Neuroshare Matlab API
Rev 2.2

Application Programming Interface for Accessing Neurophysiology Experiment Data Files

December 2003

Neuroshare Matlab Application Programming Interface

The Neuroshare Matlab import functions allow users to easily import neural data files into Matlab for further analysis. These functions were developed as part of an SBIR program funded by the National Institute for Neural Disorders and Stroke and it is currently being administered by Cyberkinetics, Inc. (formerly Bionic Technologies, LLC). The goals of the SBIR program are to create open Application Programming Interface (API) library and format standards for neurophysiological experiment data and create a set of free, open-source software tools for low-level handling and processing of neurophysiological data.

Table of Contents

Version History

Version 2.2: Added:

ns_SetLibrary: A function to specify which DLL to use for future commands

Version 2.1: Bug fixes:

- ns_GetNeuralInfo & ns_GetSegmentData: Units classified as 'noise' are now loaded in with the right unit ID.
- nsNEVLibrary.dll: The sample group packets are counted correctly (previously occasionally returned 0). Simultaneously saved NEV experimental data packets are read in the correct time order. Entities with no data are included with an ItemCount of zero.

Software was compiled with Matlab Version 6.1.

Version 2.0: Bug fixes:

- ns_GetLibraryInfo: Function returns the correct structure; some items were swapped before.
- ns_GetNeuralData: Function does not return all zeros anymore as timestamps when one of the input arguments' indices is invalid.
- ns_GetSegmentData: Function now returns the correct values when data is imported for the second time.
- Library now works with all Matlab versions 6.x and should not return the "Invalid mex-file" error message when loading the library.

Returned structures do not use Hungarian notation anymore in variable / field names (e.g. changed from 'szEntityLabel' to 'EntityLabel' and 'dwItemCount' to 'ItemCount').

Continuous analog data files for sample groups of the Cerebus system (*.nsx files) can now be imported together with their *.nev files.

Software was compiled with Matlab Version 6.1.

Version 1.1: Bug fixes:

- ns_GetAnalogInfo does not cause a segmentation fault anymore.

Software was compiled with Matlab Version 6.5.

Version 1.0: This is the first implementation of the Matlab API that conforms to the Neuroshare API Specification 1.0.

Software was compiled with Matlab Version 6.5.

Installation of the Matlab API

The API consists of the following files:

mexprog.dll
ns_SetLibrary.m
ns_GetLibraryInfo.m
ns_OpenFile.m
ns_GetFileInfo.m
ns_CloseFile.m
ns_GetEntityInfo.m
ns_GetEventInfo.m
ns_GetEventData.m
ns_GetAnalogInfo.m
ns_GetAnalogData.m
ns_GetSegmentInfo.m
ns_GetSegmentSourceInfo.m
ns_GetSegmentData.m
ns_GetNeuralInfo.m
ns_GetNeuralData.m
ns_GetIndexByTime.m
ns_GetTimeByIndex.m
ns_GetLastErrorMsg.m

There is no installation program but the files should be installed in the following locations:

mexprog.dll – The mexprog.dll file should either be in the same directory as the m-scripts or they should be on the Matlab path.

m-scripts – The m-scripts should be on the Matlab path or in the current working directory.

In addition to the aboved named files, you will also need to obtain a Neuroshare Library from your instrument vendor. This DLL can be placed anywhere on your system. It is easiest if you place the Vendor Neuroshare DLL in the same directory as the *m-scripts;* otherwise, you will need to know its location for the *ns_SetLibrary* function call.

Many of the vendor Neuroshare DLLs can be obtained from www.neuroshare.org.

Representation of Data Types

There are 17 different functions that allow the user to import certain parts of the data file or to search the data file for specific events. Neural data is divided into four basic categories or "Entity Types":

Event Entities – Discrete events that consist of small time-stamped text or binary data packets. These are used to represent data such as trial markers, experimental events, digital input values, and embedded user comments.

Analog Entities – Continuous, sampled data that represent digitized analog signals such as position, force, and other experiment signals, as well as electrode signals such as EKG, EEG and extracellular microelectrode recordings.

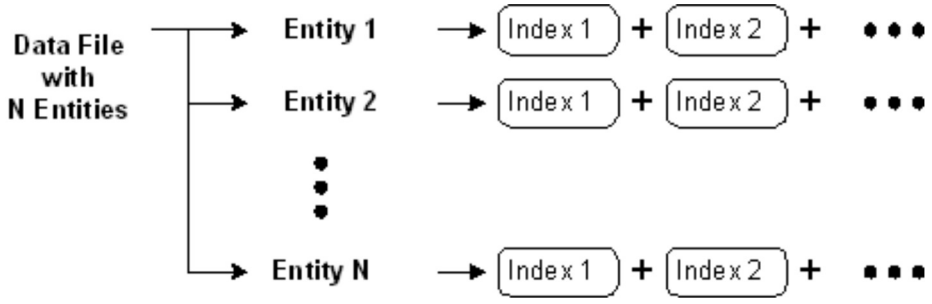
Segment Entities – Short, time-stamped segments of digitized analog signals in which the segments are separated by variable amounts of time. Segment Entities can contain data from more than one source. They are intended to represent discontinuous analog signals such as extracellular spike waveforms from electrodes or groups of electrodes.

Neural Event Entities – Timestamps of event and segment entitities that are known to represent neural action potential firing times. For example, if a segment entity contains sorted neural spike waveforms, each sorted unit is also exported as a neural entity.

The API definition also provides functions for querying information about data files and the entities contained in the data file. This information includes labels, metric units, timings, etc.

Structure of File Data

Data entities in a data file are enumerated by the library from 1 to (total number of entities). This is different than the enumeration of the similar C++ Neuroshare API. The enumeration was changed because, in contrast to C++, all arrays in Matlab start at one. Each entity is one of the four types discussed in the *Representation of Dates Types* section above and there are no requirements for ordering entities by type.



Each entity contains one or more indexed data entries that are ordered by increasing time. The API provides functions for querying the characteristics of the file, the number of entities, and the characteristics of each entity, including the number of indexes for each entity.

The structure of the indexed data entries for each entity depends on the entity type:

Each index of an **event entity** refers to a timestamp and data combination. The number of indexes is equal to the number of event entries for that event entity in the data file.

Each index of an **analog entity** refers to a specific digitized sample. Each analog entity contains samples from a single channel and the number of indexes is equal to the number of samples present for that channel.

Each index of a **segment entity** refers to a short, time-stamped segment of analog data from one or more sources. The number of indexes is equal to the number of entries for that segment entity in the file.

Each index of a **neural event entity** refers to a timestamp for each neural event. Each neural event entity contains event times for a single neural source. The number of indexes is equal to the number of entries for that neural event entity.

Summary of Library Functions

The API library functions are organized in this document according to the following categories:

Managing DLLs

ns_SetLibrary – assign the DLL to be used for future function calls

Library Version Information

ns_GetLibraryInfo – get library version information

Managing Neural Data Files

ns_OpenFile – opens a neural data file

ns_GetFileInfo – retrieves file information and entity counts

ns_CloseFile – closes a neural data file

General Entity Information

ns_GetEntityInfo – retrieves general entity information and type

Accessing Event Entities

ns_GetEventInfo – retrieves information specific to event entities

ns_GetEventData – retrieves event data by index

Accessing Analog Entities

ns_GetAnalogInfo – retrieves information specific to analog entities

ns_GetAnalogData – retrieves analog data by index

Accessing Segment Entities

ns_GetSegmentInfo – retrieves information specific to segment entities

ns_GetSegmentSourceInfo – retrieves information about the sources that generated the segment data

ns_GetSegmentData – retrieves segment data by index

Accessing Neural Event Entities

ns_GetNeuralInfo – retrieves information for neural event entities

ns_GetNeuralData – retrieves neural event data by index

Searching Entity Indexes

ns_GetIndexByTime – retrieves an entity index by time

ns_GetTimeByIndex – retrieves time range from entity indexes

Searching Entity Indexes

ns_GetLastErrorMsg – retrieves the most recent text error message

Library Function Returns

All of the Neuroshare API functions return a 32-bit integer declared as type ns_RETURN. This value is always zero (ns_OK) if the function succeeds. The complete enumeration of the return values are listed below:

Return Code	Value	Description
ns_OK	0	Function successful
ns_LIBERROR	-1	Generic linked library error
ns_TYPEERROR	-2	Library unable to open file type
ns_FILEERROR	-3	File access or read error
ns_BADFILE	-4	Invalid file handle passed to function
ns_BADENTITY	-5	Invalid or inappropriate entity identifier specified
ns_BADSOURCE	-6	Invalid source identifier specified
ns_BADINDEX	-7	Invalid entity index specified

Description of Library Functions

ns_SetLibrary – Assign the DLL to use for all function calls

Usage:

[ns_RESULT] = ns_SetLibrary('filename.dll')

Description:

Assign the DLL to use for all function names

Return Values:

ns_RESULT This function returns ns_OK if the file is successfully found. Otherwise one of the following error codes is generated:

ns_LIBERROR	Library Error

ns_GetLibraryInfo - Get library version information

Usage:

[ns_RESULT, nsLibraryInfo] = ns_GetLibraryInfo

Description:

Obtains information about the API library.

Return Values:

nsLibraryInfo ns_LIBRARYINFO structure to receive library version information.

struct ns_LIBRARYINFO	
double *LibVersionMaj;*	Major version number of this library.
double *LibVersionMin;*	Minor version number of this library.
double *APIVersionMaj;*	Major version # of API specification that library complies with
double *APIVersionMin;*	Minor version # of API specification that library complies with
char *Description[64]*;	Text description of the library.
char *Creator[64]*;	Name of library creator.
double *Time_Year;*	Year of last modification date
double *Time_Month;*	Month (0-11; January = 0) of last modification date
double *Time_Day;*	Day of the month (1-31) of last modification date
double *Flags*;	Additional library flags.
double *MaxFiles*	Maximum number of files library can simultaneously open.
double *FileDescCount*;	Number of valid description entries in the following array.
ns_FILEDESC *FileDesc[16]*;	Text descriptor of files that the DLL can interpret.
struct ns_FILEDESC	
char *Description[32];*	Text description of the file type or file family
char *Extension[8];*	Extension used on PC, Linux, and Unix Platforms.
char *MacCodes[8];*	Application and Type Codes used on Mac Platforms.
char *MagicCode[16];*	null-terminated code used at the file beginning.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_LIBERROR	Library Error

ns_OpenFile - Opens a neural data file

Usage:

[ns_RESULT, hFile] = ns_OpenFile('filename.ext')

Description:

Opens the file specified by filename and returns a file handle, hFile, that is used to access the opened file.

Parameters:

filename Pointer to a null-terminated string that specifies the name of the file to open.

Return Values:

hFile Handle/Identification number to the opened file.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_TYPEERROR	Library unable to open file type
ns_FILEERROR	File access or read error

Remarks:

All files are opened for read-only, as no writing capabilities have been implemented. If the command succeeds in opening the file, the application should call ns_CloseFile for each open file before terminating.

This function has to be called before any other Neuroshare function is called.

ns_GetFileInfo - Retrieves file information and entity counts

Usage:

[ns_RESULT, nsFileInfo] = ns_GetFileInfo(hFile)

Description:

Provides general information about the data file referenced by hFile. This information is returned in the structure nsFileInfo.

Parameters:

hFile Handle/Identification number to an open file.

Return Values:

nsFileInfo ns_FILEINFO structure that receives the file information.

struct ns_FILEINFO

char *FileType[32]*;	Human readable manufacturer's file type descriptor.
double *EntityCount*;	Number of entities in the data file. This number is used to enumerate all the entities in the data file from 1 to dwEntityCount and to identify each entity in function calls (dwEntityID).
double *TimeStampResolution*	Minimum timestamp resolution in seconds.
double *TimeSpan;*	Time span covered by the data file in seconds.
char *AppName[64]*;	Information about the application that created the file.
double *Time_Year;*	Year.
double *Time_Month;*	Month (0-11; January = 0).
double *Time_Day;*	Day of the month (1-31).
double *Time_Hour;*	Hour since midnight (0-23).
double *Time_Min;*	Minute after the hour (0-59).
double *Time_Sec;*	Seconds after the minute (0-59).
double *Time_MilliSec;*	Milliseconds after the second (0-1000).
char *FileComment[256]*;	Comments embedded in the source file.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_FILEERROR	File access or read error
ns_BADFILE	Invalid file handle passed to function

ns_CloseFile - Closes a neural data file

Usage:

ns_RESULT = ns_CloseFile(hFile)

Description:

Closes a previously opened file specified by the file handle hFile.

Parameters:

hFile Handle/Identification number to an open file.

Return Values:

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE Invalid file handle passed to function.

ns_GetEntityInfo - Retrieves general entity information and type

Usage:

[ns_RESULT, nsEntityInfo] = ns_GetEntityInfo(hFile, EntityID)

Description:

Retrieves general information about the entity, EntityID, from the file referenced by the file handle hFile. The information is passed in the structure nsEntityInfo.

Parameters:

hFile Handle/Identification number to an open file.

EntityID Identification number of the entity in the data file.

The total number of entities in the data file is provided by the member EntityCount in the ns_FILEINFO structure.

Return Values:

nsEntityInfo ns_ENTITYINFO structure to receive entity information

struct ns_ENTITYINFO

char *EntityLabel[32]*;	Specifies the label or name of the entity.
double *EntityType*;	Flag specifying the type of entity data recorded on this channel. It can be one of the following:
	Unknown entity 0
	Event entity 1
	Analog entity 2
	Segment entity 3
	Neural event entity 4
double *ItemCount*;	Number of data items for the specified entity in the file.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_FILEERROR	File access or read error

ns_GetEventInfo - Retrieves information specific to event entities

Usage:

[ns_RESULT, nsEventInfo] = ns_GetEventInfo(hFile, EntityID)

Description:

Retrieves information from the file referenced by hFile about the Event Entity, EntityID, in the structure nsEventInfo.

Parameters:

hFile Handle/Identification number to an open file.

EntityID Identification number of the entity in the data file.

Return Values:

nsEventInfo ns_EVENTINFO structure to receive the Event Entity information.

struct ns_EVENTINFO

double *EventType*;	A type code describing the type of event data associated with each indexed entry. The following information types are allowed: Text string 0 Comma separated values 1 8-bit binary values 2 16-bit binary values 3 32-bit binary values 4
double *MinDataLength*;	Minimum number of bytes that can be returned for an Event.
double *MaxDataLength*;	Maximum number of bytes that can be returned for an Event.
char *CSVDesc[128]*;	Provides descriptions of the data fields for CSV Event Entities.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_FILEERROR	File access or read error

ns_GetEventData - Retrieves event data by index

Usage:

[ns_RESULT, TimeStamp, Data, DataSize] = ns_GetEventData(hFile, EntityID, Index)

Description:

Returns the data values from the file referenced by hFile and the Event Entity EntityID. The Event data entry specified by Index is written to Data and the timestamp of the entry is returned to TimeStamp. Upon return of the function, the value at DataSize contains the number of bytes actually written to Data.

Parameters:

hFile	Handle/Identification number to an open file.
EntityID	Identification number of the entity in the data file.
Index	The index number of the requested Event data item.

Return Values:

TimeStamp	Variable that receives the timestamp of the Event data item.
Data	Variable that receives the data for the Event entry. The format of data is specified by the member EventType in ns_EVENTINFO.
DataSize	Variable that receives the actual number of bytes of data retrieved in the data buffer.
ns_RESULT	This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_BADINDEX	Invalid entity index specified
ns_FILEERROR	File access or read error

ns_GetAnalogInfo - Retrieves information specific to analog entities

Usage:

[ns_RESULT, nsAnalogInfo] = ns_GetAnalogInfo(hFile, EntityID)

Description:

Returns information about the Analog Entity associated with EntityID and the file hFile. The information is stored in nsAnalogInfo structure.

Parameters:

hFile Handle/Identification number to an open file.

EntityID Identification number of the entity in the data file.

Return Values:

nsAnalogInfo ns_ANALOGINFO structure to receive the Analog Entity information.

struct ns_ANALOGINFO

double *SampleRate;*	The sampling rate in Hz used to digitize the analog values.
double *MinVal*;	Minimum possible value of the input signal.
double *MaxVal;*	Maximum possible value of the input signal.
char *Units[16];*	Specifies the recording units of measurement.
double *Resolution;*	Minimum input step size that can be resolved. (E.g. for a +/- 1 Volt 16-bit ADC this value is .0000305).
double *LocationX;*	X coordinate of source in meters.
double *LocationY;*	Y coordinate of source in meters.
double *LocationZ;*	Z coordinate of source in meters.
double *LocationUser;*	Additional manufacturer-specific position information (e.g. electrode number in a tetrode).
double *HighFreqCorner;*	High frequency cutoff in Hz of the source signal filtering.
double *HighFreqOrder;*	Order of the filter used for high frequency cutoff.
char *HighFilterType[16];*	Type of filter used for high frequency cutoff (text format).
double *LowFreqCorner;*	Low frequency cutoff in Hz of the source signal filtering.
double *LowFreqOrder;*	Order of the filter used for low frequency cutoff.
char *LowFilterType[16];*	Type of filter used for low frequency cutoff (text format)..
char *ProbeInfo[128];*	Additional text information about the signal source.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_FILEERROR	File access or read error

ns_GetAnalogData - Retrieves analog data by index

Usage:

[ns_RESULT, ContCount, Data] = ns_GetAnalogData(hFile, EntityID, StartIndex, IndexCount)

Description:

Returns the data values associated with the Analog Entity indexed EntityID in the file referenced by hFile. The index of the first data value is StartIndex and the requested number of data samples is given by IndexCount. The requested data values are returned in the variable Data. Although the samples in an analog entity are indexed, they are not guaranteed to be continuous in time and may contain gaps between some of the indexes. When the requested data is returned, ContCount contains the number of continuous data points present in the data (starting at StartIndex). If the index range specified by StartIndex and IndexCount contains invalid indexes, the function will return ns_BADINDEX.

Parameters:

hFile	Handle/Identification number to an open file.
EntityID	Identification number of the Analog Entity in the data file.
StartIndex	Starting index number of the analog data item.
IndexCount	Number of analog values to retrieve.

Return Values:

ContCount	Number of continuous data values starting with
StartIndex	This field is ignored if the pointer is set to NULL.
Data	Array of double precision values to receive the analog data.
ns_RESULT	This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_BADINDEX	Invalid entity index or range specified
ns_FILEERROR	File access or read error

ns_GetSegmentInfo - Retrieves information specific to segment entities

Usage:

[ns_RESULT, nsSegmentInfo] = ns_GetSegmentInfo(hFile, EntityID)

Description:

Retrieves information on the Segment Entity, EntityID, in the file referenced by the handle hFile. The information is written to nsSegmentInfo.

Parameters:

hFile Handle/Identification number to an open file.

EntityID Identification number of the entity in the data file.

Return Values:

nsSegmentInfo ns_SEGMENTINFO structure that receives segment information for the requested Segment Entity.

```
struct ns_SEGMENTINFO
    double SourceCount;     Number of sources contributing to the Segment Entity data.
                            For example, with tetrodes, this number would be 4.
    double MinSampleCount;  Minimum number of samples in each Segment data item.
    double MaxSampleCount;  Maximum number of samples in each Segment data item.
    double SampleRate;      The sampling rate in Hz used to digitize source signals.
    char Units[32];         Specifies the recording units of measurement.
```

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_FILEERROR	File access or read error

ns_GetSegmentSourceInfo - Retrieves information about the sources that generated the segment data

Usage:

[ns_RESULT, nsSegmentSourceInfo] = ns_GetSegmentSourceInfo(hFile, EntityID, SourceID)

Description:

Retrieves information about the source entity, SourceID, for the Segment Entity identified by EntityID, from the file referenced by the handle hFile. The information is written to the nsSegmentSourceInfo.

Parameters:

hFile Handle/Identification number to an open file.

EntityID Identification number of the Segment Entity.

SourceID Identification number of the Segment Entity source (integer from 0 to SourceCount – 1 which can be found in ns_SEGMENTINFO structure).

Return Values:

nsSegmentSourceInfo ns_SEGSOURCEINFO structure that receives information about the source.

struct ns_SEGSOURCEINFO

double *MinVal*;	Minimum possible value of the input signal.
double *MaxVal;*	Maximum possible value of the input signal.
double *Resolution;*	Minimum input step size that can be resolved. (e.g. for a +/- 1 Volt 16-bit ADC this value is .0000305).
double *SubSampleShift*;	Time difference (in sec) between the nominal timestamp and the actual sampling time of the source probe.
double *LocationX;*	X coordinate of source in meters.
double *LocationY;*	Y coordinate of source in meters.
double *LocationZ;*	Z coordinate of source in meters.
double *LocationUser;*	Additional manufacturer-specific position information (e.g. electrode number in a tetrode).
double *HighFreqCorner;*	High frequency cutoff in Hz of the source signal filtering.
double *HighFreqOrder;*	Order of the filter used for high frequency cutoff.
char *HighFilterType[16];*	Type of filter used for high frequency cutoff (text format).
double *LowFreqCorner;*	Low frequency cutoff in Hz of the source signal filtering.
double *LowFreqOrder;*	Order of the filter used for low frequency cutoff.
char *LowFilterType[16];*	Type of filter used for low frequency cutoff (text format).
char *ProbeInfo[128];*	Additional text information about the signal source.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_BADSOURCE	Invalid source identifier specified
ns_FILEERROR	File access or read error

ns_GetSegmentData - Retrieves segment data by index

Usage:

[ns_RESULT, TimeStamp, Data, SampleCount, UnitID] =

ns_GetSegmentData(hFile, EntityID, Index)

Description:

Returns the Segment data values in entry Index of the entity EntityID from the file referenced by hFile. The data values are returned in Data. The timestamp of the entry id returned in TimeStamp. The number of samples written to Data is returned in SampleCount. The data variable should be accessed as a 2-dimensional array for samples and sources.

Parameters:

hFile	Handle/Identification number to an open file.
EntityID	Identification number of the entity in the data file.
Index	Index number of the requested Segment data item.

Remarks:

A zero unit ID is unclassified, then follow unit 1, 2, 3, etc. Unit 255 is noise.

Return Values:

TimeStamp	Time stamp of the requested Segment data item.
Data	Variable to receive the requested data.
SampleCount	Number of samples returned in the data variable.
UnitID	Unit classification code for the Segment Entity.
ns_RESULT	This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_BADINDEX	Invalid entity index specified
ns_FILEERROR	File access or read error

ns_GetNeuralInfo - Retrieves information for neural event entities

Usage:

[ns_RESULT, nsNeuralInfo] = ns_GetNeuralInfo(hFile, EntityID)

Description:

Retrieves information on Neural Event entity EntityID from the file referenced by hFile. The information is returned in the nsNeuralInfo.

Parameters:

hFile Handle/Identification number to an open file.

EntityID Identification number of the entity in the data file.

Return Values:

nsNeuralInfo ns_NEURALINFO structure to receive the Neural Event information.

struct ns_NEURALINFO

double *SourceEntityID;*	Optional ID number of a source entity. If the Neural Event is derived from other entity sources, such as Segment Entities, this value links the Neural Event back to the source.
double *SourceUnitID;*	Optional sorted unit ID number used in the source Entity.
char *ProbeInfo[128];*	Text information about the source probe or the label of a source Segment Entity.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_FILEERROR	File access or read error

ns_GetNeuralData - Retrieves neural event data by index

Usage:

[ns_RESULT, Data] = ns_GetNeuralData(hFile, EntityID, StartIndex, IndexCount)

Description:

Returns an array of timestamps for the neural events of the entity specified by EntityID and referenced by the file handle hFile. The index of the first timestamp is StartIndex and the requested number of timestamps is given by IndexCount. The timestamps are returned in Data.

Parameters:

hFile	Handle/Identification number to an open file.
EntityID	Identification number of the entity in the data file.
StartIndex	First index number of the requested Neural Events timestamp.
IndexCount	Number of timestamps to retrieve.

Return Values:

Data Array of double precision timestamps.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_BADINDEX	Invalid entity index specified
ns_FILEERROR	File access or read error

ns_GetIndexByTime - Retrieves an entity index by time

Usage:

[ns_RESULT, Index] = ns_GetIndexByTime(hFile, EntityID, Time, Flag)

Description:

Searches in the file referenced by hFile for the data item identified by the index EntityID. The flag specifies whether to locate the data item that starts before or after the time Time. The index of the requested data item is returned in Index.

Parameters:

hFile Handle/Identification number to an open file.

EntityID Identification number of the entity in the data file.

Time Time of the data to search for

Flag Flag specifying whether the index to be retrieved belongs to the data item occurring before or after the specified time Time. The flags are defined:

Return the data entry occurring before and inclusive of the time dTime.	-1
Return the data entry occurring at or closest to the time dTime.	0
Return the data entry occurring after and inclusive of the time dTime.	+1

Return Values:

Index Variable to receive the entry index.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_FILEERROR	File access or read error
ns_BADINDEX	Unable to find an valid index

ns_GetTimeByIndex - Retrieves time range from entity indexes

Usage:

[ns_RESULT, Time] = ns_GetTimeByIndex(hFile, EntityID, Index)

Description:

Retrieves the timestamp for the entity identified by EntityID and numbered Index, from the data file referenced by hFile. The timestamp is returned in Time.

Parameters:

hFile	Handle/Identification number to an open file.
EntityID	Identification number of the entity in the data file.
Index	Index of the requested data.

Return Values:

Time	Variable to receive the timestamp.
ns_RESULT	This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_BADFILE	Invalid file handle passed to function
ns_BADENTITY	Invalid or inappropriate entity identifier specified
ns_BADINDEX	Invalid entity index specified
ns_FILEERROR	File access or read error

ns_GetLastErrorMsg - Retrieves the extended error message for the last error

Usage:

[ns_RESULT, LastError] = ns_GetLastErrorMsg

Description:

Returns the last error message in text form to LastError. This function should be called immediately following a function whose return value indicates that an error occurred. The maximum size of the error message text is 256 characters.

Return Values:

LastError Variable to receive the extended last error message.

ns_RESULT This function returns ns_OK if the file is successfully opened. Otherwise one of the following error codes is generated:

ns_LIBERROR Library error

Example Matlab Program

```
function Example()

% Prompt for the correct DLL
disp(' ');  % Blank line
DLLName = input('DLL Name: ', 's');

% Load the appropriate DLL
[nsresult] = ns_SetLibrary(DLLName);
if (nsresult ~= 0)
   disp('DLL was not found!');
   return
end

% Find out the data file from user
disp(' ');  % Blank line
filename = input('Data file: ', 's');


% Load data file and display some info about the file
% Open data file
[nsresult, hfile] = ns_OpenFile(filename);
if (nsresult ~= 0)
   disp('Data file did not open!');
   return
end
clear filename;

% Get file information
[nsresult, FileInfo] = ns_GetFileInfo(hfile);
% Gives you EntityCount, TimeStampResolution and TimeSpan
if (nsresult ~= 0)
   disp('Data file information did not load!');
   return
end

% Define some variables needed for firing rates
stepsize = 0.02; % seconds
stepsize1 = 0.1; % seconds
if FileInfo.TimeSpan > 150  % Limit the timespan shown in the graphs to 150 seconds
   totaltime = 150;
else
   totaltime = FileInfo.TimeSpan; % seconds
end
time = 0 : stepsize : totaltime;   % Initialize time axis for gaussian plot

% Build catalogue of entities
[nsresult, EntityInfo] = ns_GetEntityInfo(hfile, [1 : 1 : FileInfo.EntityCount]);

NeuralList = find([EntityInfo.EntityType] == 4);    % List of EntityIDs needed to retrieve the information and data
SegmentList = find([EntityInfo.EntityType] == 3);
AnalogList = find([EntityInfo.EntityType] == 2);
EventList = find([EntityInfo.EntityType] == 1);

% How many of a particular entity do we have
cNeural = length(NeuralList);
cSegment = length(SegmentList);
cAnalog = length(AnalogList);
cEvent = length(EventList);

clear FileInfo;

if (cNeural == 0)
   disp('No neural events available!');
end
```

```
if (cSegment == 0)
   disp('No segment entities available!');
   return;     % It does not make sense to continue in this particular analysis
               % if there are no segment entities.
end

if (cAnalog == 0)
   disp('No analog entities available!');
end

if (cEvent == 0)
   disp('No event entities available!');
end

% Have user pick a channels or channels for further analysis
% Show the user how many channels are available
disp(' ');
disp(['There are ' num2str(cSegment) ' channels.']);
disp(' ');

channel = input('Which data channels would you like to display? (e.g. 1 or [1 2 3])');
if (channel > cSegment)  % Have to check that the selected channel actually exists
   disp('Channel does not exist');
   return
end
clear cNeural cSegment;

% Throw away entity infos we don't need to save memory
EntityInfo = rmfield(EntityInfo, 'EntityType');

%
% Load the waveform data and do the analysis
%

% These three functions are for demonstration purposes
[nsresult, nsSegmentInfo] = ns_GetSegmentInfo(hfile, SegmentList(channel));
[nsresult, nsSegmentSourceInfo] = ns_GetSegmentSourceInfo(hfile, SegmentList(channel), 1);

% Load the first 100 waveforms on each selected channel
[nsresult, timestamps_wf, waveforms, sampleCount, unitIDs] = ns_GetSegmentData(hfile, SegmentList(channel), [1 : 1 : 100]);
clear timestamps_wf sampleCount;

clear nsSegmentInfo;
clear nsSegmentSourceInfo;

%
% Reduce the data set to only the first 150 seconds of data
%

NeuralLabels = strvcat(EntityInfo(NeuralList).EntityLabel);

for cChannel = 1 : 1 : length(channel),
   % Have to figure out which Neural entities correspond with the selected segment entities
   list = strmatch(EntityInfo(SegmentList(channel(cChannel))).EntityLabel, NeuralLabels, 'exact');

   % Retrieve the data
   [nsresult, NeuralInfo] = ns_GetNeuralInfo(hfile, NeuralList(list));
   [nsresult, NeuralData] = ns_GetNeuralData(hfile, NeuralList(list), 1, max([EntityInfo(NeuralList(list)).ItemCount]));

   if (totaltime == 150)
      ind = find(NeuralData <= 150);
   else
      ind = [1:1:size(NeuralData, 1) * size(NeuralData, 2)];
   end

   % Get the neural timestamps
   NeuralData = reshape(NeuralData, size(NeuralData, 1) * size(NeuralData, 2), 1);
   timestamps(cChannel) = {NeuralData(ind)};
```

```
    % Match the neural events with their unit ID
    temp = ones(length(ind) / length(list), 1) * [NeuralInfo(:).SourceUnitID];
    units(cChannel) = {temp(:)};
    % Remember how many neural events were found
    ItemCount(cChannel) = length(timestamps{cChannel});

    NeuralData(:) = [];
end
clear NeuralData SegmentLabels NeuralLabels NeuralInfo;
EntityInfo = rmfield(EntityInfo, 'ItemCount');

% Close data file. Should be done by the library but just in case.
ns_CloseFile(hfile);

% Unload DLL
clear mexprog;

for cChannel = 1 : 1 : length(channel),
    % Plot neural activity on that channel
    figure;
    subplot(4, 1, 1);
    % Plot the first 100 waveforms on that channel and apply different
    % colors for different unitIDs
    % Unclassified units
    ind = find(unitIDs(:, cChannel) == 0);
    if (~isempty(ind))
        plot(waveforms(:, ind, cChannel), 'Color', [.8 .8 .8]); % in grey
        hold on;
    end
    % Classified Units
    Colors = ['y' 'm' 'c' 'r' 'g'];

    for cUnit = 1 : 1 : 5,
        ind = find(unitIDs(:, cChannel) == cUnit);
        if (~isempty(ind))
            plot(waveforms(:, ind, cChannel), Colors(cUnit));
            hold on;
        end
    end

    axis tight;
    title(['Waveforms - ' EntityInfo(SegmentList(channel(cChannel))).EntityLabel]);
    ylabel('Voltage (uV)');

    subplot(4, 1, 2);
    % This creates lines for the unit
    a = repmat(timestamps{cChannel}', 2, 1);          % X Values don't change so just two rows with the same values
    b = repmat([0.6; 1.4], 1, ItemCount(cChannel));   % Start and end points of each line
    % This plots the lines for each unit
    % Unclassified units
    ind = find(units{cChannel} == 0);
    hold on;
    if (~isempty(ind))
        plot(a(:, ind), b(:, ind), 'Color', [.8 .8 .8]); % in grey
    end

    % Classified Units
    for cUnit = 1 : 1 : 5,
        ind = find(unitIDs(:, cChannel) == cUnit);
        if (~isempty(ind))
            plot(a(:, ind), b(:, ind), Colors(cUnit));
        end
    end

    title(['Raster plot - ' EntityInfo(SegmentList(channel(cChannel))).EntityLabel]);
    xlabel('Time [s]');
    set(gca, 'ytick', []);
    xlim([0 round(totaltime)]);
    clear a b;
```

```
    % Use a sliding window 100 ms (delta t) long (equivalent to linear kernel)
    dt = 0.1; % ms
    for i = totaltime : -stepsize : 0,
        gaussdata(round(i / stepsize + 1)) = sum(exp(-(i - timestamps{cChannel}(find((timestamps{cChannel} > i - 4 * dt) &
(timestamps{cChannel} < i + 4 * dt)))').^2 / (2 * dt^2))) / sqrt(2 * 3.1415) / dt;
    end

    % Throw away firing rates higher than 200 because they are not real and show up at the beginning
    gaussdata(find(gaussdata > 200)) = 0;

    subplot(4, 1, 3);
    plot(time, gaussdata(1 : length(time)));
    title(['Approximated firing rate (Gaussian) - ' EntityInfo(SegmentList(channel(cChannel))).EntityLabel]);
    xlabel('Time [s]');
    ylabel('Rate [Hz]');
    clear gaussdata;

    % Interspike Interval Distribution (1 ms resolution)
    if (length(timestamps{cChannel}) > 1)  % Have to make sure that we have at least two data points
        nbins = 500; % ms
        dtime = (timestamps{cChannel}(2 : end) - timestamps{cChannel}(1 : end - 1)) * 1000;

        subplot(4, 1, 4);
        bar(histc(dtime, 1 : 1 : nbins));
        axis tight;
        title(['Interspike Interval Histogram - ' EntityInfo(SegmentList(channel(cChannel))).EntityLabel]);
        xlabel('Time [ms]');
        ylabel('# Spikes');
        clear dtime;
    end
end
```