|  |
| --- |
|  |
| LEVer |
| Progamer's Manual |
|  |
| **Mark Winter, Eric Wait, and Andrew Cohen** |
| **8/23/2011** |

|  |
| --- |
|  |

Contents

[Building LEVer 2](#_Toc301859092)

[Running LEVer 2](#_Toc301859093)

[Modifying/Replacing Segmentation Algorithm 2](#_Toc301859094)

[Brief Global Structures Reference 2](#_Toc301859095)

[CONSTANTS 3](#_Toc301859096)

[CellHulls 3](#_Toc301859097)

[CellTracks 3](#_Toc301859098)

[CellFamilies 4](#_Toc301859099)

[CellPhenotypes 4](#_Toc301859100)

[HashedCells 4](#_Toc301859101)

[Costs 4](#_Toc301859102)

[GraphEdits 4](#_Toc301859103)

[ConnectedDist 5](#_Toc301859104)

[SegmentationEdits 5](#_Toc301859105)

[Colors 5](#_Toc301859106)

[Log 5](#_Toc301859107)

[Brief Overview of Major Functions 6](#_Toc301859108)

[LEVer 6](#_Toc301859109)

[OpenData 7](#_Toc301859110)

[InitializeConstants 7](#_Toc301859111)

[AddConstant 7](#_Toc301859112)

[SegAndTrack 7](#_Toc301859113)

[ConvertTrackingData 7](#_Toc301859114)

[SaveLeverState 8](#_Toc301859115)

[TestDataIntegrity 8](#_Toc301859116)

[LogAction 8](#_Toc301859117)

[Segmentor 8](#_Toc301859118)

[GetDarkConnectedHulls 8](#_Toc301859119)

[WriteSegData 8](#_Toc301859120)

[MTC.exe 8](#_Toc301859121)

[ReadTrackData 8](#_Toc301859122)

[InitializeFigures 8](#_Toc301859123)

[SaveData 9](#_Toc301859124)

[updateConnectedDistance 9](#_Toc301859125)

[addToTrack 9](#_Toc301859126)

[NewCellFamily 9](#_Toc301859127)

[ProcessNewborns 9](#_Toc301859128)

[AddHullToTrack 9](#_Toc301859129)

[CreateMenuBar 9](#_Toc301859130)

[CreateContextMenuTree/CreateContextMenuCells 9](#_Toc301859131)

[FindLargestTree 9](#_Toc301859132)

# Building LEVer

Requirements:

* Microsoft Visual C++ 2008
* Matlab 2009b with the Matlab Compiler

LEVer consist of four compiled components:

1. LEVer.exe - The LEVer executable, compiled from LEVer.m scripts and dependencies.
2. Segmentor.exe - The standalone segmentation executable, compiled from Segmentor.m.
3. MTC.exe - The standalone multitemporal tracker, compiled from C++ sources listed in MTC.sln.
4. mexMAT.mexw32 - MEX tracker component, compiled from C++ sources listed in mexMAT.sln.
   1. Note: this is not included in the binary outputs, it is packaged in LEVer.exe as a dependency.

The CompileLEVer.m MATLAB script is used to build the four components of the LEVer software package and place the three output files in the binary output directory. All components can be compiled by executing the command "CompileLEVer" in a MATLAB command window. If Microsoft Visual Studio 2008 cannot be located, then an error will be displayed in the command window.

Most of the LEVer source code is written in the MATLAB scripting language, however the multitemporal tracking component of the LEVer software package is written in C++ for speed. There are two Microsoft Visual C++ solutions, MTC.sln, which builds a standalone executable tracker (MTC.exe) that is called from LEVer when tracking a newly segmented data set. The second solution, mexMAT.sln, builds a MATLAB MEX component (mexMAT.mexw32) that is used to update tracking costs during learning operations.

# Running LEVer

* To run the fully compiled version of LEVer, simple navigate to the binary output directory and run the LEVer.exe executable.
* To run LEVer from within MATLAB for debugging purposes, navigate to the LEVer MATLAB source directory and execute the command "LEVer" in the MATLAB command window.
  + ***Note*** that the CompileLEVer command (See Building LEVer) must still be run before executing LEVer within the MATLAB environment so that Segmentor.exe, MTC.exe and mexMAT.mexw32 will be available.

# Modifying/Replacing Segmentation Algorithm

The LEVer software is currently written for use on phase contrast time-lapse sequences of neural progenitor cells, the segmentation algorithm may need to be modified for other imaging protocols or for cells that greatly differ in appearance from neural progenitor cells. The current segmentation algorithm was written to be self-contained to facilitate modification. In order to change the segmentation algorithm, the Segmentor.m file must be modified or completely replaced. As long as the Segmentor interface and output format are maintained the rest of LEVer will function normally with the new segmentation algorithm.

# Brief Global Structures Reference

The global structures listed below are the main LEVer state for an image sequence. They are used throughout the LEVer codebase and represent the segmentation, tracking, and linaging results upon which the edit and learning operations of the LEVer program are based. Also, with the exception of SegmentationEdits, these structures are the set of variables saved to a LEVer file during a **SaveData** or **SaveDataAs** operation.

These structures have been roughly based on a database scheme, using index references to create relational bindings. This was done to reduce the size of any given structure and to improve searching. Some structure elements are reference one another which increases look up performance, however introduces the necessity to sync these fields. **TestDataIntegrity** function can be used to ensure that this is the case and is informative when debuging.

## CONSTANTS

CONSTANTS is a structure that stores all of the constants that pertain to a given data set. These are intended to only be set when LEVer is started. The **AddConstant** function attempts to enforce this but can be overwritten if need be.

* **CONSTANTS (1 x 1 struct)**
  + **rootImageFolder**
  + **imageDatasetName**
  + **imageSignificantDigits**
  + **imageAlpha**
  + **etc.**

## CellHulls

CellHulls is a structure array of cell segmentation information. Each entry in CellHulls corresponds to a single cell segmentation result. This structure keeps track of the convex hull points for the segmentation, the segmentation center of mass, as well as the set of pixel indices into the image sequence that the segmentation represents. The deleted and user edited fields are flags to indicate that user edits have modified a segmentation result.

* **CellHulls (1 x #Cells struct)**
  + **time - Image sequence frame of this cell segmentation result**
  + **points - List of xy-coordinates on the convex hull of this segmentation**
  + **centerOfMass - Center of mass calculated from interior pixels**
  + **indexPixels - List of linear indices of the interior pixels**
  + **imagePixels - List of intensities of interior pixels**
  + **deleted - Flag indicating this segmentation was manually removed**
  + **userEdited - Flag indicating this segmentation was manually edited**

## CellTracks

The CellTracks structure array contains an entry for each tracked cell. This contains information necessary to draw the lineage tree for a cell family including the parent and child tracks, if any, as well as track start and end times, and the associated cell family (clone). The hulls field also contains indices into the CellHulls structure for each segmentation. These are currently cleared, but not deleted, if the track has been absorbed into another track (e.g. all CellHulls for track have been added to a different track) .

* **CellTracks (1 x #Tracks struct)**
  + **familyID - Index of the track's family entry in CellFamilies**
    - reconcile with the tracks field of the coorasponding family
  + **parentTrack - Index in CellTracks of the track's parent**
    - reconcile with the childrenTracks field of the parent
  + **siblingTrack - Index in CellTracks of the track's sibling**
    - reconcile with the siblingTrack of the sibling
  + **childrenTracks - Indices of the two child tracks**
    - reconcile with the parentTrack of the children
  + **hulls - Indices into CellHulls of all segementations assigned to this track**
    - **These are hashed by taking the the frame number of a given hull, subtract the startTime, and add 1.**
    - reconcile with trackID field of Hashed Hulls
  + **startTime - Starting frame of the track**
  + **endTime - Ending frame of the track**
  + **color - Index in Colors assigned to this track**

## CellFamilies

CellFamilies contains a single structure per family (clone), the rootTrackID field gives the index into CellTracks of the root progenitor of all other cells in the family. Also contains start and end times for the entire family and a tracks field that lists all track indices in the family.

* **CellFamilies (1 x #Families struct)**
  + **rootTrackID - Root track index (into CellTracks) for the family**
    - the root track should have an empty parentTrack field
  + **tracks - Indices of all tracks in the family**
    - reconcile with familyID fields of each CellTrack
  + **startTime - Starting frame for the family**
    - reconcile with startTime field of the rootTrack
  + **endTime - Ending frame of the family**

## CellPhenotypes

The CellPhenotypes structure stores information regarding the set of phenotypes the user has created for an image sequence (only died exists by default). This structure also stores a set of segmentation-phenotype index pairs, indicating the locations where users have marked a cell with a specific phenotype. These pairs are used to visualize cell phenotypes and in certain processing operations.

* **CellPhenotypes (1 x #Phenotypes struct)**
  + **descriptions - Text description of the phenotype**
  + **contextMenuID - Context menu index of the phenotype**
  + **hullPhenoSet - Segmentation/Phenotype index pair set**

## HashedCells

HashedCells is a convenience cell array for quick access to segmentation and track information per frame. The *t*-th cell of the array contains a (1 x #Segmentations in frame *t*) structure array with hullID and trackID fields. For example usage of the HashedCells structure see the **GetTrackID** function.

* **HashedCells (1 x #Frames cell array)**
  + **hullID - Indices into CellHulls of all valid segmentations in this frame**
  + **trackID - Indices into CellTracks of corresponding track assignments**
    - reconcile with hulls field of each cellTrack

## Costs

Costs is a sparse array which contains edge costs for all valid segmentation associations as returned by the Multitemporal tracking algorithm. These costs are used during mitosis detection and also when learning from edits.

* **Costs (#Cells x #Cells sparse array)**

## GraphEdits

GraphEdits is a sparse array the same size as Costs and is used to indicate user edits to the graph structure, such as addition or removal of mitoses or changes in cell tracking.

* **GraphEdits (#Cells x #Cells sparse array)**

## ConnectedDist

The ConnectedDist contains pre-calculated forward connected-component distances for cell segmentations, these are used during tracking and learn from edits operations. This structure is built and updated by **BuildConnectedDistance** function.

* **ConnectedDist (1 x #Cells cell array)**

## SegmentationEdits

SegmentationEdits is the only global structure that is ***not*** saved to the LEVer file. This is used to track manual segmentation changes that must be considered during a learn from edits operation.

* **SegmentationEdits (1 x 1 struct)**
  + **newHulls**
  + **changedHulls**
  + **maxEditedFrame**
  + **editTime**

## Colors

Colors is an array that attempts to create a color scheme that has the maximum distinct colors. It also establishes a on current lineage tree, not on current lineage tree, and text colors per index. It is arranged so that the first three columns are the R,G,B of a selected cell, the next three columns are the R,G,B for the text, and the last three columns are the R,G,B of a unselected cell.

* **Colors (#of colors x 9 array)**

## Log

Log is an array that holds all of the edit events that has been performed on a data set. This structure is saved out as a comma delimited text file named as the [ data set ]\_log.csv. If the csv version is deleted, it is recreated when the next edit is performed or the next time the data set is opened. This struct is also intended for debug purposes. It holds the call stack for each change and when an error is caught, this is the best place to find the location of the error.

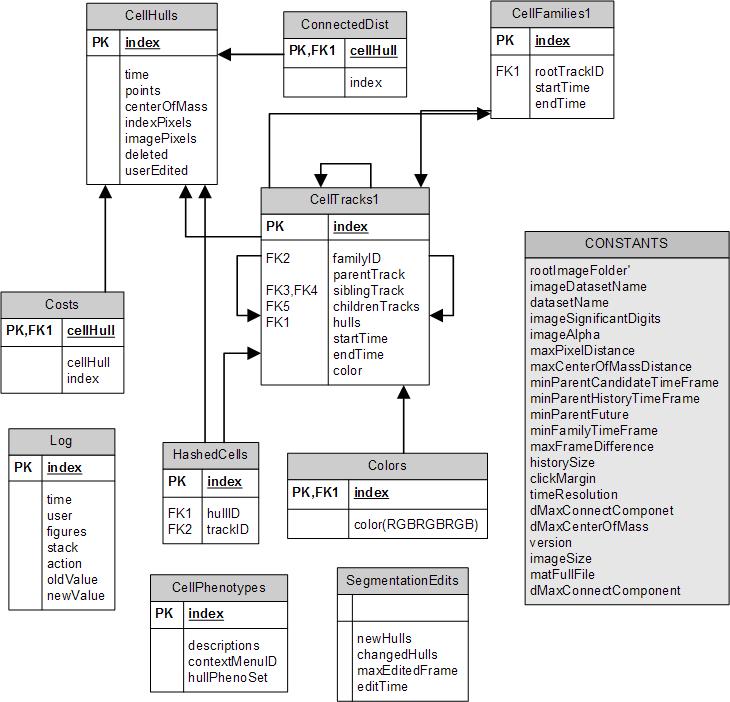


Figure (Global Variables)

# Brief Overview of Major Functions

These functions appear in the order that they are called initially when running the main function **LEVer**. All functions with the first letter capitalized are globally accessible, when the first letter is lowercase, it is locally scoped.

## LEVer

**LEVer** is the main entry point into the program. Use this to start the program or to open a new data set.

Calls:

* **OpenData**
* **InitializeFigures**
* **History**
* **DrawTree**
* **DrawCells**

## OpenData

**OpenData** is the function that handles either opening up existing LEVer data or creating new data. If LEVer is opening existing data, it checks that it is in the correct format (e.g. not a legacy format) and then sets up the figures. If LEVer needs to create data, it will run first go through the segmentation/tracking process then ensure the data is proper format, and finally set up the figures.

Calls:

* **InitializeConstants**
* **SegAndTrack**
* **ConvertTrackingData**
* **SaveLeverState**
* **TestDataIntegrity**

## InitializeConstants

**InitializeConstants** is where the constants for a given data set are set. This is mainly intended to be a convent place to find these values, it does not set them directly but calls **AddConstant**.

Calls:

* **AddConstant**

## AddConstant

**AddConstant** will add a field to the **CONSTANTS** structure if it doesn’t exist. If it does exist, it will not be overwritten unless the overWrite flag is set.

## SegAndTrack

SegAndTrack controls the processes and files associated with doing segmentation and tracking. It calls the **Segmentor** which is the data specific segmentation code compiled. This function attempts to spawn the Segmentor as many times as there are processors on the machine it is being run on. It then waits for the files to be written that signal that a particular instants of **Segmentor** as finished. It also writes and reads the documents that are used to communicate to the tracker (**MTC.exe**).

Once the tracker returns, the data is passed to **ConvertTrackingData** to convert the data the tracker returns into that of LEVer’s data structures. When that is accomplished, the data is saved and the action logged.

Calls:

* **Segmentor**
* **GetDarkConnectedHulls**
* **WriteSegData**
* **MTC.exe**
* **ReadTrackData**
* **ConvertTrackingData**
* **InitializeFigures**
* **SaveData**
* **LogAction**

## ConvertTrackingData

**ConvertTrackingData** takes a legacy data structure and converts it to the optimised data structure of the current version of LEVer.

Calls:

* **updateConnectedDistance**
* **addToTrack**
* **NewCellFamily**
* **ProcessNewborns**

## SaveLeverState

**SaveLeverState** saves all of the pertinent global variables to a file.

## TestDataIntegrity

**TestDataIntegrity** is a function used check the integrity of the data structures. This is helpful in debugging errors. It loops through each of the data structures ensuring that look up keys concur with one another. If passed a 1, it will attempt to correct any error it occurs. Errors are thrown when encountered.

## LogAction

**LogAction** keeps record of each change that is made. Each time it this function is called, the time, user, Figures global, callstack, action, old and new values are recorded. This is then exported to a csv file for user review.

## Segmentor

**Segmentor** is the cell specific segmentation code. This is intended to be compiled so that multiple instances can be ran in parallel. This function should be altered to meet the needs of the current data set.

## GetDarkConnectedHulls

**GetDarkConnectedHulls** gets the distance between cells to be passed to the tracker.

## WriteSegData

**WriteSegData** writes out the current data to a text file to be read by the tracker.

## MTC.exe

**MTC.exe** is compiled c code that does the tracking.

## ReadTrackData

**ReadTrackData** reads in the text file that the tracker has created.

## InitializeFigures

**InitializeFigures** creates two new figures for the lineage tree and cell images respectfully. It also creates handlers for automatic playback of the sequence.

Calls:

* **CreateMenuBar**
* **CreateContexMenuCells**
* **CreateContexMenuTree**
* **FindLargestTree**

## SaveData

**SaveData** takes care of where the data should be saved and enableing the save menu.

Calls:

* **SaveLeverState**

## updateConnectedDistance

**updateConnectedDistance** is a local function to **ConverTrackingData** to ensure that the distance matrix is in the correct format for the current LEVer.

## addToTrack

**addToTrack** is a local function to **ConverTrackingData** that creates the initial family and track that a particular cell is a member of and then each subsequent cell in a given track is added to that track.

Calls:

* **NewCellFamily**
* **AddHullToTrack**

## NewCellFamily

**NewCellFamily** creates a new family with one track with one hull.

Calls:

* **NewCellTrack**

## ProcessNewborns

**ProcessNewborns** takes any family that does not start at the first frame and attempts to attach it to an existing track. If the constraints are met, the family is merged with the found track. When the constraints are not met, the family is unchanged.

Calls:

* **ChangeTrackParent**

## AddHullToTrack

**AddHullToTrack** will add a cell to a given track.

## CreateMenuBar

**CreateMenuBar** set up the menu bar for both figures and sets the functions to be called by each item.

## CreateContextMenuTree/CreateContextMenuCells

**CreateContexMenu(Tree/Cells)** creates the context menus for each figure respectively and the function calls necessary.

## FindLargestTree

FindeLargestTree will find the lineage tree that has the most nodes and display it. This is typically the tree that either has been edited or is a good candidate to edit.

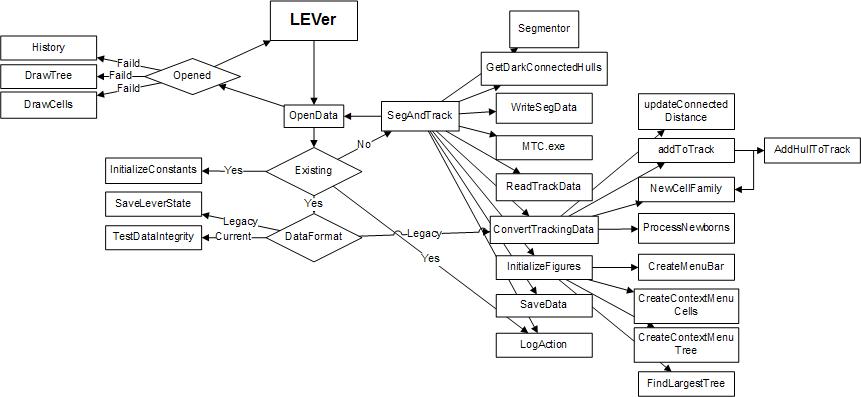


Figure (Initial Function Call Stack)