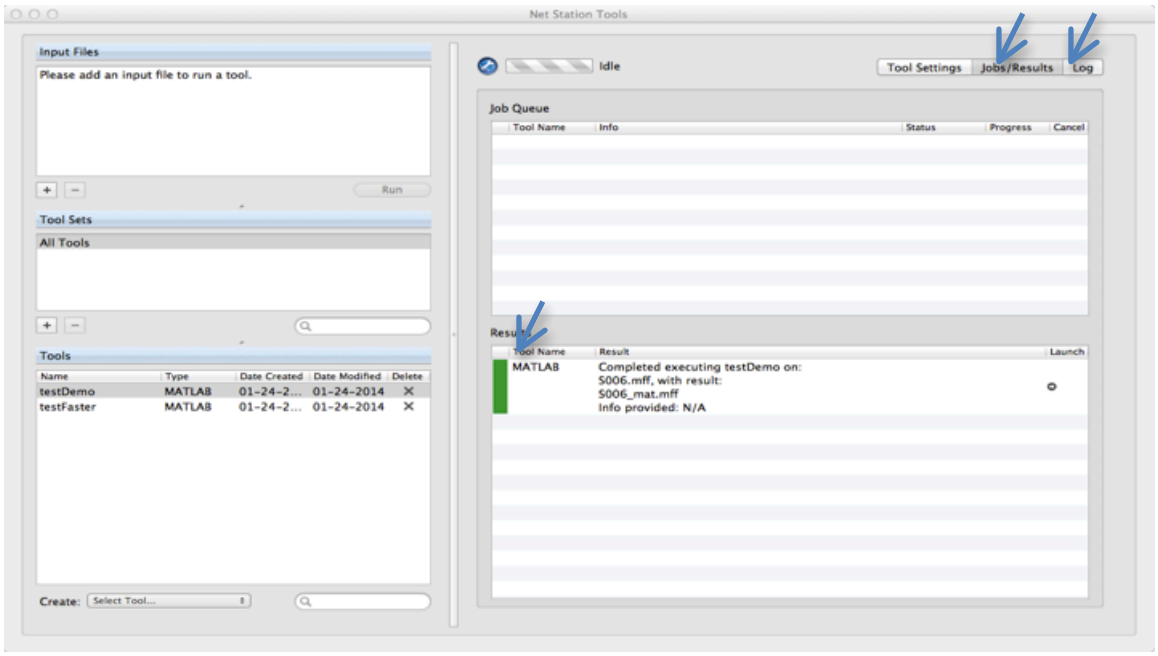
Matlab will launch, and the Net Station Tools will remain in the “Working” state until the tool is finished.

If your Matlab code is unable to finish (e.g. if it errors out), the Net Station Tools will remain stuck in the “Working” state. You can fix this situation by manually quitting the Matlab session.



4. Check the status of the MATLAB tool

If the MATLAB tool finished running successfully, a green bar will be shown in the "Jobs/Results" section. Click "Log" to review additional information.



Appendix: Testing

While it's not possible to guarantee that any software is 100% bug free, this section briefly describes the testing that was done for this release of the Matlab MFF API.

The demoMFF3Release script, provided with this software was run on the three computers described below. This involves the five datasets included with the release, which include in various combinations:

- Continuous data & segmented data
- Data with and without PIB channels
- Single-subject and multi-subject data

In addition, a much more detailed test script was run that used the above five data sets plus four others. This script was also run on the three computers described below. This test script:

- In the case where we have data from older MFF versions that were converted to the current version, the script reads the old data with the previous Matlab MFF API, and the converted data with the current Matlab MFF API, and compares them.
- Does basic data integrity testing.
- Compares the data to data previously read with earlier releases of this code as a regression test.
- Randomly picks sample ranges and sensor ranges and reads those subsets of data, and compares the results to the same sample and sensor ranges when reading the entire dataset.

Theoretically, the Matlab MFF API should work on all computers with all versions of Matlab, EEGLAB and FieldTrip. However we did the testing described above on three different computers, including two Macs and one Windows PC. The testing spanned three operating systems, and two different versions of Matlab. The specific details on the systems we tested on are provided in the tables below, generated by Matlab's "ver" command.

```
MATLAB Version 7.11.0.584 (R2010b)
MATLAB License Number: XXXXXX
Operating System: Mac OS X  Version: 10.6.8 Build: 10K549
Java VM Version: Java 1.6.0_65-b14-462-10M4609 with Apple Inc. Java HotSpot(TM)
64-Bit Server VM mixed mode

MATLAB                                Version 7.11       (R2010b)
ERP PCA Toolbox                         Version 2.23
FastICA for Matlab 7.x and 6.x              Version 2.5,       October 19
FieldTrip                           Version unknown    fieldtriptoolbox.org
Image Processing Toolbox                Version 7.1        (R2010b)
```

Signal Processing Toolbox	Version 6.14	(R2010b)
Statistical Parametric Mapping	Version 8.3	(SPM8)
Yokogawa MEG Reader toolbox for MATLAB	Version 1.04.01	

MATLAB Version: 8.2.0.701 (R2013b)
MATLAB License Number: 261386
Operating System: Mac OS X Version: 10.9.1 Build: 13B42
Java Version: Java 1.7.0_11-b21 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode

MATLAB	Version 8.2	(R2013b)
Image Processing Toolbox	Version 8.3	(R2013b)
Signal Processing Toolbox	Version 6.20	(R2013b)

MATLAB Version: 8.2.0.701 (R2013b)
MATLAB License Number: 261386
Operating System: Microsoft Windows 7 Version 6.1 (Build 7601: Service Pack 1)
Java Version: Java 1.7.0_11-b21 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode

MATLAB	Version 8.2	(R2013b)
Image Processing Toolbox	Version 8.3	(R2013b)
Signal Processing Toolbox	Version 6.20	(R2013b)