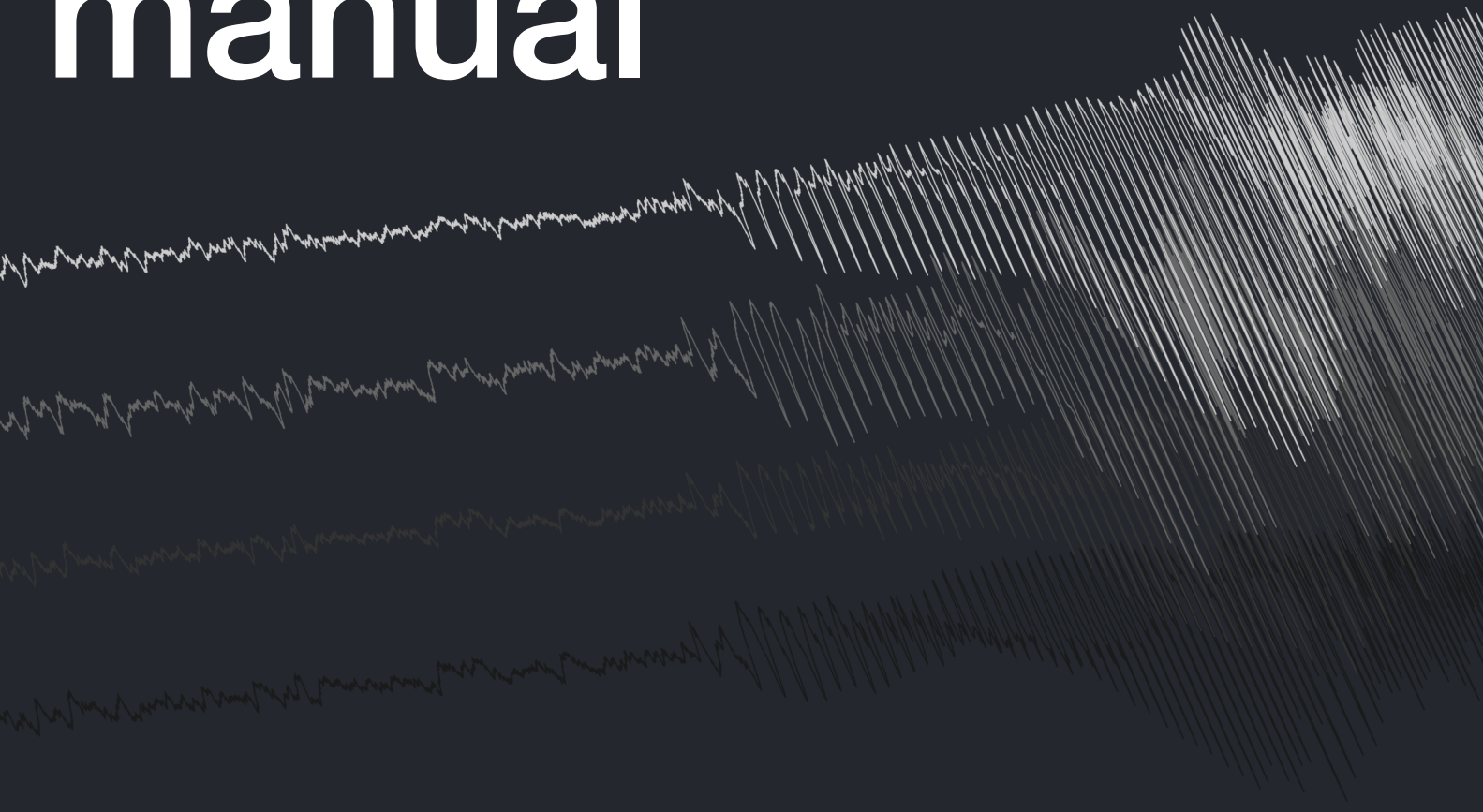


PatientDB manual



**A guide to the Matlab toolbox
and database for storing and
accessing patient recordings**

Updated:
May 13, 2025

github.com/edmerix/PatientDB

Edward M. Merricks, Ph.D.

Columbia University Medical Center

Contents:

- [Introduction/concepts](#)
- [Getting started \(quick start\)](#)
- [Accessing data](#)
- [Adding new data](#)
- [Saving the database](#)
- [Built-in properties & methods](#)
- ["Subway map" \(schematic overview\)](#)

Introduction

What is this?

Keeping track of where specific data are stored in huge datasets of continuous recordings in patients is unwieldy. Making changes to the original, raw recordings to add notes for when various things occur is risky, so we want to have a separate entity that knows where everything is, and points to the right locations in a human-readable manner. PatientDB is a combination of a "database" (storage of raw data) and "methods" (ways to interact with the data) in a single *object-oriented* toolbox to achieve this format.

Concepts

PatientDB is entirely MATLAB code. This allows for raw data to be stored in-line, and for the interactive methods to exist within the same object. It also allows the data to be stored in an "object-oriented" manner, meaning if you change info about a specific item in the PatientDB, whatever you update will immediately apply everywhere else in the database that the item is referenced/stored, avoiding mismatched, out-of-sync data or broken links.

This means that data can be accessed *circularly*, and *infinitely/recursively*. It also means that data access follows an "object" approach such that data within an item is accessed via a "dot". So if you have a variable called `patient`, which has specific seizures in it—stored in an array called `seizures`—and each one had a `file` in it, you'd access the second seizure's file with:

```
patient.seizures(2).file
```

From there, each object (i.e. each item before or after a dot) has its own methods and properties, while the PatientDB as a whole has methods to access most of the data intuitively.

Check out the [schematic](#) ("subway map") at the end of this document for an overview of how the data interact with each other, and the individual file [overviews](#) for their specific methods.

N.B. Within MATLAB, we usually store the PatientDB object in a variable called `pdb`, so the rest of this document uses that terminology. This is a variable though, and you can call as many PatientDB objects as many different names as you want, and they'll work in the same way, completely independently of one another. And they can all be saved in whatever filename you care for.

Be aware, however, that if you copy a PatientDB object (e.g. `pdb2 = pdb;`) it will be passed by "reference" meaning changes to `pdb2` will also be enacted in `pdb`.

Getting started

Loading a pre-made database

PatientDB objects can be created from scratch (see [later examples](#) or included script [PatientDB_basic_examples.m](#)), but in most cases a pre-existing database will be being interacted with. After ensuring the directory containing the PatientDB files themselves (i.e., PatientDB.m, PDBpatient.m, etc.) and the directory containing the PatientDB database (e.g., PatientDB.mat) are both on the MATLAB path, the database can be loaded simply with:

```
load PatientDB
```

Alternatively, a full path can be given to avoid adding extra directories to the MATLAB path, e.g.,

```
load('/absolute/path/to/data/PatientDB.mat')
```

In either case, the workspace should now contain a variable called `pdb` (or whatever name the database had been saved under). If working on the server at Columbia University Medical Center, both the software and the latest database can be made available with

```
addpath(genpath('/storage/software/PatientDB/'))
```

Quick start examples

As a general rule, data can be accessed from the root object, e.g., all patients are found within `pdb.patients`, all seizures within `pdb.seizures`, all electrodes in `pdb.electrodes`, etc. These all internally reference each other, meaning typical first usage would be to interact via the `patients` field. For example, to get information about the third seizure from a patient with identifier "PTID04" (not a real patient identifier):

```
pdb.patients.PTID04.seizures(3)
```

Say we want to find the filename and path containing the raw recording from the microelectrodes, and how many seconds into that file this seizure occurred:

```
file = pdb.patients.PTID04.seizures(3).micros.fullpath;  
onset = pdb.patients.PTID04.seizures(3).micros.onset;
```

If `NSxFile` is also on the MATLAB path, there is a shorthand to immediately open a link to the relevant file and store seizure onset and offset without having to repeatedly call PatientDB:

```
[nsx,onset,offset] = pdb.patients.PTID04.seizures(3).loadMicros();
```

`nsx` now contains an `NSxFile` object, allowing data to be read from, for example, the first 5 channels from 30 seconds before seizure onset until 30 seconds after seizure termination:

```
nsx.read('channels',1:5,'time',[onset-30 offset+30]);
```

Using the catalog

PatientDB contains methods to enable searching by criteria and returning arrays of matching data. For example:

Get all complex-partial seizures (to use terminology from the latest ILAE guidelines use the pre-release version):

```
CPS_seizures = pdb.getSeizures('type','CPS');
```

Get all GTC seizures that had electrodes labeled 'uLHC' AND 'u*H' ('*' is a wildcard, so this would match uLH, uRH and uHH, for example):

```
specific_data = pdb.getSeizures('type','GTC','label',{'uLHC','u*H'});
```

If you just want all seizures available:

```
all_seizures = pdb.seizures;
```

Say PTID04 was a re-implant of CU_ID9, we would find this information in:

```
pdb.patients.PTID04.reimplantOf.id
```

The output will be CU_ID9 since we requested the ID of the re-implant information, confirming their being the same patient. Say we therefore want all single units from this patient, across both implants:

```
units = [pdb.patients.PTID04.units; pdb.patients.PTID04.reimplantOf.units];
```

We could have simply requested `pdb.patients.CU_ID9.units`, of course, but since PatientDB maintains automatic links between patients, we can bypass manually searching for the patient identifier and just use the `reimplantOf` shorthand instead.

Similarly, units across recording sessions can be linked to one another if they are deemed to be activity from the same neuron. Let's say the 16th and 21st units that were isolated in PTID04's first and second seizure recordings were deemed to be the same neuron—in this case:

```
pdb.patients.PTID04.seizures(1).units(16).sameUnit(1)
```

would return the same `PDBunit` variable as:

```
pdb.patients.PTID04.seizures(2).units(21)
```

and vice versa, i.e.,

```
unit = pdb.patients.PTID04.seizures(2).units(21).sameUnit(1);  
unit == pdb.patients.PTID04.seizures(1).units(16)
```

will return `true`. This enables building analysis workflows that automatically track identities, in order to avoid re-counting the same neuron as a new statistical entity, or to check how neuronal responses shift over time. See [PDBunit page](#) for details of how to “link” units.

Adding data

Data can be added to the PatientDB object in any order, using one of the following methods:

Method	Description
<code>addPatient(ptID, ...)</code>	Add a new patient with the specified identifier, <code>ptID</code> . Optional extra arguments follow the PDBpatient properties.
<code>addSeizure(ptID, num, ...)</code>	Add a new seizure recorded in the patient with the specified identifier, <code>ptID</code> , and numbered by <code>num</code> . If the patient identifier doesn't already exist in the database, it will be created first. Optional extra arguments follow the PDBseizure properties.
<code>addElectrode(ptID, ...)</code>	Add an electrode from the given patient, <code>ptID</code> , to the database. If the patient identifier doesn't already exist in the database, it will be created first. Optional extra arguments follow the PDBelectrode properties. The <code>channel</code> or <code>label</code> can be added here or after the fact.
<code>addUnit(ptID, ...)</code>	Add a single unit that was recorded in the given patient, <code>ptID</code> . If the patient identifier doesn't already exist in the database, it will be created first. Optional extra arguments follow the PDBunit properties.
<code>addMultiunit(ptID, ...)</code>	Add a multi-unit that was recorded in the given patient, <code>ptID</code> . If the patient identifier doesn't already exist in the database, it will be created first. Optional extra arguments follow the PDBmultiunit properties.
<code>addEvent(ptID, num, ...)</code>	Add a numbered event that was recorded in the given patient, <code>ptID</code> . If the patient identifier doesn't already exist in the database, it will be created first. Optional extra arguments follow the PDBevent properties.

For example, to add a patient with the identifier “PT05” that had Behnke-Fried electrodes to the database, and then add a seizure that was recorded in that patient:

```
pt = pdb.addPatient('PT05', 'implantType', 'BF');
sz = pdb.addSeizure('PT05', 1, 'type', 'CPS');
```

The return values (`pt` & `sz`) are not necessary, but if information was not added during initial creation, it can be added to the database via these output handles to save time, for example:

```
pt.reimplantOf = pdb.PT01;
```

Then to link the newly added seizure to a specific file, including marking when seizure onset and offset occurred in that file (in seconds):

```
fl = sz.addFile('name', 'example.ns5', 'path', '/not/a/real-path/');
fl.onset = 305; % could have been done inline above, if preferred
fl.offset = 358;
```

Note that the PDBfile object has a `findLostFile()` method that can automatically update the database if the file is moved later.

More information on adding specific events, linking them to one another, modifying their contents, etc., can be found in the methods pages for each object later in this document, or in the included `PatientDB_basic_examples.m` file as part of the PatientDB repository.

Saving backups (recommended before making any edits)

Calling `pdb.backup()` without any arguments will create a backup file in the “backups” directory where the PatientDB code is stored.

Adding a `'path'` argument will save the backup file in that location, e.g.:

```
pdb.backup('path', '~/preferred-directory/');
```

If desired, setting a `'name'` argument will save the PatientDB object under that variable name in the file, however due to Matlab's handling of `'matfile'` objects that include classes, this causes considerably larger files, and so leaving the default `pdb` name is recommended.

The files saved via this backup method are labeled by the date and time the backup was created, using `'uuuuMMdd_HHmmss'` format.

Saving the database

To update the main database as stored to disk, call `pdb.save()`, which will overwrite the main file. For this reason, we recommend regularly storing backups (see above, or run manual backups of the file) prior to making edits to the core PatientDB file.

By default, this saves at `'/storage/software/PatientDB/PatientDB.mat'`, but this can be overridden by passing in a path to the function, e.g.:

```
pdb.save('~/preferred-main-directory/PatientDB.mat');
```

Note that the file can be named anything you wish, but for simplicity we use `'PatientDB.mat'` for consistency within this guide.

For a full list of methods available in each object (e.g., patient, seizure, files, etc.) see their individual manual pages below.

PatientDB

This is the root object of PatientDB, returned on entering `pdb` at the command window

PatientDB structure

Property	Type	Description
creationDate	<i>datetime</i>	When this PatientDB was first created
modifiedDate	<i>datetime</i>	When this PatientDB was last modified
patients	<i>struct</i>	A struct pointing to each patient's data
seizures	<i>PDBseizure</i>	Array pointing to each seizure
electrodes	<i>PDBelectrode</i>	Array pointing to each electrode
units	<i>PDBunit</i>	Array pointing to each single unit
multiunits	<i>PDBmultiunit</i>	Array pointing to each multi-unit
markers	<i>PDBevent</i>	Array pointing to each marked event
info	<i>char</i>	Any relevant text information

PatientDB methods

- `.addElectrode(ptID, arguments)`
Add an electrode to the database for the patient with ID `ptID`.
Optional `arguments` from *PDBelectrode*.
- `.addEvent(ptID, num, arguments)`
Add an event to the database for the given `ptID` and event number, `num`.
Optional `arguments` from *PDBevent*.
- `.addMultiunit(ptID, arguments)`
Add a multi-unit to the database for the given `ptID`.
Optional `arguments` from *PDBmultiunit*.
- `.addPatient(ptID, arguments)`
Add a patient with the unique identifier `ptID` to the database.
Optional `arguments` from *PDBpatient*.
- `.addSeizure(ptID, num, arguments)`
Add a seizure to the database for the given `ptID` and seizure number, `num`.
Optional `arguments` from *PDBseizure*.
- `.addUnit(ptID, arguments)`
Add a single-unit to the database for the given `ptID`.
Optional `arguments` from *PDBunit*.
- `.backup(arguments)`
Save a backup of the database. Optional `arguments` include `path` and `name`.
- `.getSeizures(arguments)`
Returns an array of seizures that meet your selection criteria in the `arguments`.
- `.linkBundles(ptID)`
Automatically finds electrodes that are from the same bundle. With no input, runs on whole database, otherwise limits to that `ptID`.
- `.listElectrodes()`
Lists all electrodes in the format: `[ptIDs, channels, labels]`.
- `.listEvents()`
Lists all events in the format: `[ptIDs, event numbers]`.
- `.listSeizures()`
Lists all seizures in the format: `[ptIDs, seizure numbers]`.
- `.save(path)`
Saves the core database in code root directory unless an optional `path` is given.

PDBpatient

The basic structure to handle patient metadata and handles to associated data and methods.

PDBpatient structure

Property	Type	Description
id	char	Details for the associated patient
implantType	char	Details of the implant type, e.g., BF, cereplex
reimplantOf	<i>PDBpatient</i>	Array of other PDBpatient objects from the same patient during a different implant
seizures	<i>PDBseizure</i>	Array pointing to each seizure for this patient
electrodes	<i>PDBelectrode</i>	Array pointing to each electrode for this patient
units	<i>PDBunit</i>	Array pointing to single units in this patient
multiunits	<i>PDBmultiunit</i>	Array pointing to multi-units in this patient
markers	<i>PDBevent</i>	Array pointing to marked events for this patient
notes	char	Any relevant text information

PDBpatient acts as a parent object of other classes, each with their own methods, and other than the constructor has none of its own.

PDBseizure

The methods and structure to handle data about recorded seizures.

PDBseizure structure

Property	Type	Description
patient	<i>PDBpatient</i>	Details for the associated patient
number	<i>uint16</i>	What seizure number this was logged as
type	<i>PDBseizureType</i>	Enumeration for seizure type (see map)
files	<i>PDBfile</i>	Array of associated files with raw data
units	<i>PDBunit</i>	Array pointing to associated single units
multiunits	<i>PDBmultiunit</i>	Array pointing to associated multi-units
notes	<i>char</i>	Any relevant text information

PDBseizure methods

`.addFile(arguments)`

Add a file that is associated with this seizure.

Optional `arguments` from *PDBfile*.

`.addOnsetInfo(selectorType, selector, {patternList})`

Mark the selected (`selector`) channel/electrode (`selectorType`) with the provided cell-array list of onset patterns (`{patternList}`).

`.loadMacros()`

If *NSxFile* is on the path and the seizure has a file listed for the macros, open a handle to that file and return the onset and offset time of the seizure within that file in seconds, in the format:

`[nsx, onset, offset]`.

`.loadMicros()`

If *NSxFile* is on the path and the seizure has a file listed for the micros, open a handle to that file and return the onset and offset time of the seizure within that file in seconds, in the format:

`[nsx, onset, offset]`.

`.macros()`

If a file that recorded the macros for this seizure has been added, return its *PDBfile* object. Also returns any file marked as containing “both”.

`.micros()`

If a file that recorded the micros for this seizure has been added, return its *PDBfile* object. Also returns any file marked as containing “both”.

`.printOnsetInfo()`

For each channel/electrode from this recording, print what seizure onset type has been listed.

PDBfile

The methods and structure to handle and load associated external data files.

PDBfile structure

Property	Type	Description
name	char	The name of the file
path	char	Where the file is (absolute path preferred)
type	PDBfileType	Enumeration for the file type (see map)
onset	double	Time event/seizure starts in the file (seconds)
offset	double	Time event/seizure stops in the file (seconds)

PDBfile methods

`.findLostFile(searchPath, overwrite, extension)`

If the original file has been moved, search for it again, in the default data storage directory. Alternatively, include a `searchPath` to limit/alter where it searches.

Optionally, set `overwrite` to true to allow it to automatically update the PatientDB structure after locating it, otherwise it will ask for confirmation first. `pdb.save()` must still be called to write changes to disk.

Provide `extension` if multiple files share the same name but different extensions, e.g., “ns3” versus “ns5”, otherwise it determines based on the stored `PDBfile.type` value.

Returns the file path and whether the object was updated in the format:

`[filepath, updated]`

Note that this is designed for usage on a *nix based system.

`.fullPath()`

Returns the correctly formatted absolute path for this file as a string. Works out the correct file extension based on the `PDBfile.type` value.

PDBunit & PDBmultiunit

The methods and structures to handle data spike sorted single- and multi-units.

PDBunit & PDBmultiunit structure

Property	Type	Description
UID	<i>int32</i>	Unique identifier for this unit (auto-assigned)
patient	<i>PDBpatient</i>	Details for the associated patient
electrode	<i>PDBelectrode</i>	Details for the electrode for this unit
waveform	<i>double</i>	The mean waveform from this unit
waveformSD	<i>double</i>	The standard deviation of the unit's waveforms
Fs	<i>single</i>	The sampling frequency
times	<i>double</i>	An array of timestamps when the unit fired
wideband *	<i>double</i>	The mean waveform from the unfiltered data
widebandSD *	<i>double</i>	The standard deviation of the above
type *	<i>PDBcellType</i>	Enumeration for what cell-type it is (see map)
typeProb *	<i>double</i>	The confidence of the cell-type assignment
metrics	<i>PDBunitMetrics</i>	A structure of metrics for the unit
relatedEvent	<i>PDBevent</i>	A link to an associated event, if any
relatedSeizure	<i>PDBseizure</i>	A link to an associated seizure, if any
sameUnit *	<i>PDBunit</i>	Links to the same unit recorded at other times
file	<i>PDBfile</i>	The file this unit recording arose from
notes	<i>char</i>	Any relevant text information

* unique to PDBunit

PDBunit & PDBmultiunit methods

`.calculateFiringMetrics(arguments)`

Calculate the firing pattern metrics. Optional *arguments* are *'bins'* and *'epoch'*. Auto-populates the *metrics* field.

`.calculateWaveformMetrics()`

Calculate the metrics based on the mean wideband waveform. Optional *arguments* are *'troughIndex'*, *'uprate'*, *'idealized'*, and *'idealizedOrder'*. Auto-populates the *metrics* field. Unique to PDBunit.

`.getBundle()`

Returns a combined object of all multi-units from this bundle of electrodes, as well as an array of each individually, in the format: *[combined, multi-units]*. Unique to PDBmultiunit.

`.linkUnits(PDBunit)`

Adds this unit to the *sameUnit* list of the input *PDBunit* object, and vice versa. Unique to PDBunit.

PDBevent

The methods and structure to handle data about recorded events.

PDBevent structure

Property	Type	Description
patient	<i>PDBpatient</i>	Details for the associated patient
electrode	<i>PDBelectrode</i>	The electrode the event is associated with
time	<i>datetime</i>	The timestamp for this event
files	<i>PDBfile</i>	Array of associated files with raw data
units	<i>PDBunit</i>	Array pointing to associated single units
multiunits	<i>PDBmultiunit</i>	Array pointing to associated multi-units
notes	<i>char</i>	Any relevant text information

PDBevent methods (inherited from PDBseizure)

.addFile(arguments)

Add a file that is associated with this event.

Optional *arguments* from PDBfile.

.loadMacros()

If NSxFile is on the path and the seizure has a file listed for the macros, open a handle to that file and return the onset and offset time of the event within that file in seconds, in the format:

[nsx, onset, offset].

.loadMicros()

If NSxFile is on the path and the seizure has a file listed for the micros, open a handle to that file and return the onset and offset time of the event within that file in seconds, in the format:

[nsx, onset, offset].

.macros()

If a file that recorded the macros for this event has been added, return its PDBfile object. Also returns any file marked as containing “both”.

.micros()

If a file that recorded the micros for this event has been added, return its PDBfile object. Also returns any file marked as containing “both”.

PDBelectrode

The methods and structure to handle storage of and access to electrodes across recordings.

PDBelectrode structure

Property	Type	Description
patient	<i>PDBpatient</i>	<i>Details for the associated patient</i>
label	<i>char</i>	<i>Any label the electrode was given</i>
channel	<i>int16</i>	<i>The channel number for the electrode</i>
bundle	<i>PDBelectrode</i>	<i>Array of other electrodes on the same bundle</i>
units	<i>PDBunit</i>	<i>Array pointing to associated single units</i>
multiunits	<i>PDBmultiunit</i>	<i>Array pointing to associated multi-units</i>

PDBelectrode methods

.addOnsetInfo(seizureNumber, {patternList})

Mark the selected seizure with the provided cell-array list of onset patterns ({patternList}) for this electrode.

.getBank()

If this electrode is part of a bank/bundle then return the label for that group (i.e., remove the unique number from this electrode's label).

.getBundle()

Alias for the above .getBank() for backwards compatibility.

.getName()

If this electrode isn't just a channel number, return the electrode name.

.printOnsetInfo()

For each seizure captured on this electrode, print what seizure onset type has been listed.

PDBunitMetrics

The methods and structure to handle standardized metrics about single- and multi-units.

PDBunitMetrics structure

Property	Type	Description
ACarea	double	The area under the cumulative sum of the unit's autocorrelation over a specified duration
firingRate	double	The mean firing rate for the unit, in spikes s^{-1}
FWHM	double	The full width at half the maximum amplitude of the mean waveform, in milliseconds
gmFalseNeg	double	False negative rate based on overlap between fitted Gaussians across units on the same electrode and assessing overlap
gmFalsePos	double	False positive rate based on overlap between fitted Gaussians across units on the same electrode and assessing overlap
matchConfidence	double	An array of confidences that each spike was a true match to this neuron
meanAC	double	The mean value in the autocorrelogram over a specified duration, in milliseconds
missingRate	double	The proportion of a fitted Gaussian to the detected troughs for the unit that falls sub-threshold for detection
recoverySlope	double	The angle of the slope after the peak, i.e., recovery from after-hyperpolarization, in $\mu V ms^{-1}$
repolarizationSlope	double	The angle of the slope during repolarization, in $\mu V ms^{-1}$
rpvRate	double	The percentage of spikes that violated the refractory period
threshold	double	The threshold for spike detection on the channel this unit came from
troughToPeak	double	The duration from the detected trough to the following peak in the mean waveform, in milliseconds

PDBunitMetrics methods

`.falsePositiveRate()`

Calculate and return the false-positive rate as described in [Hill et al. \(2011\)](#).

`.falseNegativeRate()`

Calculate and return the false-negative rate as described in [Hill et al. \(2011\)](#).

PatientDB structure map

(makes it look more complicated than it is...)

