Shared Near Infrared Spectroscopy Format (SNIRF) Specification

- **Document Version**: v1.1
- **License**: This document is in the public domain.

Table of Content

  - aux
  - aux.name
  - aux.dataTimeSeries
  - aux.dataUnit
  - aux.time
  - aux.timeOffset
- Appendix
- Acknowledgement

Introduction

The file format specification uses the extension `.snirf`. These are HDF5 format files, renamed with the `.snirf` extension. For a program to be "SNIRF-compliant", it must be able to read and write the SNIRF file.

The development of the SNIRF specification is conducted in an open manner using the GitHub platform. To contribute or provide feedback visit https://github.com/fNIRS/snirf.

Data format

The HDF5 specifications are defined by the HDF5 group and found at https://www.hdfgroup.org. It is expected that HDF5 future versions will remain backwards compatibility in the foreseeable future.

The HDF5 format defines "groups" (`H5G` class) and "datasets" (`H5D` class) that are the two primary data organization and storage classes used in the SNIRF specification.

The structure of each data file has a minimum of required elements noted below.

For each element in the data structure, one of the 4 types is assigned, including

- `group`: a structure containing sub-fields (defined in the `H5G` object class). Arrays of groups, also known as the indexed-groups, are denoted with numbers at the end (e.g. `/nirs/data1`, `/nirs/data2`) starting with index 1. Array indices should be contiguous with no skipped values (an empty group with no sub-member is permitted).
- `string`: a variable-length, null-terminated sequence of characters, i.e. `H5T_C_S1` with size set to `H5T_VARIABLE`. At this time HDF5 does not have a UTF16 native type, so `H5T_NATIVE_B16` will need to be converted to/from unicode-16 within the read/write code).

  Strings MUST be stored in null-terminated 'variable-length' format to be considered valid. Fixed-length strings and variable-length strings are loaded differently by HDF5 interface implementations.* - `integer`: the native integer types `H5T_NATIVE_INT` `H5T` datatype (alias of `H5T_STD_I32BE` or `H5T_STD_I32LE`). Use of 64-bit `long` string types such as `H5T_STD_I64LE` is *not recommended*, although most HDF5 interface implementations will not have issues converting between the two implicitly. - `numeric`: one of the native double or floating-point types; `H5T_NATIVE_DOUBLE` or `H5T_NATIVE_FLOAT` in `H5T` (alias of `H5T_IEEE_F64BE`,`H5T_IEEE_F64LE`, i.e. "double", or `H5T_IEEE_F32BE`, `H5T_IEEE_F32LE`, i.e. "float")

Datasets which are not arrays must be saved in scalar dataspaces. It is NOT VALID to save Datasets which are not specified as arrays in simple dataspaces with 1 dimension and with size 1. HDF5 interface implementations distinguish between these two formats and exhibit different behavior depending on the format of the file.

Valid arrays MUST:

- Contain elements of a correct type as described above.
- Occupy a simple dataspace.
- Have exactly the number of dimensions specified. A SNIRF field specified by this document as a `numeric 1-D array` must occupy a dataspace with `rank` of 1.

  For code samples in various programming languages which demonstrate the writing of SNIRF-specified formats, see the Appendix.

SNIRF file specification

The SNIRF data format must have the initial `H5G` group type `/nirs` at the initial file location.

All indices (source, detector, wavelength, datatype etc) start at 1.

All SNIRF data elements are associated with a unique HDF5 location path in the form of `/root/parent/.../name`. All paths must use `/nirs` or `/nirs#` (indexed group array).
Note that the root `/nirs` can be either indexed or a non-indexed single entry.

If a data element is an HDF5 group and contains multiple sub-groups, it is referred to as an **indexed group**. Each element of the sub-group is uniquely identified by appending a string-formatted index (starting from 1, with no preceding zeros) in the name, for example, `/.../name1` denotes the first sub-group of data element `name`, and `/.../name2` denotes the 2nd element, and so on.

In the below sections, we use the notations `"(i)"` `"(j)"` or `"(k)"` inside the HDF5 location paths to denote the indices of sub-elements when multiplicity presents.

SNIRF data format summary

Note that this table serves as machine-readable schema for the SNIRF format. Its format may not be altered.

SNIRF-formatted NIRS data structure	Meaning of the data	Type
`/formatVersion`	* SNIRF format version	`"s"` *
`/nirs{i}`	* Root-group for 1 or more NIRS datasets	`{i}` *
`metaDataTags`	* Root-group for metadata headers	`{.}` *
`SubjectID`	* Subject identifier	`"s"` *
`MeasurementDate`	* Date of the measurement	`"s"` *
`MeasurementTime`	* Time of the measurement	`"s"` *
`LengthUnit`	* Length unit (case sensitive)	`"s"` *
`TimeUnit`	* Time unit (case sensitive)	`"s"` *
`FrequencyUnit`	* Frequency unit (case sensitive)	`"s"` *
…	* Additional user-defined metadata entries	
`data{i}`	* Root-group for 1 or more data blocks	`{i}` *
`dataTimeSeries`	* Time-varying signals from all channels	`[[<f>,...]]`*
`time`	* Time (in `TimeUnit` defined in metaDataTag)	`[<f>,...]` *
`measurementList{i}`	* Per-channel source-detector information	`{i}` *
`sourceIndex`	* Source index for a given channel	`<i>` *
`detectorIndex`	* Detector index for a given channel	`<i>` *
`wavelengthIndex`	* Wavelength index for a given channel	`<i>` *
`wavelengthActual`	* Actual wavelength for a given channel	`<f>`
`wavelengthEmissionActual`	* Actual emission wavelength for a channel	`<f>`
`dataType`	* Data type for a given channel	`<i>` *
`dataUnit`	* SI unit for a given channel	`"s"`
`dataTypeLabel`	* Data type name for a given channel	`"s"`
`dataTypeIndex`	* Data type index for a given channel	`<i>` *
`sourcePower`	* Source power for a given channel	`<f>`
`detectorGain`	* Detector gain for a given channel	`<f>`
`moduleIndex`	* Index of the parent module (if modular)	`<i>`
`sourceModuleIndex`	* Index of the source's parent module	`<i>`
`detectorModuleIndex`	* Index of the detector's parent module	`<i>`
`stim{i}`	* Root-group for stimulus measurements	`{i}`
`name`	* Name of the stimulus data	`"s"` +
`data`	* Data stream of the stimulus channel	`[[<f>,...]]` +
`dataLabels`	* Names of additional columns of stim data	`["s",...]`
`probe`	* Root group for NIRS probe information	`{.}` *
`wavelengths`	* List of wavelengths (in nm)	`[<f>,...]` *
`wavelengthsEmission`	* List of emission wavelengths (in nm)	`[<f>,...]`
`sourcePos2D`	* Source 2-D positions in `LengthUnit`	`[[<f>,...]]`*[1]

SNIRF-formatted NIRS data structure	Meaning of the data	Type
sourcePos3D	* Source 3-D positions in LengthUnit	[[<f>,...]]*1
detectorPos2D	* Detector 2-D positions in LengthUnit	[[<f>,...]]*2
detectorPos3D	* Detector 3-D positions in LengthUnit	[[<f>,...]]*2
frequencies	* Modulation frequency list	[<f>,...]
timeDelays	* Time delays for gated time-domain data	[<f>,...]
timeDelayWidths	* Time delay width for gated time-domain data	[<f>,...]
momentOrders	* Moment orders of the moment TD data	[<f>,...]
correlationTimeDelays	* Time delays for DCS measurements	[<f>,...]
correlationTimeDelayWidths	* Time delay width for DCS measurements	[<f>,...]
sourceLabels	* String arrays specifying source names	[["s",...]]
detectorLabels	* String arrays specifying detector names	["s",...]
landmarkPos2D	* Anatomical landmark 2-D positions	[[<f>,...]]
landmarkPos3D	* Anatomical landmark 3-D positions	[[<f>,...]]
landmarkLabels	* String arrays specifying landmark names	["s",...]
coordinateSystem	* Coordinate system used in probe description	"s"
coordinateSystemDescription	* Description of coordinate system	"s"
useLocalIndex	* If source/detector index is within a module	<i>
aux{i}	* Root-group for auxiliary measurements	{i}
name	* Name of the auxiliary channel	"s" +
dataTimeSeries	* Data acquired from the auxiliary channel	[[<f>,...]] +
dataUnit	* SI unit of the auxiliary channel	"s"
time	* Time (in TimeUnit) for auxiliary data	[<f>,...] +
timeOffset	* Time offset of auxiliary channel data	[<f>,...]

In the above schema table, the used notations are explained below: * {.} represents a simple HDF5 group * {i} represents an HDF5 group with one or multiple sub-groups (i.e. an indexed-group) * <i> represents an integer value * <f> represents a numeric value * "s" represents a string of arbitrary length * [...] represents a 1-D vector (dataset), can be empty * [[...]] represents a 2-D array (dataset), can be empty * ... (optional) additional elements similar to the previous element * * in the last column indicates a required subfield * * in the last column indicates that at least one of the subfields in the subgroup identified by n is required * + in the last column indicates a required subfield if the optional parent object is included

SNIRF data container definitions

/formatVersion

- **Presence**: required
- **Type**: string
- **Location**: /formatVersion

This is a string that specifies the version of the file format. This document describes format version "1.0"

/nirs(i)

- **Presence**: required
- **Type**: indexed group
- **Location**: /nirs(i)

This group stores one set of NIRS data. This can be extended by adding the count number (e.g. /nirs1, /nirs2,…) to the group name. This is intended to allow the storage of 1 or more complete NIRS datasets inside a single SNIRF document. For example, a two-subject hyperscanning can be stored using the notation * /nirs1 = first subject's data * /nirs2 = second subject's data The use of a non-indexed (e.g. /nirs) entry is allowed when only one entry is present and is assumed to be entry 1.

/nirs(i)/metaDataTags

- **Presence**: required

- **Type**: group
- **Location**: `/nirs(i)/metaDataTags`

The `metaDataTags` group contains the metadata associated with the measurements. Each metadata record is represented as a dataset under this group - with the name of the record, i.e. the key, as the dataset's name, and the value of the record as the actual data stored in the dataset. Each metadata record can potentially have different data types. Sub-groups should not be used to organize metadata records: a member of the `metaDataTags` Group must be a Dataset.

The below five metadata records are minimally required in a SNIRF file

/nirs(i)/metaDataTags/SubjectID

- **Presence**: required as part of `metaDataTags`
- **Type**: string
- **Location**: `/nirs(i)/metaDataTags/SubjectID`

This record stores the string-valued ID of the study subject or experiment.

/nirs(i)/metaDataTags/MeasurementDate

- **Presence**: required as part of `metaDataTags`
- **Type**: string
- **Location**: `/nirs(i)/metaDataTags/MeasurementDate`

This record stores the date of the measurement as a string. The format of the date string must either be `"unknown"`, or follow the ISO 8601 date string format `YYYY-MM-DD`, where - `YYYY` is the 4-digit year - `MM` is the 2-digit month (padding zero if a single digit) - `DD` is the 2-digit date (padding zero if a single digit)

/nirs(i)/metaDataTags/MeasurementTime

- **Presence**: required as part of `metaDataTags`
- **Type**: string
- **Location**: `/nirs(i)/metaDataTags/MeasurementTime`

This record stores the time of the measurement as a string. The format of the time string must either be `"unknown"` or follow the ISO 8601 time string format `hh:mm:ss.sTZD`, where - `hh` is the 2-digit hour - `mm` is the 2-digit minute - `ss` is the 2-digit second - `.s` is 1 or more digit representing a decimal fraction of a second (optional) - `TZD` is the time zone designator (`Z` or `+hh:mm` or `-hh:mm`)

/nirs(i)/metaDataTags/LengthUnit

- **Presence**: required as part of `metaDataTags`
- **Type**: string
- **Location**: `/nirs(i)/metaDataTags/LengthUnit`

This record stores the **case-sensitive** SI length unit used in this measurement. Sample length units include “mm”, “cm”, and “m”. A value of “um” is the same as “ m”, i.e. micrometer.

/nirs(i)/metaDataTags/TimeUnit

- **Presence**: required as part of `metaDataTags`
- **Type**: string
- **Location**: `/nirs(i)/metaDataTags/TimeUnit`

This record stores the **case-sensitive** SI time unit used in this measurement. Sample time units include “s”, and “ms”. A value of “us” is the same as “ s”, i.e. microsecond.

/nirs(i)/metaDataTags/FrequencyUnit

- **Presence**: required as part of `metaDataTags`
- **Type**: string
- **Location**: `/nirs(i)/metaDataTags/FrequencyUnit`

This record stores the **case-sensitive** SI frequency unit used in this measurement. Sample frequency units "Hz", "MHz" and "GHz". Please note that "mHz" is milli-Hz while "MHz" denotes "mega-Hz" according to SI unit system.

We do not limit the total number of metadata records in the `metaDataTags`. Users can add additional customized metadata records; no duplicated metadata record names are allowed.

Additional metadata record samples can be found in the below table.

Metadata Key Name	Metadata value
ManufacturerName	"Company Name"
Model	"Model Name"
SubjectName	"LastName, FirstName"
DateOfBirth	"YYYY-MM-DD"
AcquisitionStartTime	"1569465620"
StudyID	"Infant Brain Development"
StudyDescription	"In this study, we measure …."
AccessionNumber	"############################"
InstanceNumber	2
CalibrationFileName	"phantomcal_121015.snirf"
UnixTime	"1569465667"

The metadata records `"StudyID"` and `"AccessionNumber"` are unique strings that can be used to link the current dataset to a particular study and a particular procedure, respectively. The `"StudyID"` tag is similar to the DICOM tag "Study ID" (0020,0010) and `"AccessionNumber"` is similar to the DICOM tag "Accession Number"(0008,0050), as defined in the DICOM standard (ISO 12052).

The metadata record `"InstanceNumber"` is defined similarly to the DICOM tag "Instance Number" (0020,0013), and can be used as the sequence number to group multiple datasets into a larger dataset - for example, concatenating streamed data segments during a long measurement session.

The metadata record `"UnixTime"` defines the Unix Epoch Time, i.e. the total elapse time in seconds since 1970-01-01T00:00:00Z (UTC) minus the leap seconds.

/nirs(i)/data(j)

- **Presence**: required
- **Type**: indexed group
- **Location**: `/nirs(i)/data(j)`

This group stores one block of NIRS data. This can be extended adding the count number (e.g. `data1`, `data2`,…) to the group name. This is intended to allow the storage of 1 or more blocks of NIRS data from within the same `/nirs` entry * `/nirs/data1` = data block 1 * `/nirs/data2` = data block 2

/nirs(i)/data(j)/dataTimeSeries

- **Presence**: required
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/data(j)/dataTimeSeries`

This is the actual raw or processed data variable. This variable has dimensions of `<number of time points> x <number of channels>`. Columns in `dataTimeSeries` are mapped to the measurement list (`measurementList` variable described below).

`dataTimeSeries` can be compressed using the HDF5 filter (using the built-in `deflate` filter or 3rd party filters such as `305-LZO` or `307-bzip2`

Chunked data is allowed to support real-time streaming of data in this array.

/nirs(i)/data(j)/time

- **Presence**: required
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/data(j)/time`

The `time` variable. This provides the acquisition time of the measurement relative to the time origin. This will usually be a straight line with slope equal to the acquisition frequency, but does not need to be equal spacing. For the special case of equal sample spacing an array of length `<2>` is allowed where the first entry is the start time and the second entry is the sample time spacing in `TimeUnit` specified in the `metaDataTags`. The default time unit is in second ("s"). For example, a time spacing of 0.2 (s) indicates a sampling rate of 5 Hz.

- **Option 1** - The size of this variable is `<number of time points>` and corresponds to the sample time of every data point
- **Option 2**- The size of this variable is `<2>` and corresponds to the start time and sample spacing.

Chunked data is allowed to support real-time streaming of data in this array.

/nirs(i)/data(j)/measurementList(k)

- **Presence**: required
- **Type**: indexed group
- **Location**: `/nirs(i)/data(j)/measurementList(k)`

The measurement list. This variable serves to map the data array onto the probe geometry (sources and detectors), data type, and wavelength. This variable is an array structure that has the size `<number of channels>` that describes the corresponding column in the data matrix. For example, the `measurementList3` describes the third column of the data matrix (i.e. `dataTimeSeries(:,3)`).

Each element of the array is a structure which describes the measurement conditions for this data with the following fields:

/nirs(i)/data(j)/measurementList(k)/sourceIndex

- **Presence**: required
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/sourceIndex`

Index of the source.

/nirs(i)/data(j)/measurementList(k)/detectorIndex

- **Presence**: required
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/detectorIndex`

Index of the detector.

/nirs(i)/data(j)/measurementList(k)/wavelengthIndex

- **Presence**: required
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/wavelengthIndex`

Index of the "nominal" wavelength (in `probe.wavelengths`).

/nirs(i)/data(j)/measurementList(k)/wavelengthActual

- **Presence**: optional
- **Type**: numeric
- **Location**: `/nirs(i)/data(j)/measurementList(k)/wavelengthActual`

Actual (measured) wavelength in nm, if available, for the source in a given channel.

/nirs(i)/data(j)/measurementList(k)/wavelengthEmissionActual

- **Presence**: optional
- **Type**: numeric
- **Location**: `/nirs(i)/data(j)/measurementList(k)/wavelengthEmissionActual`

Actual (measured) emission wavelength in nm, if available, for the source in a given channel.

/nirs(i)/data(j)/measurementList(k)/dataType

- **Presence**: required
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/dataType`

Data-type identifier. See Appendix for list possible values.

/nirs(i)/data(j)/measurementList(k)/dataUnit

- **Presence**: optional
- **Type**: string
- **Location**: `/nirs(i)/data(j)/measurementList(k)/dataUnit`

International System of Units (SI units) identifier for the given channel. Encoding should follow the CMIXF-12 standard, avoiding special unicode symbols like U+03BC ( ) or U+00B5 (µ) and using '/' rather than 'per' for units such as `V/us`. The recommended export format is in unscaled units such as V, s, Mole.

/nirs(i)/data(j)/measurementList(k)/dataTypeLabel

- **Presence**: optional
- **Type**: string
- **Location**: `/nirs(i)/data(j)/measurementList(k)/dataTypeLabel`

Data-type label. Only required if dataType is "processed" (`99999`). See Appendix for list of possible values.

/nirs(i)/data(j)/measurementList(k)/dataTypeIndex

- **Presence**: required
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/dataTypeIndex`

Data-type specific parameter indices. The data type index specifies additional data type specific parameters that are further elaborated by other fields in the probe structure, as detailed below. Note that the Time Domain and Diffuse Correlation Spectroscopy data types have two additional parameters and so the data type index must be a vector with 2 elements that index the additional parameters. One use of this parameter is as a stimulus condition index when `measurementList(k).dataType = 99999` (i.e, `processed` and `measurementList(k).dataTypeLabel = 'HRF ...'` .

/nirs(i)/data(j)/measurementList(k)/sourcePower

- **Presence**: optional
- **Type**: numeric
- **Location**: `/nirs(i)/data(j)/measurementList(k)/sourcePower`

The units are not defined, unless the user takes the option of using a `metaDataTag` as described below.

/nirs(i)/data(j)/measurementList(k)/detectorGain

- **Presence**: optional
- **Type**: numeric
- **Location**: `/nirs(i)/data(j)/measurementList(k)/detectorGain`

Detector gain

/nirs(i)/data(j)/measurementList(k)/moduleIndex

- **Presence**: optional
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/moduleIndex`

Index of a repeating module. If `moduleIndex` is provided while `useLocalIndex` is set to `true`, then, both `measurementList(k).sourceIndex` and `measurementList(k).detectorIndex` are assumed to be the local indices of the same module specified by `moduleIndex`. If the source and detector are located on different modules, one must use `sourceModuleIndex` and `detectorModuleIndex` instead to specify separate parent module indices. See below.

/nirs(i)/data(j)/measurementList(k)/sourceModuleIndex

- **Presence**: optional
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/sourceModuleIndex`

Index of the module that contains the source of the channel. This index must be used together with `detectorModuleIndex`, and can not be used when `moduleIndex` presents.

/nirs(i)/data(j)/measurementList(k)/detectorModuleIndex

- **Presence**: optional
- **Type**: integer
- **Location**: `/nirs(i)/data(j)/measurementList(k)/detectorModuleIndex`

Index of the module that contains the detector of the channel. This index must be used together with `sourceModuleIndex`, and can not be used when `moduleIndex` presents.

For example, if `measurementList5` is a structure with `sourceIndex=2`, `detectorIndex=3`, `wavelengthIndex=1`, `dataType=1`, `dataTypeIndex=1` would imply that the data in the 5th column of the `dataTimeSeries` variable was measured with source #2 and detector #3 at wavelength #1. Wavelengths (in nanometers) are described in the `probe.wavelengths` variable (described later). The data type in this case is 1, implying that it was a continuous wave measurement. The complete list of currently supported data types is found in the Appendix. The data type index specifies additional data type specific parameters that are further elaborated by other fields in the `probe` structure, as detailed below. Note that the Time Domain and Diffuse Correlation Spectroscopy data types have two additional parameters and so the data type index must be a vector with 2 elements that index the additional parameters.

`sourcePower` provides the option for information about the source power for that channel to be saved along with the data. The units are not defined, unless the user takes the option of using a `metaDataTag` described below to define, for instance, `sourcePowerUnit`. `detectorGain` provides the option for information about the detector gain for that channel to be saved along with the data.

Note: The source indices generally refer to the optode naming (probe positions) and not necessarily the physical laser numbers on the instrument. The same is true for the detector indices. Each source optode would generally, but not necessarily, have 2 or more wavelengths (hence lasers) plugged into it in order to calculate deoxy- and oxy-hemoglobin concentrations. The data from these two wavelengths will be indexed by the same source, detector, and data type values, but have different wavelength indices. Using the same source index for lasers at the same location but with different wavelengths simplifies the bookkeeping for converting intensity measurements into concentration changes. As described below, optional variables `probe.sourceLabels` and `probe.detectorLabels` are provided for indicating the instrument specific label for sources and detectors.

/nirs(i)/stim(j)

- **Presence**: optional
- **Type**: indexed group
- **Location**: `/nirs(i)/stim(j)`

This is an array describing any stimulus conditions. Each element of the array has the following required fields.

/nirs(i)/stim(j)/name

- **Presence**: required as part of `stim(i)`
- **Type**: string
- **Location**: `/nirs(i)/stim(j)/name`

This is a string describing the jth stimulus condition.

/nirs(i)/stim(j)/data

- **Presence**: required as part of `stim(i)`
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/stim(j)/data`
- **Allowed attribute**: `names`

This is a numeric 2-D array with at least 3 columns, specifying the stimulus time course for the jth condition. Each row corresponds to a specific stimulus trial. The first three columns indicate `[starttime duration value]`.
The starttime, in seconds, is the time relative to the time origin when the stimulus takes on a value; the duration is the time in seconds that the stimulus value continues, and value is the stimulus amplitude. The number of rows is not constrained. (see examples in the appendix).

Additional columns can be used to store user-specified data associated with each stimulus trial. An optional record `/nirs(i)/stim(j)/dataLabels` can be used to annotate the meanings of each data column.

/nirs(i)/stim(j)/dataLabels

- **Presence**: optional
- **Type**: string array
- **Location**: `/nirs(i)/stim(j)/dataLabels(k)`

This is a string array providing annotations for each data column in `/nirs(i)/stim(j)/data`. Each element of the array must be a string; the total length of this array must be the same as the column number of `/nirs(i)/stim(j)/data`, including the first 3 required columns.

/nirs(i)/probe

- **Presence**: required
- **Type**: group
- **Location**: `/nirs(i)/probe`

This is a structured variable that describes the probe (source-detector) geometry. This variable has a number of required fields.

/nirs(i)/probe/wavelengths

- **Presence**: required
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/wavelengths`

This field describes the "nominal" wavelengths used (in `nm` unit). This is indexed by the `wavelengthIndex` of the measurementList variable. For example, `probe.wavelengths` = [690, 780, 830]; implies that the measurements were taken at three wavelengths (690 nm, 780 nm, and 830 nm). The wavelength index of `measurementList(k).wavelengthIndex` variable refers to this field. `measurementList(k).wavelengthIndex` = 2 means the kth measurement was at 780 nm.

Please note that this field stores the "nominal" wavelengths. If the precise (measured) wavelengths differ from the nominal wavelengths, one can store those in the `measurementList.wavelengthActual` field in a per-channel fashion.

The number of wavelengths is not limited (except that at least two are needed to calculate the two forms of hemoglobin). Each source-detector pair would generally have measurements at all wavelengths.

This field must present, but can be empty, for example, in the case that the stored data are processed data (`dataType=99999`, see Appendix).

/nirs(i)/probe/wavelengthsEmission

- **Presence**: optional
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/wavelengthsEmission`

This field is required only for fluorescence data types, and describes the "nominal" emission wavelengths used (in `nm` unit). The indexing of this variable is the same wavelength index in measurementList used for `probe.wavelengths` such that the excitation wavelength is paired with this emission wavelength for a given measurement.

Please note that this field stores the "nominal" emission wavelengths. If the precise (measured) emission wavelengths differ from the nominal ones, one can store those in the `measurementList.wavelengthEmissionActual` field in a per-channel fashion.

/nirs(i)/probe/sourcePos2D

- **Presence**: at least one of `sourcePos2D` or `sourcePos3D` is required
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/probe/sourcePos2D`

This field describes the position (in `LengthUnit` units) of each source optode. The positions are coordinates in a flattened 2D probe layout. This field has size `<number of sources> x 2`. For example, `probe.sourcePos2D(1,:) = [1.4 1]`, and `LengthUnit='cm'` places source number 1 at x=1.4 cm and y=1 cm.

/nirs(i)/probe/sourcePos3D

- **Presence**: at least one of `sourcePos2D` or `sourcePos3D` is required
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/probe/sourcePos3D`

This field describes the position (in `LengthUnit` units) of each source optode in 3D. This field has size `<number of sources> x 3`.

/nirs(i)/probe/detectorPos2D

- **Presence**: at least one of `detectorPos2D` or `detectorPos3D` is required
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/probe/detectorPos2D`

Same as `probe.sourcePos2D`, but describing the detector positions in a flattened 2D probe layout.

/nirs(i)/probe/detectorPos3D

- **Presence**: at least one of `detectorPos2D` or `detectorPos3D` is required
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/probe/detectorPos3D`

This field describes the position (in `LengthUnit` units) of each detector optode in 3D, defined similarly to `sourcePos3D`.

/nirs(i)/probe/frequencies

- **Presence**: optional
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/frequencies`

This field describes the frequencies used (in `FrequencyUnit` units) for frequency domain measurements. This field is only required for frequency domain data types, and is indexed by `measurementList(k).dataTypeIndex`.

/nirs(i)/probe/timeDelays

- **Presence**: optional
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/timeDelays`

This field describes the time delays (in `TimeUnit` units) used for gated time domain measurements. This field is only required for gated time domain data types, and is indexed by `measurementList(k).dataTypeIndex`. The indexing of this field is paired with the indexing of `probe.timeDelayWidths`.

/nirs(i)/probe/timeDelayWidths

- **Presence**: optional
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/timeDelayWidths`

This field describes the time delay widths (in `TimeUnit` units) used for gated time domain measurements. This field is only required for gated time domain data types, and is indexed by `measurementList(k).dataTypeIndex`. The indexing of this field is paired with the indexing of `probe.timeDelays`.

/nirs(i)/probe/momentOrders

- **Presence**: optional
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/momentOrders`

This field describes the moment orders of the temporal point spread function (TPSF) or the distribution of time-of-flight (DTOF) for moment time domain measurements. This field is only required for moment time domain data types, and is indexed by `measurementList(k).dataTypeIndex`.
Note that the numeric value in this array is the exponent in the integral used for calculating the moments. For detailed/specific definitions of moments, see Wabnitz et al, 2020; for general definitions of moments see here.

In brief, given a TPSF or DTOF N(t) (photon counts vs. photon arrival time at the detector):
momentOrder = 0: total counts: `N_total = \intergral N(t)dt`
momentOrder = 1: mean time of flight: `m = <t> = (1/N_total) \integral t N(t) dt`
momentOrder = 2: variance/second central moment: `V = (1/N_total) \integral (t - <t>)^2 N(t) dt`
Please note that all moments (for orders >=1) are expected to be normalized by the total counts (i.e. n=0); Additionally all moments (for orders >= 2) are expected to be centralized.

/nirs(i)/probe/correlationTimeDelays

- **Presence**: optional
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/correlationTimeDelays`

This field describes the time delays (in `TimeUnit` units) used for diffuse correlation spectroscopy measurements. This field is only required for diffuse correlation spectroscopy data types, and is indexed by `measurementList(k).dataTypeIndex`. The indexing of this field is paired with the indexing of `probe.correlationTimeDelayWidths`.

/nirs(i)/probe/correlationTimeDelayWidths

- **Presence**: optional
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/probe/correlationTimeDelayWidth`

This field describes the time delay widths (in `TimeUnit` units) used for diffuse correlation spectroscopy measurements. This field is only required for gated time domain data types, and is indexed by `measurementList(k).dataTypeIndex`. The indexing of this field is paired with the indexing of `probe.correlationTimeDelays`.

/nirs(i)/probe/sourceLabels

- **Presence**: optional
- **Type**: string 2-D array
- **Location**: `/nirs(i)/probe/sourceLabels(j)`

This is a string array providing user friendly or instrument specific labels for each source. Each element of the array must be a unique string among both `probe.sourceLabels` and `probe.detectorLabels`.This can be of size `<number of sources>x 1` or `<number of sources> x <number of wavelengths>`. This is indexed by `measurementList(k).sourceIndex` and `measurementList(k).wavelengthIndex`.

/nirs(i)/probe/detectorLabels

- **Presence**: optional
- **Type**: string array
- **Location**: `/nirs(i)/probe/detectorLabels(j)`

This is a string array providing user friendly or instrument specific labels for each detector. Each element of the array must be a unique string among both `probe.sourceLabels` and `probe.detectorLabels`. This is indexed by `measurementList(k).detectorIndex`.

/nirs(i)/probe/landmarkPos2D

- **Presence**: optional
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/probe/landmarkPos2D`

This is a 2-D array storing the neurological landmark positions projected along the 2-D (flattened) probe plane in order to map optical data from the flattened optode positions to brain anatomy. This array should contain a minimum of 2 columns, representing the x and y coordinates (in `LengthUnit` units) of the 2-D projected landmark positions. If a 3rd column presents, it stores the index to the labels of the given landmark. Label names are stored in the `probe.landmarkLabels` subfield. An label index of 0 refers to an undefined landmark.

/nirs(i)/probe/landmarkPos3D

- **Presence**: optional
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/probe/landmarkPos3D`

This is a 2-D array storing the neurological landmark positions measurement from 3-D digitization and tracking systems to facilitate the registration and mapping of optical data to brain anatomy. This array should contain a minimum of 3 columns, representing the x, y and z coordinates (in `LengthUnit` units) of the digitized landmark positions. If a 4th column presents, it stores the index to the labels of the given landmark. Label names are stored in the `probe.landmarkLabels` subfield. An label index of 0 refers to an undefined landmark.

/nirs(i)/probe/landmarkLabels(j)

- **Presence**: optional
- **Type**: string array
- **Location**: `/nirs(i)/probe/landmarkLabels(j)`

This string array stores the names of the landmarks. The first string denotes the name of the landmarks with an index of 1 in the 4th column of `probe.landmark`, and so on. One can adopt the commonly used 10-20 landmark names, such as "Nasion", "Inion", "Cz" etc, or use user-defined landmark labels. The landmark label can also use the unique source and detector labels defined in `probe.sourceLabels` and `probe.detectorLabels`, respectively, to associate the given landmark to a specific source or detector. All strings are ASCII encoded char arrays.

/nirs(i)/probe/coordinateSystem

- **Presence**: optional
- **Type**: string
- **Location**: `/nirs(i)/probe/coordinateSystem`

Defines the coordinate system for sensor positions. The string must be one of the coordinate systems listed in the BIDS specification (Appendix VII) such as "MNI152NLin2009bAsym", "CapTrak" or "Other". If the value "Other" is specified, then a defition of the coordinate system must be provided in `/nirs(i)/probe/coordinateSystemDescription`. See the FieldTrip toolbox web page for detailed descriptions of different coordinate systems.

/nirs(i)/probe/coordinateSystemDescription

- **Presence**: optional
- **Type**: string
- **Location**: `/nirs(i)/probe/coordinateSystemDescription`

Free-form text description of the coordinate system. May also include a link to a documentation page or paper describing the system in greater detail. This field is required if the `coordinateSystem` field is set to "Other".

/nirs(i)/probe/useLocalIndex

- **Presence**: optional
- **Type**: integer
- **Location**: `/nirs(i)/probe/useLocalIndex`

For modular NIRS systems, setting this flag to a non-zero integer indicates that `measurementList(k).sourceIndex` and `measurementList(k).detectorIndex` are module-specific local-indices. One must also include `measurementList(k).moduleIndex`, or when cross-module channels present, both `measurementList(k).sourceModuleIndex` and `measurementList(k).detectorModul` in the `measurementList` structure in order to restore the global indices of the sources/detectors.

/nirs(i)/aux(j)

- **Presence**: optional
- **Type**: indexed group
- **Location**: `/nirs(i)/aux(j)`

This optional array specifies any recorded auxiliary data. Each element of `aux` has the following required fields:

/nirs(i)/aux(j)/name

- **Presence**: optional; required if `aux` is used
- **Type**: string
- **Location**: `/nirs(i)/aux(j)/name`

This is string describing the jth auxiliary data timecourse. While auxiliary data can be given any title, standard names for commonly used auxiliary channels (i.e. accelerometer data) are specified in the appendix.

/nirs(i)/aux(j)/dataTimeSeries

- **Presence**: optional; required if `aux` is used
- **Type**: numeric 2-D array
- **Location**: `/nirs(i)/aux(j)/dataTimeSeries`

This is the aux data variable. This variable has dimensions of `<number of time points> x <number of channels>`. If multiple channels of related data are generated by a system, they may be encoded in the multiple columns of the time series (i.e. complex numbers). For example, a system containing more than one accelerometer may output this data as a set of `ACCEL_X`/`ACCEL_Y`/`ACCEL_Z` auxiliary time series, where each has the dimension of `<number of time points> x <number of accelerometers>`. Note that it is NOT recommended to encode the various accelerometer dimensions as multiple channels of the same `aux` Group: instead follow the `"ACCEL_X"`, `"ACCEL_Y"`, `"ACCEL_Z"` naming conventions described in the appendix. Chunked data is allowed to support real-time data streaming.

/nirs(i)/aux(j)/dataUnit

- **Presence**: optional
- **Type**: string
- **Location**: `/nirs(i)/aux(j)/dataUnit`

International System of Units (SI units) identifier for the given channel. Encoding should follow the CMIXF-12 standard, avoiding special unicode symbols like U+03BC ( ) or U+00B5 (µ) and using '/' rather than 'per' for units such as `V/us`. The recommended export format is in unscaled units such as V, s, Mole.

/nirs(i)/aux(j)/time

- **Presence**: optional; required if `aux` is used
- **Type**: numeric 1-D array
- **Location**: `/nirs(i)/aux(j)/time`

The time variable. This provides the acquisition time (in `TimeUnit` units) of the aux measurement relative to the time origin. This will usually be a straight line with slope equal to the acquisition frequency, but does not need to be equal spacing. The size of this variable is `<number of time points>` or `<2>` similar to definition of the `/nirs(i)/data(j)/time` field.

Chunked data is allowed to support real-time data streaming

/nirs(i)/aux(j)/timeOffset

- **Presence**: optional
- **Type**: numeric
- **Location**: `/nirs(i)/aux(j)/timeOffset`

This variable specifies the offset of the file time origin relative to absolute (clock) time in `TimeUnit` units.

Appendix

Supported `measurementList(k).dataType` values in `dataTimeSeries`

- 001-100: Raw - Continuous Wave (CW)
- 001 - Amplitude
- 051 - Fluorescence Amplitude
- 101-200: Raw - Frequency Domain (FD)
- 101 - AC Amplitude
- 102 - Phase
- 151 - Fluorescence Amplitude
- 152 - Fluorescence Phase
- 201-300: Raw - Time Domain - Gated (TD Gated)
- 201 - Amplitude
- 251 - Fluorescence Amplitude

- 301-400: Raw - Time domain – Moments (TD Moments)
- 301 - Amplitude
- 351 - Fluorescence Amplitude
- 401-500: Raw - Diffuse Correlation Spectroscopy (DCS):
- 401 - g2
- 410 - BFi
- 99999: Processed

Supported `measurementList(k).dataTypeLabel` values in `dataTimeSeries`

Tag Name	Meanings
"dOD"	Change in optical density
"dMean"	Change in mean time-of-flight
"dVar"	Change in variance (2nd central moment)
"dSkew"	Change in skewness (3rd central moment)
"mua"	Absorption coefficient
"musp"	Scattering coefficient
"HbO"	Oxygenated hemoglobin (oxyhemoglobin) concentration
"HbR"	Deoxygenated hemoglobin (deoxyhemoglobin) concentration
"HbT"	Total hemoglobin concentration
"H2O"	Water content
"Lipid"	Lipid concentration
"BFi"	Blood flow index
"HRF dOD"	Hemodynamic response function for change in optical density
"HRF dMean"	HRF for change in mean time-of-flight
"HRF dVar"	HRF for change in variance (2nd central moment)
"HRF dSkew"	HRF for change in skewness (3rd central moment)
"HRF HbO"	Hemodynamic response function for oxyhemoglobin concentration
"HRF HbR"	Hemodynamic response function for deoxyhemoglobin concentration
"HRF HbT"	Hemodynamic response function for total hemoglobin concentration
"HRF BFi"	Hemodynamic response function for blood flow index

Supported `/nirs(i)/aux(j)/name` values

Tag Name	Meanings
"ACCEL_X"	Accelerometer data, first axis of orientation
"ACCEL_Y"	Accelerometer data, second axis of orientation
"ACCEL_Z"	Accelerometer data, third axis of orientation
"GYRO_X"	Gyrometer data, first axis of orientation
"GYRO_Y"	Gyrometer data, second axis of orientation
"GYRO_Z"	Gyrometer data, third axis of orientation
"MAGN_X"	Magnetometer data, first axis of orientation
"MAGN_Y"	Magnetometer data, second axis of orientation
"MAGN_Z"	Magnetometer data, third axis of orientation

Examples of stimulus waveforms

Assume there are 10 time points, starting at zero, spaced 0.1s apart. If we assume a stimulus to be a 0.2 second off, 0.2 second on repeating block, it would be specified as follows:

```
[0.2 0.2 1.0]
[0.6 0.2 1.0]
```

Code samples

The following code demonstrates how to use the Python `h5py` and `numpy` libraries and the MATLAB `H5ML.hdf5lib2` "low-level" interface to write specified SNIRF datatypes to disk as HDF5 Datasets of the proper format.

String "s"

MATLAB

```
fid = H5F.open(<SNIRF file path>, 'H5F_ACC_RDWR', 'H5P_DEFAULT')
sid = H5S.create('H5S_SCALAR')
tid = H5T.copy('H5T_C_S1');
H5T.set_size(tid, 'H5T_VARIABLE');
did = H5D.create(fid, <dataset location>, tid, sid, 'H5P_DEFAULT')
H5D.write(did, tid, 'H5S_ALL', 'H5S_ALL', 'H5P_DEFAULT', <value of string>)
```

Python

```
file = h5py.File(<SNIRF file path>, 'r+')
varlen_str_dtype = h5py.string_dtype(encoding='ascii', length=None)
file.create_dataset(<dataset location>, dtype=varlen_str_dtype, data=<value of string>)
```

numeric <f>

MATLAB

```
fid = H5F.open(<SNIRF file path>, 'H5F_ACC_RDWR', 'H5P_DEFAULT')
tid = H5T.copy('H5T_NATIVE_DOUBLE')
sid = H5S.create('H5S_SCALAR')
H5D.create(fid, <dataset location>, tid, sid, 'H5P_DEFAULT')
h5write(<SNIRF file path>, <dataset location>, <value of numeric>)
```

Python

```
file = h5py.File(<SNIRF file path>, 'r+')
file.create_dataset(<dataset location>, dtype='f8', data=<value of numeric>)
```

integer <i>

MATLAB

```
fid = H5F.open(<SNIRF file path>, 'H5F_ACC_RDWR', 'H5P_DEFAULT')
tid = H5T.copy('H5T_NATIVE_INT')
sid = H5S.create('H5S_SCALAR')
H5D.create(fid, <dataset location>, tid, sid, 'H5P_DEFAULT')
h5write(<SNIRF file path>, <dataset location>, <value of integer>)
```

Python

```
file = h5py.File(<SNIRF file path>, 'r+')
file.create_dataset(<dataset location>, dtype='i4', data=<value of integer>)
```

string array ["s",...]

MATLAB

```
fid = H5F.open(<SNIRF file path>, 'H5F_ACC_RDWR', 'H5P_DEFAULT')

str_arr = {'Hello', 'World', 'foo', 'bar'}  % values to write, a cell array of strings of any length

sid = H5S.create_simple(1, numel(str_arr), H5ML.get_constant_value('H5S_UNLIMITED'));

tid = H5T.copy('H5T_C_S1');
```

```
H5T.set_size(tid, 'H5T_VARIABLE');

pid = H5P.create('H5P_DATASET_CREATE');
H5P.set_chunk(pid, 2);

did = H5D.create(fid, <dataset location>, tid, sid, pid)

H5D.write(did, tid, 'H5S_ALL', 'H5S_ALL', 'H5P_DEFAULT', str_arr)
```

Python

```
array = numpy.array(<list of strings>).astype('O')  # A list of strings must be converted to a NumPy list wit
file = h5py.File(<SNIRF file path>, 'r+')
varlen_str_dtype = h5py.string_dtype(encoding='ascii', length=None)
file.create_dataset(<dataset location>, dtype=varlen_str_dtype, data=array)
```

numeric array [<f>,...] or [[<f>,...]]

MATLAB > Note: Because MATLAB has no notion of arrays with fewer than 2 dimensions, using `size(data)` as the 3rd argument of `h5create` will erroneously save arrays with 1 dimension as a row or column vector of 2 dimensions. In the 1D case, use `length(data)` as the 3rd argument of `h5create`.

```
data = <numeric array>
h5create(<SNIRF file path>, <dataset location>, length(data) / size(data), 'Datatype', 'double')
h5write(<SNIRF file path>, <dataset location>, data)
```

Python

```
array = numpy.array(<numeric array>).astype(numpy.float64)  # A list or nested list of values should be conve
file = h5py.File(<SNIRF file path>, 'r+')
file.create_dataset(<dataset location>, dtype='f8', data=array)
```

integer array [<i>,...] or [[<i>,...]]

MATLAB > Note: Because MATLAB has no notion of arrays with fewer than 2 dimensions, using `size(data)` as the 3rd argument of `h5create` will erroneously save arrays with 1 dimension as a row or column vector of 2 dimensions. In the 1D case, use `length(data)` as the 3rd argument of `h5create`.

```
data = <integer array>
h5create(<SNIRF file path>, <integer array dataset location>, length(data) / size(data), 'Datatype', 'int32')
h5write(<SNIRF file path>, <integer array dataset location>, data)
```

Python

```
array = numpy.array(<integer array>).astype(int)  # A list or nested list of values should be converted to a
file = h5py.File(<SNIRF file path>, 'r+')
file.create_dataset(<integer array dataset location>, dtype='i4', data=array)
```

Acknowledgement

This document was originally drafted by Blaise Frederic (bbfrederick at mclean.harvard.edu) and David Boas (dboas at bu.edu).

Other significant contributors to this specification include: - Theodore Huppert (huppert1 at pitt.edu) - Jay Dubb (jdubb at bu.edu) - Qianqian Fang (q.fang at neu.edu)

The following individuals representing academic, industrial, software, and hardware interests are also contributing to and supporting the adoption of this specification:

Software

- Ata Akin, Acıbadem University
- Hasan Ayaz, Drexel University
- Joe Culver, University of Washington, neuroDOT
- Hamid Deghani, University of Birmingham, NIRFAST
- Adam Eggebrecht, University of Washington, neuroDOT
- Christophe Grova, McGill University, NIRSTORM
- Felipe Orihuela-Espina, Instituto Nacional de Astrofísica, Óptica y Electrónica, ICNNA
- Luca Pollonini, Houston Methodist, Phoebe
- Sungho Tak, Korea Basic Science Institute, NIRS-SPM
- Alessandro Torricelli, Politecnico di Milano
- Stanislaw Wojtkiewicz, University of Birmingham, NIRFAST
- Robert Luke, Macquarie University, MNE-NIRS
- Stephen Tucker, Boston University
- Michael Lührs, Maastricht University, Brain Innovation B.V., Satori
- Robert Oostenveld, Radboud University, FieldTrip

Hardware

- Hirokazu Asaka, Hitachi
- Rob Cooper, Gower Labs Inc
- Mathieu Coursolle, Rogue Research
- Rueben Hill, Gower Labs Inc
- Jörn Horschig, Artinis Medical Systems B.V.
- Takumi Inakazu, Hitachi
- Lamija Pasalic, NIRx
- Davood Tashayyod, fNIR Devices and Biopac Inc
- Hanseok Yun, OBELAB Inc
- Zahra M. Aghajan, Kernel