U.L.E.M.A. Tutorial for developers

This tutorial is for developers only.

How to add protocol parameters:

- Choose a name for new parameters.
- Open the file *Prot / ProtOptsDescr.m*. For each new parameter add a row to the cell matrix **d**. Set the first column as the parameter name, the second one as the processing section name, the third as the handle to a function able to compare the parameters (useful when paramters are not string or scalar number but vector, cell-arrays,...) and returning true or false.
- Modify the file *Prot / ProtDB.mat* by adding the new parameter values for each protocol.
- Open the function *Core / updateProtocolInfo.m* and create a convenience variable into handles to be futher used by the specific parameters editing sub-GUI, as follows for instance. This convenient structure can be custom; however, normally for values not in a list (in the sub-GUI) it should contain the parameter value itself in the filed *Value* , and for values in a list it should contain the element index in **Value**, and the list of items in **String**.

  handles.<proc-section>.<param-name>.Value = handles.protDB.protList(n).<param-name>;

- Edit the sub-GUI relative to a processing section to include the new parameters. Available sub-GUIs can be found inside the folder *SubGUIs*. The sub-GUI will use the convenience structure previously created, and it will return an updated version.
- In the file *Core / editProtocolParam.m*, in the correct section, add the following section for each new parameter:

```
ind = strcmp(d(:,1),<param-name>);
… extract parameter value from convenience structure …
test = feval(d{ind,3}, <value-from-convenience-struct>, handles.protDB.protList(n).<param-name>);
if test == 0
        somethingChanged = true;
        fprintf('\n\n<param-name> changed\n\n');
end
```

Variables available for processing functions:

The function *Core / runProcessing.m* calls all the function aimed at data processing / exportation. These are the input variables:

- **engineInfo, engineIdx**: for future use;
- **lastUsedDir**: empty string, for future use;
- **recoveryPath**: string, recovery path;
- **savePath**: string, path where to save MAT created by computation process;
- **protDB**: struct, content of the file Prot/ProtDB.mat
- **sectionsToComp**: struct, field names are alla sections, values are true (if section has to be processed) or false.
- **evConfig**: parsing of Seg/EventsConfig.txt (see Seg/parseConfigFile.m)
- **trToUseDB, trDB**: struct. The first one contains file names selected for processing in the main GUI. The second one contains all the file names. The two structures are very similar. Relevant fields are:

- subjects(i).name: name of subject i;
- subjects(i).sessions(j).name: name of session j for subject i;
- subjects(i).sessions(j).trials(k).name: name of trial k for sesion j and subject i;
- subjects(i).sessions(j).trials(k).protocolData. struct, contains protocol parameters for the trial. Content is the same as in *Prot / ProtDB.mat* / **protList(z)**. This field can always be used to get protocol settings for processing.

- **export**: struct, contains settings for exportation panel in the main GUI (see *mainGUI_OpeningFcn()* in *mainGUI.m*). Exportation settings are *not* stored in the protocol settings.
- **ageMatch**: cell-matrix, represents the content of the table *Subject match table* in *MAP options*. Empty or partially filled rows are filtered out (see *Core / cleanTableFromEmptyLines.m*)
- **refFileData**: struct, content of the reference file loaded in *MAP options*.
- **forceDataOverwriting**: bool, if force data overwriting is set into the main GUI.
- **splitProcDataLevels**: cell-array, represeting *Processing → Options → Split proc. data into .mat files for*. It can contain strings *subject*, *session*, *trial*.
- **procDataSaving**: string, method for data saving indicated by Processing → Options → Proc. data saving. It can be *overwrite* or *safe_merge*.

Other important variables:

- structure **subject**, created in the beginning of each subject loop in *Core / runProcessing.m*, contains all the processed data for a single subject. The struct is similar to the one of **trDBToUse**, but with processed data added, under **trials** and **sessions** fields.

<u>Main processing functions:</u>

These functions are all called directly from *Core / runProcessing.m*, apart from kinematics functions, called indirectly.

<u>Kinematics</u>:

- *Engines / BodyMech3.06.01/IProcKine_BM30601.m:*
  - (IN) see chapter "*Variables available for processing functions".*
  - The function fills the following fields to subject.sessions(j).trials(k):
    - static, staticRef: struct containing static file name, points data, acquisition frequency and frame used, respectively for defining technical reference frames (static file) and the reference posture for absolut angles (reference static file)
    - calib: struct-array, where each item contains name, points data (pointerand technical markers), and position in the technical reference frame, of an anatomical landmark
    - DJC: cell-matrix containing data relative to dynamic joint center calculation
    - data.points: struct containing all the technical markers and anatomical points for the trial
    - data.angles. struct containing all the angles data.for the trial

<u>Segmentation:</u>

- *Seg / startSegmentation.m:*
  - (IN) **data**: struct, with the following fields:
    - stParam.eventsRaw.eventTime: array of event instant (in seconds)
    - stParam.eventsRaw.eventType: cell-array of event types (e.g. HS, TO)
    - stParam.eventsRaw.eventSide: cell-array of event sides (e.g. Right, Left)

        - <data-section>.<data-curve>: struct containing data sections (e.g. points, angles) which contains data curves in time (rows represents time instants, columns data curves such as cooridnate x, y, z for a point).
    - (IN) **evConfig**: parsing of *Seg / EventsConfig.txt* (see *Seg / parseConfigFile.m*)
    - (IN) **firstLastOverlapped**: bool, true if two consecutive cycles share an event (last one for first cycle and first one for second cycle)
    - (IN) **mergeContexts**: deprecated, use 0
    - (IN) **contexts**: cell-matrix representing table *Contexts* in *Segmentation options* in the main GUI. Empty or partially filled rows have to be filtered out first
    - (IN) **sideForMerge**: deprecated, value ignored if mergeContexts is 0
    - (IN) **freq**: acquisition frequency for kinematic data (marker data)
    - (IN) **toCutAndNorm**: cell-array of data section to split into cycles (see input data) and normalize in time between 0 and 100
    - (IN) **f**: deprecated, use vector of 1s as long as **toCutAndNorm**
    - (OUT) **cycles**: struct with the following fields:
        - <context>(i).data.<data-section>.<phase>.<data-curve>: data curves splitted and normalized, for cycle i. <context> correspond to a found side from input stParam.eventsRaw.eventSide; <phase> is the phase name inside of the cycle (Phase1,...,Phase1) or EntireCycle.
        - <found-side>(i).name: cycle name
        - <found-side>(i).movingSide: anatomical moving side for cycle

<u>Spatio-temporal parameters calculation:</u>

- *Seg / calcSTParams.m*:
    - (IN) **cycles**: struct (see output of *Seg / StartSegmentation.m*)
    - (IN) **stParPoints**: cell-matrix, representing table *spatio-temporal parameters* in *Segmentation options* in main GUI. Empty or partially filled rows have to be filtered out first
    - (IN) **freq**: acquisition frequency for kinematic data (marker data)
    - (IN) **anglesMinMaxEv**: bool, if to calculate min, max, and events values for angles
    - (IN) **timing, speed, trajectory, jerk**: bool. If to process spatio-temporal parameters for points in table spatio-temporal-parameters
    - (OUT): **cycles**: struct (see output of *Seg / StartSegmentation.m*), following structure is added: <context>(i).data.stParam, containing spatio-temporal parameters data.

<u>Best cycles selection:</u>

- *BestCy / getBestCycles.m:*
    - (IN) **trials**: structure array (see output of *Engines* / *BodyMech3.06.01 / IProcKine_BM30601.m*) of trials related to the same task
    - (IN) **context**: context name
    - (IN) **phase**: phase name
    - (IN) **anglesList**: cell-array, list of angle names to be used for best cycles ranking
    - (IN) **bestCyclesN**: number of best cycles to keep. If greater than the number of existing cycles, this will be lowered to the number of existing cycles automatically.
    - (OUT) **out**: struct with the following fields:
        - formattedData.RMSETableForBestCycles: cell-matrix, contains RMSE between each cycle (for each angle) and the average of the remaning cycles. Angles are indicated in rows, while cycles are in columns.
        - cycles: struct-array, whose length is the minimum between **bestCyclesN** and

the number of existing cycles. It contains best cycles data, fields are:
- ➢ name: cycle name
- ➢ movingSide: moving side for cycle
- ➢ trial: trial name which the cycle comes from
- ➢ data.anglesCut: struct, angles data for cycle, not normalized
- ➢ data.anglesNorm: struct, angles data for cycle, normalized between 0% and 100%
- ➢ data.pointsCut: struct, points data for cycle, not normalized
- ➢ data.pointsNorm: struct, points data for cycle, normalized between 0% and 100%
- ➢ data.stParam: spatio-temporal data as from *Seg / calcSTParams.m*
- ➢ anglesForSelection: cell-array of angle names used for best cycles selection

Clinical parameters calculation:

- ➢ *ClinParams / getScores.m:*
  - ➢ (IN) **data**: struct as out from *BestCy / getBestCycles.m*
  - ➢ (IN) **refDataStruct**: struct, containing data fom reference cycles. It has the following fields:
    - ➢ rawData.<ref-cycle-name>.<angle-name>: reference angle for a reference cycle
  - ➢ (IN) **anglesList**: cell-array of angle names used to calculate the scores. These names can differ from the ones inside rawData.<ref-cycle-name> by a prepended 'R' or 'L'
  - ➢ (IN) **logTransVS**: deprecated, use 1
  - ➢ (IN) **VSName**: named for a value score (e.g. AVS, arm value score)
  - ➢ (IN) **PSName**: named for a profile score (e.g. APS, arm profile score)
  - ➢ (OUT) **data**: struct as in input, but with the following fileds added:
    - ➢ formattedData.RMSETableVSPerCycle: cell-array, each containing formatted data for a specific best cycle. Formatted data consists of a table where each row is VS for an angle, and the final row is PS, the root of summed squares of VS.
    - ➢ formattedData.RMSETableVSandPSPerAngle: summary cell-matrix table where columns are VS values for angles, and rows represent mean and std. dev. data.
    - ➢ formattedData.MAP: very similar to RMSETableVSandPSPerAngle, but now mean and std dev. data is in form of median, IQR+ and IQR-.

How to add processed data to trials after processing:

- ➢ *In Core / runProcessing.m*, or in sub-processing functions, add data to the variable subject loaded by *Core / loadStructData.m*.
- ➢ In Core/loadStructData.m, add the new data fields:

data.sessions(j).trials(k).<new-field> = tempDB.subjects(subInd).sessions(sesInd).trials(trInd).<new-field>;

- ➢ In *Core / currFileVersion.m* increment version by 1
- ➢ In *Core / importVirtualDBStruct.m*, add default values for the new fields, normally []:

trDB.subjects(i).sessions(j).trials(k).<new-field> = <default-value>;

- ➢ In *Core / runProcessing.m*, structure for trial k is available into

subject.sessions(j).trials(k).version. This is useful to check for older versions than the number returned by *Core / currFileVersion.m*.

Where to place new functions for processing sections:

The main U.L.E.M.A.folder already contains sub-folders relative to each processing section. New files should be place there. For kinematics, new functions should be place into *Engines / BodyMech3.06.01 / Core / Bmnewfunctions*. This folder also contains improved versions of native BodyMech functions. It is recommended to use them instead of the original ones.