Overview

This document contains a description of the different record formats that Neuralynx uses to store data recorded with Cheetah. If you are creating your own programs for reading data from a Neuralynx file or monitoring data over NetCom, you may want to use the structures provided in the NetCom Development Package (Nlx_DataTypes.h for C/C++ and MNetComClient*.DLL for .NET languages).

File Header

All files contain a 16 kilobyte ASCII text header. This header can be read using most text editors and is provided for informational purposes only. It is intended to give information regarding what settings were used when the data in the file was recorded. The information in the header will vary based on the type of data contained in the file, and items may be added, changed or removed from the header in newer versions of Cheetah. If you do use values from the header in your analysis, avoid hard coding specific strings in the header into your analysis program.

File Data

The data portion of the file begins immediately following the header. All Neuralynx files contain data that is stored as a sequence of individual records in binary format. The size and data contained in each record varies based on the type of data being stored in the file.

Record Formats

The record formats listed below store data in sequential memory in the order they are listed below. Individual records are also sent over NetCom when streaming data to a NetCom application. All data types listed below are using .NET format data types.

Continuously Sampled Record

Storage format for continuously sampled channel (CSC) recorded data. These files end in the NCS extension.

UInt64	qwTimeStamp	Cheetah timestamp for this record. This corresponds to the sample time for the first data point in the snSamples array. This value is in microseconds.
UInt32	dwChannelNumber	The channel number for this record. This is NOT the A/D channel number.
UInt32	dwSampleFreq	The sampling frequency (Hz) for the data stored in the snSamples Field in this record.
UInt32	dwNumValidSamples	Number of values in snSamples containing valid data.
Int16[ ]	snSamples	Data points for this record. Cheetah currently supports 512 data points per record. At this time, the snSamples array is a [512] array.

Event Record

Storage format for the event record. These files end in the NEV extension.

Int16	nstx	Reserved
Int16	npkt_id	ID for the originating system of this packet.
Int16	npkt_data_size	This value should always be two (2).
UInt64	qwTimeStamp	Cheetah timestamp for this record. This value is in microseconds.
Int16	nevent_id	ID value for this event.
Int16	nttl	Decimal TTL value read from the TTL input port.
Int16	ncrc	Record CRC check from Cheetah. Not used in consumer applications.
Int16	ndummy1	Reserved
Int16	ndummy2	Reserved
Int32[]	dnExtra	Extra bit values for this event. This array has a fixed length of eight (8).
String	EventString	Event string associated with this event record. This string consists of 127 characters plus the required null termination character. If the string is less than 127 characters, the remainder of the characters will be null.

Single Electrode Spike Record

Storage format for a single electrode spike record. These files end in the NSE extension.

UInt64	qwTimeStamp	Cheetah timestamp for this record. This value is in microseconds.
UInt32	dwScNumber	The spike acquisition entity number for this record. This is NOT the A/D channel number.
UInt32	dwCellNumber	The classified cell number for this record. If no cells have been classified, this number will be zero.
UInt32[]	dnParams	Array of selected feature data for this spike channel. Cheetah currently allows eight (8) selected features, so this array is fixed at length eight.
Int16[,]	snData	Data points for this record. Cheetah currently supports 32 data points per spike record. The array is organized as [DATAPOINTS, CHANNEL]. At this time, the snData array is a [32, 1] array.

Stereotrode Spike Record

Storage format for astereotrodespike record. These files end in the NST extension.

UInt64	qwTimeStamp	Cheetah timestamp for this record. This value is in microseconds.
UInt32	dwScNumber	The spike acquisition entity number for this record. This is NOT the A/D channel number.
UInt32	dwCellNumber	The classified cell number for this record. If no cells have been classified, this number will be zero.
UInt32[]	dnParams	Array of selected feature data for this spike channel. Cheetah currently allows eight (8) selected features, so this array is fixed at length eight.
Int16[,]	snData	Data points for this record. Cheetah currently supports 32 data points per spike record. The array is organized as [DATAPOINTS, CHANNEL]. At this time, the snData array is a [32, 2] array.

Tetrode Spike Record

Storage format for atetrodespike record. These files end in the NTT extension.

UInt64	qwTimeStamp	Cheetah timestamp for this record. This value is in microseconds.
UInt32	dwScNumber	The spike acquisition entity number for this record. This is NOT the A/D channel number.
UInt32	dwCellNumber	The classified cell number for this record. If no cells have been classified, this number will be zero.
UInt32[]	dnParams	Array of selected feature data for this spike channel. Cheetah currently allows eight (8) selected features, so this array is fixed at length eight.
Int16[,]	snData	Data points for this record. Cheetah currently supports 32 data points per spike record. The array is organized as [DATAPOINTS, CHANNEL]. At this time, the snData array is a [32, 4] array.

Video Tracker Record

Storage format for avideo tracker record. These files end in the NVT extension.

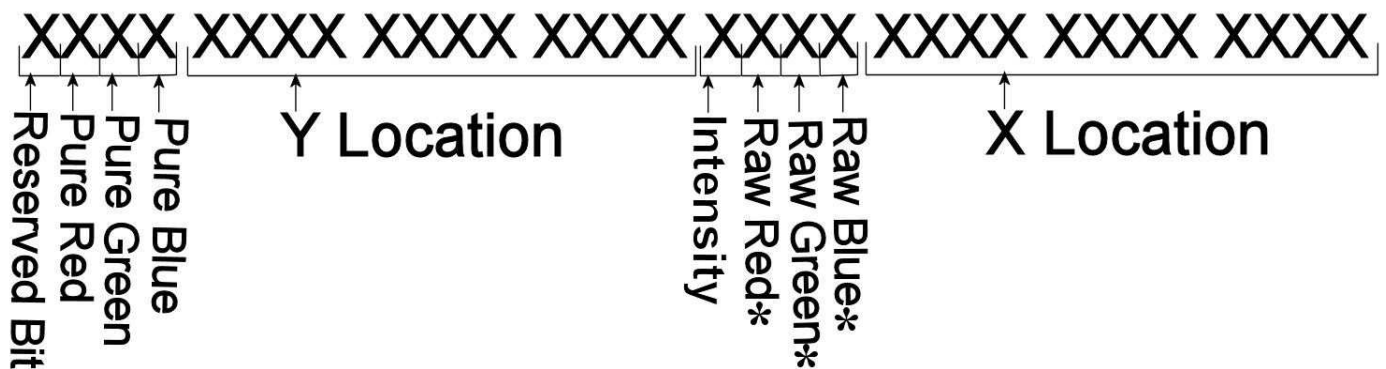| UInt16	swstx	Value indicating the beginning of a record. Always 0x800 (2048).
UInt16	swid	ID for the originating system of this record.
UInt16	swdata_size	Size of a VideoRec in bytes.
UInt64	qwTimeStamp	Cheetah timestamp for this record. This value is in microseconds.
UInt32[]	dwPoints	Points with the color bitfield values for this record. This is a 400 element array. See Video Tracker Bitfield Information below.
Int16	sncrc	Unused*
Int32	dnextracted_x	Extracted X location of the target being tracked.
Int32	dnextracted_y	Extracted Y location of the target being tracked.
Int32	dnextracted_angle	The calculated head angle in degrees clockwise from the positive Y axis. Zero will be assigned if angle tracking is disabled.**
Int32[]	dntargets	Colored targets using the samebitfield format used by the dwPoints array. Instead of transitions, the bitfield indicates the colors that make up each particular target and the center point of that target. This is a 50 element array sorted by size from largest (index 0) to smallest (index 49). A target value of 0 means that no target is present in that index location. See Video Tracker Bitfield Information below.

*Prior to Cheetah 5.0.0, this value was only used by the Cheetah160 VT to check for bit errors between the Cheetah160 box and the PC. It can safely be ignored for all versions of Cheetah.

**Prior to Cheetah 5.0.0 this contains invalid data and should be ignored.

Video Tracker Bitfield Information:

The pixel data consists of four hundred 32 bit values (one 32 bit value per pixel). The target data consists of fifty 32 bit values. These data have the same bit-field format which means that the 32 bit value is broken up into sub data to describe the X location (pixel number in the line), Y location (line number of the frame) and the tracker colors which were above and below threshold.

The X and Y values are allocated 12 bits each, but their maximum value is determined by the resolution that is used when tracking. See the header of your file for information about the resolution used when your file was recorded. The other bits indicate which of the color values were above (1) or below (0) their respective threshold setting. The layout of these bit fields can be visualized by the following:

These color signals are calculated using the following values from the RGB pixel:

Color Signal	Value
Intensity**	(Red + Green + Blue)/3
Raw Red*	Red
Raw Green*	Green
Raw Blue*	Blue
Pure Red***	Red - Intensity
Pure Green***	Green –Intensity
Pure Blue***	Blue –Intensity

*These values are only used in Cheetah 4.97 or earlier. In Cheetah 5 and newer, these values are always zero (0).
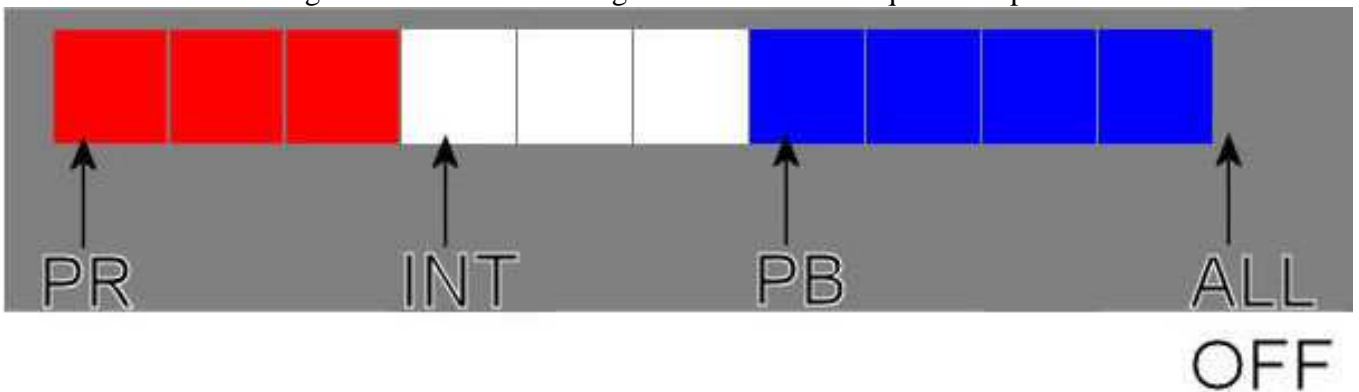** This value is labeled as Luminance in Cheetah 4.97 or earlier.
*** Cheetah 4.97 and earlier uses the following calculations:

Pure Red	Red - (Green + Blue)
Pure Green	Green - (Red + Blue)
Pure Blue	Blue - (Red + Green)

Example:

There is space in each video record for 400 transitions. This means that a total of 400 rising and falling transitions can occur for each record without overflowing the video record. Note that a single pixel may have more than a single color transition. If the Intensity, PureRed, PureGreen and PureBlue thresholds are enabled the following 4 transitions would be generated from the sequence of pixels shown:

Note that the PureRed transitions OFF at the first whitepixel and Intensity transitions off at the first blue pixel (where PureBlue transitions ON). Cheetah 5 and newer also has a built in noise filter that ignores transitions that do not last long enough to be considered a trackable object.

Raw Data A/D Record

Storage format for a raw dataA/D record. These files end in the NRD extension.* A/D records are not guaranteed to be complete, see the Processing Raw Data files section following this table for more information.

Int32	STX	Start of transmission identifier for the AD record. This value is always 2048.
Int32	Packet ID	Value to identify the type of packet. There is only one (1) type of packet for the NRD file format. This value is always 1.
Int32	Packet Size	This value is equal to the number of AD channels plus the number of Extra valuesin the packet.
UInt32	Timestamp High	The 32 high order bits that make up the 64 bit timestamp value. This value must be combined with the Timestamp Low value to create the full 64 bit Cheetah timestamp whose value is in microseconds. Please see Step 4 below for more information.
UInt32	Timestamp Low	The 32 low order bits that make up the 64 bit timestamp value. This value must be combined with the Timestamp High value to create the full 64 bit Cheetah timestamp whose value is in microseconds. Please see Step 4 below for more information.
Int32	Status	Reserved
UInt32	Parallel Input Port	The current value of the Parallel Input Port.
Int32[]	Extras	Reserved. This is currently an array ten (10) values containing reserved information from the hardware. While actual value of this array is not needed for data analysis, the number of values in this array needs to be taken into account when calculating the size of the Data block from the Packet Size value detailed above.
Int32[]	Data	Data values for the current record. The number of data values is based on the packet size value. The number of values in this array depends on the number of channels in your acquisition system. The size of this array is calculated by subtracting the number of Extra values from the Packet Size value detailed above.
Int32	CRC	The CRC value for the current record only.

*DMA Logfiles recorded with versions of Cheetah prior to 5.0.0 used the NDMA file extension. NDMA files recorded with Digital Lynx systems are compatible with the NRD A/D record format as described above. For information on NDMA files recorded with other acquisition systems, please contact support@neuralynx.com.

Processing Raw Data Files

A/D records within the file are not guaranteed to be valid or in sequential order. The most common occurrence of invalid records occurs at the beginning and end of recording sessions, where the recording process is started while the hardware is in the middle of processing a single record. These bad records can usually be ignored without affecting the quality of the recorded data. However, this means that all A/D records must first be checked for validity before data from the NRD file can be used. There are several steps that must be taken in order to verify the validity of each A/D record in the file.

Step 1: Finding the start of a record

First, the start of a valid record must be found. Data must be searched in 32 bit integer increments to find a valid STX value. A valid STX value of 2048must be found before attempting to verify the validity of the entire A/D record.

Step 2: Processing a record header

Next, the record's header (consisting of the STX, Packet ID and Packet Size fields) must be processed. The STX field must be equal to 2048 and the Packet ID field must be equal to1. The next integer value represents the size of the record to follow. As described in the above table, the Packet Size is the size of the Events array (currently 10 32bit integers) plus the length of the Data array. If any of these fields are incorrect, then the record is either incomplete or corrupt and another STX must be found via the first step in this process.

Step 3: Verifying the record CRC

The third step is to verify the validity of the record by performing a CRC check on all values in the A/D record. In order to accomplish this, a logical OR must be performed on all fields in the recordresulting in a value of 0.The following example illustrates how to check the CRC of a record:

C++ Example Code for CRC check

```
intrecordHeaderFooterSize = 18;
intadChannelCount = 32;
intrecordFieldCount = recordHeaderFooterSize + adChannelCount;
intfieldIndex = 0;
DWORD crcValue = 0;

//loop through each field in the record
for(fieldIndex = 0; fieldIndex<recordFieldCount; fieldIndex++ ) {
    //logically or all values in packet
crcValue ^= currentPacket[fieldIndex];
}

//check to see if we have a valid CRC value
if(crcValue != 0 ) {
    //Throw error here and discard packet
} else {
    //Process packet data
}
```

Step 4: Processing the record timestamp

Once the record header has been verified and the CRC check has been completed, it is now time to process data from the record. The first piece of data we will extract will be the timestamp. All Neuralynx timestamps are stored as 64 bit unsigned integers. Because AD record fields are all 32 bit integers, we must extract the Timestamp Low and Timestamp High fields from the record and combine them to build a 64 bit value. In this step, the order of the timestamps is checked to ensure that the current timestamp is greater than the previous one. If the timestamps are out of order, the current record should be discarded and the STX search process from step one should be repeated.

C++ example code for extracting and building the record timestamp

```
unsigned __int64 timestamp = 0;
inttimestampOffset = 3;
unsignedinttimestampLow = 0;

//extract the timestamp high value
timestamp = currentPacket[timestampOffset];

//because this value represents the high order bits of the timestamp we must
//shift the bits by 32 to go from the low order 32 bits to the high order 32
//bits
timestamp<<= 32;

//assign timestamp low field to variable
timestampLow = currentPacket[timestampOffset+1];

//load the low order bits from the timestamp low field variable into our current
//timestamp value
Timestamp += timestampLow;

// Because AD records are not guaranteed to arrive in order, the timestamp order
//should be checked by comparing it to the previous records timestamp value
if( timestamp <mLastPacketTimestamp ) {
    //Throw error here
} else {
mLastPacketTimestamp = timestamp;
}
```

Step 5: Processing data

The final step is to extract the data from the record. The data is stored sequentially as 32 bit integers and does not require any formatting. Each value in the Data block is stored according to the A/D channel where the data originated (e.g. Data[0] contains a 32 bit sample from A/D Channel 0, Data[23] from A/D Channel 23, etc.) Data in a Raw Data file is stored independently of the acquisition entity (AE) setup. So if two AE share the same A/D channel, only a single value will be stored for that A/D channel in the raw data file.