Neuroshare API Specification Rev0.9a

Application Programming Interface for Accessing Neurophysiology Experiment Data Files

March 2002

IMPORTANT: This specification is presently in development and not ready for general release. Revision suggestions are welcome at http://www.neuroshare.org.

AFFILIATIONS

This standard is being developed and maintained through the Neuroshare Project. The purpose of this project is to create open, standardized methods for accessing neurophysiological experiment data from a variety of different data formats, as well as open-source software tools based on these methods. All standards and software resulting from the Neuroshare Project are distributed and revised through the http://www.neuroshare.org web site. Additional contact information and project history can also be accessed through this site.

REVISIONS

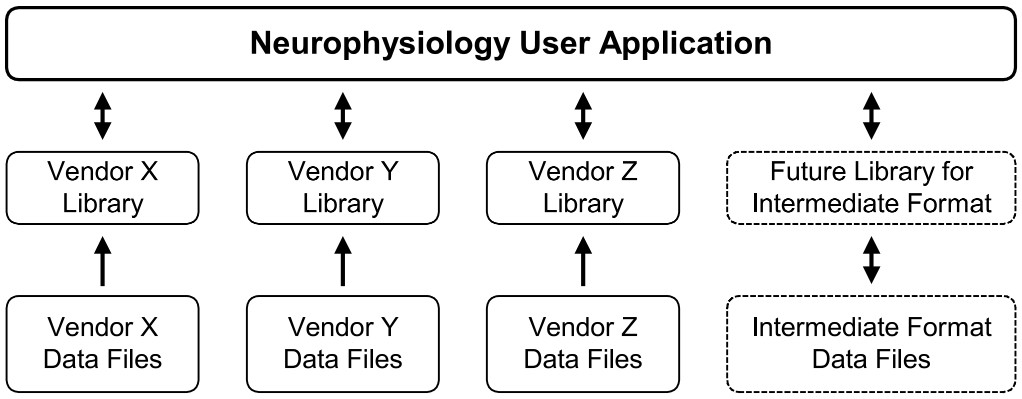
This is the first beta release of the Neuroshare API Specification for public review. This specification is currently in development and not for general usage.

Table of Contents

Intended Scope and Usage **4**
Representation of Data Types **5**
Representation of Time **5**
Conventions used in this Library Specification **6**
Structure of File Data **6**
Summary of Library Functions **8**
Primitive Data Types **9**
Library Function Returns **9**
Managing Neural Data Files **10**
ns_OpenFile *10*
ns_GetFileInfo *11*
ns_FILEINFO *12*
ns_CloseFile *13*
General Entity Information **14**
ns_GetEntityInfo *14*
ns_ENTITYINFO *15*
Accessing Event Entities **16**
ns_GetEventInfo *16*
ns_EVENTINFO *17*
ns_GetEventData *18*
Accessing Analog Entities **19**
ns_GetAnalogInfo *19*
ns_ANALOGINFO *20*
ns_GetAnalogData *21*
Accessing Segment Entities **22**
ns_GetSegmentInfo *22*
ns_SEGMENTINFO *23*
ns_GetSegmentSourceInfo *24*
ns_SOURCEINFO *25*
ns_GetSegmentData *26*
Accessing Neural Event Entities **27**
ns_GetNeuralEventInfo *27*
ns_NEURALINFO *28*
ns_GetNeuralData *29*
Searching Entity Indexes **30**
ns_GetIndexByTime *30*
ns_GetTimeByIndex *31*
Win32 DLL Structure **32**
Revision History **35**

Intended Scope and Usage

The purpose of this Application Programming Interface (API) standard is to define a common interface for accessing neurophysiology experiment data files. This common interface allows neurophysiology applications to access data in a variety of proprietary file formats through vendor-specific libraries. Such applications can include extracellular spike sorting programs, data visualization utilities, and high-level neuroscience data analysis programs.



As of this revision, the API only defines functions necessary to extract information from data files. At a future time, extensions may be provided to allow user applications to write/modify data files. It is also possible that a simple intermediate data file format, based on the data structures in this API, may be created for storing processed experiment information and sorted spike timing information generated from other spike classification programs.

When complete, this document will contain all of the information required to develop both libraries and user applications. Additional utilities and example source code for supporting the development of libraries and applications will be made available on the neuroshare.org web site as they are developed.

As of this revision, the specification is oriented towards 32-bit Microsoft Windows applications through the of Dynamic Link Libraries (DLLs). Whenever possible, portable coding conventions will be used to support future ports to other 32-bit and 64-bit operating systems.

Representation of Data Types

Although there are many types of data sources possible in neurophysiological experiments, this API definition abstracts data into four basic catagories or "Entity Types". These are:

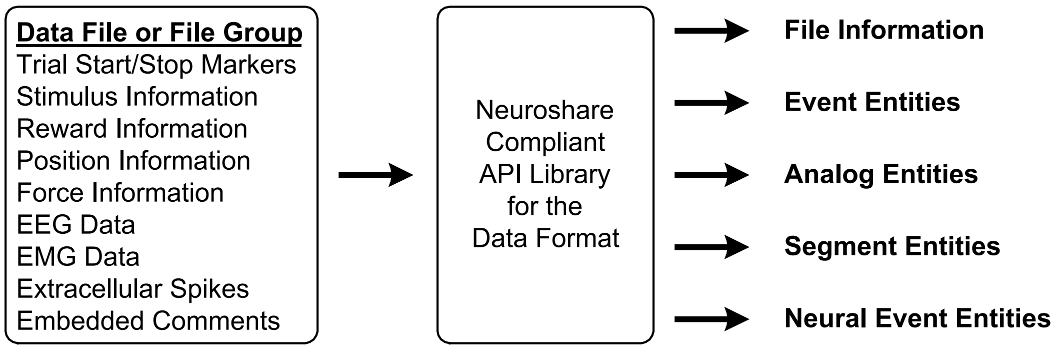
Event Entities – Discrete events that consist of small time-stamped text or binary data packets. These are used to represent data such as trial markers, experimental events, digital input values, and embedded user comments.

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 – Neural Events are a special case of Event Entities that contain only timestamps for representing neuron firing times. This type is included in addition to Event Entities due to the special significance of neural events to neurophysiology experiments.

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.



Representation of Time

The API definition assumes that each data file consists of a single span of time. The timings of all data presented by the library to user applications are referenced to the beginning of this span.

Some data formats organize data according to trials in which time is recorded within each trial, but not between trials. Libraries that access these types of files must combine these trials into a single time span and present an event entity which marks the beginning of each trial. Although this is somewhat awkward, this abstraction makes the organization of trial-based files equivalent to single time span files that use event markers to delineate trials.

Conventions used in this Library Specification

The function definitions and data structures presented in this document will be specified according to the C language syntax and convention. However, the actual language used to write the libraries is irrelevant as libraries use a common linkage format for exported functions.

All Neuroshare-specific functions, constants and data types will include a "ns_" prefix.

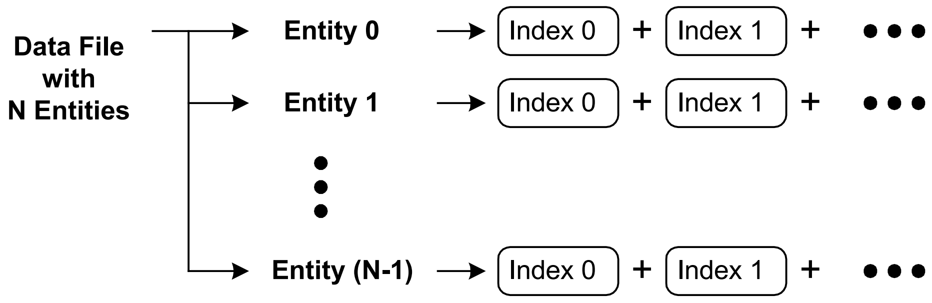
The API functions in this specification utilize several text fields for descriptions, such as labels, user comments, electrode locations, etc. The use of human readable text is encouraged wherever possible in these fields along with simplified data representations. For example, if a vendor uses a proprietary data packet format for position information in experiments, the vendor is encouraged to include library code that presents this data as analog entities with labels such as "POS X" and "POS Y". In this initial version of the specification, all text information will be reported in 8-bit ASCII format.

All analog values in this library, including time, shall use a 64-bit double-precision floating point representation. All analog entities also include a text field for reporting measurement units such as "meters", "MPa", "kg". The use of metric units is strongly encouraged. Time is always reported in seconds.

Structure of File Data

Data entities in a data file are enumerated by the library from 0 to (total number of entities –1). 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.

The API provides unified functions for searching for index ranges of an entity of any type by time range, and functions are also provided to report the timing of an entity index.

The data abstraction listed above is somewhat demanding on the libraries as it requires them to organize, temporally sort and report data in the file according to type. This structure was chosen to simplify the data representation for user applications that must analyze the data in these files. The libraries were selected as the best place for this re-organization of data to occur as most libraries have access to special knowledge about the particular file formats that they must handle. It would be highly inefficient and complicated for user applications to import data from serial packet streams into catalogs of available data with time and index search capabilities.

The prototypical loading sequence for the library and data files can be summarized by the following pseudo-code:

```
Load Needed Library;

Open Neural Data File;

     Query Number of Entities;

     For Each Entity,
          Get Entity Type;
          Get Type Specific Entity Information;

     Repeat Main Operational Loop,
          Determine Entities of Interest;
          Search for Needed Indexes of Relevant Entities;
          Retrieve the Data for the Relevant Entities;
          Do Application-Specific Processing and Display;
     While Still Interested;

Close Neural Data File;

Unload Library;
```

Although this pseudo-code sequence outlines the typical operations for accessing a single file, the API functions are specified to allow multiple files or file groups to be opened simultaneously. Libraries must properly manage memory to support this functionality.

Summary of Library Functions

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

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

All Neuroshare-compliant libraries must export all of the above functions along with platform specific functions for opening, closing and dynamically linking libraries (e.g., the DllMain() function in Win32 DLLs).

The following sections that provide the details of these functions also include data structure definitions used by its functions.

Primitive Data Types

To avoid ambiguity across platforms, the following primitive data types are explicitly defined:

`char`	8-bit character value normally reserved for ASCII strings
`int8`	8-bit (1 byte) signed integers
`uint8`	8-bit (1 byte) unsigned integers
`int16`	16-bit (2 byte) signed integers
`uint16`	16-bit (2 byte) unsigned integers
`int32`	32-bit (4 byte) signed integers
`uint32`	32-bit (4 byte) unsigned integers
`double`	64-bit, double precision floating point value

All of the data structures and functions detailed in this specification will use the above data types. In this API specification, data types in functions and structures are rigidly defined so that endianess issues should not be a problem in properly written code. Developers are discouraged from making assumptions about byte ordering in the above primitive data types.

Library Function Returns

All of the Neuroshare API functions return a 32-bit unsigned 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`	`0x00`	Function Successful
`ns_LIBERROR`	`0x01`	Linked Library Error
`ns_TYPEERROR`	`0x02`	Library unable to open file type
`ns_FILEERROR`	`0x04`	File access or read Error
`ns_BADFILE`	`0x10`	Invalid file handle passed to function
`ns_BADENTITY`	`0x20`	Invalid or inappropriate entity identifier specified
`ns_BADSOURCE`	`0x40`	Invalid source identifier specified
`ns_BADINDEX`	`0x80`	Invalid entity index specified

Managing Neural Data Files

The following functions open and close neurophysiological data files and provide general file information.

ns_OpenFile

Usage

ns_RESULT ns_OpenFile (const char **pszFilename,* uint32 **hFile*)

Description

Opens the file specified by *pszFilename* and returns a file handle, *hFile*, that can be used to access the opened file.

Parameters

pszFilename	Pointer to a null-terminated string that specifies the name of the file to open.
hFile	Handle to the opened file. This value is returned by the function and is used for subsequent file operations.

Return Values

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.

ns_GetFileInfo

Usage

ns_RESULT ns_GetFileInfo (uint32 *hFile*, ns_FILEINFO **pFileInfo*,
uint32 *dwFileInfoSize*);

Description

Provides general information about the data file referenced by *hFile*. This information is returned in the structure pointed to by *pFileInfo*. The size of the file information structure is given by *dwFileInfoSize*.

Parameters

hFile	Handle to an open file.
pFileInfo	Pointer to the ns_FILEINFO structure that receives the file information.
dwFileInfoSize	Size of the ns_FILEINFO structure in bytes.

Return Values

This function returns ns_OK if the file information is successfully retrieved. 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_FILEINFO

```
typedef struct {
    char szFileType[32];             // Human readable manufacturer’s file type descriptor.
    uint32 dwEntityCount;            // Number of entities in the data file.  This number is used
                                     // to enumerate all the entities in the data file from 0 to
                                     // (dwEntityCount –1) and to identify each //entity in
                                     // function calls (dwEntityID).
    double dwTimeStampResolution     // Minimum timestamp resolution.
    double dTimeRange;               // Time range covered by the data file in seconds.
    char szAppName[64];              // Information about the application that created the file.
    uint16 wTime_Year;               // Year.
    uint16 wTime_Month;              // Month (0-11; January = 0).
    uint16 wTime_DayofWeek;          // Day of the week (0-6; Sunday = 0).
    uint16 wTime_Day;                // Day of the month (1-31).
    uint16 wTime_Hour;               // Hour since midnight (0-23).
    uint16 wTime_Min;                // Minute after the hour (0-59).
    uint16 wTime_Sec;                // Seconds after the minute (0-59).
    uint16 wTime_MilliSec;           // Milliseconds after the second (0-1000).
    char szFileComment[256];         // Comments embedded in the source file.
} ns_FILEINFO;
```

Remarks

The time variables in the ns_FILEINO structure refer to the beginning of the time span to which the data is referenced.

ns_CloseFile

Usage

ns_RESULT ns_CloseFile (uint32 *hFile*);

Description

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

Parameters

hFile	Handle to an open file.

Return Values

This function always returns ns_OK.

General Entity Information

The functions described below provide general information about the data entities in the file. The total number of data entities available can be obtained from the ns_FILEINFO structure. The entities are enumerated from 0 to (the number of entities - 1). All of the subsequent information and data access functions require an entity to be specified in the *dwEntitityID* field.

ns_GetEntityInfo

Usage

ns_RESULT ns_GetEntityInfo (uint32 *hFile,* uint32 *dwEntityID*,
ns_ENTITYINFO **pEntityInfo,* uint32 *dwEntityInfoSize*)**;**

Description

Retrieves general information about the entity, *dwEntityID,* from the file referenced by the file handle *hFile*. The information is passed in the structure pointed to by *pEntityInfo* with a size of *dwEntityInfoSize* bytes.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file. The total number of entities in the data file is provided by the member *dwEntityCount* in the ns_FILEINFO structure.
pEntityInfo	Pointer to a ns_ENTITYINFO structure to receive entity information.
dwEntityInfoSize	Size of ns_ENTITYINFO structure in bytes.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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_ENTITYINFO

```
typedef struct {
    char szEntityLabel[32];        // Specifies the label or name of the entity.
    uint32 dwEntityType;           // Flag specifying the type of entity data recorded on this
                                   // channel.  It can be one of the following:
                                   // # define ns_ENTITY_EVENT             1
                                   // # define ns_ENTITY_ANALOG            2
                                   // # define ns_ENTITY_SEGMENT           3
                                   // # define ns_ENTITY_NEURALEVENT       4
    int32 dwItemCount;             // Number of data items for the specified entity in the file.
} ns_ENTITYINFO;
```

<u>Accessing Event Entities</u>

The following functions retrieve information and data for Event Entities.

ns_GetEventInfo

<u>Usage</u>

ns_RESULT ns_GetEventInfo (uint32 *hFile,* uint32 *dwEntityID*,
ns_EVENTINFO **pEventInfo,* uint32 *dwEventInfoSize*);

<u>Description</u>

Retrieves information from the file referenced by hFile about the Event Entity, *dwEntityID,* in the structure pointed to by *pEventsInfo.* The structure has a size of *dwEventInfoSize* bytes.

<u>Parameters</u>

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
pEventsInfo	Pointer to a ns_EVENTINFO structure to receive the Event Entity information.
dwEventInfoSize	Size of the ns_EVENTINFO structure in bytes.

<u>Return Values</u>

This function returns ns_OK if the information is successfully retrieved. 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_EVENTINFO

```
typedef struct {
    uint32 dwEventType;         // A type code describing the type of event data associated with
                                // each indexed entry.  The following information types are
                                // allowed:
                                // #define ns_EVENT_TEXT    0   //text string
                                // #define ns_EVENT_CSV     1   //comma separated values
                                // #define ns_EVENT_BYTE    2   // 8-bit binary value
                                // #define ns_EVENT_WORD    3   //16-bit unsigned integer
                                // #define ns_EVENT_DWORD   4        //32-bit unsigned
                                                                integer
  uint32 dwMinDataLength;       // Minimum number of bytes that can be returned for an Event.
  uint32 dwMaxDataLength;       // Maximum number of bytes that can be returned for an Event.
  char szCSVDesc [128];         // Provides descriptions of the data fields for CSV Event Entities.
} ns_EVENTINFO;
```

ns_GetEventData

Usage

ns_RESULT ns_GetEventData (uint32 *hFile*, uint32 *dwEntityID*, uint32 *nIndex*, double **pdTimeStamp*, void **pData,* uint32 **pdwDataSize*);

Description

Returns the data values from the file referenced by *hFile* and the Event Entity *dwEntityID*. The Event data entry specified by *nIndex* is written to *pData* and the timestamp of the entry is returned to *pdTimeStamp*. Upon return of the function, the value at *pdwDataSize* contains the number of bytes actually written to *pData*.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
nIndex	The index number of the requested Event data item.
pdTimeStamp	Pointer to a variable that receives the timestamp of the Event data item.
pData	Pointer to a buffer that receives the data for the Event entry. The format of Event data is specified by the member *dwEventType* in ns_EVENTINFO.
pdwDataSize	Pointer to a variable that receives the actual number of bytes of data retrieved in the data buffer.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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

Accessing Analog Entities

The following functions retrieve information and data for Analog Entities.

ns_GetAnalogInfo

Usage

ns_RESULT ns_GetAnalogInfo (uint32 *hFile*, uint32 *dwEntityID,*
ns_ANALOGINFO **pAnalogInfo,*
uint32 *dwAnalogInfoSize*);

Description

Returns information about the Analog Entity associated with *dwEntityID* and the file *hFile*. The information is stored in a ns_ANALOGINFO structure, addressed by the pointer *pAnalogSourceInfo*. The size of ns_ANALOGINFO is specified by *dwAnalogInfoSize*.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
pAnalogSourceInfo	Pointer to a ns_ANALOGINFO structure.
dwAnalogInfoSize	Size in bytes of ns_ANALOGINFO structure.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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_ANALOGINFO

```
typedef struct{
    double dSampleRate;         // The sampling rate in Hz used to digitize the analog values.
    double dMinVal;             // Minimum possible value of the input signal.
    double dMaxVal;             // Maximum possible value of the input signal.
    char szUnits[32];           // Specifies the recording units of measurement.
    double dResolution;         // Minimum input step size that can be resolved.
                                // (E.g. for a +/- 1 Volt 16-bit ADC this value is .0000305).
    double dLocationX;          // X coordinate of source in meters.
    double dLocationY;          // Y coordinate of source in meters.
    double dLocationZ;          // Z coordinate of source in meters.
    double dLocationUser;       // Additional manufacturer-specific position information
                                // (e.g. electrode number in a tetrode).
    double dHighFreqCorner;     // High frequency cutoff in Hz of the source signal filtering.
    uint32 dwHighFreqOrder;     // Order of the filter used for high frequency cutoff.
    char szHighFilterType[16];  // Type of filter used for high frequency cutoff (text format).
    double dLowFreqCorner;      // Low frequency cutoff in Hz of the source signal filtering.
    uint32 dwLowFreqOrder;      // Order of the filter used for low frequency cutoff.
    char szLowFilterType[16];   // Type of filter used for low frequency cutoff (text format)..
    char szProbeInfo[128];      // Additional text information about the signal source.
} ns_ANALOGINFO;
```

ns_GetAnalogData

Usage

ns_RESULT ns_GetAnalogData (uint32 *hFile*, uint32 *dwEntityID*, uint32 *dwStartIndex*, uint32 *dwIndexCount*, double **pData*);

Description

Returns the data values associated with the Analog Entity indexed *dwEntityID* in the file referenced by *hFile*. The index of the first data value is *nStartIndex* and the requested number of data values is given by *dwIndexCount*. The requested data values are returned in the buffer pointed to by *pData*.

If the index range specified by *dwStartIndex* and *dwIndexCount* contains invalid indexes, the function will return ns_BADINDEX.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
dwStartIndex	Starting index number of the analog data item.
dwIndexCount	Number of analog values to retrieve.
pData	Pointer to an array of double precision values to receive the analog data. The user application must allocate sufficient space to hold *dwIndexCount* double values or *dwIndexCount**sizeof(double) bytes.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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 or range specified
`ns_FILEERROR`	File access or read error

Accessing Segment Entities

The following functions retrieve information and data for Segment Entities.

ns_GetSegmentInfo

Usage

```
ns_RESULT ns_GetSegmentInfo (uint32 hFile, uint32 dwEntityID,
                             ns_SEGMENTINFO *pdwSegmentInfo,
                             uint32 dwSegmentInfoSize);
```

Description

Retrieves information on the Segment Entity, *dwEntityID,* in the file referenced by the handle *hFile*. The information is written to the ns_SEGMENTINFO structure at *pdwSegmentInfo.* The size of the ns_SEGMENTINFO structure is specified by *dwSegmentInfoSize*.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
pdwSegmentInfo	Pointer to the structure ns_SEGMENTINFO that receives general segment information for the Segment Entity.
dwSegmentInfoSize	Size of the structure ns_SEGMENTINFO in bytes.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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_SEGMENTINFO

```
typedef struct {
    uint32 dwSourceCount;        // Number of sources contributing to the Segment Entity data.
                                 // For example, with tetrodes, this number would be 4.
    uint32 dwMinSampleCount;     // Minimum number of samples in each Segment data item.
    uint32 dwMaxSampleCount;     // Maximum number of samples in each Segment data item.
    double dSampleRate;          // The sampling rate in Hz used to digitize source signals.
    char szUnits[32];            // Specifies the recording units of measurement.
} ns_SEGMENTINFO;
```

ns_GetSegmentSourceInfo

Usage

```
ns_RESULT ns_GetSegmentSourceInfo (uint32 hFile, uint32 dwEntityID,
                                   uint32 dwSourceID,
                                   ns_SOURCEINFO *pSourceInfo,
                                   uint32 dwSourceInfoSize);
```

Description

Retrieves information about the source entity, *dwSourceID,* for the Segment Entity identified by *dwEntityID*, from the file referenced by the handle *hFile*. The information is written to the ns_SOURCEINFO structure pointed to by *pSourceInfo*. The size of the structure is given by *dwSourceInfoSize* in bytes (sizeof(ns_SOURCEINFO)).

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
dwSourceID	Identification number of the Segment Entity source.
pSourceInfo	Pointer to a ns_SOURCEINFO structure that receives Segment Entity information about the source.
dwSourceInfoSize	Size of ns_SOURCEINFO structure in bytes.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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

Remarks

The value of *dwSourceID* is an integer index value ranging from 0 to *dwSourceCount -1* (which is returned by the function ns_GetSegmentInfo).

ns_SOURCEINFO

```
typedef struct {
    double dMinVal;              // Minimum possible value of the input signal.
    double dMaxVal;              // Maximum possible value of the input signal.
    double dResolution;          // Minimum input step size that can be resolved.
                                 // (E.g. for a +/- 1 Volt 16-bit ADC this value is .0000305).
    double dSubSampleShift;      // Time difference (in sec) between the nominal timestamp
                                 // and the actual sampling time of the source probe. This
                                 // value will be zero when all source probes are sampled
                                 // simultaneously.
    double dLocationX;           // X coordinate of source in meters.
    double dLocationY;           // Y coordinate of source in meters.
    double dLocationZ;           // Z coordinate of source in meters.
    double dLocationUser;        // Additional manufacturer-specific position information
                                 // (e.g. electrode number in a tetrode).
    double dHighFreqCorner;      // High frequency cutoff in Hz of the source signal filtering.
    uint32 dwHighFreqOrder;      // Order of the filter used for high frequency cutoff.
    char szHighFilterType[16];   // Type of filter used for high frequency cutoff (text format).
    double dLowFreqCorner;       // Low frequency cutoff in Hz of the source signal filtering.
    uint32 dwLowFreqOrder;       // Order of the filter used for low frequency cutoff.
    char szLowFilterType[16];    // Type of filter used for low frequency cutoff (text format)..
    char szProbeInfo[128];       // Additional text information about the signal source.
} ns_SOURCEINFO;
```

ns_GetSegmentData

Usage

ns_RESULT ns_GetSegmentData (uint32 *hFile*, uint32 *dwEntityID*, int32 *nIndex*,
double **pdTimeStamp,* double **pData,*
uint32 **pdwSampleCount*)**;**

Description

Returns the Segment data values in entry *nIndex* of the entity *dwEntityID* from the file referenced by *hFile.* The data values are returned in the buffer pointed to by *pData.* The timestamp of the entry is returned at the address pointed to by *pdTimeStamp.* The number of samples written to the buffer is returned at *pdwSampleCount*.

The data buffer should be accessed as a 2-dimensional array for samples and sources.

In C, the array would be declared as `double data[maxsamplecount][sourcecount];`
and the values would be referenced by `data[sample][source]`

With pointers, the reference would be `*(pData+(sample*sourcecount)+source)`

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
nIndex	The index number of the requested Segment data item.
pdTimeStamp	Pointer to the time stamp of the requested Segment data item.
pData	Pointer to the buffer that is to receive the requested data.
pdwSampleCount	Pointer to the number of samples returned in the data buffer.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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

Accessing Neural Event Entities

The following functions retrieve information and data for Neural Entities.

ns_GetNeuralEventInfo

Usage

ns_RESULT ns_GetNeuralInfo (uint32 *hFile,* uint32 *dwEntityID*,
ns_NEURALINFO **pNeuralInfo,*
uint32 *dwNeuralInfoSize*)**;**

Description

Retrieves information on Neural Event entity *dwEntityID* from the file referenced by *hFile.* The information is returned in the structure ns_NEURALINFO at the address *pnNeuralInfo* The size of the structure ns_NEURALINFO is given by *dwNeuralInfoSize* in bytes.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
pNeuralInfo	Pointer to the ns_NEURALINFO structure to receive the Neural Event information.
dwNeuralInfoSize	Size of the structure ns_NEURALINFO in bytes.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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_NEURALINFO

```
typedef struct {
    uint32 dwSourceEntityID;     // 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.
    uint32 dwSourceUnitID;       // Optional sorted unit ID number used in the source Entity.
    char szProbeInfo[128];       // Text information about the source probe or the label of a
                                 // source Segment Entity.
} ns_NEURALINFO;
```

ns_GetNeuralData

Usage

ns_RESULT ns_GetNeuralData(uint32 *hFile*, uint32 *dwEntityID*, uint32 *dwStartIndex*, uint32 *dwIndexCount*, double **pData*)

Description

Returns an array of timestamps for the neural events of the entity specified by *dwEntityID* and referenced by the file handle *hFile*. The index of the first timestamp is *nStartIndex* and the requested number of timestamps is given by *dwIndexCount*. The timestamps are returned in the buffer pointed to by *pData*.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
dwStartIndex	First index number of the requested Neural Events timestamp.
dwIndexCount	Number of timestamps to retrieve.
pData	Pointer to an array of double precision timestamps. The user application must allocate sufficient space (*dwIndexCount**sizeof(double) bytes) to hold the requested data.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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

Searching Entity Indexes

All of the data access functions defined in this API enumerate their data entries by index. The functions described in this section can be used to link these indexes with time.

ns_GetIndexByTime

Usage

ns_RESULT ns_GetIndexByTime(uint32 *hFile,* uint32 *dwEntityID*, double *dTime*, int32 *nFlag*, uint32 **pdwIndex*)

Description

Searches in the file referenced by *hFile* for the data item identified by the index *dwEntityID*. The flag specifies whether to locate the data item that starts before the time *dTime* or after the time *dTime*. The index of the requested data item is returned at *pdwIndex.*

Parameters

hFile	Handle to an open file
dwEntityID	Identification number of the entity in the data file.
dTime	Time of the data to search for
nFlag	Flag specifying whether the index to be retrieved belongs to the data item occurring before or after the specified time *dTime*. The flags are defined:

```
#define ns_BEFORE    -1    // return the data entry occuring before
                           // (but not coincident with) the time dTime.

#define ns_CLOSEST   0     // return the data entry occuring at or closest
                           // to the time dTime

#define ns_AFTER     +1    // return the data entry occuring after
                           // (but not coincident with) the time dTime.
```

pdwIndex	Pointer to variable to receive the entry index.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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 given the search parameters

ns_GetTimeByIndex

Usage

ns_RESULT ns_GetTimeByIndex(uint32 *hFile,* uint32 *dwEntityID*, uint32 *dwIndex*,
double **pdTime*)

Description

Retrieves the timestamp for the entity identified by *dwEntityID* and numbered *dwIndex*, from the data file referenced by *hFile.* The timestamp is returned at *pdTime.*

Parameters

hFile	Handle to an open file
dwEntityID	Identification number of the entity in the data file.
dwIndex	Index of the requested data.
pdTime	Pointer to the variable to receive the timestamp.

Return Values

This function returns ns_OK if the information is successfully retrieved. 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

Win32 DLL Structure

On windows architectures neuroshare library DLLs should be kept in a "neuroshare" folder in the windows directory (e.g. C:\WinNT\neuroshare) for easy location and management. Applications may query the pathname of the Windows OS directory by using the Win32 GetWindowsDirectory() function or checking the "windir" environmental variable.

All Win32 DLLs contain a Version resource with multiple fields for defining titles, versions, languages, architecture information, copyrights, and comments. These fields can be viewed by looking at the "Version" panel under the files properties within Windows, or accessed by applications through the Win32 GetFileVersionInfo() function. The Version resource for Neuroshare-compliant DLL Libraries must contain a "Language Neutral, Unicode (0x000004b0)" block descriptor and there are special requirements for the FileDescription and Comments fields of this block descriptor:

> The FileDescription text field must begin with "Neuroshare API Library" and may include additional text such as "for .MNO and .XYZ files" .
>
> The Comments text field must be composed of comma-delimited text fields that describe the API specification version and the types of files that the library can open. The specification revision is provided in the format "Major.minor" so that libraries written according to revision 1.0 would have "1.0," at the beginning of their comment fields. The next pair of fields should be a plain text descriptor of the types of files that the library can interpret and the file extension. For example, "NeuroCompany NEU file, *.neu" would be used for a company with a *.neu file type. If a library can recognize additional types, additional description/extension pairs can be added.

The Version resource of the DLL may contain additional block descriptors for other languages. However, the "Language Neutral" block should be the first in the series. Unicode is used here for portability among different language versions of Windows.

These requirements for the Version resource make it possible to query the specification version and capabilities of a neuroshare-compliant DLL library without actually loading the DLL into memory. Sophisticated applications should use this information for properly formatted File-Open dialog windows. Simple applications can ignore this information and present the user with a list of available DLLs so that the user can directly select the intended library.

The following Windows application code provides an example of how to read the FileDescription and Comments fields from the Version resource of a DLL. In this code, the version information buffer is statically allocated to 2048 bytes (enough for the FileDesc and Comments fields) to simplify the programming.

```
#include <winver.h>     // NOTE: must also link with version.lib

char   *filedesc_str;
char   *comments_str;
UINT   filedesc_len;
UINT   comments_len;
DWORD  dummyhandle;
BYTE   version_info_buf[2048];

GetFileVersionInfo(   "C:\\WinNT\\Neuroshare\\neuroinc.dll",
                      dummyhandle, 2048, &version_info_buf );

if ( !VerQueryValue(  &version_info_buf,
                      TEXT("\\StringFileInfo\\000004b0\\FileDescription"),
                      (void**)&filedesc_str, &filedesc_len ) ) return ERROR;

if ( !VerQueryValue(  &version_info_buf,
                      TEXT("\\StringFileInfo\\000004b0\\Comments"),
                      (void**)&comments_str, &comments_len ) ) return ERROR;
```

The Version resource should also include the VOS_WINDOWS32 and VFT_DLL flags in the FILEOS and FILETYPE descriptor fields.

The functions in Neuroshare Win32 DLLs must be declared compiled for Run-Time Dynamic Linking so that they can be loaded through the Win32 LoadLibrary() and GetProcAddress() functions. The preferred convention is to include a DllMain() function to internally manage the loading and attachment of the library to running threads.

The mechanics of writing Win32 DLLs are described in the "Platform SDK / Windows Base Functions / Executables / Dynamic Link Libraries" section of the Microsoft Developers Network (MSDN) help system that is included with Visual C++. This topic can also be reached through Microsoft's developer website (http://msdn.microsoft.com/). Example source code will also be made available through the neuroshare.org web site.

The following Win32 code section demonstrates how to load a 32-bit Windows DLL and execute the DLL function named "Function1."

```
//Define the function prototype
int  Function1(double dOneParam, int nTwoParam);

//Define the type for the function to use for type-casting
typedef int (*fnType1)( double dOneParam, int nTwoParam);

//Load library and get a handle to it
fnType1   function1;   //Function pointer
HINSTANCE hInstDLL;    //Handle to library DLL
hInstDLL = LoadLibrary("C:\\WinNT\Neuroshare\\neuroinc.dll ");

if (hInstDLL != NULL)
{
    //Get pointer to the function with the name "Function1" in the library
    function1= (fnType1) GetProcAddress(hInstDLL, "Function1");

    if (!function1)
    {
            // handle the error
            FreeLibrary(hDLL);
            return ;
    }
    else
    {
            //Call the function.  Parenthesis around the function tells the compiler
            //the it is a pointer to a function and to call the function pointed at.
            double dParam1 = 123.45;
            int nParam2 = -10;
            int nRetVal = (function1)( dParam1, nParam2);
    }
}

//Clean up
FreeLibrary(hInstDLL);
```

Revision History

Revision 0.9a –Beta draft produced after the first working group meeting (Jan 16-18, 2002). This meeting included Tim Bergel (Cambridge Electronic Design Ltd.), Charlotte Gruner (Pronghorn Engineering), Shane Guillory (Bionic Technologies, LLC), Hans Löffler (Multi Channel Systems MCS GmbH), Thane Plummer (Neuralynx Inc.), Tony Reina (The Neurosciences Institute), Casey Stengel (Neuralynx Inc.), Angela Wang (Bionic Technologies, LLC), Harvey Wiggins (Plexon Inc.), and Willard Wilson (Tucker-Davis Technologies). Draft compiled by Shane Guillory and Angela Wang and published for public review and comment on March 27, 2002.