Neuroshare API Specification
Rev 0.9c

Application Programming Interface for Accessing Neurophysiology Experiment Data Files

January 2003

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.

DISCLAIMER

This specification document is provided "as is" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. The Neuroshare project and the working group disclaim all liability relating to the use of information in this specification.

TRADEMARKS

Windows and Microsoft are registered trademarks of the Microsoft Corporation. All other product names are trademarks or servicemarks of their respective owners.

REVISIONS

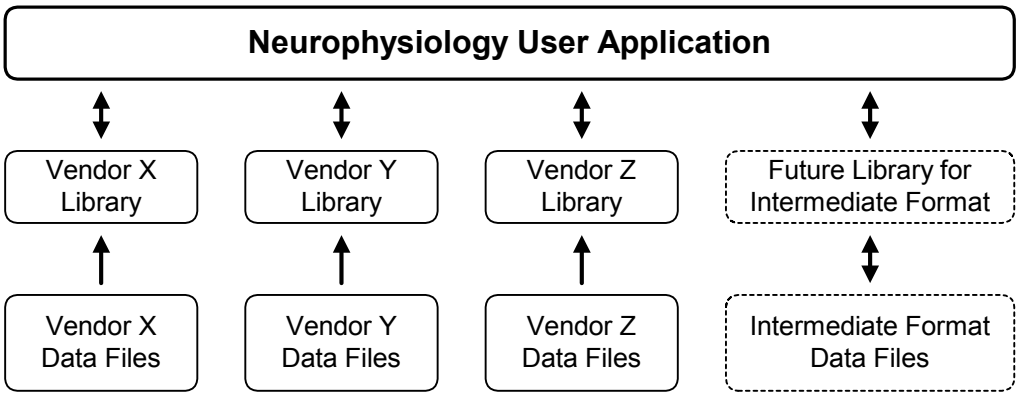
This is the first beta release of the Neuroshare API Specification for public review. This specification is currently in development and not for general usage.

COPYRIGHT AND DISTRIBUTION

Contents

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 use 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 categories or "Entity Types". These are:

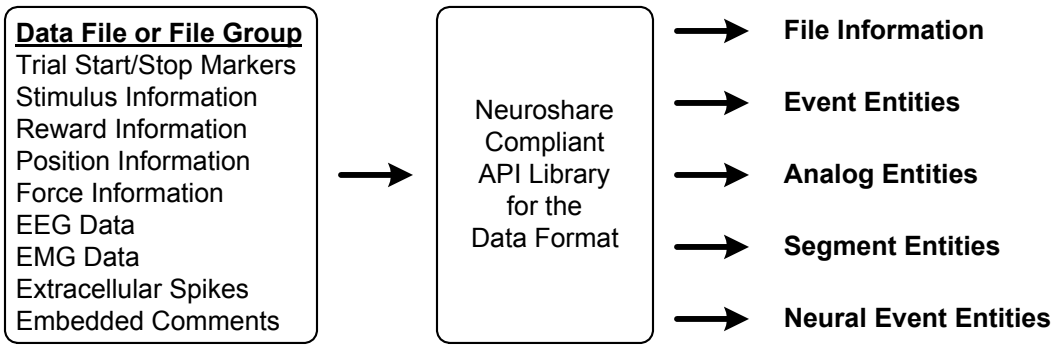
Event Entities – Discrete events that consist of small time-stamped text or binary data packets. These are used to represent data such as trial markers, experimental events, digital input values, and embedded user comments.

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. Analog Entities may also contain gaps in time from data files that do not record data between experimental trials.

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. If an event entity is known by the library to only contain neuron firing times, it should be exported as a neural event entity instead of an event entity. This entity provides a simple, efficient representation of neural firing times for high-level neuroscience analysis programs. It avoids the problems of requiring applications to look for spike timing within different entity types and the problems associated with decoding one or more unit firing times from within single segment entries.

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

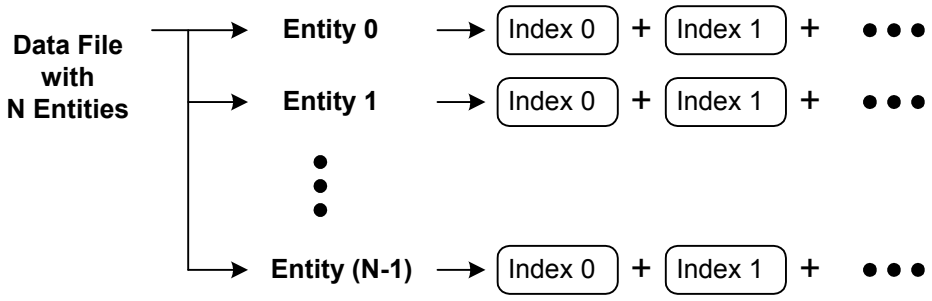
The API definition assumes that each data file consists of a single span of time. The timings of all data presented by the library to user applications are referenced to the beginning of this span.

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 that 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.

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. The time interval between successive samples is not always the sampling period as there may be gaps in the acquisition of the data.

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;
      Get General File Information;
      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 a **minimum of 64** open files. If system resources constrain the number of simultaneously opened files, the library open-file function reports a system error.

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.

Summary of Library Functions

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

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

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 data structures required by the above functions are defined following the calling function specification.

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.

The default alignment for library data structure members is 4 bytes and the structures have been declared with this alignment and should require no padding. Future revisions of this library format will add fields to the end of these structures. Unsupported or unused fields in data structures should return zero.

Library Function Arguments

Modifiable arguments are passed to functions by pointers in this specification. If a reference argument, or pointer, is not to retrieve data values, it is set to NULL in the function call.

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

Multiple Instance and Multithreaded Operation

It is important to recognize that API DLLs may be loaded simultaneously by more than one application. In Win32 operating systems, each DLL is loaded and executed within its own virtual memory space by default. However, it is possible for DLLs to register global memory spaces for data that are shared between multiple executing copies of the same DLL. In situations where multiple applications use a library to access the same data file, it may be advantageous for libraries to share some memory regarding data files. This type of memory sharing is beyond the scope of this library definition and left to the developers of each API DLL.

In modern multithreaded operating systems, it is possible for users to write an application that can call DLL functions simultaneously in more than one thread. For applications using Neuroshare libraries, this will probably not lead to increased data throughput due to disk and operating system bottlenecks. However, there are some situations that might benefit from multithreaded access to libraries and data. For example, it is not uncommon for spike-sorting applications to allow users to display and configure one channel while operating on another.

For Neuroshare API libaries, the decision of whether or not to make a library safe for multithreaded operation is left to the library developer. Libraries can report their thread safety level in the ns_LIBRARYINFO data structure returned by the ns_GetLibraryInfo function. Libraries that claim to be safe for simultaneous, pre-empted calls to their functions must include the spin locks or key atomic accesses necessary for this mode of operation.

Library Loading and Resource Allocation

Libraries will need to allocate system resources when loaded to manage internal variables and open files. However, applications may open several libraries simply to call their ns_GetLibraryInfo functions as part of an open-file dialog box. Because of this, library developers are encouraged to minimize the amount of system resources used by libraries until files are opened with the library.

Library Version and File Support Information

ns_GetLibraryInfo

Usage

ns_RESULT ns_GetLibraryInfo (ns_LIBRARYINFO **pLibraryInfo,*
uint32 *dwLibraryInfoSize*)

Description

Obtains information about the API library.

Parameters

pLibInfo	Pointer to structure to receive library version information.
dwLibInfoSize	Size in bytes of ns_LIBRARYINFO structure.

Return Values

This function returns ns_OK if the data is successfully retrieved. Otherwise one of the following error codes is generated:

`ns_LIBERROR`	Library Error

ns_LIBRARYINFO

```
typedef struct {
    uint32 dwLibVersionMaj;          // Major version number of library.
    uint32 dwLibVersionMin;          // Minor version number of library.
    uint32 dwAPIVersionMaj;          // Major version number of API specification that library complies with
    uint32 dwAPIVersionMin;          // Minor version number of API specification that library complies with
    char szDescription[64];          // Text description of the library.
    char szCreator[64];              // Name of library creator.
    uint32 dwTime_Year;              // Year of last modification date
    uint32 dwTime_Month;             // Month (0-11; January = 0) of last modification date
    uint32 dwTime_Day;               // Day of the month (1-31) of last modification date
    uint32 dwFlags;                  // Additional library flags.
    uint32 dwMaxFiles                // Maximum number of files library can simultaneously open.
    uint32 dwFileDescCount;          // Number of valid description entries in the following array.
    ns_FILEDESC FileDesc[16];        // Text descriptor of files that the DLL can interpret.
} ns_LIBRARYINFO;
```

Remarks

Flags defined at this time are:

```
#define ns_LIBRARY_DEBUG          0x01   // includes debug info linkage
#define ns_LIBRARY_MODIFIED       0x02   // file was patched or modified
#define ns_LIBRARY_PRERELEASE     0x04   // pre-release or beta version
#define ns_LIBRARY_SPECIALBUILD   0x08   // different from release version
#define ns_LIBRARY_MULTITHREADED  0x10   // library is multithread safe
```

The dwFileDescCount and FileDesc fields provide a method for the library to describe the file types that it is capable of opening. The ns_LIBRARYINFO structure provides room for up to 16 file types. The number of valid ns_FILEDESC structures are reported in dwFileDescCount. Unused ns_FILEDESC structures should be set to all zeros or not returned.

Neural Event Files File formats that consist of pools of files in a directory that belong to a single data set should be opened with an index file or one of the pool member files.

```
typedef struct {
    char szDescription[32];          // Text description of the file type or file family
    char szExtension[8];             // Extension used on PC, Linux, and Unix Platforms.
    char szMacCodes[8];              // Application and Type Codes used on Mac Platforms.
    char szMagicCode[16];            // null-terminated code used at the file beginning.
} ns_FILEDESC;
```

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 is 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 within the library.

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.

The file handle *hFile* is a file enumeration created by the library and is recognizable only within the library. If the file is invalid or there is no file associated with it, a NULL file handle is returned.

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 dTimeStampResolution     // Minimum timestamp resolution in seconds.
    double dTimeSpan;               // Time span covered by the data file in seconds.
    char szAppName[64];             // Information about the application that created the file.
    uint32 dwTime_Year;             // Year.
    uint32 dwTime_Month;            // Month (0-11; January = 0).
    uint32 dwTime_Day;              // Day of the month (1-31).
    uint32 dwTime_Hour;             // Hour since midnight (0-23).
    uint32 dwTime_Min;              // Minute after the hour (0-59).
    uint32 dwTime_Sec;              // Seconds after the minute (0-59).
    uint32 dwTime_MilliSec;         // Milliseconds after the second (0-1000).
    char szFileComment[256];        // Comments embedded in the source file.
} ns_FILEINFO;
```

Remarks

The time and date variables in the ns_FILEINO structure refer to the beginning (time zero in the source file) 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 returns ns_OK when the file is successfully closed. Otherwise the following error code is generated:

`ns_BADFILE` Invalid file handle passed to function.

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 index 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_UNKNOWN        0
                                 // # 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;
```

Accessing Event Entities

The following functions retrieve information and data for Event Entities.

ns_GetEventInfo

Usage

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

Description

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.

Parameters

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.

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_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 values
                               // #define ns_EVENT_WORD      3   //16-bit binary values
                               // #define ns_EVENT_DWORD     4   //32-bit binary values
  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 *dwIndex*,
double **pdTimeStamp*, void **pData,*
uint32 *dwDataBufferSize*, uint32 **pdwDataRetSize*);

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*. *dwDataBufferSize* specifies the size in bytes allocated to the buffer pointed to by *pData*, and *pdwDataRetSize* specifies the actual amount of data in bytes retrieved to the buffer.

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the entity in the data file.
dwIndex	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.
dwDataBufferSize	The number of bytes allocated to the receiving data buffer.
pdwDataRetSize	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[16];           // 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*, uint32 **pdwContCount,*

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 samples is given by *dwIndexCount*. The requested data values are returned in the buffer pointed to by *pData*.

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, *pdwContCount* contains the number of continuous data points present in the data (starting at *dwStartIndex)*.

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 Analog Entity in the data file.
dwStartIndex	Starting index number of the analog data item.
dwIndexCount	Number of analog values to retrieve.
pdwContCount	Number of continuous data values retrieved. This field is ignored if the pointer is set to NULL.
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. If this pointer is NULL, no data is returned

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 requested 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_SEGSOURCEINFO **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_SEGSOURCEINFO structure pointed to by *pSourceInfo*. The size of the structure is given by *dwSourceInfoSize* in bytes (sizeof(ns_SEGSOURCEINFO)).

Parameters

hFile	Handle to an open file.
dwEntityID	Identification number of the Segment Entity.
dwSourceID	Identification number of the Segment Entity source.
pSourceInfo	Pointer to a ns_SEGSOURCEINFO structure that receives information about the source.
dwSourceInfoSize	Size of ns_SEGSOURCEINFO 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_SEGSOURCEINFO

```
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_SEGSOURCEINFO;
```

ns_GetSegmentData

Usage

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

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 size in bytes allocated to the data buffer is specified by *dwDataBufferSize*. The timestamp of the entry is returned at the address pointed to by *pdTimeStamp.* The actual number of samples written to the data 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.
dwDataBufferSize	Size in bytes allocated to the data buffer indicated by *pData*
pdwSampleCount	Pointer to the number of samples returned in the data buffer.
pdwUnitID	Pointer to the unit classification code for the Segment Entity.

Remarks

The *pdwUnitID* field is a bit-field supporting multiple classification codes. A zero unit ID is unclassified, bit 0 is set if the segment is noise or an artifact, bit 1 indicates unit 1 is present, bit 2 indicates that unit 2 is present, etc.

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_GetNeuralInfo

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 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
                          // and inclusive of 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
                          // and inclusive of 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

Extended Error Message Handler

The following function reports an extended text message about the last error returned from a function call.

ns_GetLastErrorMsg

Usage

ns_RESULT ns_GetLastErrorMsg(char *pszMsgBuffer, uint32 dwMsgBufferSize)

Description

Returns the last error message in formatted text form to the buffer pointed to by *pszMsgBuffer*. This function should be called immediately following a function whose return value indicates that such a call will return useful data. Otherwise, the error set by the failed function may be wiped out by more recent function calls. *dwMsgBufferSize* specifies the number of characters that can be stored in the message buffer.

The maximum size of the error message text is 256 characters.

Parameters

pszMsgBuffer Pointer to buffer to receive the text error message.
dwMsgBufferSize Size in bytes of the error message buffer.

Return Values

This function returns ns_OK if the information is successfully retrieved. Otherwise one of the following error codes is generated:

`ns_LIBERROR` Library 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. Applications should also check their root directory first for DLLs and give these DLLs precedence.

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 DLL Libraries should contain a "Language Neutral, Unicode (0x000004b0)" block descriptor and the FileDescription field of this block should begin with "Neuroshare API Library". For example, "Neuroshare API Library for .MNO and .XYZ files".

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.

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
           //that 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.

Revision 0.9b –Revisions made after first public review. Changes compiled by Shane Guillory and Angela Wang. Clarified the role of Neural Event Entities as abstractions of the neural timing information from Event and Segment entities. Added GetLibraryInfo function and supporting data structure and eliminated the DLL version method of getting library information. Added sections to discuss multi-instance, multi-threaded issues and provided method for libraries to report multithread support in the Get Library Info function. Changed analog data gap reporting method and clarified the descriptions. Added Unit Identification code field to the Segment Entity data functions. Changed error codes to negative, sequential values. Added minor language and grammatical corrections. July 19, 2002.

Revision 0.9c

1) P8. line 26. In order to emphasize that multiple data files can be opened at once, the relevant words are put in bold font. A minimum of 64 simultaneously open data files is required, system resources allowing.
2) A function to allow for extended error information reporting has been added.
   GetLastErrorMsg(char *pszMsgBuffer, uint32 dwMsgBufferSize).
4) Add parameter to indicate the size in bytes of the allocated buffer to receive the data in the following functions:
   ns_GetEventData (uint32 *hFile*, uint32 *dwEntityID*, uint32 *dwIndex*,
   double **pdTimeStamp*, void **pData,*
   uint32 *dwDataSize*, uint32 **pdwDataRetSize*);
   ns_GetSegmentData (uint32 *hFile*, uint32 *dwEntityID*, int32 *nIndex*,
   double **pdTimeStamp,* double **pData,*
   uint32 *dwDataBufferSize*, uint32 **pdwSampleCount*,
   uint32 **pdwUnitID)*
6) p 14, ln 26. Invalid file handles are defined to be NULL..
7) p 10. NULL function parameters that point to data, mean that no data for that parameter is to be returned
8) p 13. Two parameters added to ns_LIBRARYINFO to indicate the version number of the Neuroshare API Specification that the library complies with. Jan-28-2003 AW