PURPOSE

cellListObj is an interface for Oufti generated cellists. This wrapper screens the cellList for valid cells and grants the user direct access to valid cells without the user screening the entire list. By default, a valid cell is one with a mesh field of size greater than 6. To organize the cellList, cellListObj internally constructs multiple indices of the cellList that reference cellList data for straightforward user access.

To maintain backward compatability, cellListObj provides meshData and cellId which are identical to the fields from the original Oufti cellList. Objects of this class can be used in any instance requiring an original Oufti cellList.

INITIALIZATION

cellListWrapper = cellListObj(cellList);

cellListObj takes in an Oufti cellList as the sole input argument. At initialization, the cellListObj traverses the input cellList and constructs default indexing methods. See addRestriction for more information on adding conditions that must be met for a cell to be considered valid and added to the index.

FUNCTION CALLS

addRestriction

USE

cellListWrapper.addRestriction(@functionHandle)

EXPLANATION

addRestriction accepts a function handle as an input argument and adds it as a condition that must be met for a cell to be valid. The function pointed to by functionHandle must return true or false for any input argument. functionHandle should take in a single cell from a cellList and return logical operators indicating that the restriction has been met and the cell is valid (true) or that the cell is not valid (false). Once a new restriction condition has been added, the cellListObj will automoatically recalculate valid indexing methods.

advanceCell

USE

cellListWrapper.advanceCell()

cellListWrapper.advanceCell(N)

EXPLANATION

advanceCell increments to the next valid cell in the index. If the operation was successful, advanceCell returns true. If not (indicating the end of valid cells in the cellList has been reached), advanceCell returns false. If an input argument is passed, it must be an integer and indicates the number of steps forward that should be taken in the index.

calculateForAllCells

USE

cellListWrapper.calculateForAllCells(@functionHandle)

EXPLANATION

calculateForAllCells applies the function pointed to by functionHandle to all valid cells in the cellList. The output is stored as a cell array. The output for each biological cell is returned in its own cell in the cell array.

calculateForLineage

USE

cellListWrapper.calculateForLineage(@functionHandle)

EXPLANATION

Applies the function pointed to by functionHandle to each cell in a lineage. The function pointed to must take in a single Oufti cell; there are no restrictions on the output that it may produce.

calculateForAllLineages

USE

cellListWrapper.calculateForAllLineages(@functionHandle)

EXPLANATION

Applies the function pointed to by functionHandle to each cell in a lineage. The function pointed to must take in a single Oufti cell; there are no restrictions on the output that it may produce. Equivalent to running calculateForLineage on all cellIds.

cellAtFrameIndex

USE

cellListWrapper.cellAtFrameIndex(Frame, Index)

EXPLANATION

Provides redundant access as cellList.meshData{Frame}{Index}. No restriction checks are performed to return a cell with this function call and Frame and Index is not required to be found in the valid indices array.

cellAtFrameId

USE

cellListWrapper.cellAtFrameId(Frame,cellId)

EXPLANATION

Returns the cell at the indicated frame with the specified cellId. If no cell is found, 0 is returned. No restriction checks are performed with this function call.

getCurrentCell

USE

cellListWrapper.getCurrentCell()

EXPLANATION

Returns the current valid Oufti cell as determined by default restriction settings and user supplied restriction rules (if supplied through addRestriction). cellListObj internally keeps an index of valid cells and allows the user to directly access cells within this list. Use this function in conjunction with advanceCell.

getIndex

USE

cellListWrapper.getIndex

EXPLANATION

Returns to the user the list of indices (n rows of [Frame, Index]) determined to be valid, whether by the default method or after the addition of some restriction.

getLineage

USE

cellListWrapper.getLineage(cellId)

EXPLANATION

Assembles the lineage for the cell corresponding to cellId. A cl\_access object is returned. Objects of type cl\_access are compatible with cellListObj and have corresponding functions (see cl\_access, below).

getLineageIndex

USE

cellListWrapper.getLineageIndex

EXPLANATION

Returns the list of indices for each lineage. The index of the first cellId will be in the first cell array. E.g., cellListWrapper.getLineageIndex{1} is the index of the lineage corresponding to cellListWrapper. uniqueCellIDs(1)

listMethods

USE

cellListWrapper.listMethods

EXPLANATION

Prints a list of methods available to the cellList object.

modifyCellList

USE

cellListWrapper.modifyCellList(@functionHandle)

EXPLANATION

Modifies each valid cell with the output from the function pointed to by function handle. Each cell will be replaced with the output from the function; the function pointed to by functionHandle should return all aspects of the cell that the user intends to keep.

randomSample

USE

cellListWrapper.randomSample

EXPLANATION

Returns a randomly selected cell from the valid cells.

removeRestriction

USE

cellListWrapper.removeRestriction

EXPLANATION

Removes any restriction put in place by addRestriction, and resets the index of valid cells along with iteration index along the valid cells.

resetIndex

USE

cellListWrapper.resetIndex

EXPLANATION

Resets the index of valid cells as well as current location in the index. Any sorting or restrictions are eliminated. Any restriction functions are not removed and the index will be recalculated with them in place.

sortBy

USE

cellListWrapper.sortBy(@functionHandle)

cellListWrapper.sortBy(@functionHandle, direction)

EXPLANATION

The function pointed to by functionHandle must return a sortable type. Cells will be sorted in the direction describted by the optional argument direction, which must be either ‘ascend’,’descend’ If direction is not specified sorting will be performed in ascending order.