Svoboda Lab data format – General Information
For data sets from Svoboda lab at CRCNS.org
Version 0.9 (Sept. 8, 2014)

Overview

This document outlines the data format used in the Svoboda lab for both, electrophysiology and optophysiology. A dataset contains three components:

1) a data structure containing processed data
2) meta data describing the experiment
3) the 'heavy' raw data (in a subdirectory)

1) and 2) together are stored in the relatively light-weight "session object", which contains all data pertinent to one experimental session (data from one day) except the 'heavy' raw data (images, extracellular voltages). The session object references the raw data, but for most data analyses the session object will be sufficient. This separation was done because the raw data (5-d image stacks; many voltage traces representing single electrodes) are extremely voluminous (many TB / project). 'Light' raw data types have been moved into the session object and are not provided in raw format (e.g., for imaging data, the position signal of the z piezo is stored in the session object; this also holds for most behavioral events).

The experiment description is stored in the field metaDataHash within the session object. The fieldnames and vocabulary conform to the meta_data_template file (provided as an Excel file). Shared data should include this vocabulary.

The specific session objects are explained in more detail in documents in the example_session_* folders.

Three different types of experiment are currently implemented:

- multi-electrode electrophysiological (silicon probes; tetrodes etc)
- cellular calcium imaging data
- single electrode electrophysiological (cell attached or intracellular)

Most of the data in the session objects is stored in a series of **hash tables**. We prefer hash tables to fixed structures with static fieldnames because they facilitate searching and simplify code that uses the session objects. Each hash key is accompanied by a description field so that users can understand the underlying data in the key-value pair. Hash tables reduce the overhead for future field addition. Adding a field is simple. Hash tables also enforce field check: code cannot simply assume that a field exists, as is possible with structure fields,

as one has to explicitly retrieve a key-associated value. We use static fields for data that will always be present, independent of the experiment.

Basic Outline

Most data can be divided into two major types: *time series*, a sequence of data points evenly spaced in time; *events*, temporally sparse data with time stamps. For example, a membrane potential recording would be a time series, whereas spike train would be a series of events (spikes). The session object consists of these two fundamental types, as well as meta-data.

The most basic types are **timeSeries** and **eventSeries**. Because arrays of each of these are handled in a different manner, there is a **timeSeriesArray** and **eventSeriesArray**. Finally, everything is encapsulated in a **session** container.

(Although these were originally developed as MATLAB classes, the outline below does deviate from the classes that can be obtained from the Janelia SVN repository, so as to be more general. The session objects are stored as structures. Much of the MATLAB functionality – e.g., scripts that allow one to extract salient events from a time series, or convert a series of events into a time series – can be used, but for now, this is purely a data model.)

timeSeriesArray

Each timeSeriesArray should have the following fields:

Descriptive & metadata:

timeSeriesArray.id: vector of integers, not necessarily in order but unique.
timeSeriesArray.idStr: a cell array of descriptive tags which should be brief.
timeSeriesArray.idStrDetailed: a cell array of more detailed descriptive tags, if needed.

timeSeriesArray.descrHash: for storing metadata that pertains only to this timeSeriesArray (e.g., roi positioning).

timeSeriesArray.timeUnit: unit of time.

Actual data (all vectors should be of same size):

timeSeriesArray.time: vector indicating the (common) time at which datapoints were acquired, ideally in standard UNIX format (i.e., time since Jan 1, 1970).
timeSeriesArray.trial: vector of integers indicating which trial each time point sits in. Convenient for indexing.
timeSeriesArray.valueMatrix: matrix where the number of rows corresponds to the number of *timeSeriesArray.id* values, and the number of columns is the same as the length of *time* and *trial* fields.

timeSeriesArray.sourceFileList: cell array with pathnames of all data files.
timeSeriesArray.sourceFileIdx: two-row matrix, with first row pointing to one of the sourceFileList elements, and the second pointing to the indexed position within the file (if applicable).

eventSeriesArray

This type should be used to store arrays anything that are best described as a set of time/value pairs (e.g., spikes). Each eventSeriesArray structure should have the following fields:

Descriptive & metadata:

eventSeriesArray.id: a vector of integers, not necessarily sorted but unique.
eventSeriesArray.idStr: a cell array of descriptive tags, which should be brief.
eventSeriesArray.idStrDetailed: a cell array of more detailed descriptive tags, if needed.

eventSeriesArray.descrHash: for storing metadata that pertains only to this eventSeriesArray (e.g., roi positioning).

eventSeriesArray.type: a vector of integers, 1 or 2, indicating type (1: single events 2: paired events -- start/end).

Actual data:

eventSeriesArray.eventTimes: a cell array of vectors, where for *eventSeriesArray.id(x)* with *idStr{x}*, etc., you store data. *eventSeriesArray.eventTimes{x}*
eventSeriesArray.eventTrials: cell array of vectors with trial numbers.
eventSeriesArray.eventPropertiesHash: cell array of Hash tables. These may be blank.

session

This is the overall container. It should contain all the data for a single behavioral session.

Descriptive & metadata:

session.metaDataHash: contains all of the relevant metadata fields, with values; conforms to the metadata definition file.

session.timeUnitIds: hard coded numerical indexing for timeUnits used throughout; vector of integers.
session.timeUnitNames: cell array of strings describing each time unit, same indexing.

session.trialTimeUnit: specifies time unit used in trialTimes.
session.trialTypeStr: cell array of strings; for each trial type, a description.

session.trialIds: this contains a vector of trial numbers. This allows you to map the trial numbers from, e.g., trialTimes, to the next few vectors/matrices, in case of misordering or not starting at one.
session.trialStartTimes: same indexing as trialIds; stores the trial start time.
session.trialTypeMat: boolean matrix of size n_types x n_trials ; value is 1 if the trial belongs to a type, 0 if not. Trials can thereby be of multiple types at the same time.
session.trialPropertiesHash: a hash table with any number of fields; the only constraint is that the elements must have n_trials entries (for instance, to store # touches).

Actual data:
session.timeSeriesArrayHash: a hash that contains timeSeriesArray elements.
session.eventSeriesArrayHash: a hash that contains eventSeriesArray elements.

Special Definitions

Hash: this is not a type that is native to MATLAB. We specify the following fields:

Hash.keyNames: cell array of strings with names.
Hash.descr: cell array with a detailed description per key.
Hash.values: cell array of whatever – usually cell array of numerical vectors.

Imaging Data

Imaging data that is raster-scanned will generally require definition of regions-of-interest (ROIs). By convention, this should be stored in the descrHash of the relevant timeSeriesArray. The field "ROIs" will have one ROI map per imaging plane. This is a structure with fields:

roiMap.roiIds: vector representation of roiIds; corresponds to timeSeriesArray ids.
roiMap.masterImage: average across session image for this plane.
roiMap.rois: array with ROIs.

Each ROI within **roiMap.rois** contains:

id: the id of the roi; consistent with other ids.
cornersXY: Coordinates of the X and Y corners for the ROI.
indicesWithinImage: In the image, stores the exact pixels were used for the mask, indexed in MATLAB convention.