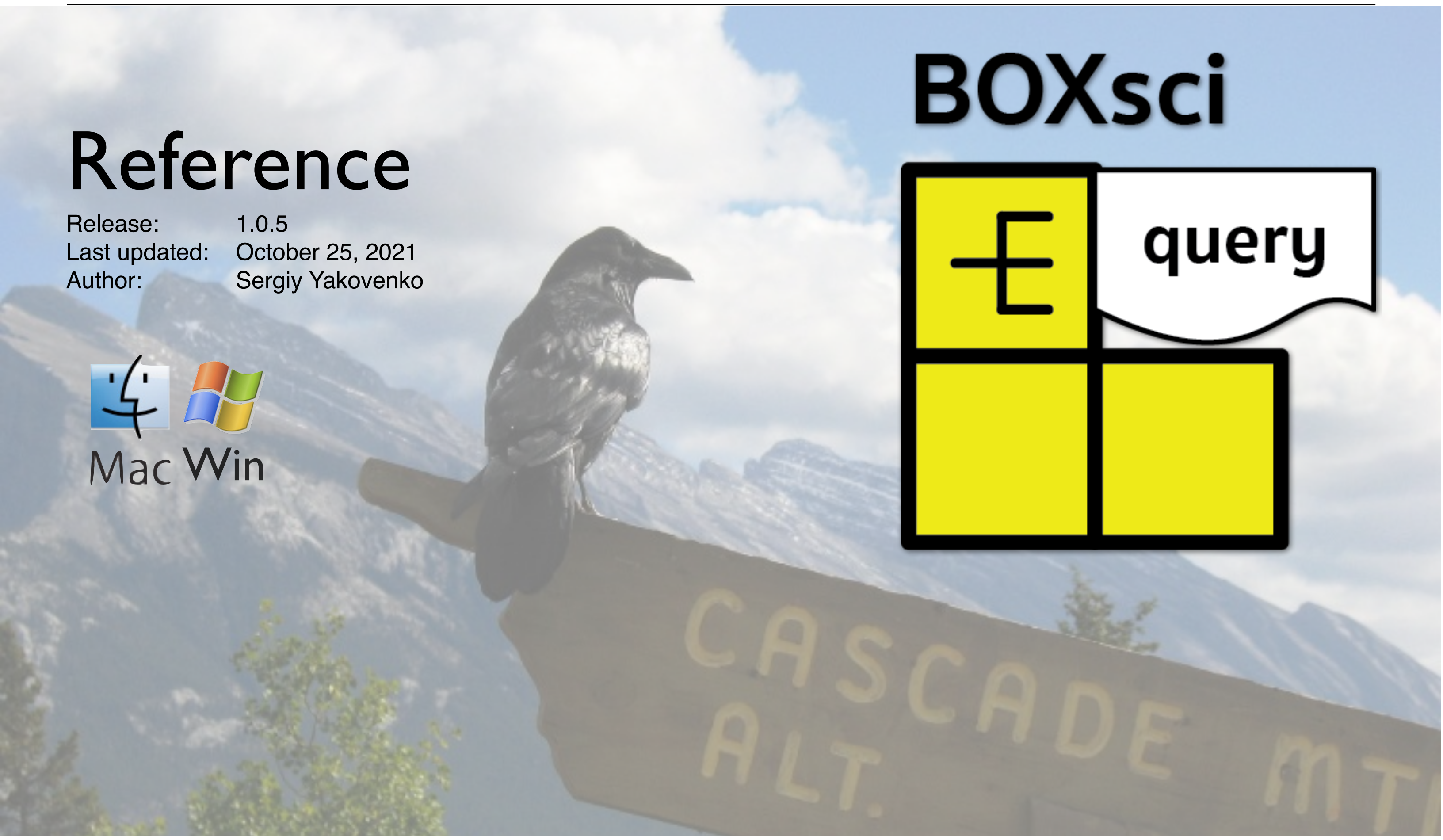


BOXsci

query

Reference

Release: 1.0.5
Last updated: October 25, 2021
Author: Sergiy Yakovenko

Mac Win

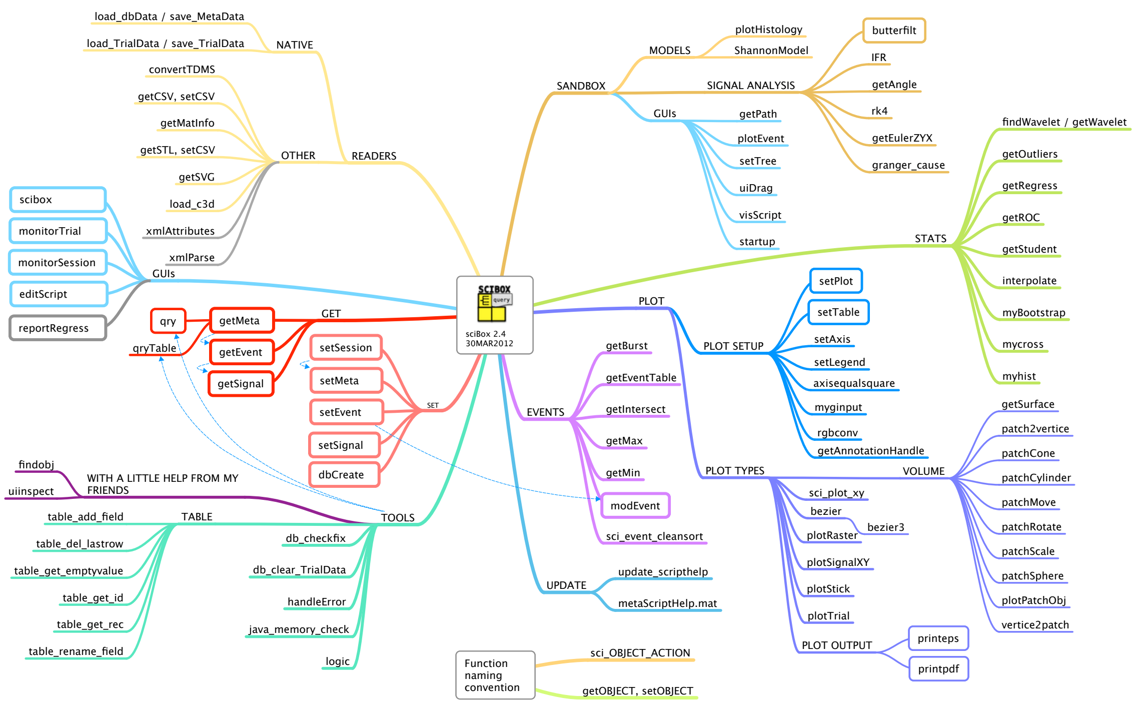
GitHub: https://github.com/Neural-Engineering-Laboratory/boxsci.git
Web: https://sites.google.com/site/matboxsite/

Table of Contents

1. Introduction

The **relational model** for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar F. Codd. The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries. (source: http://en.wikipedia.org/wiki/Relational_model)

Figure 1.1. The complete layout of boxsci functions where the backbone scripting functions are highlighted with box outlines.



Contents of QuickStart

- Quick Q&A
- Installation Instructions
- GUIs at a Glance
- Scripting Cascade

Q: What is Boxsci for Matlab?
A: Boxsci is a relational database for analysis and storage of neurophysiological/biomechanical/biophysical data implemented *fully* in Matlab. Additional licenses for other toolboxes, external software or SQL skills are **not** required. All you need is a copy of basic Matlab and rudimentary knowledge about data types.

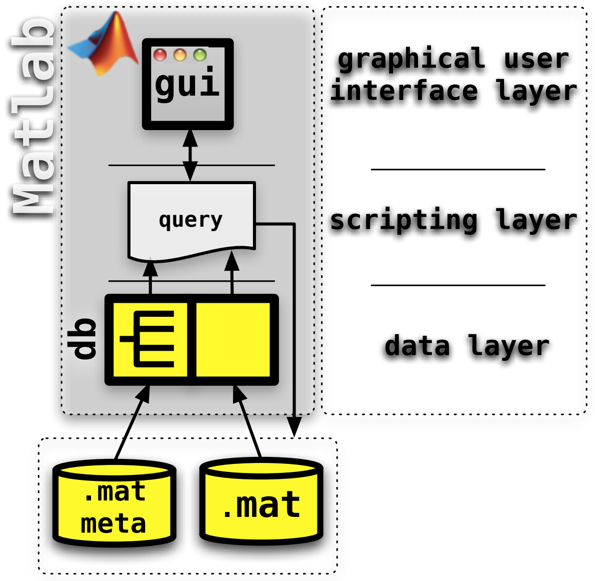


Figure 1.2. **Workflow schematic of Boxsci.** The application consists of several transparent levels of access. The relational database is represented by a global structure in workspace. It is dynamically manipulated to load and analyze data stored in MAT files.

Q: What are the features of Boxsci?
A: Boxsci simplifies access to complex scientific data sets. It allows three levels of data access (see Fig.1.2):

1. The data layer is a low-level access to the database structure containing <*meta*> and <*signal*> tables with standard Matlab data inspectors, which allow independent from SciBox data manipulation for users with advanced programming skills. Any additional <*meta*> information that satisfies few organizational constraints can be appended directly to the database structure and becomes part of it.

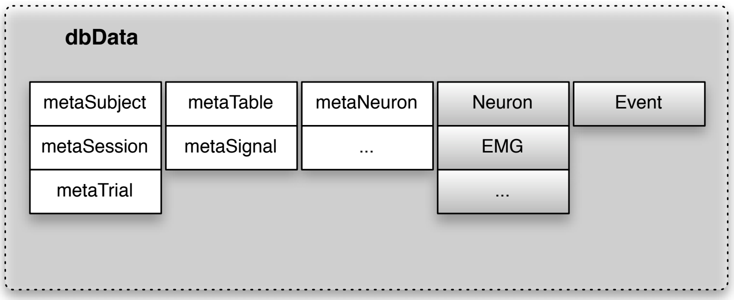
2. The scripting layer allows data queries that can execute on-the-fly signal processing analysis that accepts any Matlab script, and plots intermediate and final results. The group of essential query scripts consists of only three functions **getMeta**, **getEvent**, **getSignal**. These functions reflect three basic steps in data analysis flow of standard analysis of neurophysiological data (see Examples of Data Analysis of two projects below: the studies of the motor cortex contribution to movement control in cats and humans from two laboratories in University of Montreal).

3. The graphical user interface (**GUI**) layer allows users without programming skills to access and analyze data in two typical views: a trial-by-trial view of multiple signals or a multiple sessions with multiple trials view of several signals. In these data views, events and signals can be manipulated manually with a drag-and-drop interface or with automatic event detection scripts, e.g. see example of the automatic burst detection during locomotion (#.#). Additional report generators enable users to perform typical analyses on the whole database, e.g. averages of signals, raster plots of neuron spiking, temporal and magnitude regressions.

2. dbData • Low-level Access

dbData is the main database variable. *dbData* is a structure array with dynamically loaded on-demand information about recorded analog and digital signals in *meta* & *signal* tables.

dbData variable in Matlab Workspace contains the following tables:



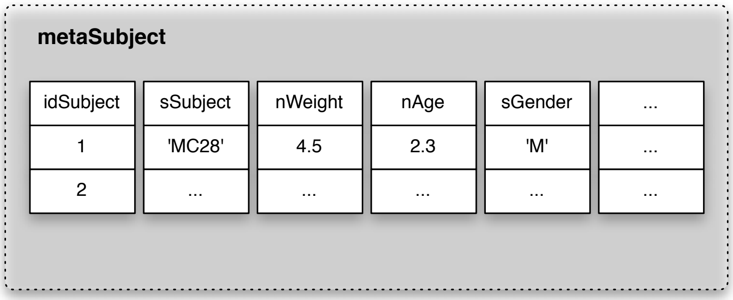
Tables	**Table Type**	Description of Fields
meta *	meta	id, sTable,sDescription, nHierarchy,sType
metaSubject	meta	idSubject, sSubject, sPrefix
metaDate *	meta	idDate, nDate, sDate, nTrack
metaSession	meta	idSession, idSubject, idDate, idFileRaw
metaTable	meta	idTable, sTable, bTable, sTableInfo
metaSignal	meta	idSignal, idTable, sSignal, nRate
metaTrial	meta	idTrial, idSession, idSubject, idTrialType, ...
metaEvent	meta	idEventType, sEvent, ...
metaTrialType	meta	idTrialType, sTrialType, ...
metaSync	meta	synchronization time for each data table and each trial. Fields: idTrial, .(sTable)=tSync
metaFile	meta	idTrial, sFile, sPath
metaFileRaw *	meta	idFile, sFile, sPath
metaLoad	meta	boolean flag of loaded data for each trial. Fields: idTrial, .(sTable).(sSignal)=true/false
Event	signal	tEvent, idEventType, idSignal

(sTable)	signal	.(sSignal){idTrial} = nDataTrial;
		for example
		dbData.EMG.TibialisAnterior{idTrial} = [1 2 3]

* optional tables

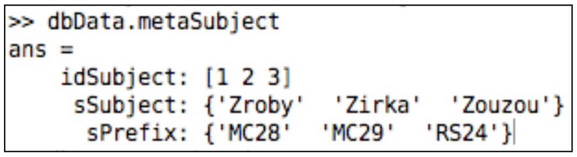
metaSubject

Describes information about subjects. A **subject** is a hierarchical parent of **session**, which is a parent of **trial**.



metaSubject

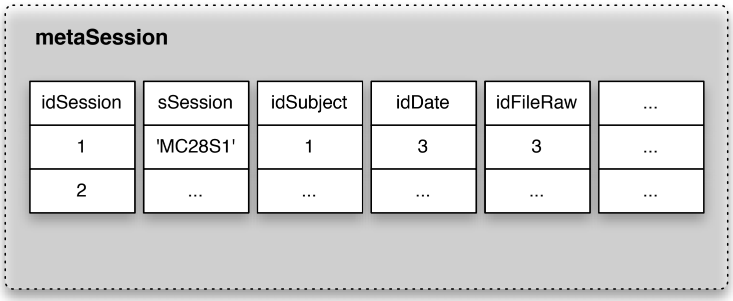
idSubject	sSubject	nWeight	nAge	sGender	...
1	'MC28'	4.5	2.3	'M'	...
2	...	...	...	...	...



```
>> dbData.metaSubject
ans =
    idSubject: [1 2 3]
     sSubject: {'Zroby'  'Zirka'  'Zouzou'}
      sPrefix: {'MC28'  'MC29'  'RS24'}
```

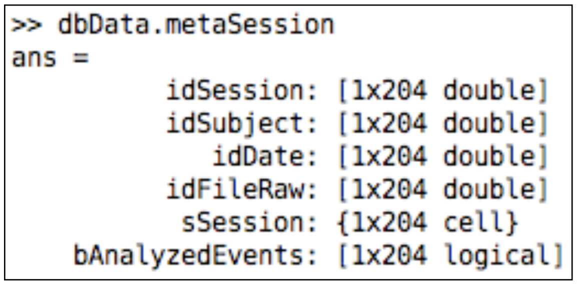
metaSession

Describes information about sessions. The required fields are idSession, sSession (used for naming files and GUI access). For example:
sSession = 'Subject1May12Session3' or sSession = 'MC29T65R2'.



metaSession

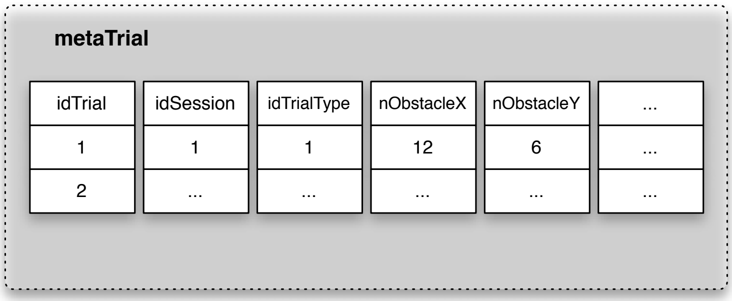
idSession	sSession	idSubject	idDate	idFileRaw	...
1	'MC28S1'	1	3	3	...
2	...	...	...	...	...



```
>> dbData.metaSession
ans =
          idSession: [1x204 double]
          idSubject: [1x204 double]
             idDate: [1x204 double]
          idFileRaw: [1x204 double]
           sSession: {1x204 cell}
     bAnalyzedEvents: [1x204 logical]
```

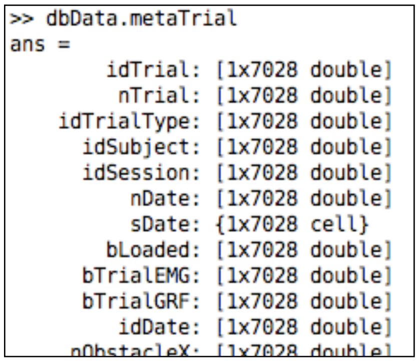
metaTrial

The information in this main meta table describes each trial. It is recommended to keep as much meta data as possible at this low level in the hierarchy to streamline queries.
Required fields: idTrial, idSession, idTrialType.



metaTrial

idTrial	idSession	idTrialType	nObstacleX	nObstacleY	...
1	1	1	12	6	...
2	...	...	...	...	...



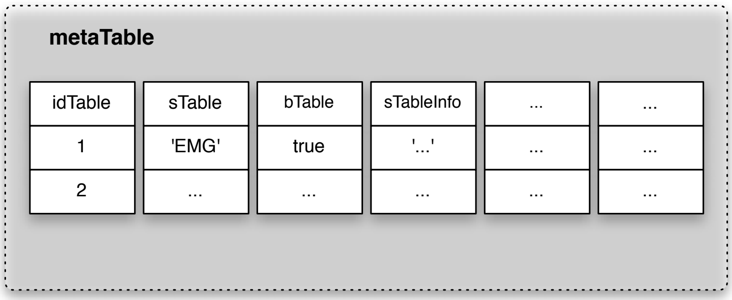
```
>> dbData.metaTrial
ans =
        idTrial: [1x7028 double]
         nTrial: [1x7028 double]
    idTrialType: [1x7028 double]
      idSubject: [1x7028 double]
      idSession: [1x7028 double]
          nDate: [1x7028 double]
          sDate: {1x7028 cell}
        bLoaded: [1x7028 double]
      bTrialEMG: [1x7028 double]
      bTrialGRF: [1x7028 double]
         idDate: [1x7028 double]
     nObstacleX: [1x7028 double]
```

metaTrialType

The information about different tasks in a session is kept in this table. Each trial type could be repeated in multiple trials recored in metaTrial table. For example, the separate behavioral tasks of center-out reaching to multiple targets can be listed as separate TrialTypes with stated properties, e.g., directions, speed, limb, etc. However, if the database is small then this information could be inserted into metaTrial (not recommended for large databases).
Required fields: idTrialType, sTrialType.

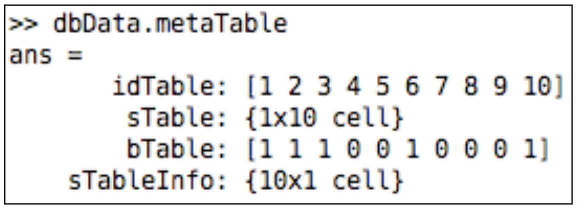
metaTable

The list of tables in the database is described in metaTable. These tables organize signals usually recorded with the same DAQ with the same sampling rate. For example, electromyograms recorded with two different systems (Delsys with 4kHz, Myoband with 500Hz) might be convenient to place into two different tables.
Required fields: idTable, sTable, bTable.
bTable is a boolean flag that indicates if a specific table is used in this database.



metaTable

idTable	sTable	bTable	sTableInfo	...	...
1	'EMG'	true	'...'	...	...
2	...	...	...	...	...



```
>> dbData.metaTable
ans =
       idTable: [1 2 3 4 5 6 7 8 9 10]
        sTable: {1x10 cell}
        bTable: [1 1 1 0 0 1 0 0 0 1]
    sTableInfo: {10x1 cell}
```

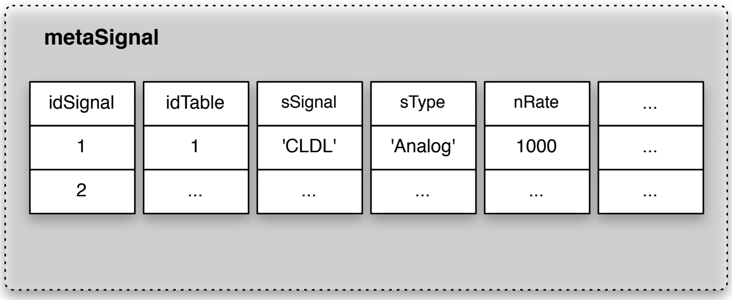
metaSignal

Signal information in this meta table describes all available signals in the database. Note that signals are organized in tables defined by user. For example all EMG signals could be combined in one table *'EMG'* or in several tables defined by user's preference, e.g. DELSYS and MYOBAND EMGs in *'EMG_DELSYS'* & *'EMG_MYO'*. Simpler is better, so then keep signals with the same characteristics in one table.

Required fields: idSignal, idTable, sSignal, nRate, nDim.

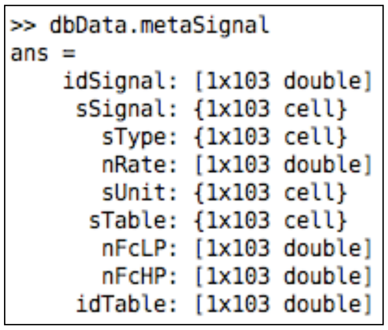
nRate is the sampling frequency in Hz

nDim is the dimensionality of data (e.g. 1 for regular signals, and 3 for kinematics in vector form)



metaSignal

idSignal	idTable	sSignal	sType	nRate	...
1	1	'CLDL'	'Analog'	1000	...
2	...	...	...	...	...



```
>> dbData.metaSignal
ans =
    idSignal: [1x103 double]
     sSignal: {1x103 cell}
       sType: {1x103 cell}
       nRate: [1x103 double]
       sUnit: {1x103 cell}
      sTable: {1x103 cell}
       nFcLP: [1x103 double]
       nFcHP: [1x103 double]
     idTable: [1x103 double]
```

metaEvent

Description of event types and their properties. The main types of events are:

tStamp - time stamp event

tSpike - time stamp of spike (restrictions and special plotting is applied)

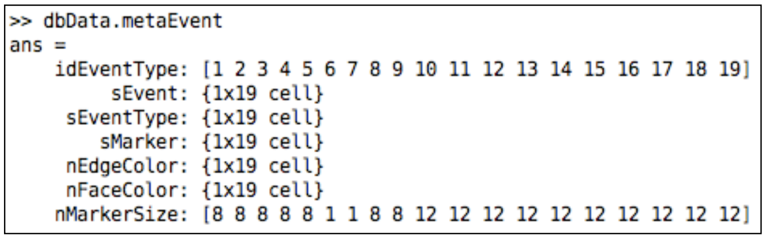
tPeriod - time stamp with a non-zero duration

tText - time for a text note

idEventType	sEvent	sEventType	sMarker	nEdgeColor	nFaceColor	nMarkerSize
1	'on'	'tStamp'	'>'	[1 0 0]	[1 0 0]	6
7	'cycle'	'tPeriod'	''	[0 0 0]	[0 0 0]	6

```
% metaEvent
dbData.metaEvent.idEventType = [1 2 3 4 5 6 7];
dbData.metaEvent.sEvent = {'on','off','peak','trough','special','spike1','cycle'};
dbData.metaEvent.sEventType =
{'tStamp','tStamp','tStamp','tStamp','tStamp','tStamp','tPeriod'};
```

```
dbData.metaEvent.sMarker = {'>','<','^','v','*','.',''};
dbData.metaEvent.nEdgeColor = {[1 0 0],[0 0 1],[0 1 0],[0 1 0],[1 1 0],[0 0 0],[0 0 0]};
dbData.metaEvent.nFaceColor = {[1 0 0],[0 0 1],[0 1 0],[0 1 0],[1 1 0],[0 0 0],[0 0 0],
[0 0 0]};
dbData.metaEvent.nMarkerSize = [6 6 6 6 6 6 6 6];
```



```
>> dbData.metaEvent
ans =
    idEventType: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]
         sEvent: {1x19 cell}
     sEventType: {1x19 cell}
        sMarker: {1x19 cell}
     nEdgeColor: {1x19 cell}
     nFaceColor: {1x19 cell}
    nMarkerSize: [8 8 8 8 8 1 1 8 8 12 12 12 12 12 12 12 12 12 12]
```

Note: Additional events can be added by appending UserDefined values to fields above.

metaFile

This meta table contains information about where analog data is located. When importing new trials make sure that the path is stated for the first trial in the database, which will be used by default when sPath is not stated:

dbData.metaFile.sPath{1} = '/Users/sergiy/Projects/Motor Cortex/dbCortex/data';

The default format of saved files is as follows:

sFile = [dbData.metaSession.sSession{idSession},'_Trial',num2str(idTrial),'.mat'];
% e.g. sFile = 'MC28T6R1_Trial23.mat'

metaSync

This meta table contains information about synchronization between different tables. Often different modalities of signals, e.g. EMG, force, and kinematics, are recorded with different acquisition boards with respect to a synchronization event. The synchronization between different signal modalities is stored in META table metaSync as the temporal difference in acquisition onset. The general form of this table is:

```
dbData.metaSync.(sTable).(sSignal)(idTrial) = tSync;
```

For example, EMG signals in trial `348203` are preceding the sync event by `2.815`:

```
dbData.metaSync.EMG(348203) = 2.8150
```

Note that these values are all zero for tables recorded with the same acquisition card.

metaLoad

This table keeps track of loaded tables. It contains the following fields: idTrial, EMG, Force, …, (sTable). When importing data tables (sTable) of a set of trials (idTrialList), set the following fields to true:

    dbData.metaLoad.(sTable)(idTrialList) = 1;

This operation should be done only during data import to indicates to internal functions that the data is loaded. Once this information is available within dbData all data management (LOAD/UNLOAD/SAVE) can be done using UIs. For example, to save all imported trials follow the checklist:

- set loaded state of imported trials in dbData.metaLoad.(sTable)(idTrialList) = 1;
- check that dbData.metaFile.sPath points to the desired save path
- using Boxsci-Main Window click on *Save Database and Data Signals in Trials*

metaDate (optional)

Describe date information.
Required fields:
idDate, sDate

meta (optional)

This table helps to keep track of additional tables in the database. nHierarchy fields might be used for vertical queries in the future versions of Boxsci.

4. Data Tables • Low-level Access

The database is designed to accommodate both analog and discrete data types.

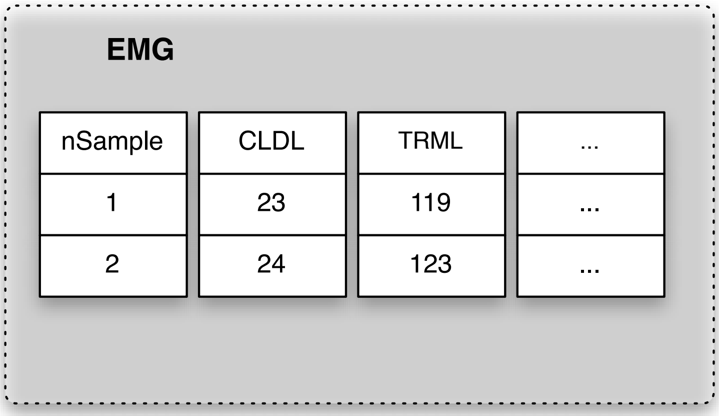
Analog Data Tables

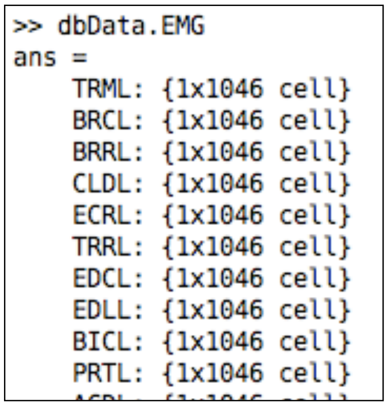
The analog data tables in the database are described in *metaTable*.They can be accessed by querying with the following script:

```
sTable = 'EMG'; sSignal = 'CLDL'; idTrial = 1;
dbData.(sTable).(sSignal){idTrial}
```

As can be seed from the previous line individual signals are stored in *cells* that correspond to trials in the database. New trials can be added by assigning trial data to:

```
dbData.(sTable).(sSignal){idTrial} = single(nDataTrial);
```





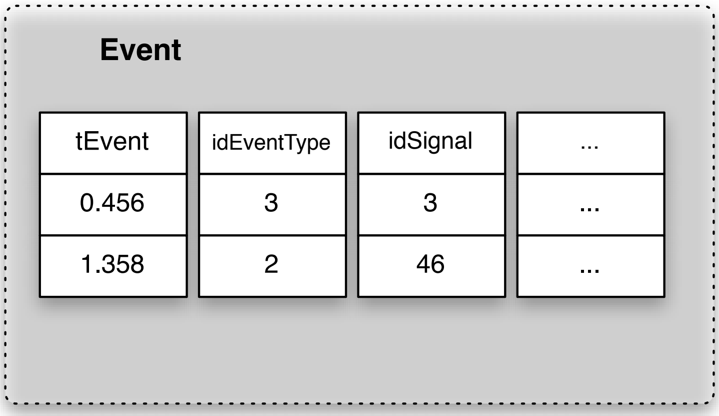
```
>> dbData.EMG
ans =
    TRML: {1x1046 cell}
    BRCL: {1x1046 cell}
    BRRL: {1x1046 cell}
    CLDL: {1x1046 cell}
    ECRL: {1x1046 cell}
    TRRL: {1x1046 cell}
    EDCL: {1x1046 cell}
    EDLL: {1x1046 cell}
    BICL: {1x1046 cell}
    PRTL: {1x1046 cell}
```

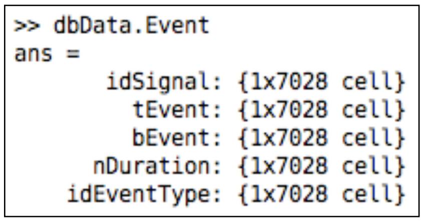
Discrete Data Table: Event

tEvent	idEventType	idSignal	bEvent	nDuration
0.024	1	5	1	0
-0.735	7	6	0	0.53

where

- *tEvent* contains numeric value in sec of event occurrence within a trial,
- *idEventType* refers to a set type of the event defined in metaEvent, eg. idEventType=7 refers to event type CYCLE that has a non-zero nDuration value.
- *bEvent* marks events selected for deletion (bEvent = false),
- *nDuration* contains numeric value in sec of event duration. It can also be used as an additional field to store UserData in events that are not type tPeriod, e.g. tStamp.





```
>> dbData.Event
ans =
       idSignal: {1x7028 cell}
         tEvent: {1x7028 cell}
         bEvent: {1x7028 cell}
      nDuration: {1x7028 cell}
    idEventType: {1x7028 cell}
```

5. Graphics objects & manipulation functions

The database supports 3D patch objects to support anatomical/histological reconstructions, kinematic animations, and dynamic simulations.
Status: * HARD-HATS *

Data importers/exporters
(•) svg2mat
(•) load_stl, save_stl

Visualization
(•) Common patch objects: patchCone, patchCylinder, patchSphere() (- add patchBox)

Manipulation
(•) patchMove, patchScale, patchRotate
(•) patch2vertice, vertice2patch

Example
```
figure
hold on
hCPG1 = patchSphere('facecolor','b');
hCPG2 = patchSphere('facecolor','r');
hCPG3 = patchSphere('facecolor','b');
hCPG4 = patchSphere('facecolor','r');

co{1} = [-2,0,0];
co{2} = [-2,-4,0];
co{3} = [2,0,0];
co{4} = [2,-4,0];

patchMove(hCPG1,co{1})
patchMove(hCPG2,co{2})
patchMove(hCPG3,co{3})
patchMove(hCPG4,co{4})
```

6. Appendix

Notes on importing data manually (without setMeta, setSession, setEvent, setSignal)

Example 1

Developing a functional importer for your data is the most difficult step simplified by the importer functions (see Section 5 above). Once the data is imported in Matlab Workspace the rest can be handled with simple-to-use Boxsci importers following these steps:

1. Import data to Matlab Workspace.
2. Initialize dbData with dbCreate :
   a. run command dbData=dbCreate
   b. note that empty.dbData.mat file is automatically saved in the current directory
3. Create new SUBJECT and SESSION entries with setSession.
4. Import continuous signals and create new TABLES and SIGNALS using setSignal.
5. Add events with setEvent.
6. Add other information with setMeta.
7. Save database by setting bSave flag in setSignal to TRUE while importing data.

```
Use the following steps
idTable = find if metaTable already has this id assigned
if isempty(idTable)
        assign new idTable and fill in all the required fields
end
<continue to the next table>
```

```
% idSubject
idSubject = find(dbData.metaSubject.idSubject==idSubject, 1);
if isempty(idSubject)
   idSubject = numel(dbData.metaSubject.idSubject)+1;
   dbData.metaSubject.idSubject(idSubject) = idSubject;
   dbData.metaSubject.sSubject(idSubject) = {'Undefined'};
   dbData.metaSubject.sPrefix(idSubject) = {'MC X'};
end
% idSession
idSession = find(dbData.metaSession.idSession==idSession, 1);
if isempty(idSession)
   idSession = dbData.metaSession.idSession(end)+1;
   dbData.metaSession.idSession(idSession) = idSession;
   dbData.metaSession.idSubject(idSession) = idSubject;
   dbData.metaSession.idDate(idSession) = idDate;
   dbData.metaSession.idSession(idSession) = idSession;
end
```

Example 2

An example of a generic data importer is in EXAMPLES folder of Boxsci. However, if you would like to develop your importer from low-level commands follow this checklist to streamline the process. It is recommended to write an importer that cycles through single sessions and updates *meta* tables in the process.

1. Import data to Matlab Workspace.
2. Initialize dbData with dbCreate.m.
   a. run command dbData=dbCreate
   b. note that empty.dbData.mat file is automatically saved in the current directory
3. Update *meta* tables by descending down the hierarchy (see **Example 1**). Use setMeta for adding new values and creating new fields.
   a. find idSubject of metaSubject
   b. find idSession of metaSession
   c. find idTrial of metaTrial
   d. update auxiliary *meta* tables

4. Import *analog* & *discrete data* tables with a triple FOR loop Table > Signal > Trial (**see Example 2**).
   a. update metaTable, metaSignal tables as needed
   b. in the Trial for-loop update metaSync
   c. import *analog* signals with
dbData.(sTable).(sSignal){idTrial}=nData;
   d. import *discrete* signals with
dbData.Event.tEvent{idTrial}(iEvent)=tEvent;

…

5. Save database. Follow these step-by-step instructions to save data:
    a. Assign metaLoad.(sTable)(idTrialLoaded)=1;
    b. Check that metaFile.sPath{1} = sPathDesired;

e.g. execute the following command in Command Window:

```
>> dbData.metaFile.sPath{1} = '/Users/sergiy/Projects/Motor Cortex/dbCortex/data'
```

    c. Use toolbar in Boxsci • Main Window to save the database *with signals in trials*.

```
for iTable = 1:nTable
    % Update metaTable table
    ...
    for iSignal = 1:nSignal
        % Update metaSignal table
        ...
        % Define idTrialList
        for idTrial = idTrialList
            % Import Analog Data
            % Import Discrete Data
            % Update meta Tables
            ...
        end
    end
end
```

6. Known bugs

Java out of memory issue

Matlab has java memory leaks associated with GUIs, which may cause Out of Memory Error when generating hundreds of reports with many separate graphical objects, e.g. spikes, in a single Matlab session. The problem "disappears" when Java Memory assignment is increased from the default value in

File/Preferences…

General/Java Heap Memory

monitorTrial events with processing scripts

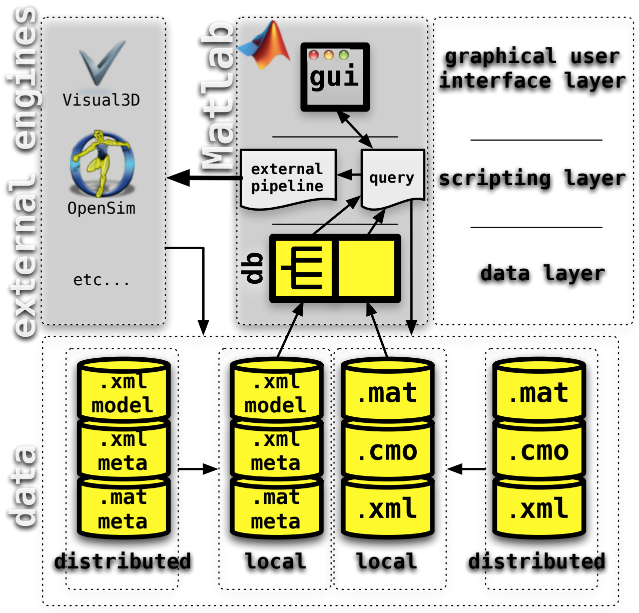
When a processing script is added to the signal in monitorTrial the dragged events are plotted with the unprocessed signals. The bug is not critical, but it is inconvenient.

Resolved bugs

see Release Notes in Boxsci folder for details

7. Future directions

boxsci development



Development of organization

1. metaSession information - different parameters change between sessions.
   1.1. nRate, nGain, etc.
2. On-demand access to META tables & Objects - save/load objects organized by trial and id.
   2.1. metaObject {id, Properties}, Object {id, Data}
   2.2. metaTable, metaLoad
3. Distributed trial information for import/export.
4. ...

Reference sections

1. Notes on scripting style.
2. Essential topics of Matlab.
    a. data types and structures, in particular
    b. basic arithmetics, e.g. a+b, mean(x)
3. Troubleshooting and bug reports.

8. GitHub interface through terminal

Terminal/shell commands

Using git commands in Terminal in .../boxsci folder:

```
git add file
git commit --message "my commit"
git push
```

Fetch and pull changes from the remote master/origin:

```
git checkout master
git pull
```