

**CARMEN**

ENHANCING COLLABORATION  
IN NEUROSCIENCE

# **NDF data format and API implementation**

Dr. Bojian Liang  
Department of Computer Science  
University of York  
25 March 2010



## What is NDF?

The **Neurophysiology Data translation Format (NDF)** is a data format developed within the CARMEN project. NDF provides a standard for sharing data, specifically for data transfer between the various analysis software applications / service developed within the CARMEN project. NDF can be also a useful data format on a researcher's desktop.



# CARMEN

ENHANCING COLLABORATION  
IN NEUROSCIENCE

## What we have achieved?

- A low level C library for NDF I/O. Available currently for Windows and Linux platforms. Support for other platforms to come soon.
- A data converter translates other data formats to NDF. Currently uses Neuroshare as input module. Will extend to other input modules.
- A high level MATLAB toolbox for NDF I/O under MATLAB environment. The toolbox provides convenient methods for using NDF either as a researcher desktop tool or an I/O module for CARMEN web services.



## Term conventions

- ❖ **Primary data**

Data generated from the data acquisition system.  
Sometimes called raw data.

- ❖ **Secondary data**

Data generated by data processing software.

- ❖ **NDF internal data types**

Data that can be read/written with the NDF API.

- ❖ **NDF external data types**

Data that cannot be read/written by the NDF API but  
“wrapped” in the NDF XML configuration file.

## Numerical primary data types

There are three kinds of numerical primary data types used for data acquisition/storage as raw data.

- 1) Continuous time-series data.
- 2) Segmented time-series data.
- 3) Neural event data.

These are the data types supported by most of the currently used neural data formats (the fourth data type is the experimental event data and is not necessarily numerical).

## Continuous time-series Data:

1. One dimensional data series representing the variable values in time order ( $v_1, v_2, \dots, v_n$ ).
2. Usually integers as output by an ADC system.
3. The real values are calculated by

$$V = V_0 + \Delta V_i$$

where.  $V$  is the real signal value.

$V_0$  is the zero offset

$\Delta$  is the quantisation step

$V_i$  is the integer ADC value.

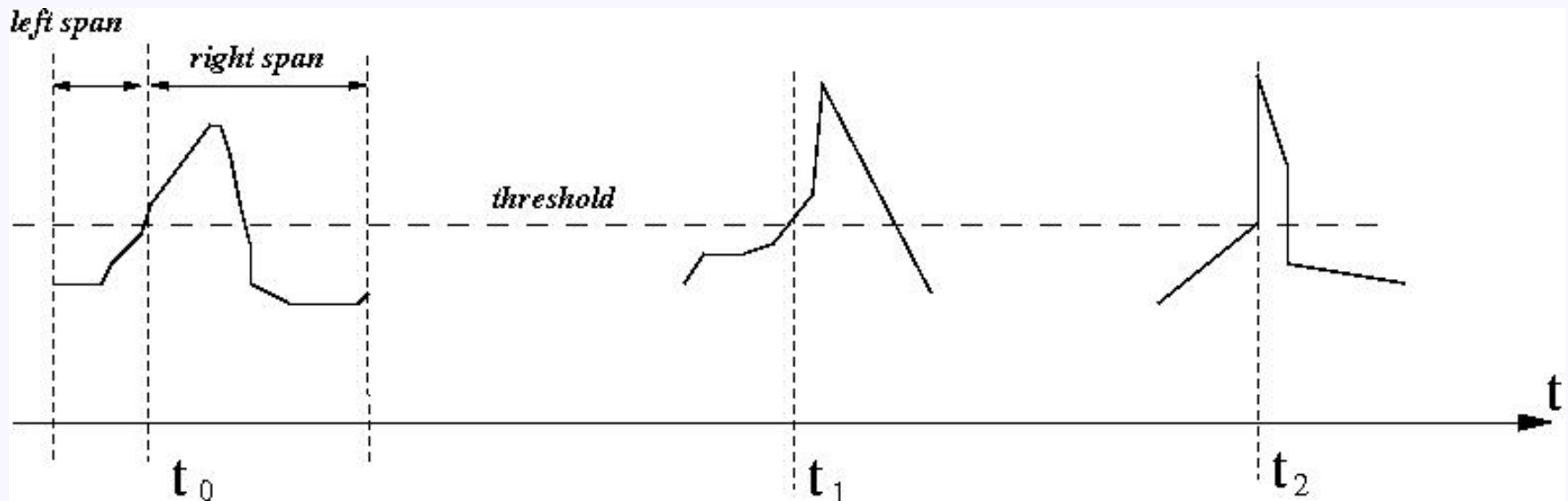
# Segmented time-series data

## Segmented time-series data

1. A collection of time-series data representing variable values from multiple equal length time intervals.  
 $(v_{11}, v_{12}, \dots, v_{1m}), (v_{21}, v_{22}, \dots, v_{2m}), \dots,$   
 $(v_{n1}, v_{n2}, \dots, v_{nm}).$
2. Data representation is similar to continuous time-series data: in addition the time offset and the length of the data segment are also provided.
3. In most cases, the offsets of the start point and end point of the data interval associated with the reference points are also provided as “left span” and “right span”.
4. For some spike detection data, there may also be a channel of data representing the spike sorted ID (spike train number).

## Representation of segmented time-series data.

In NDF, the offsets ( $t_1, t_2 \dots t_n$ ) are stored in the first cell of a cell array; the data values are stored in the second cell (as an array of dimension (segment width  $m$ ) \* (number of segments  $n$ )). The spike (sorted) IDs are optionally stored in the third cell. The cell array has length 3.



## Neural event data

### Neural event data

1. One dimensional data series representing the time at which an event (such as a spike) occurred.
2. Data values of neural event data are time offsets from the start of data ( $T_1, T_2, \dots, T_n$ ).
3. For a given a time resolution value, neural event data can be represented as an integer (or any numerical data type). The real time values are the product of the time resolution and the data value in the series:

$$t_i = r_t T_i$$



## NDF data file structure (1)

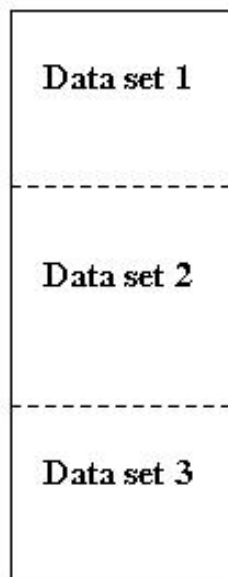
- ❖ Each NDF data set contains one header file in XML format and multiple data files.
- ❖ One data channel can be stored into multiple data files. This allows NDF to handle file of up to  $2^{64}$  bytes in either a 32-bit or 64-bit operating system. The NDF API automatically splits the data into chunks and manages multiple data file chains. All this is transparent to user.
- ❖ There are three modes for saving data into file: single data chunk mode, split data mode and redirected data chain mode.



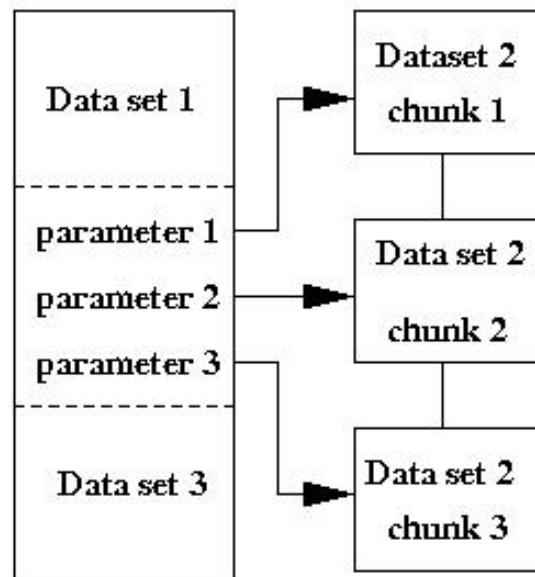
# CARMEN

ENHANCING COLLABORATION  
IN NEUROSCIENCE

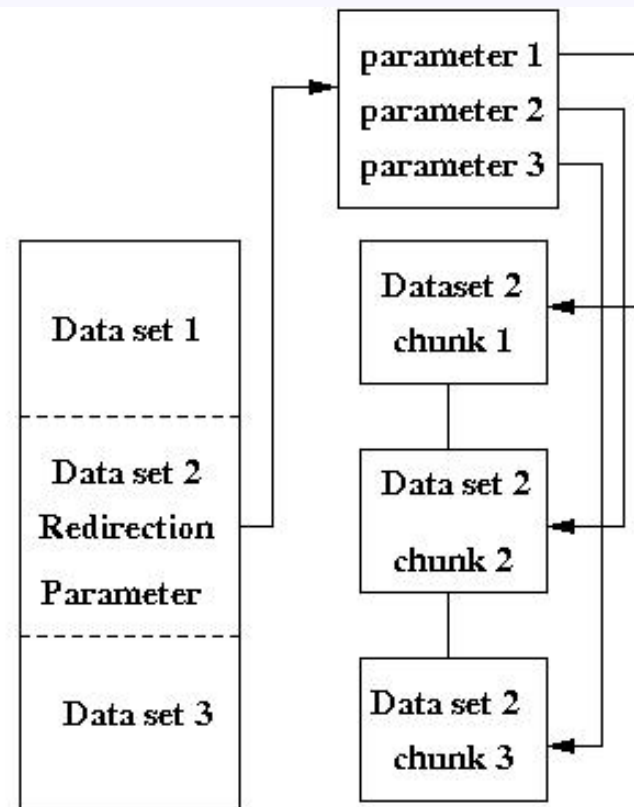
## NDF data file structure (2)



Single Chunk Mode



Split Data Mode



Redirected Data Chain mode



## What NDF provides? (1)

- ❖ Minimizes the network traffic load for extracting Meta-data from a NDF data set from remote site.
- ❖ Easy to display the meta-data on the browser and provides additional information about the data.
- ❖ Allows download of part of data rather than the whole data set for processing.
- ❖ Interactive client code can be easily written based on information extracted from the NDF header file. This is essential to provide parameters to remote data processing services or download data on demand (for example, a service may need information to check if the input data is valid).

**Solution:** Use an XML header file to store meta-data. Separate the data set into different data files.



## What NDF provides? (2)

- ❖ Normally, neural data formats only provide support for primary numerical data types plus experimental event data such as descriptions or time markers.
- ❖ CARMEN system needs to handle outputs from data processing services. Hence NDF should provide support for secondary data that may not be represented by formats used for primary data.

**Solution:** Generic Matrix (a high dimensional array) as internal data, and user defined data as external data to provide additional support.

Slide 14

## What NDF provides? (2)

### --- continue

Using the NDF semi-defined data type GenericMatrix or UserDefinedData to declare a new application dependent data type for a service:

1. A service can declare its output as “NDF + GenericMatrix + Application-ID” to publish a new application dependent data type. The application-ID is one of the meta-data defined on the NDF header.
2. A new service takes a particular NDF application dependent data type as its input by declaring the input as “NDF + GenericMatrix + Application-ID”. This will provide information for a data type pre-verification module to verify the connection between two services in a workflow train.
3. UserDefinedData works in the same way.

A service should declare its input and output data types on registering to the CARMEN system. How the application ID is defined is a system design issue and is out of the scope of the NDF design.



## What NDF provides? (3)

- ❖ For a data processing chain a history or “foot-prints” of each previous process should be included in the output data. This information may be useful/required for later processing or reference. In particular, other researchers can easily repeat the work by reference to the data processing history records.

**Solution:** Add additional data type to record the data processing parameters and relevant information.



## What NDF provides? (4)

- ❖ The data format for numeric data should be designed so that it minimizes the impact for most of current tools used.
- ❖ To reduce the network traffic load, choose an efficient data type for the application. For example, Neuroshare data format can only use double precision type, which, in most cases, increases the traffic load by a factor of 4.
- ❖ API for data accessing should be easy to implement.

**Solution:** Using MAT file format to store numeric data.



## What NDF provides? (5)

- ❖ Experimental event data can be easily added/edited after the data is generated.
- ❖ Additional information such as pictures, sound files or scientific papers can be easily attached.
- ❖ Annotation for the data set can be done easily.

**Solution:** In addition to the conventional experimental event data, a separate XML file can be used to store the experimental event data, annotation and additional third party data objects. Any text editor tool can be used to add information to the XML file.



## What NDF provides? (6)

- ❖ Support image data or image sequence data.
- ❖ NDF should use industrial standards to store the image data rather than a new image data format.
- ❖ It is preferred that these kinds of data can be directly accessed using the NDF APIs, i.e. designed as an internal data type.

**Solution:** Design NDF as a container to wrap the image data in separate data files. There are open source codes available for image data accessing and will be used as part of the NDF API.



## What NDF provides? (7)

The other features that NDF can provide include:

- ❖ A (**Universally Unique Identifier**) UUID field in the NDF configuration file provides identification for the data set. The API automatically generates a UUID for each data set unless told not to do so – this will be ignored if the original data does not include a valid UUID string.
- ❖ Zipped stream is fully supported for MAT file I/O in all valid array types defined in the NDF specification.
- ❖ Partially data read/write from/to a MAT file is supported for all valid array types defined in the NDF specification.



## What NDF provides? (8)

Partial data I/O is crucial in the CARMEN system.

- ❖ Data size may be too large to load into memory at one run.
- ❖ Whole Data may be loadable into memory but this may use too much memory. Consequently, the server will struggle, swapping data between physical memory and virtual memory. We will get a much, much longer processing time than using a progressive data processing procedure using part of the data at a time.
- ❖ Without a partial data read, implementation of services which download part of the data becomes impossible.



**CARMEN**

ENHANCING COLLABORATION  
IN NEUROSCIENCE

# Programming Using the NDF API

The ***NdtfDataInfo*** data structure

- ❖ Is the top level data info structure.
- ❖ It is a C style tree structure corresponding to the XML DOM tree, containing all meta-data of the NDF data set.
- ❖ It consists of three sub-elements accommodating general info, data set info and history data.
- ❖ It also provides information about the NDF specification version number and the data ID.

## The *NdtfDataInfo* data structure (continued)

- ❖ The structure and all its sub-elements are automatically filled when calling the API function “`ndtfReadXMLHeader()`”;
- ❖ The User should fill this structure before creating a NDF configuration XML file by calling the API function “`ndtfWriteXMLHeader()`”.

The ***NdtfGeneralInfo*** data structure:

- ❖ It is a sub-element of the top level ***NdtfDataInfo*** data structure.
- ❖ It is a single level C style structure which contains general information about the data set, such as description of the data, investigator name, etc.
- ❖ Part of the CARMEN meta-data may be extracted automatically from this info structure.

The ***NdtfDataSetInfoList*** data structure.

- ❖ One of the sub-elements of the top level structure. It is a C style data structure used to store information about the NDF data files.
- ❖ It consists of seven elements corresponding to the seven data types defined within the NDF.
- ❖ Each element accommodates information for that type of data and the number of info sets is defined by the sub-element “*nItemCnt*”.
- ❖ Data info are stored in sub-element “*pInfoList*” for each type of data.

# Programming Using the NDF API (5)

The *NdtfHistoryInfoList* data structure.

- ❖ One of the sub-elements of the top level *NdtfDataInfo* data structure
- ❖ Contains information about the processing history, the processing parameters of all the previous tools/services applied to the data.
- ❖ It can also be used to store descriptions of previous processing.



# Programming Using the NDF API (6)

## Memory management

- ❖ All NDF data structures need to be initialized before use and released after use.
- ❖ All NDF data structures are initialized and released in the same manner – call the relevant APIs.

***ndtfxxxxxxInit(\*)***

***ndtfxxxxxxRelease(\*)***

- ❖ If a NDF data structure is allocated dynamically (heap based), it needs to be “freed” explicitly after the contents are released by the NDF API function.

# Programming Using the NDF API (7)

The NDF basic data types.

- ❖ The NDF API defines a set of basic data types in the form of “ndtf\_xxxx\_t” for NDF numerical data storage.
- ❖ It is, in particular, crucial to cast a pointer to an array returned from the MAT accessing API function. This can avoid conflict or memory overflow when compiling the code in different platforms.
- ❖ There are 8 NDF basic data types for integers: signed/unsigned, from 8-bit to 64-bit, plus 2 additional types for single and double length floating point data.

# Programming Using the NDF API (8)

## The NDF Generic Matrix data type

- ❖ It is an application specified data type used to store data that can not be represented using the three data formats for primary data.
- ❖ A application identification number is used to identify the data. This ID number is used by the other services to see if the data is a valid input.
- ❖ It is a semi-defined data type. Applications that create the data ID can define additional tags within the structure. The tags can be XML.

# Programming Using the NDF API (9)

## The NDF User defined data type

- ❖ It is a user specified data type used to store data that can not be stored by the NDF internal data types.
- ❖ A identification number is used to identify the data. This ID number is used by the other service to see if the data is a valid input.
- ❖ It is semi-defined data type. Applications that create the data ID can define additional tags within the structure. The tags can be XML tag or any other style tags.

# Programming Using the NDF API (10)

The NDF User defined data type (continued)

- ❖ The NDF API cannot access the raw data file of a user defined data type – i.e. it is an external data type. Data accessing can only be done by an application that identifies a valid data ID from the data.
- ❖ Information about user defined data type will not be possible to extract from raw data in the NDF header automatically.
- ❖ When a user defined data becomes popular, it may be included into the internal data type by a later NDF version.

# Programming Using the NDF API (11)

## The NDF experimental event data type

- ❖ A internal data type: this can be a conventional time-value pair data list or an XML file for annotation data.
- ❖ Annotation file contains time markers and time interval markers as experimental event data time stamp.
- Each time marker attached to a description string and can be used to annotate the data.
- It can attach one or more third party objects to the time marker. Examples of the third part objects include a photo of the experiment, a sound file, a video or a research paper.

# Programming Using the NDF API (12)

## The compact NDF header element

- ❖ Normally, one XML element in the header represents one data channel. However, to reduce the file size, multiple data channels with the same parameters can be integrated into one element – the compact element.
- ❖ For low level programming, one needs to extract the data info for a single channel from the compact element.
- ❖ Level-2 API function, treats all channels of the same type as whole and accesses the channel by index, e.g. extract the i-th data info of type X.



**CARMEN**

ENHANCING COLLABORATION  
IN NEUROSCIENCE

## The NDF MATLAB toolbox



## The NDF MATLAB toolbox

The NDF MATLAB toolbox provides high level support for NDF data I/O. These include:

- ❖ Main data I/O classes *ndtread()* and *ndfwrite()* provide required NDF read/write support.
- ❖ NDF Datainfo classes acts as carrier to take meta-data from *ndfread()* object or insert meta-data into *ndfwrite()* data object.
- ❖ NDF data classes acts like Info classes but are used for numeric data.
- ❖ All NDF classes are designed as handle classes.



## The NDF MATLAB toolbox

All NDF numeric data used in MatLab environment are MatLab native numeric classes. The following conventions are applied:

- ❖ All one-dimension data are in  $n$ -by-1 arrays.
- ❖ For segment data, segments are represented as column vectors, i.e. a  $m$ -by- $n$  array represents  $n$  segments each with length  $m$ .
- ❖ Data read from NDF by the NDF-toolkit maintain their original data type in order to minimize memory usage.



# The NDF MATLAB toolbox

## In summary:

1. Time-series data is in a  $n$ -by-1 numeric matrix. Any numeric data type is supported.
2. Neural event data (spike time) is in a  $n$ -by-1 numeric matrix. Any numeric data type is supported.
3. Segment data is in a 2-by-1 or 3-by-1 cell array. The first cell ( $n$ -by-1 numeric matrix) represents time offset. The second cell ( $m$ -by- $n$  numeric matrix) represents  $n$  segments each with length  $m$ . The optional third cell ( $n$ -by-1 uint8 type matrix) represents the sorted ID.
4. Binary experimental event data is in a 2-by-1 cell array: the two cells are both  $n$ -by-1 numeric matrices. The first one is the time offset of the events. The second one is the event values.

---

Why do we use the *handle class*.

- ❖ All MATLAB native data classes are value classes.
  - A new object will be created if a variable is assigned to an old object and we start to use it.
  - When a variable is used as an argument in calling a function, a new object is created once it is used. Embedded calling will create multiple objects for a single variable.
- ❖ Using value class can cause serious memory problems in a service. A handle class is similar to a reference on C++ and won't cause unnecessary memory overuse.

# The NDF MATLAB toolbox

```
function foo1(A)
    disp(A)
    foo2(A)
end
function foo2(A)
    disp(A)
end
...
A = ones(100000, 1)
B = A;
disp(B)
foo1(B)
```

```
function foo1(A)
    disp(A.data);
    foo2(A)
end
function foo2(A)
    disp(A.data);
end
...
A.data = ones(100000, 1);
B = A;
disp(B.data);
foo1(B)
```

1. Create a data input object from an NDF data set:

$$A = ndfread('ndf\_filename')$$

This will create an NDF data input object '*A*'. It is a read only object. Information about the specific data file will be read into memory. General data info and the number of channels in each stream type are displayed by default. The object manages all data information extraction and data read operations.

2. Extraction of data information from the input object:

*$B = A.gettimeseriesdatainfo(channel\_num);$*

This will extract the data information about a given channel into object *B*. For other stream types, change string *timeseries* to the name of the stream type. Use *B.dispall* view to full data information.

Where necessary, use *B.DataInfo.fieldname = xx* to assign a value to particular parameter, where *fieldname* is the name displayed when *B.dispall* is called.

## Using the NDF MATLAB toolbox (3)

3. Reading data from NDF data file:

```
D = A.gettimeseriesdata(channel_num);
```

Read time-series data from a specified channel to data object D (full channel read).

```
D = A.gettimeseriesdata(channel_num, idx);
```

Read time-series data of specified channel data from *idx* to end into data object D.

```
D = A.gettimeseriesdata(channel_num, idx1, idx2);
```

Read time-series data of specified channel data from *idx1* to *idx2* into data object D.

## Using the NDF MATLAB toolbox (4)

4. Create an NDF data output object:

```
O = ndfwrite('output-filepath');
```

will create an NDF data output object O. The NDF output object acts like a status machine to control all data integration, writing and coupling operation.

```
O = ndfwrite('output-filepath', 'template_ndf');
```

works the same as the above except that it also loads a general info header from the template file. A general Info header provides information about the experiment such as Investigator and specimen ID.

5. Create an NDF data object to write to file:

```
T = ndftimeseriesdata();
```

will create an NDF time-series data object *T*. The object contains meta-data stored in data member *T.DataInfo*. The numerical data (a N-by-1 matrix) must be assigned to data member *T.Data* before the data can be written to file. The numerical data in MATLAB contains information about the data size, you don't need to change the parameters on *T.DataInfo*.

## Using the NDF MATLAB toolbox (6)

6. Write the data to file:

*`O.writedata(T);`*

will write time series data  $T$  to specified MAT file. The NDF output object detects the type of  $T$  and selects appropriate methods to save the data. On return,  $T$  is assigned to the correct meta-data about the output data set. NDF output object  $O$  also registers the data object  $T$  and assigns an ID to it in order to couple  $T$  to an NDF DataInfo object later. You can clean up  $T.Data$  using  $T.Data = []$ , but don't delete object  $T$  at this stage.

## Using the NDF MATLAB toolbox (7)

7. Couple an NDF DataInfo object to an NDF Data object that has registered to the output object O.

Firstly, create an NDF DataInfo object:

```
I = ndftimeseriesdatainfo();
```

then call:

```
O.updatedatainfo( T, I);
```

The NDF output object O will assign the same ID from the Data object T to DataInfo object I. This procedure will also copy parameters from the Data object T to the DataInfo object I.

8. Add the NDF DataInfo object to the output object. Before doing this, some parameters may need to be set. Use *l.dispall* to view the available fieldnames and assign a value to it if necessary. For example:

*l.DataInfo.samplingRate = 50000*

then call:

*O.adddatainfo(l).*

The NDF output object will add *l* to the DataInfo list and make a record about *T* and *l* to prevent further appending of the same data for a second time.

9. We have now written one channel of time-series data together with the meta-data to file. More data can be added in the same way. Then we need to set up the history of the processing:

```
O.history = ndfhistory( "", 'cmdline', 'comment');
```

and append the previous history data to *O*:

```
O.history + x.history
```

(this assumes that the previous history data was stored in NDF input object *x*. )

## 10. Finalizing the data output.

*O.writendfheader();*

will write out all datainfo stored in the object to the specified NDF header file. After the writing, the object is locked to prevent further writing to the same NDF file. If it is necessary, call

*O.unlockoverwritten()*

to unlock it and obtain is again in order to overwrite the file.

11. Advanced NDF data writing procedure.

The NDF output object supports two writing modes: one-run and multiple-run. To initialize the multiple-run mode, set the total number of items to be saved before call method *writedata()*

*T.setnitemtosave(nnnn).*

then call *O.writedata(T)* multiple times until the total item sum reaches the set number 'nnnn', where T contains number of items less than 'nnnn'.

## Using the NDF MATLAB toolbox (11) –continue

For data streaming, the total number of items is not known in advance. In this case, set the total item number to -1. However, explicitly finalizing must be applied to finish the saving of the data;

*T.setnitemtosave(nnnn)*

where *nnnn* is the number of data items saved up till now plus items in the last chunk of data. Then call

*O.writedata(T)*

to save the last chunk and finish the writing.