Module Documentation

C-LAB 26.05.0

May, 2026

Contents

1 About

This document describes the API of the C-LAB Python module distributed with C-LAB 26.05.0. Get the latest version from https://www.c-lab.tech.

For additional information, support, or feedback, contact us at info@c-lab.tech.

2 Classes

C-LAB module defines 6 custom classes, as listed below. Refer to each section for further information.

clab.Array	Stores data as a contiguous IEEE 754 double floating-point precision (64-bits) array.
clab.Container	Container with all variables (as TimeSeries) read from file.
clab.Mesh	Stores mesh data to be used in surface 3D plots.
clab.Model	Simulation model object.
clab.Result	Container with data results from a simulation model.
clab.TimeSeries	Stores series of data indexed in chronological order.

2.1 clab.Array

`clab.Array` stores data as a contiguous IEEE 754 double floating-point precision (64-bits) array.

Constructor

An instance of `clab.Array` can be created directly using the following functions:

`clab.Array(list)`

`list` [mandatory] Python list with numeric values. Behaviour with non-numeric types is undefined.

`clab.Array(other)`

`other` [mandatory] `clab.Array`. This constructor will shallow-copy `other` to the new object. Therefore, both objects will share the same underlying data values: modifications done in one object will reflect upon the other. Check the `clone` function if a deep-copy is required.

`clab.Array(len)`

`len` [mandatory] Length of the new array. The new `clab.Array` will be zero-initialized.

Member Variables

`clab.Array` has no publicly exposed member variables.

Member Functions

`clab.Array` implements the Number Protocol, supporting standard arithmetic operations: addition, subtraction, multiplication, remainder, division, exponentiation, bitwise AND (&), bitwise XOR (∧), bitwise OR (|), negation (∼), and absolute value. Inplace equivalents are also supported.

`clab.Array` implements the Sequence Protocol, allowing element access via the standard `[]` operator.

Besides standard protocols, the following member functions are available:

`out = clab.Array.clone()`

`out` `clab.Array`.

Return a new `clab.Array` containing a deep-copy of the caller's values.

`out = clab.Array.to_list()`

`out` Python list with the same length as the `clab.Array`.

Export array values as a standard Python list.

`out = clab.Array.avg()`

`out` Float.

Return the arithmetic mean (average) of the array's values.

`out = clab.Array.sum()`

```
        out  Float.
```

Return the sum of all elements in the array.

```
out = clab.Array.count_eq(val)
        out  Integer.
        val  [mandatory] Numeric value.
```

Return the number of elements in the array that are exactly equal to `val`.

```
out = clab.Array.count_gt(val)
        out  Integer.
        val  [mandatory] Numeric value.
```

Return the count of elements that are strictly greater than `val`.

```
out = clab.Array.count_lt(val)
        out  Integer.
        val  [mandatory] Numeric value.
```

Return the count of elements that are strictly less than `val`.

```
out = clab.Array.count_bt(val1, val2)
        out  Integer.
       val1  [mandatory] Numeric lower bound.
       val2  [mandatory] Numeric upper bound.
```

Return the count of elements that are strictly between the provided arguments, i.e., $>$ `val1` and $<$ `val2`.

```
out = clab.Array.logical_and(rhs)
        out  Integer.
```

Return a new array (containing 0s and 1s) resulting from the *logical AND* operation between the current array and `clab.Array` `rhs`.

```
out = clab.Array.logical_or(rhs)
        out  clab.Array.
        rhs  clab.Array.
```

Return a new array (containing 0s and 1s) resulting from the *logical OR* operation between the current array and `clab.Array` `rhs`.

```
out = clab.Array.logical_not()
        out  clab.Array.
```

Return a new array (containing 0s and 1s) representing the *logical NOT* operation on all elements.

```
out = clab.Array.logical_xor(rhs)
```

out `clab.Array`.

rhs `clab.Array`.

Return a new array (containing 0s and 1s) resulting from the *logical XOR* operation between the current array and `clab.Array` `rhs`.

2.2 clab.Container

The `clab.Container` class stores variables loaded from a file or acquired from a real-time source (experimental feature). Each `clab.TimeSeries` is also exposed as an attribute of the container, with its name automatically converted into a valid Python identifier.

Constructor

An instance of `clab.Container` can be created using:

```
clab.Container(path, plugin='', realtime=False, samples=300)
```

path [mandatory] Full path to the input file.
If the file is compressed, C-LAB will first decompress it into the cache directory and then load the extracted file using the appropriate loader.

plugin [optional] Full path to a plugin.
If omitted, C-LAB automatically selects a compatible loader by testing installed plugins and built-in loaders. If provided, the specified plugin is used directly, bypassing compatibility checks.

realtime [optional][beta] Enables acquisition from a real-time data source when set to `True`.
This feature is experimental and its API may change in future releases.

samples [optional][beta] Maximum number of samples stored when acquiring real-time data.
Once this limit is reached, older samples are discarded to accommodate new data. This feature is experimental and subject to change.

Member Variables

`clab.Container.id64` *integer*
Read-only. Unique 64-bit identifier.

`clab.Container.flags` *integer*
Read-only. Reserved for internal use.

Member Functions

`clab.Container` partially implements the Mapping Protocol, allowing access to individual `clab.TimeSeries` objects via the `[]` operator using their source names.

In addition to standard protocol behavior, the following member functions are available:

`out = clab.Container.get_path()`

`out` List.

Returns the list of file paths associated with the container.

`out = clab.Container.get_size()`

`out` Integer.

Returns the number of `TimeSeries` objects in the container.

`clab.Container.axpb(a, b)`

`a` [mandatory] Numeric value.

`b` [mandatory] Numeric value.

Applies the transformation $A \cdot X + B$ to the x-axis of all variables.

If the data is not yet loaded, the operation is deferred until loading completes.

`clab.Container.aypb(a, b)`

`a` [mandatory] Numeric value.

`b` [mandatory] Numeric value.

Applies the transformation $A \cdot Y + B$ to the y-axis of all variables.

If the data is not yet loaded, the operation is deferred until loading completes.

`clab.Container.merge(path)`

`path` [mandatory] File path.

Merges the contents of the specified file into the container.

All `TimeSeries` loaded after this call will use data from this file in addition to previously loaded sources. Already loaded `TimeSeries` remain unchanged.

`clab.Container.replace_xdata(ts)`

`ts` [mandatory] `clab.TimeSeries` used as the source for x-axis data.

Replaces the x-axis data of all contained variables using the y-data from `ts`.

The source data must be *strictly increasing* (monotonically increasing with no repeated or decreasing values).

If data sizes differ, the maximum compatible subset is copied without overflow, and a warning message is issued.

`clab.Container.rt_read()`

Refreshes data from a real-time source by invoking the Load function (see 5.1.2).

`clab.Container.rt_start()`

Starts real-time data acquisition by invoking the Start function (see 5.1.2).

`clab.Container.rt_stop()`

Stops real-time data acquisition by invoking the Stop function (see 5.1.2).

2.3 clab.Mesh

clab.Mesh stores clab.Array data of the x, y and z-axis to be plotted in a surface mesh plot. X and Z are the input of the mesh function. Y is the output matrix, where x values are stored contiguously:

$$Y = \begin{pmatrix} y_{x:0,z:0} & y_{x:1,z:0} & y_{x:2,z:0} & \cdots \\ y_{x:0,z:1} & y_{x:1,z:1} & y_{x:2,z:1} & \cdots \\ \vdots & \vdots & \vdots & \\ y_{x:0,z:N} & y_{x:1,z:0} & y_{x:2,z:N} & \cdots \end{pmatrix}$$

Constructor

An instance of clab.Mesh can be created directly using the following functions:

`clab.Mesh()`

Create an empty clab.Mesh object.

Member Variables

`clab.Mesh.id64` *integer*

Read only. Unique 64-bits identifier assigned upon creation.

`clab.Mesh.flags` *integer*

Read only. Internal use only for implementation details.

`clab.Mesh.xdata` *clab.Array*

clab.Array containing the data points for the x-axis coordinates.

`clab.Mesh.ydata` *clab.Array*

clab.Array containing the data values for the y-coordinate, corresponding to the specified x and z pairs.

`clab.Mesh.zdata` *clab.Array*

clab.Array containing the data points for the z-axis coordinates.

Member Functions

clab.Mesh has no public member functions.

2.4 clab.Model

The `clab.Model` class represents a simulation model. It can be created programmatically or loaded from a JSON file and used for validation, execution, and code generation.

Constructor

An instance of `clab.Model` can be created using:

`clab.Model()`

Creates an empty `clab.Model`.

`clab.Model(path)`

`path` String with the full path to a JSON file.

Creates a `clab.Model` by loading the JSON file located at the given path.

Member Variables

`clab.Model.flags` *integer*

Read only. Internal use only.

Member Functions

`clab.Model.get(key, id)`

`key` [mandatory] String with the parameter name. It can be one of the following values:

inportCount	Number of inports.
outportCount	Number of outports.
x	Block position (x-axis).
y	Block position (y-axis).
width	Block dimension.
height	Block dimension.
red	Block color channel.
green	Block color channel.
blue	Block color channel.
ti	Simulation initial time (seconds).
tf	Simulation final time (seconds).
ts	Simulation step time (seconds).

`id` [optional] Block unique identifier. Required when accessing block-specific parameters.

Returns model-level or block-level parameters.

Examples

```
model.get(key='ts')
model.get(key='x', id=1234)
```

```
clab.Model.set(...)
```

- id [optional] Block unique identifier. Required when modifying block parameters.
- x [optional] Block position (x-axis).
- y [optional] Block position (y-axis).
- width [optional] Block dimension.
- height [optional] Block dimension.
- red [optional] Block color channel. Values outside the range [0.0, 1.0] are ignored.
- green [optional] Block color channel. Values outside the range [0.0, 1.0] are ignored.
- blue [optional] Block color channel. Values outside the range [0.0, 1.0] are ignored.
- ti [optional] Simulation initial time (seconds).
- tf [optional] Simulation final time (seconds).
- ts [optional] Simulation step time (seconds).

Sets model-level or block-level parameters.

Examples

```
model.set(ti=5.0, tf=10.0)
model.set(id=1234, x=10, y=20)
```

```
out = clab.Model.create(class, ...)
```

- out Block unique identifier.

- class [mandatory] Block class. It can be one of the following values:
  - abs Calculates the absolute value of the input.
  - acc Outputs the sum of values from the first port; resets the accumulation to the value of the second port if the third port is True.
  - acos Calculates the arccosine of the input.
  - add Computes the sum of all inputs.
  - asin Calculates the arcsine of the input.
  - atan Calculates the arctangent of the input.
  - cast Converts the input to a different data type, as specified by the type parameter (`'bool'`, `'int8'`, `'int16'`, `'int32'`, `'int64'`, `'uint8'`, `'uint16'`, `'uint32'`, `'uint64'`, `'flt32'`, `'flt64"'`).
  - clamp Limits the value of the first input based on the boundaries set by the second a third inputs.

constant	Represents a constant value source, with its value defined by the `expression` parameter.
cos	Calculates the cosine of the input.
debounce	Debounces input signals, introducing a delay to filter out rapid changes or noise.
delta	Computes the difference between the current sample and the previous one.
display	Displays input data.
divide	Computes the quotient of the first input divided by the second input.
enable	Enables or disables sub-system simulation. By default, *enable* ports are assigned to the port number `-1`.
from	Used with *goto* as an alternative for line connections. This block connects to its *goto* counterpart by its `target` parameter.
goto	Used with *from* as an alternative for line connections. This block connects to its *from* counterpart by its `target` parameter.
hysteresis	The output will be TRUE when input is greater than `valueOn` and FALSE when is less than `valueOff`. Otherwise it'll keep its last value.
if	Implements conditional logic.
inport	Represents an input port for receiving external signals into a subsystem. Its number is set by the `portNumber` parameter (zero-based indexing).
lerp	Computes linear interpolation between two values. Effectively, it outputs $A + U \times (B - A)$. The equation coefficients are set by the `coeffA` and `coeffB` parameters.
log	Computes the logarithm (base e) of the input.
logical	Performs logical operations on input signals. The logical operation is set by the `operator` parameter (`'and'`, `'or'`, `'nand'`, `'nor'`).
lut1d	Performs a one-dimensional lookup table operation. Its values are set by the `out` and `out` parameters.
lut2d	Performs a two-dimensional lookup table operation. Its values are set by the `out`, `bp1` and `bp2` parameters.
max	Determines the maximum value among all input samples.
memory	Delays input sample, by storing its current value and outputing it on the next simulation loop.
min	Determines the minimum value among all input samples.
mod	Calculates the remainder of the division of the first input by the second input.
multiply	Computes the multiplication of all input values.
not	Performs logical negation, converting `True` to `False` and vice versa.
outport	Represents an output port for transmitting signals out of the subsystem. Its number is set by the `portNumber` parameter (zero-based indexing).
poly1st	Outputs the result of a first-order polynomial, as $f(x) = A \times x + B$. The equation coefficients are set by the `coeffA` and `coeffB` parameters.
pow	Raises the first input to the power of the second input.
random	Generates random values, defined by the `min`, `max` and `seed` parameters.
refsys	Represents a reference to a subsystem defined in a external file, set by the `path` parameter.
relational	Compares input values based on relational operations, set by the `operator` parameter (`'eq'`, `'ge'`, `'gt'`, `'le'`, `'lt'`, `'ne'`).

sinewave	Generates a sinusoidal waveform, as specified by the `frequency` and `phase` parameters. The frequency is given in hertz and the phase, in degrees.
squarewave	Generates a square waveform, as specified by the `timeOn`, `timeOff`, `valueOn` and `valueOff` parameters.
step	Outputs a step function, defined by the `initialValue`, `finalValue` and `stepTime` parameters.
sin	Calculates the sine of the input.
sqrt	Calculates the square root of the input.
subsys	Represents a subsystem or a modular block that encapsulates a specific functionality within a larger system.
subtract	Calculates the difference between the first input and the second input.
switch	Routes input signals to the output based on a control signal.
tan	Calculates the tangent of the input.
time	Outputs the current simulation time.
unpack	Unpacks the input value, selecting the desired bits and applying gain and bias (bit extraction), as specified by the `lsb`, `msb`, `coeffA` and `coeffB` parameters. The bits indexing is zero-based.
from_workspace	Imports data from a clab.TimeSeries variable, set by the `obj` parameter.
to_workspace	Exports to a clab.TimeSeries variable, whose name is set by the `name` parameter.

name	[optional] Block name.
parent	[optional] Parent block identifier. If omitted, the block is created in the root subsystem.

Creates a new block and returns its unique identifier.

Examples

```
model = clab.Model()
c0 = model.create('constant', expression=3.14)
s0 = model.create('sinewave', frequency=2.0)
```

```
clab.Model.link(blockFrom, blockTo, portFrom, portTo, name)
```

blockFrom	[mandatory] Source block identifier.
blockTo	[mandatory] Destination block identifier.
portFrom	[mandatory] Source port index (zero-based).
portTo	[mandatory] Destination port index (zero-based).
name	[optional] Connection label.

Creates a connection between blocks.

Examples

```
model.link(blockFrom=b1, blockTo=b2, portFrom=0, portTo=1)
```

```
clab.Model.codegen(lang, path)
```

lang	[mandatory] Target language: `'c'` or `'python'`.
path	[mandatory] Output directory path.

Generates source code from the model. Raises an exception on failure.

```
out = clab.Model.load(path)
```

`out` Error code.

`path` [mandatory] JSON file path.

Loads a model from disk.

```
clab.Model.run()
```

Runs the simulation. Raises an exception on failure.

```
clab.Model.validate()
```

Validates the model. Raises an exception if errors are found.

2.5 clab.Result

The `clab.Result` class stores the output data generated by a simulation. Each subsystem in the model is represented by a corresponding `clab.Result` object.

Constructor

An instance of `clab.Result` can be created using:

```
clab.Result()
```

Creates an empty `clab.Result`.

Member Variables

`clab.Result.flags` *integer*

Read only. Internal use only.

Member Functions

`clab.Result` does not provide any member functions.

2.6 clab.TimeSeries

clab.TimeSeries stores two `clab.Array` instances with data related to the x-axis (typically time, hence the name TimeSeries) and the y-axis. These arrays have the same size. It's expected that the x-axis values to increase and its indexes increases, i.e., $xdata_k < xdata_{k+1}$ for any given k.

Constructor

An instance of`clab.TimeSeries` can be created directly using the following functions:

`clab.TimeSeries()`

Create an empty`clab.TimeSeries` object. It's recommended not to use this function. It's listed here only for the sake of completeness.

`clab.TimeSeries(id64)`

id64 [mandatory] Unique identifier.

Create a`clab.TimeSeries` from a valid variable identifier `id64`. This is for internal usage only.

`clab.TimeSeries(xvalues, yvalues)`

xvalues [mandatory] Python List or Python Tuple with the x-axis values. All elements in the sequence must be numeric.

yvalues [mandatory] Python List or Python Tuple with the y-axis values. All elements in the sequence must be numeric.

Create a`clab.TimeSeries` from two Python Lists or from two Python Tuples. The first one has the values for the x-axis and the second one for the y-axis.

`clab.TimeSeries(rhs)`

rhs [mandatory] `clab.TimeSeries`.

Create a`clab.TimeSeries` from another`clab.TimeSeries` `rhs`. Both objects will share the same xdata and ydata objects (shallow copy).

Member Variables

`clab.TimeSeries.id64` *integer*

Read only. Unique 64-bits identifier.

`clab.TimeSeries.flags` *integer*

Read only. Internal use only.

`clab.TimeSeries.xdata` *clab.Array*

Read/write. `clab.Array` object that stores data to be used in the x-axis (typically time in seconds).

`clab.TimeSeries.ydata` *clab.Array*

Read/write. `clab.Array` object that stores data to be used in the y-axis.

Member Functions

clab.TimeSeries implements the Number Protocol, ie,`clab.TimeSeries` supports the standard operations: add, subtract, multiply, remainder, divide, power, and (&), xor (^), or (|) (and their inplaces equivalent), negative ($\sim$), absolute. For all the aforementioned functions that requires two or more operands, the ydata is aligned over time using the xdata values.

clab.TimeSeries partially implements the Mapping Protocol, meaning that it's possible to access each`clab.TimeSeries` in the container using the [] operator providing the child's source name.

clab.TimeSeries implements the Sequence Protocol, meaning that it's possible to read each x-y pair in the TimeSeries using the [] operator. Each item contains a two-element tuple: the first element contains the x value and the second, the y value.

Besides the standard protocols, the following member functions are available:

```
out = clab.TimeSeries.get_name()
```

`out` String.

Return variable's name. Only available for data loaded directly from sources, i.e, calculated variables will return an empty string.

```
out = clab.TimeSeries.get_fs()
```

`out` Sampling frequency in hertz.

Return the estimated sampling frequency in hertz. If the object has not been loaded yet, this function will force it.

```
out = clab.TimeSeries.get_ts()
```

`out` Sampling period in seconds.

Return the estimated sampling period in seconds. If the object has not been loaded yet, this function will force it.

```
out = clab.TimeSeries.axpb(a, b)
```

`a` [mandatory] Numeric value.

`b` [mandatory] Numeric value.

Apply $A \cdot X + B$ for all values in the x-axis. If the data is empty, this operation will be postponed until the data has been successfully loaded.

```
out = clab.TimeSeries.aypb(a, b)
```

`a` [mandatory] Numeric value.

`b` [mandatory] Numeric value.

Apply $A \cdot Y + B$ for all values in the y-axis. If the data is empty, this operation will be postponed until the data has been successfully loaded.

```
out = clab.TimeSeries.avg()
```

```
        out  Float value.
```

Return the average of the object's yvalues.

```
out = clab.TimeSeries.max()
        out  Float value.
```

Return the maximum of the object's yvalues.

```
out = clab.TimeSeries.min()
        out  Float value.
```

Return the minimum of the object's yvalues.

```
out = clab.TimeSeries.sum()
        out  Float value.
```

Return the sum of the object's yvalues.

```
out = clab.TimeSeries.index(val, op='eq')
        out  Integer value.
        val  [mandatory] Numeric value.
         op  [optional] Operator used by search. Valid operators are eq, ne, le, lt, ge or gt.
```

Return sample index that match the search criteria.

```
clab.TimeSeries.inspect()
```

Print TimeSeries internal information for debugging and troubleshooting.

```
clab.TimeSeries.load()
```

Load TimeSeries values from source file, if they haven't been already. On success, the members `xdata` and `ydata` are filled with the loaded values.

```
out = clab.TimeSeries.slice(tini, tend)
        out  clab.TimeSeries.
        tini  [mandatory] Numeric value. Slice initial time.
       tend  [mandatory] Numeric value. Slice final time.
```

Return a new object containing a slice (or subsequence) of the source object, within the limits defined by the arguments.

```
clab.TimeSeries.sort()
```

Sort samples chronologically (i.e. by x-axis values).

```
out = clab.TimeSeries.tag(val)
        out  String value.
```

Return the tag associated to this objects's value. If there's no tag associated, it will return `'n/a'`.

```
out = clab.TimeSeries.time_eq(val)
```

out Numeric value.

val [mandatory] Numeric value.

Return total time in which y-axis values are equal to `val`.

```
out = clab.TimeSeries.time_ne(val)
```

out Numeric value.

val [mandatory] Numeric value.

Return total time in which y-axis values are not equal to `val`.

```
out = clab.TimeSeries.time_gt(val)
```

out Numeric value.

val [mandatory] Numeric value.

Return total time in which y-axis values are greater than `val`.

```
out = clab.TimeSeries.time_lt(val)
```

out Numeric value.

val [mandatory] Numeric value.

Return total time in which y-axis values are less than `val`.

```
out = clab.TimeSeries.time_bt(val1, val2)
```

out Integer value.

val [mandatory] Numeric value.

Return total time in which y-axis values are between the provided arguments, i.e., simultaneously greater than `val1` and less than `val2`.

3 Settings

Settings can be configured through the dedicated tab in the application UI or via the module functions (see Section 3.3).

3.1 UI Settings

3.1.1 UI Theme

Controls the color theme of the user interface. Default: `'dark'`.

Setting this option to `'lite'` enables the lite theme.

Changes are applied immediately.

3.1.2 Export Theme

Controls the color theme used when exporting charts to files. Default: `'dark'`.

Setting this option to `'lite'` enables the lite theme for exported charts. This allows dark UI users to export charts with a light background without changing the UI theme.

Changes are applied immediately.

3.2 General Settings

3.2.1 Autosave

Enables autosave and defines the interval for saving charts and models in memory to a backup file.

Autosave files are stored in: `c:\Users\<user>\.clab\cache`.

3.2.2 Log Severity Level

Defines the minimum severity level to be logged. Supported values (in increasing order): *Trace*, *Debug*, *Info*, *Warning*, *Error*, *Fatal*.

Logs are stored in: `c:\Users\<user>\.clab\logs`.

Restart required.

3.2.3 Tags File Path

Path to a TSV file used to assign *tags* to specific values in data tips.

If both the parameter name and value match an entry, the corresponding *tag* is displayed.

Useful for parameters with discrete meanings (e.g., *states*, *modes*).

An example file is available in the `example` directory of the installation folder.

Required TSV columns:

Source	Parameter name.
Value	Parameter value.
Tag	Displayed text.

Restart not required.

3.2.4 Unpack File Path

Path to a TSV file used to automatically unpack `clab.TimeSeries` data. If a parameter name matches an entry, its data is unpacked into one or more derived series, equivalent to `clab.unpack()` (see Section 6.2.8).

An example file is available in the `example` directory of the installation folder.

Required TSV columns:

Source	Parameter name used for matching.
Alias	Name of the generated series.
MSB	Most significant bit (see *Indexing*).
LSB	Least significant bit (see *Indexing*).
Resolution	Scaling factor in the transformation.
Offset	Offset in the transformation.
Unit	Documentation only.
Type	Conversion type: `2C`, `BCD`, or `BNR`.
Indexing	Bit indexing scheme (e.g., 0 for zero-based).

Transformation: `alias = source[msb:lsb]` $\times$ `resolution` $+$ `offset`

Restart not required.

3.2.5 Python Dynamic Library Path

Specifies the path to the Python 3 dynamic library (.dll) used by C-LAB.

If not set, the embedded version is used: `c:\<clab folder>\python\python3.dll`.

> NOTE: Requires Python 3.11 or newer.

Restart required.

3.2.6 Memory Allowed

Defines the maximum fraction of system RAM that C-LAB may use. Default: 80%.

If the limit is exceeded, least frequently used data is unloaded until usage falls below the threshold.

Memory usage (used, available, total) is displayed in the application status area.

Restart required.

3.3 Using the Module

3.3.1 Read Configuration Values

```
out = clab.get_config(key, group)
```

out	Value associated with the specified key.
key	[required] Setting key.
group	[optional] Setting group. Default: `'general'`.

Examples

```
value = clab.get_config(key='memory')
```

3.3.2 Write Configuration Values

Some changes are applied only after the module is reloaded, as with updates made through the UI. See Sections 3.1 and 3.2 for details.

```
clab.set_config(key, group, value)
```

key	[required] Setting key. Supported values:
file	Export theme. Values: `'dark'`, `'lite'`. Group: `'theme'`.
ui	UI theme. Values: `'dark'`, `'lite'`. Group: `'theme'`.
x	Window horizontal position. Group: `'window'`.
y	Window vertical position. Group: `'window'`.
height	Window height. Group: `'window'`.
width	Window width. Group: `'window'`.
group	[optional] Setting group. Default: `'general'`.
value	[required] Setting value.

Examples

```
clab.set_config(key='memory', value='80')
clab.set_config(key='ui', group='theme', value='dark')
clab.set_config(key='width', group='window', value=800)
```

3.3.3 Reload Configuration Values from File

Reloads configuration data from file. Required only if the configuration file is modified externally.

```
clab.reload_config()
```

4 Loading Data From Files

C-LAB supports loading data from a wide range of file formats using either native loaders or external plugins.

C-LAB does not require the user to explicitly specify the file format. Instead, format detection is performed automatically.

Format Resolution

When a file is opened, C-LAB determines the appropriate loader using the following procedure:

1. C-LAB queries all available plugins to determine compatibility with the target file. Each plugin **MUST** evaluate the file and indicate whether it can handle the format.
2. If no plugin reports compatibility, C-LAB attempts to load the file using its native loaders.

Compressed Files

If the input file is a compressed archive, C-LAB will extract its contents to a cache directory prior to loading.

After extraction, C-LAB applies the standard format resolution process to the extracted file(s).

4.1 Built-In Loaders

4.1.1 ArduPilot Dataflash Logs

File Extensions `*.bin`, `*.log`

Supported Data Types Numeric and string values only.

Reference https://ardupilot.org/copter/docs/common-downloading-and-analyzing-data-logs-in-mission-planner
html

4.1.2 Delimiter-Separated Values

File Extensions `*.csv`, `*.tsv`, `*.txt`

Format Values will be separated by a delimiter character.

Supported Delimiters comma (ASCII 0x2C), semicolon (ASCII 0x3B), pipe (ASCII 0x7C), tab (ASCII 0x09).

Header The first row will contain parameter names.

Time Axis The first column will contain time values.

4.1.3 Hierarchical Data Format (HDF5)

File Extensions `*.hdf5`, `*.h5`

Supported Numeric Types One-dimensional arrays of signed/unsigned integers (8, 16, 32, 64-bit) and IEEE 754 single/double precision.

Supported String Types Two-dimensional arrays of fixed-length or variable-length ASCII strings.

String Handling String values will be mapped to unique numeric identifiers.

Time Axis Array indices will be used as x-axis values.

4.1.4 Microsoft Excel (2007–2013)

File Extensions `*.xlsx`

Header The first row will contain parameter names.

Time Axis The first column will contain time values.

Limitations Formulas are not supported; only evaluated values may be processed.

4.1.5 MATLAB/Octave MAT Files

File Extensions `*.mat`

Supported Numeric Types Arrays of signed/unsigned integers (8, 16, 32, 64-bit) and IEEE 754 single/double precision.

Time Axis Array indices will be used as x-axis values.

Limitations Cells, structures, and non-numeric objects are not supported.

Supported Versions Versions 4, 5, and 7.3.

4.1.6 Parquet Files

File Extensions `*.parquet`, `*.gzip`

Data Mapping Each column will be treated as an independent variable.

Supported Types Only numeric columns are processed; non-numeric columns are ignored.

Null Handling Null samples will be skipped.

Time Axis If one or more time-typed columns are present, the first (lowest index) will be used as the time vector.

If no time-typed column is present, the time vector will be generated as a strictly increasing sequence starting at zero.

4.1.7 Waveform Audio File Format (WAV)

File Extensions `*.wav`

Description Standard uncompressed audio file format.

4.2 Custom DLL Plugin Loaders

C-LAB supports custom Dynamic Link Libraries (DLLs) as user-defined plugins for loading data from files.

A plugin **MUST** implement the API defined in Section 4.2.2 and **MUST** be located in the plugins directory at `C:\Users\<user>\.clab\plugins`.

A DLL will be loaded only once per process. Therefore, a plugin implementation **MUST** support concurrent handling of multiple files.

NOTE: Identifiers containing a period (.) are interpreted as hierarchical paths. For example, `parent.child` is rendered as node `child` under `parent`.

NOTE: Plugin management is available through a dedicated tab in the application interface.

NOTE: A reference implementation is provided in the installation directory under `plugindll`. This example defines all required functions and data types.

4.2.1 Plugin Lifecycle

When a file is opened, C-LAB determines plugin compatibility and executes the following sequence:

1. C-LAB calls `IsCompatible()` for each enabled plugin.
2. If a compatible plugin is found, C-LAB calls `Initialize()`. The plugin **MUST** initialize internal state and allocate required resources.
3. C-LAB calls `GetParameterName()` to enumerate available parameters. The plugin **MUST** return consistent and deterministic results.
4. C-LAB calls `Load()` for each requested parameter. The plugin **MUST** return synchronized x- and y-value arrays of equal length.
5. When the file is closed or no longer required, C-LAB calls `Terminate()`. The plugin **MUST** release all associated resources.

Error Handling

If `Initialize()` or `Load()` returns a non-zero value, C-LAB will call `GetErrorMessage()` (if implemented) to retrieve diagnostic information.

If no error is reported, C-LAB **MAY** call `GetInfoMessage()` to retrieve informational messages.

4.2.2 Plugin API

`GetPluginAuthor`	Returns the plugin author.
`GetPluginDate`	Returns the plugin release date.
`GetPluginHTML`	Returns plugin metadata formatted as HTML.
`GetPluginExtension`	Returns the file extension used for filtering in file dialog boxes.
`GetPluginName`	Returns the plugin name. Used in file dialog filters.
`GetPluginVersion`	Returns the plugin version.
`GetErrorMessage`	Returns a human-readable error message associated with the most recent failure of `Initialize()` or `Load()`.
`GetInfoMessage`	Returns a human-readable informational message.
`GetConfigCount`	Returns the number of configuration entries.
`GetConfigKey`	Returns the configuration key for a given index.
`GetConfigHint`	Returns a short description for a configuration entry.
`GetConfigType`	Returns the configuration value type for a given index. Valid values are `file`, `folder`, `text`.
`IsCompatible`	Determines whether the plugin supports the given file. Return 0 if not compatible. Any other value is interpreted as compatible.
`Initialize`	Initializes the plugin using the provided file path and configuration values. Return 0 on success. Any other value is interpreted as error.
`Terminate`	Releases resources associated with the provided file path. Return 0 on success. Any other value is interpreted as error.
`GetParameterCount`	Returns the number of available parameters for the given file path.
`GetParameterName`	Returns the parameter name for a given index and file path.
`Load`	Loads parameter data. – The plugin **MUST** return two arrays: x-values and y-values. – Both arrays **MUST** have identical length. – X-values typically represent time (in seconds).

4.3 Custom Python Plugin Loaders

C-LAB supports custom Python® 3 modules as user-defined plugins for loading data from files.

A plugin **MUST** implement the API defined in Section 4.3.2 and **MUST** be located in the plugins directory at `C:\Users\<user>\.clab\plugins`.

A Python module is imported only once per interpreter session. Therefore, a plugin implementation **MUST** support concurrent handling of multiple files.

NOTE: Identifiers containing a period (.) are interpreted as hierarchical paths. For example, `parent.child` is rendered as node `child` under parent `parent`.

NOTE: Plugin management is available through a dedicated tab in the application interface.

NOTE: A reference implementation is provided in the installation directory as `plugin.py`. This example defines all required functions and data types.

4.3.1 Plugin Lifecycle

When a file is opened, C-LAB determines plugin compatibility and executes the following sequence:

1. C-LAB calls `compatible()` for each enabled plugin.
2. If a compatible plugin is found, C-LAB calls `init()`. The plugin **MUST** initialize internal state and allocate required resources.
3. C-LAB calls `list_names()` to enumerate available parameters. The plugin **MUST** return consistent and deterministic results.
4. C-LAB calls `load()` for each requested parameter. The plugin **MUST** return synchronized x- and y-value sequences of equal length.
5. When the file is closed or no longer required, C-LAB calls `terminate()`. The plugin **MUST** release all associated resources.

Error Handling

If `init()` or `load()` returns a non-zero value, the operation is considered to have failed. Implementations **SHOULD** provide meaningful error signaling via return codes or exceptions.

4.3.2 Plugin API

`plugin_author` Returns the plugin author.

`plugin_date` Returns the plugin release date.

`plugin_doc` Returns plugin metadata formatted as HTML.

`plugin_ext` Returns the file extension used for filtering in file dialog boxes.

`plugin_name` Returns the plugin name. Used in file dialog filters.

`plugin_version` Returns the plugin version.

`config_keys` Returns a sequence containing all required configuration keys.

`config_hints` Returns a sequence of descriptions corresponding to each configuration key.

`compatible` Determines whether the plugin supports the given file.

Return 0 if not compatible. Any other value is interpreted as compatible.

`init` Initializes the plugin using the provided file path and configuration values.

Return 0 on success. Any other value is interpreted as error.

`terminate` Releases resources associated with the provided file path.

Return 0 on success. Any other value is interpreted as error.

`list_names` Returns a sequence containing the parameter names for the given file path.

`load` Loads parameter data.

Returns a tuple containing two sequences: x-values and y-values.

- Both sequences **MUST** be provided.
- Both sequences **MUST** have identical length.
- x-values typically represent time (in seconds).

5 Loading Data From Real-Time Sources

NOTE: This feature is currently experimental.

C-LAB provides support for loading data from real-time sources via external custom plugins, enabling dynamic data integration during runtime.

5.1 Custom DLL Plugin Loaders

C-LAB supports custom Dynamic Link Libraries (DLLs) as user-defined plugins for loading data from real-time sources.

A plugin **MUST** implement the API described in Section 5.1.2 and **MUST** be located in the plugins directory at `C:\Users\<user>\.clab\plugins`.

NOTE: Identifiers containing a period (.) are interpreted as hierarchical paths. For example, `parent.child` is rendered as node `child` under `parent`.

NOTE: Plugin management is available through a dedicated tab in the application interface.

NOTE: A reference implementation is provided in the installation directory under `plugindll_rt`. This example defines all required functions and data types.

5.1.1 Plugin Lifecycle

The plugin lifecycle is defined as follows:

1. C-LAB calls `Initialize()` to initialize the plugin with a file path and configuration values. The plugin **MUST** allocate and prepare all required resources during this phase.
2. C-LAB calls `GetParameterName()` to enumerate available parameters. The plugin **MUST** return consistent and deterministic results for a given input.
3. C-LAB calls `Start()` to signal the beginning of data acquisition. The plugin **SHOULD** begin producing data after this call.
4. C-LAB calls `Load()` periodically for each requested parameter. The plugin **MUST** return synchronized x- and y-value arrays of equal length.
5. C-LAB calls `Stop()` to signal the end of data acquisition. The plugin **SHOULD** cease data production promptly.
6. C-LAB calls `Terminate()` when the plugin is no longer required. The plugin **MUST** release all allocated resources.

Error Handling

If `Initialize()` or `Load()` returns a non-zero value, C-LAB will invoke `GetErrorMessage()` (if implemented) to obtain diagnostic information.

If no error is reported, C-LAB **MAY** invoke `GetInfoMessage()` to retrieve informational messages for display.

5.1.2 Plugin API

`GetPluginAuthor`	Returns the plugin author.
`GetPluginDate`	Returns the plugin release date.
`GetPluginHTML`	Returns plugin metadata formatted as HTML.
`GetPluginName`	Returns the plugin name. Used in file dialog filters.
`GetPluginVersion`	Returns the plugin version.
`GetErrorMessage`	Returns a human-readable error message associated with the most recent failure of `Initialize()` or `Load()`.
`GetInfoMessage`	Returns a human-readable informational message.
`GetConfigCount`	Returns the number of configuration entries.
`GetConfigKey`	Returns the configuration key for a given index.
`GetConfigHint`	Returns a short description for a configuration entry.
`GetConfigType`	Returns the configuration value type for a given index. Valid values are `file`, `folder`, `text`.
`Initialize`	Initializes the plugin using the provided file path and configuration values. Return 0 on success. Any other value is interpreted as error.
`Terminate`	Releases resources associated with the provided file path. Return 0 on success. Any other value is interpreted as error.
`Start`	Signals the start of data acquisition.
`Stop`	Signals the stop of data acquisition.
`GetParameterCount`	Returns the number of available parameters for the given file path.
`GetParameterName`	Returns the parameter name for a given index and file path.
`Load`	Loads parameter data. – The plugin **MUST** return two arrays: x-values and y-values. – Both arrays **MUST** have identical length. – X-values typically represent time (in seconds).

6 Built-In Functions

The C-LAB module provides a set of built-in functions for processing `clab.Array`, `clab.Mesh`, and `clab.TimeSeries` objects, in addition to those described in their respective sections.

6.1 Standard Math Functions

6.1.1 Exponential Functions

Return a new object, whose values are equal to e (Euler's number, $2.7182818...$) raised to power of each value in the input argument.

```
out = clab.exp(in)
        out  clab.TimeSeries object
         in  [mandatory] clab.TimeSeries object
```

6.1.2 Logarithm Functions

Return a new object, whose values are equal to the natural (base e) logarithm of the argument. There is also a version that computes the logarithm with base 10 of the argument.

```
out = clab.log(in)
        out  clab.TimeSeries object
         in  [mandatory] clab.TimeSeries object
```

```
out = clab.log10(in)
        out  clab.TimeSeries object
         in  [mandatory] clab.TimeSeries object
```

6.1.3 Rounding Functions

Computes the nearest integer value to the argument, following the specified rounding rules:

ceil	Computes the smallest integer value not less than the argument (ceiling).
floor	Computes the largest integer value not greater than the argument (floor).
round	Computes the nearest integer value to the argument, rounding halfway cases away from zero.
trunc	Computes the integer value by discarding the fractional part (truncation towards zero).

```
out = clab.<operation>(in)
        out  clab.TimeSeries object
         in  [mandatory] clab.TimeSeries object
```

6.1.4 Square Root Functions

Return a new object, whose values are equal to the square root of each value in the input argument.

```
out = clab.sqrt(in)
        out  clab.TimeSeries object
         in  [mandatory] clab.TimeSeries object
```

6.1.5 Trigonometric Functions

Computes standard trigonometric operations. All results for $\sin, \cos, \tan,$ arc ... are in **radians** unless the degree suffix is used.

cos	Computes the cosine (radians).
sin	Computes the sine (radians).
tan	Computes the tangent (radians).
cosd	Computes the cosine (degrees).
sind	Computes the sine (degrees).
tand	Computes the tangent (degrees).
acos	Computes the arc cosine (radians).
asin	Computes the arc sine (radians).
atan	Computes the arc tangent (radians).
acosd	Computes the arc cosine (degrees).
asind	Computes the arc sine (degrees).
atand	Computes the arc tangent (degrees).

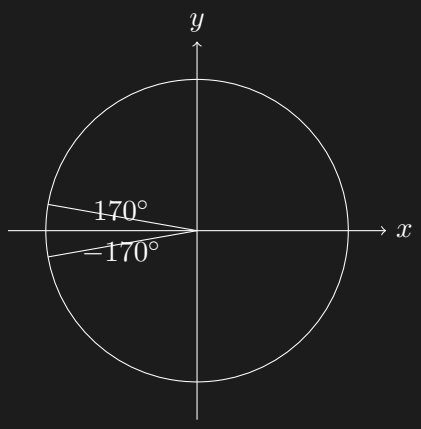
```
out = clab.<operation>(in)
        out  clab.TimeSeries object
         in  [mandatory] clab.TimeSeries object
```

6.2 Custom Functions

6.2.1 Angular Distance

Calculate the minimum difference between two angles in a circle.

The function computes `lhs - rhs`, but taking into account the angle position in the circle. For instance, the distance between 0.0 and 360.0 is 0.0 and between +170.0 and -170.0 is 20.0.




```
out = clab.angular_distance(lhs, rhs, type='deg')
```

out	clab.TimeSeries.
lhs	[mandatory] `clab.TimeSeries`.
rhs	[mandatory] `clab.TimeSeries`.
type	[optional] String with the angle value type. It can be either `deg` (for degrees) or `rad` (for radians).

Examples

```
diff = clab.angular_distance(y2, y1, type='rad')
```

6.2.2 Discrete Derivative

The output is calculated as follows:

$$out_k = \frac{in_k - in_{k-1}}{t_k - t_{k-1}}$$

```
out = clab.derivative(in)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.

Examples

```
dY = clab.derivative(y)
```

6.2.3 Discrete Integration

Discrete integration (numerical quadrature), calculated using the trapezoidal rule.

```
out = clab.integral(in)
        out  clab.TimeSeries.
         in  [mandatory] clab.TimeSeries.
```

Examples

```
yint = clab.integral(y)
```

6.2.4 Adjacent Differences

Differences between adjacent elements, calculated as

$$out_k = in_k - in_{k-1}$$

```
out = clab.diff(in)
        out  clab.TimeSeries.
         in  [mandatory] clab.TimeSeries.
```

Examples

```
yint = clab.diff(y)
```

6.2.5 Accumulate

The nth element is the sum of the 1...n elements of the argument.

```
out = clab.accumulate(in)
        out  clab.TimeSeries.
         in  [mandatory] clab.TimeSeries.
```

Examples

```
acc = clab.accumulate(y)
```

6.2.6 Estimate Sampling Frequency

The output represents the estimated instantaneous sampling frequency corresponding to the provided sample argument.

```
out = clab.estimate_sampling_frequency(in)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.

Examples

```
fs = clab.estimate_sampling_frequency(y)
```

6.2.7 Estimate Sampling Period

The output represents the estimated instantaneous sampling period corresponding to the provided sample argument.

```
out = clab.estimate_sampling_period(in)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.

Examples

```
ts = clab.estimate_sampling_period(y)
```

6.2.8 Unpack

Unpack values, selecting the desired bits and applying gain and bias (bit extraction).

The output is calculated as

$$out_k = unpacked_k[msb : lsb] \times gain + bias$$

where $unpacked$ is obtained by converting the input using the selected conversion *type*.

```
out = clab.unpack(in, msb=0, lsb=0, base=0, gain=1.0, bias=0.0, type='USGN')
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.
msb	[optional] Most significant bit, indexed from base.
lsb	[optional] Least significant bit, indexed from base.
base	[optional] Smallest value in the bit indexing system.
gain	[optional] Value gain.
bias	[optional] Value bias.
type	[optional] Type conversion method. It can be `2C` (two's complement), `BCD` (binary coded decimal), `USGN` (unsigned binary) or `SGN` (signed binary).

Examples

```
unpacked1 = clab.unpack(in, msb=10, lsb=2)
unpacked2 = clab.unpack(in, msb=32, lsb=28, gain=0.5, bias=10.0)
unpacked3 = clab.unpack(in, msb=28, lsb=10, type='2C')
```

6.2.9 Debounce

Delay output of boolean signal after rising or falling edge. Persistences should be provided in the same unit of the argument's time vector.

```
out = clab.debounce(in, on=0.0, off=0.0)
```

out	clab.TimeSeries.
in	[mandatory] clab.TimeSeries.
on	[optional] Persistence (in x-data units) for output 'on' signal.
off	[optional] Persistence (in x-data units) for output 'off' signal.

Examples

```
out1 = clab.debounce(in, on=10)
out2 = clab.debounce(in, off=0.5)
```

6.2.10 Hysteresis

Output 1 if the argument is bigger than the upper threshold, 0 if the argument is smaller than the lower threshold and keep the last value otherwise.

```
out = clab.hysteresis(in, lower=0.0, upper=0.0)
```

out	clab.TimeSeries.
in	[mandatory] clab.TimeSeries.
lower	[optional] Lower threshold.
upper	[optional] Upper threshold.

Examples

```
out = clab.hysteresis(in, lower=90.0, upper=180.0)
```

6.2.11 Clamp

For each input sample, if value is less than lower limit, returns lower limit. If argument is greater than upper limit, returns higher limit. Otherwise returns input value.

```
out = clab.clamp(in, lower=-inf, upper=+inf)
```

out	clab.TimeSeries.
in	[mandatory] clab.TimeSeries.
lower	[optional] Lower limit.
upper	[optional] Upper limit.

Examples

```
out = clab.clamp(in, lower=90.0, upper=180.0)
```

6.2.12 Drop

Create a new TimeSeries dropping values that are within either of the specified limits. The function will drop a value y that evaluates true for either op1(y, lim1) or op2(y, lim2).

Valid operators are eq, ne, le, lt, ge or gt.

```
out = clab.drop(in, lim1=-inf, lim2=+inf, op1='le', op2='ge')
```

out	clab.TimeSeries.
in	[mandatory] clab.TimeSeries.
lim1	[optional] Limit 1.
lim2	[optional] Limit 2.
op1	[optional] Operator applied to limit 1.
op2	[optional] Operator applied to limit 2.

Examples

```
out = clab.drop(in, lim1=90.0, lim2=180.0, op1='lt', op2='gt')
```

6.2.13 In Range

For each input sample, if its value is less than the upper limit and greater than the lower limit for longer than the define persistency time, returns 1.0. Otherwise returns 0.0.

```
out = clab.in_range(in, lower=-inf, upper=+inf, pers=0.0)
```

out	clab.TimeSeries.
in	[mandatory] clab.TimeSeries.
lower	[optional] Lower limit.
upper	[optional] Upper limit.
pers	[optional] Persistency time (in x-data units).

Examples

```
out1 = clab.in_range(in, lower=90.0, upper=180.0)
out2 = clab.in_range(in, upper=180.0, pers=10.0)
```

6.2.14 Stale

This function detects staleness of the input argument during the defined persistency window.

It returns `1.0` to indicate staleness, or `0.0` if any sample has changed within the previous samples persistency window.

```
out = clab.stale(in, pers)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.
pers	[mandatory] Persistency time (in x-data units).

Examples

```
out = clab.stale(in, pers=2.0)
```

6.2.15 Low-pass Filter

First order low pass filter.

```
out = clab.lowpass(in, tau)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.
tau	[mandatory] Filter time constant.

Examples

```
filtered = clab.lowpass(in, tau=2)
```

6.2.16 High-pass Filter

First order high pass filter.

```
out = clab.highpass(in, tau)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.
tau	[mandatory] Filter time constant.

Examples

```
filtered = clab.highpass(in, tau=2)
```

6.2.17 Linear Interpolation

Computes `a + in × (b - a)`, i.e. the linear interpolation between `a` and `b` for the parameter `in` (or extrapolation, when `in` is outside the range [0,1]).

```
out = clab.lerp(in, a, b)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.
a	[mandatory] Start value.
b	[mandatory] End value.

Examples

```
out = clab.lerp(in, a=2.0, b=4.0)
```

6.2.18 Lookup Table Interpolation (1D)

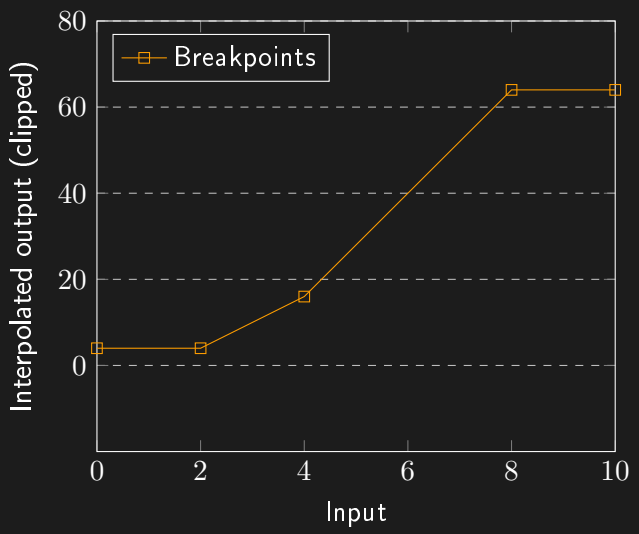
Computes intermediate values of the input from the provided set of discrete data points using interpolation techniques.

```
out = clab.lut1d(in, bp, val, clip=True)
```

out	clab.TimeSeries.
in	[mandatory] `clab.TimeSeries`.
bp	[mandatory] Breakpoints list.
val	[mandatory] Values list.
clip	[optional] If true, values outside the boundaries of the breakpoint list will be clipped.

Examples

```
out = clab.lut1d(in, bp=[2.0, 4.0, 8.0], val=[4.0, 16.0, 64.0], clip=True)
```



6.2.19 Lookup Table Interpolation (2D)

Computes intermediate values of the inputs from the provided set of discrete data points using interpolation techniques.

```
out = clab.lut2d(in1, in2, bp1, bp2, val, clip=True)
```

out	clab.TimeSeries.
in1	[mandatory] `clab.TimeSeries`. This is the *row* variable in the output values table.
in2	[mandatory] `clab.TimeSeries`. This is the *column* variable in the output values table.
bp1	[mandatory] Breakpoints list for `in1`.
bp2	[mandatory] Breakpoints list for `in2`.
val	[mandatory] Values matrix. This should be a list of lists.
clip	[optional] If true, values outside the boundaries of the breakpoint list will be clipped.

Examples

```
table = [
[1.0, 2.0, 3.0, 4.0],
[2.0, 3.0, 4.0, 5.0],
[3.0, 4.0, 5.0, 6.0],
[4.0, 5.0, 6.0, 7.0],
[5.0, 6.0, 7.0, 8.0]]

out = clab.lut2d(in1=ts1, in2=ts2, bp1=[1.0, 2.0, 3.0, 4.0, 5.0], bp2=[0.0, 1.0, 2.0, 3.0],
val=table)
```

6.2.20 Fast Fourier Transform Data

This function shares similarities with the Fast Fourier Transform Plot (see 7.1.6), except that it doesn't create a graphical plot; instead, it provides the data as a tuple.

```
out = clab.fftdata(in, ini=-inf, end=+inf)
```

out	2×N tuple, where the first element is the frequency in hertz and the second, the FFT component for that frequency.
in	[mandatory] `clab.TimeSeries`.
ini	[optional] Initial time of the range to be analysed.
end	[optional] Final time of the range to be analysed.

Examples

```
out = clab.fftdata(in)
```

6.2.21 HeatMap Data

This function shares similarities with the HeatMap Plot (see 7.1.8), except that it doesn't create a graphical heatmap; instead, it provides the data as a tuple.

```
out = clab.heatmapdata(in1, in2, ini=-inf, end=+inf, cols=30, rows=30)
```

- **out** Tuple with the heatmap data, formatted as described:
  - Index 0 Matrix (tuple of tuples) `cols`×`rows`. Each cell contains the number of samples that fit that square.
  - Index 1 List with the column labels.
  - Index 2 List with the row labels.
- **in1** [mandatory] `clab.TimeSeries`.
- **in2** [mandatory] `clab.TimeSeries`.
- **ini** [optional] Initial time of the range to be analysed.
- **end** [optional] Final time of the range to be analysed.
- **cols** [optional] Number of columns in the heatmap. Value is limited between 3 and 1000.
- **rows** [optional] Number of rows in the heatmap. Value is limited between 3 and 1000.

Examples

```
out = clab.heatmapdata(in1, in2, cols=8, rows=8)
```

6.2.22 Histogram Data

This function shares similarities with the Histogram Plot (see 7.1.9), except that it doesn't create a graphical histogram; instead, it provides the data as a tuple.

```
out = clab.histdata(in, ini=-inf, end=+inf, nbins=8, bins=[])
```

- **out** Matrix (tuple of tuples) with the histogram data. The first element of each tuple is the bin's label and second the bin's value.
- **in** [mandatory] `clab.TimeSeries`.
- **ini** [optional] Initial time of the range to be analysed.
- **end** [optional] Final time of the range to be analysed.
- **nbins** [optional] Number of sections. Value cannot be less than 3.
- **bins** [optional] List with the bins limits. Its length cannot be less than 2. If this argument is provided along with `nbins`, the latter will be ignored.

Examples

```
out = clab.histdata(in, nbins=4)
out = clab.histdata(in, bins=[1,3,5,7])
```

6.2.23 Convert Units

Converts a TimeSeries object values to the target unit.

```
out = clab.convert(in, source, target)
```

- out: clab.TimeSeries.
- in: [mandatory] `clab.TimeSeries`.
- source: [mandatory] String with unit name to convert from.
- target: [mandatory] String with unit name to convert to. It can be one of the following:
  - Angle `'deg'`, `'rad'`.
  - Length `'feet'`, `'inches'`, `'kilometers'`, `'meters'`, `'miles'`, `'nautical miles'`.
  - Mass `'kilograms'`, `'pounds'`.
  - Speed `'knots'`, `'km/h'`, `'m/s'`, `'mph'`.
  - Temperature `'celsius'`, `'fahrenheit'`, `'kelvin'`.

Examples

```
out = clab.convert(in, 'rad', 'deg')
```

6.3 Export Functions

6.3.1 Export To File

Export a clab.Container or a list of`clab.TimeSeries` objects to a delimiter-separated file, to a HDF5 file or to a Parquet file.

If the samples windows (defined by `tini` and `tend`) are not the same among the variables being exported, C-LAB will select the lowest initial time and the highest final time across all provided objects to define the overall export window.

```
clab.export_to_file(path, delimiter, step, tini, tend, src, tags=False)
```

- path: [mandatory] Full path of the output file. C-LAB will select the output type based on this argument extension (e.g., `.tsv`, `.hdf5` or `.parquet`).
- tini: [mandatory] Start time for the export window.
- tend: [mandatory] End time for the export window.
- delimiter: [optional] Character used to separate the values in the exported file. Used only when exporting to a delimiter-separated file.
- step: [optional] Sampling step period. Used only when exporting to a delimiter-separated file.
- src: [mandatory] Source data to export. This can be one of the following:
  - List of tuples: Each tuple must contain a`clab.TimeSeries` object and the desired name to be used as the column header in the file. E.g., `[(obj1, name1), (obj2, name2)]`.
  - clab.Container object: The function will use each `clab.TimeSeries` name within the container as a column header.

tags [optional] If True, the exported file will contain the column headers defined by the tag names instead of the corresponding value data.

Examples

```
clab.export_to_file(path='c:/output.tsv', step=0.2, tini=0.0, tend=100.0, delimiter='\t',
src=[(obj1, 'name1'),(obj2, 'name2')])
clab.export_to_file(path='c:/output.hdf5', tini=0.0, tend=100.0, src=f)
```

6.4 Factory Functions

6.4.1 Create Random clab.Array

Create an clab.Array with the length specified and all values set to random values in the interval [-1.0, +1.0].

```
out = clab.create_array_random(len)
```

out clab.Array object.

len [mandatory] Length of the new array.

Examples

```
out = clab.create_array_random(500)
```

6.4.2 Create All-One clab.Array

Create an clab.Array with the length specified and all values set to 1.0.

```
out = clab.create_array_ones(len)
```

out clab.Array object.

len [mandatory] Length of the new array.

Examples

```
out = clab.create_array_ones(500)
```

6.4.3 Create All-Zero clab.Array

Create an clab.Array with the length specified and all values set to 0.0.

```
out = clab.create_array_zeros(len)
```

out clab.Array object.

len [mandatory] Length of the new array.

Examples

```
out = clab.create_array_zeros(500)
```

6.4.4 Create clab.Container from Clipboard

Create an clab.Container object from clipboard content.

This functionality is also available from the UI, allowing the user to edit a data file (an Excel file for instance), CTRL+C its content into the clipboard and import it into C-LAB without saving it to the disk.

```
out = clab.create_from_clipboard()
```

out	clab.Container object.

6.4.5 Create Chirp Signal clab.TimeSeries

Create aclab.TimeSeries object with a chirp signal (sine sweep) values in the y-axis.

```
out = clab.create_chirp(xmin=0.0, xmax=100.0, xdiff=0.01, freq_start=0.0, freq_end=10.0)
```

out	clab.TimeSeries object.
xmin	[optional] Minimum value for the x-axis (in engineering units, usually time in seconds).
xmax	[optional] Maximum value for the x-axis (in engineering units, usually time in seconds).
xdiff	[optional] Difference between adjacent elements in the x-axis (in engineering units).
freq_start	[optional] Start frequency (in hertz).
freq_end	[optional] End frequency (in hertz).

Examples

```
out = clab.create_chirp(freq_start=8.0, freq_end=0.0)
```

6.4.6 Create Constant clab.TimeSeries

Create aclab.TimeSeries object with a constant value in the y-axis.

```
out = clab.create_const(value=1.0, xmin=0.0, xmax=100.0, xdiff=0.01)
```

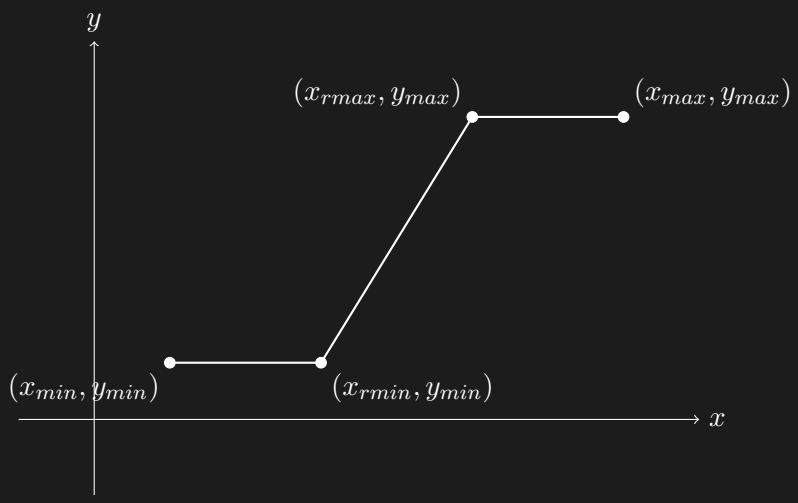
out	clab.TimeSeries object.
value	[optional] Constant value.
xmin	[optional] Minimum value for the x-axis (in engineering units, usually time in seconds).
xmax	[optional] Maximum value for the x-axis (in engineering units, usually time in seconds).
xdiff	[optional] Difference between adjacent elements in the x-axis (in engineering units).

Examples

```
out = clab.create_const(value=3.1415, xmin=10, xmax=20, xdiff=0.5)
```

6.4.7 Create Ramp clab.TimeSeries

Create `aclab.TimeSeries` object with a ramp signal in the y-axis, defined between y_{min} and y_{max} over the x-range $[x_{min}, x_{max}]$.




```
out = clab.create_ramp(rxmin, rxmax, ymin, ymax, xmin, xmax, xdiff)
```

out	clab.TimeSeries object.
rxmin	[mandatory] Value for the x-axis at the start of the ramp (x_{min}).
rxmax	[mandatory] Value for the x-axis at the end of the ramp (x_{max}).
ymin	[mandatory] Minimum value for the y-axis.
ymax	[mandatory] Maximum value for the y-axis.
xmin	[mandatory] Minimum value for the x-axis (in engineering units, usually time in seconds).
xmax	[mandatory] Maximum value for the x-axis (in engineering units, usually time in seconds).
xdiff	[mandatory] Difference between adjacent elements in the x-axis (in engineering units).

Examples

```
out = clab.create_ramp(rxmin=1, rxmax=5, ymin=0.75, ymax=4, xmin=1, xmax=7, xdiff=0.5)
```

6.4.8 Create Random clab.TimeSeries

Create `aclab.TimeSeries` object with random values in the y-axis, generated from a specified distribution.

```
out = clab.create_random(dist='uniform', type='double', seed=0, ymin=0.0, ymax=1.0, mean=0.0,
stddev=1.0, xmin=0.0, xmax=100.0, xdiff=0.01)
```

out	clab.TimeSeries object.
dist	[optional] Type of distribution for the random values. Options are 'uniform' or 'normal'.
type	[optional] Type of data. Options are 'double' or 'int'.
seed	[optional] Seed for the pseudo-random generator.
ymin	[optional] Minimum value for the y-axis. Used when 'uniform' distribution is selected.
ymax	[optional] Maximum value for the y-axis. Used when 'uniform' distribution is selected.
mean	[optional] Mean value for the y-axis. Used when 'normal' distribution is selected.
stddev	[optional] Standard deviation value for the y-axis. Used when 'normal' distribution is selected.
xmin	[optional] Minimum value for the x-axis (in engineering units, usually time in seconds).

xmax [optional] Maximum value for the x-axis (in engineering units, usually time in seconds).
xdiff [optional] Difference between adjacent elements in the x-axis (in engineering units).

Examples

```
out = clab.create_random(dist='normal', mean=1.0, stddev=2.0)
out = clab.create_random(dist='uniform', ymin=-1.0, ymax=1.0, xmin=10, xmax=20, xdiff=0.5)
```

6.4.9 Create Sine Signal clab.TimeSeries

Create aclab.TimeSeries object with a sine wave in the y-axis.

```
out = clab.create_sine(freq=1.0, phase=0.0, amplitude=1.0, xmin=0.0, xmax=100.00, xdiff=0.01)
```

out clab.TimeSeries object.
freq [optional] Frequency in hertz of the sine wave.
phase [optional] Phase in degrees of the sine wave.
amplitude [optional] Minimum absolute value for the y-axis.
xmin [optional] Minimum value for the x-axis (in engineering units, usually time in seconds).
xmax [optional] Maximum value for the x-axis (in engineering units, usually time in seconds).
xdiff [optional] Difference between adjacent elements in the x-axis (in engineering units).

Examples

```
out = clab.create_sine(freq=10, phase=90, amplitude=10.0, xmin=10, xmax=20, xdiff=0.5)
```

6.4.10 Create Step Signal clab.TimeSeries

Create aclab.TimeSeries object with a step signal in the y-axis. The signal will start with the minimum value defined and switch to the maximum when the time is equal to or greater than step.

```
out = clab.create_step(step=50.0, ymin=0.0, ymax=1.0, xmin=0.0, xmax=100.0, xdiff=0.01)
```

out clab.TimeSeries object.
step [optional] Step in engineering units, usually time in seconds.
ymin [optional] Minimum value for the y-axis.
ymax [optional] Maximum value for the y-axis.
xmin [optional] Minimum value for the x-axis (in engineering units, usually time in seconds).
xmax [optional] Maximum value for the x-axis (in engineering units, usually time in seconds).
xdiff [optional] Difference between adjacent elements in the x-axis (in engineering units).

Examples

```
out = clab.create_step(step=1.0, xmin=10, xmax=20, xdiff=0.5)
```

6.4.11 Create Sinc Signal clab.Mesh Grid

Create a clab.Mesh object with a sinc function signal in the y-axis.

For 2 dimensions, $sinc$ is defined as

$$sinc(X) = \frac{\sin(X)}{X}$$

For 3 dimensions, sinc is defined as

$$sinc(X, Z) = \frac{\sin(R)}{R}$$

where $R = \sqrt{X^2 + Z^2}$

```
out = clab.create_mesh_sinc(xmin=-10.0, xmax=+10.0, xdiff=0.5, zmin=-10.0, zmax=+10.0,
zdiff=0.5)
```

out	clab.Mesh object.
xmin	[optional] Minimum value for the x-axis.
xmax	[optional] Maximum value for the x-axis.
xdiff	[optional] Step value for the x-axis.
zmin	[optional] Minimum value for the z-axis.
zmax	[optional] Maximum value for the z-axis.
zdiff	[optional] Step value for the z-axis.

Examples

Create a mesh with 1600 values in the Y matrix

```
out = clab.create_mesh_sinc()
```

Create a mesh with 40000 values in the Y matrix

```
out = clab.create_mesh_sinc(xdiff=0.1, zdiff=0.1)
```

6.5 Find Functions

6.5.1 Find a Specific clab.TimeSeries

Find the first `clab.TimeSeries` object whose name matches the regular expression provided.

```
out = clab.find(regex)
```

out	clab.TimeSeries.
regex	[mandatory] Valid regular expression.

Examples

```
out1 = clab.find('*.value')
out2 = clab.find('value')
```

6.5.2 List Matching clab.TimeSeries

List all `clab.TimeSeries` objects whose names match the regular expression provided.

```
out = clab.list(regex)
```

- `out` List of`clab.TimeSeries` objects.
- `regex` [mandatory] Valid regular expression.

Examples

```
out1 = clab.list('*.value')
out2 = clab.list('value')
```

6.5.3 List Paths From Opened clab.Container

List the file system paths of all `clab.Container` objects currently loaded or accessible by the module.

```
out = clab.list_paths()
```

- `out` List of strings (file system paths).

Examples

```
out1 = clab.list_paths()
```

6.6 Fit Functions

6.6.1 Linear Data Fit

Computes linear regression to fit observed data into a first order linear function

$$Y = A \times X + B$$

```
out = clab.fit_linear(in)
```

- `out` Tuple with the following values:
  - 0 Correlation coefficient between X and Y.
  - 1 Coefficient A (slope).
  - 2 Coefficient B (Y-intercept).
- `in` [mandatory] `clab.TimeSeries`.

6.6.2 Exponential Data Fit

Computes exponential regression to fit observed data into a function

$$Y = A \times e^{(X \times B)}$$

```
out = clab.fit_exp(in)
```

out	Tuple with the following values:
0	Correlation coefficient between X and Y.
1	Coefficient A.
2	Coefficient B.
in	[mandatory] `clab.TimeSeries`.

6.6.3 Polynomial Data Fit

Computes polynomial regression to fit observed data into an arbitrary order function

$$Y = B_0 + B_1 \times X + B_2 \times X^2 + \cdots + B_N \times X^N$$

```
out = clab.fit_polynomial(in, order)
```

out	Tuple with the following values:
0	Correlation coefficient between X and Y.
1...order+1	Polynomial coefficients $(B_1, B_2, \ldots, B_{N+1})$.
in	[mandatory] `clab.TimeSeries`.
order	[mandatory] Polynomial order (N).

6.7 Merge Functions

6.7.1 Merge File Into clab.Container

Merge the content of a file into an existing `clab.Container`, provided the data structures are compatible.

If you need to force merging the content of a file into a specific `clab.Container` (regardless of compatibility), use the corresponding member function (refer to section 2.2).

```
out = clab.merge(path)
```

out	If successful, returns the updated clab.Container object. Returns `None` if the merge fails due to incompatibility or error.
path	[mandatory] File path to the data to be merged.

Examples

```
clab.merge(path='c:/user/data.csv')
```

7 Charts Functions

The C-LAB module provides functions to create and manipulate charts.

Each chart has a unique identifier (a 64-bit unsigned integer) which is returned by the plotting functions. Henceforth, this unique identifier will be referred to as *id*. This *id* shall be passed as an argument to all functions that modify a chart.

7.1 Functions to Create 2D Charts

All functions in C-LAB require either a `clab.TimeSeries` object or the name of a variable. When only the variable's name is supplied, C-LAB will attempt to locate an object whose name corresponds to the given string. If both the object and the name are provided, C-LAB will utilize the data from the specified object and label it using the provided variable's name.

7.1.1 Time Series Plot

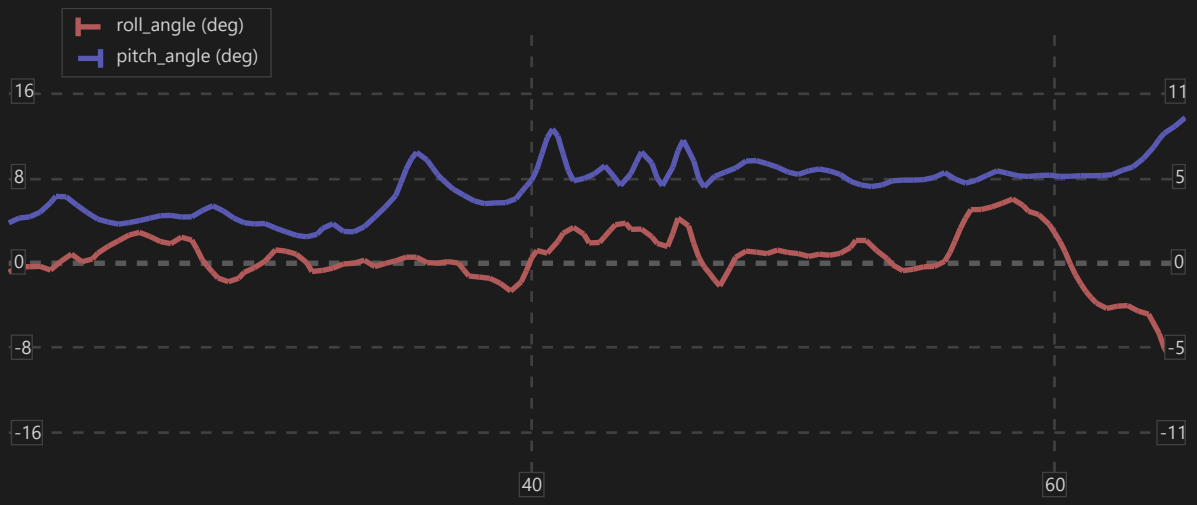
Plot a `clab.TimeSeries` object, displaying the engineering values in the y-axis and time in the x-axis. If the object was created using the built-in loaders, the x-axis will display time in seconds.

```
out = clab.plot(id=0, obj=null, name='', side='left', style='line')
```

out	Chart's unique identifier.
id	[optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
obj	[optional] `clab.TimeSeries` object.
name	[optional] Variable's name.
side	[optional] Y-axis side. It can be either 'left' or 'right'.
style	[optional] Draw style. It can be 'bool', 'dots' or 'line'.

Examples

```
chart_id1 = clab.plot()
clab.plot(id=chart_id1, obj=f['roll_angle (deg)'])
clab.plot(id=chart_id1, obj=f['pitch_angle (deg)'], side='right')
```



7.1.2 Bode Diagram

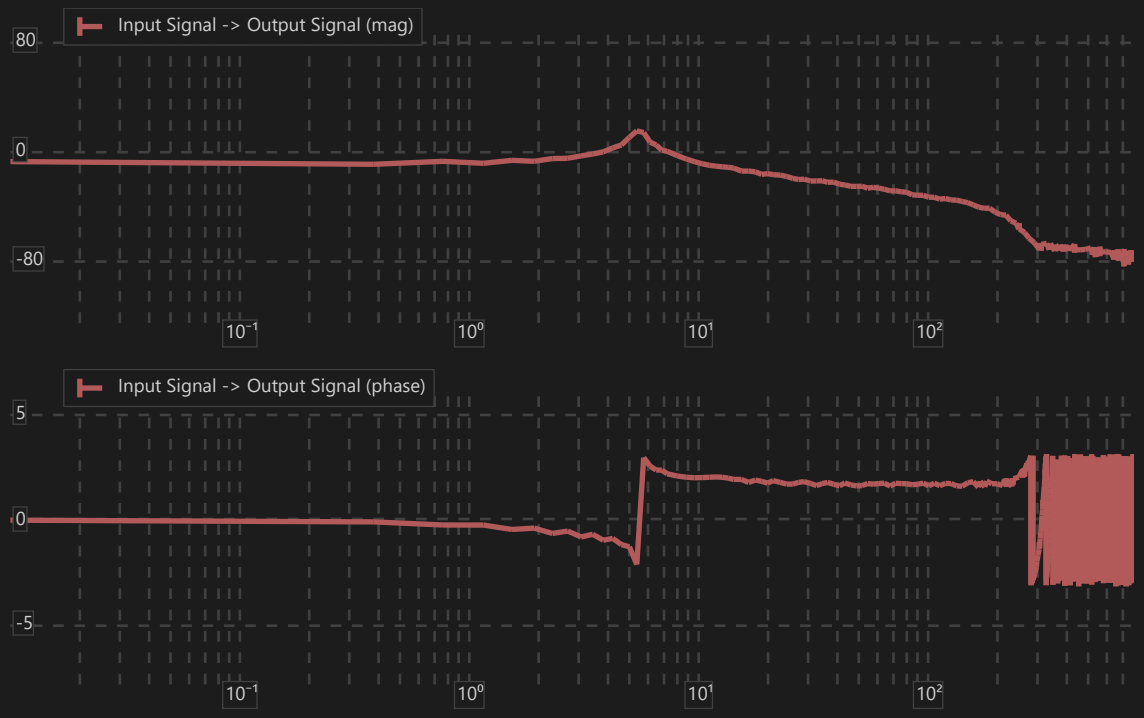
Estimate the transfer function of a system using the FFT of the input and output signals and plot its Bode Diagram. The first signal added to the plot will be used as *input*, and the second as *output*.

```
out = clab.bode(id=0, obj=null, name='')
```

out	Chart's unique identifier.
id	[optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
obj	[optional] `clab.TimeSeries` object.
name	[optional] Variable's name.

Examples

```
chart_id1 = clab.bode()
clab.bode(id=chart_id1, obj=f['Input Signal'])
clab.bode(id=chart_id1, obj=f['Output Signal'])
```



7.1.3 Boolean Plot

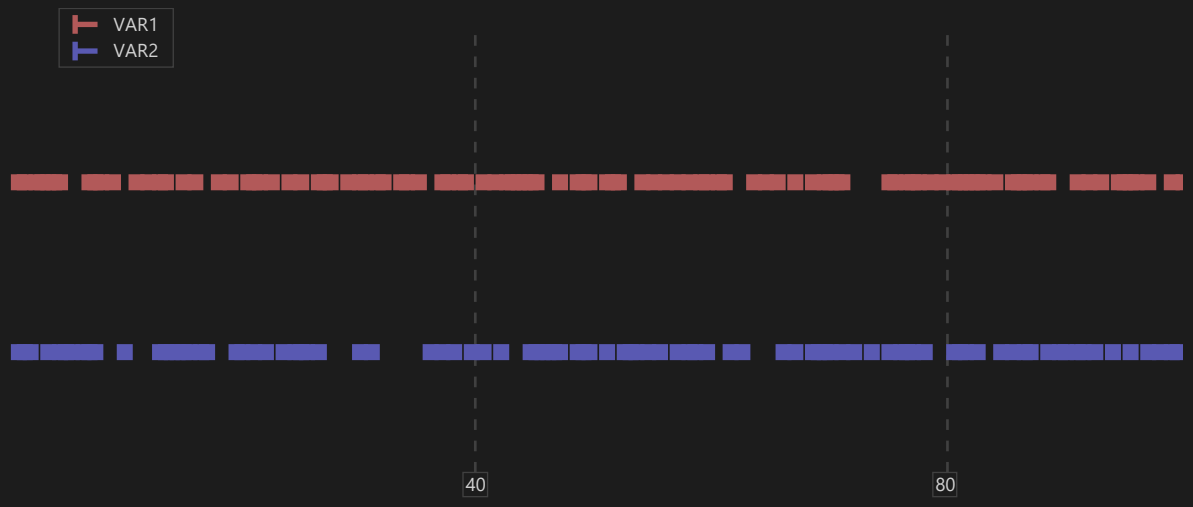
This function plots data points based on boolean values. A dot is plotted for each value different from zero in the data; zero values result in no plot for that specific data point.

```
out = clab.boolplot(id=0, obj=null, name='')
```

out	Chart's unique identifier.
id	[optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
obj	[optional] `clab.TimeSeries` object.
name	[optional] Variable's name.

Examples

```
ts1 = clab.create_random(seed=1234, type='int', ymin=0.0, ymax=1.0, xdiff=0.5)
ts2 = clab.create_random(seed=4321, type='int', ymin=0.0, ymax=1.0, xdiff=0.5)
chart_id1 = clab.boolplot()
clab.boolplot(id=chart_id1, obj=ts1, name='VAR1')
clab.boolplot(id=chart_id1, obj=ts2, name='VAR2')
```



7.1.4 Cross Plot

Plot a `clab.TimeSeries` object against another one, displaying the engineering values in both the x-axis and y-axis.

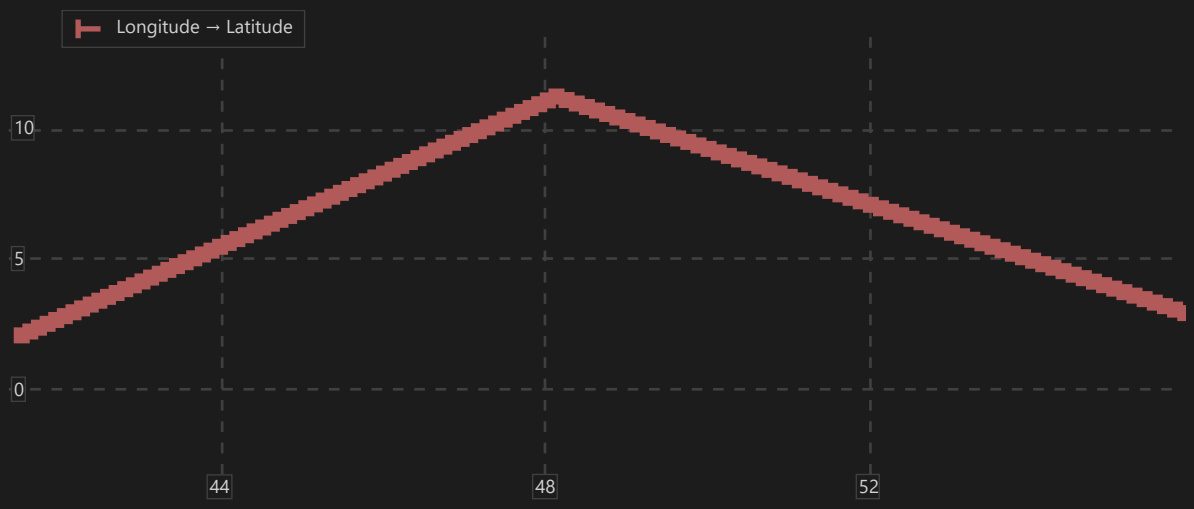
The plotting is done by pairs: the first variable added to the chart will be the x-axis variable and the second, the y-axis. If there is an odd number of variables in the chart, the last pair will be incomplete and will not be displayed.

```
out = clab.crossplot(id=0, obj=null, name='', side='left')
```

- **out** Chart's unique identifier.
- **id** [optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
- **obj** [optional] `clab.TimeSeries` object.
- **name** [optional] Variable's name.
- **side** [optional] Y-axis side. It can be either 'left' or 'right'. For the first variable, this value is ignored.

Examples

```
chart_id1 = clab.crossplot()
clab.crossplot(id=chart_id1, obj=f['Latitude'])
clab.crossplot(id=chart_id1, obj=f['Longitude'])
```



7.1.5 Cross-Correlation Plot

Calculate and plot the cross-correlation (also known as a sliding dot product or sliding inner-product) of a `clab.TimeSeries` object against another one (the reference variable).

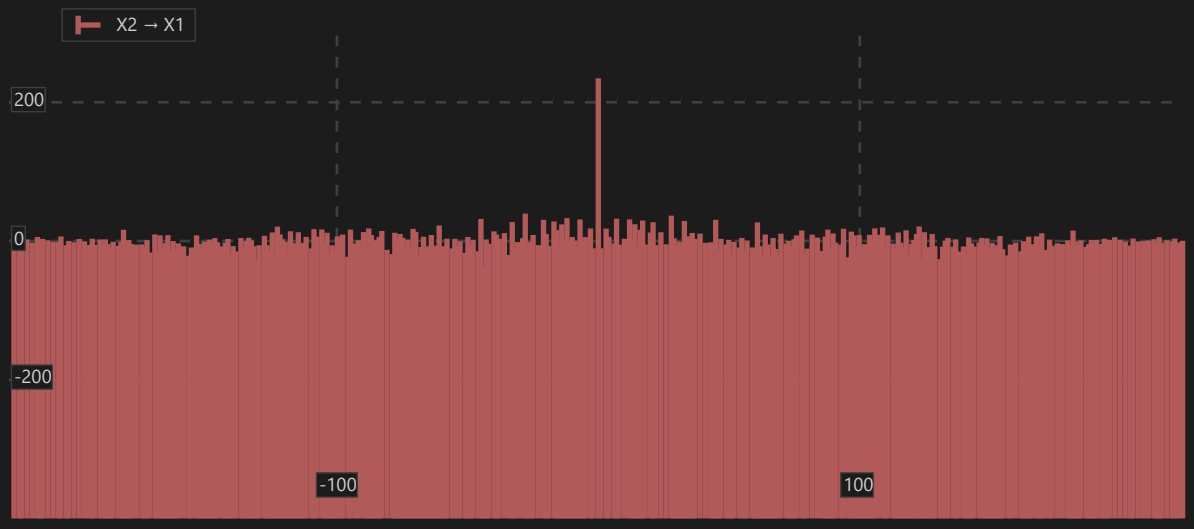
The first variable added to the chart will be the reference variable.

```
out = clab.xcorrplot(id=0, obj=null, name='', side='left')
```

- **out** Chart's unique identifier.
- **id** [optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
- **obj** [optional] `clab.TimeSeries` object.
- **name** [optional] Variable's name.
- **side** [optional] Y-axis side. It can be either 'left' or 'right'. For the first variable, this value is ignored.

Examples

```
chart_id1 = clab.xcorrplot()
clab.xcorrplot(id=chart_id1, obj=f['X1'])
clab.xcorrplot(id=chart_id1, obj=f['X2'])
```



7.1.6 Fast Fourier Transform Plot

Plot the Fast Fourier Transform of a `clab.TimeSeries` object. The x-axis will display frequency in hertz.

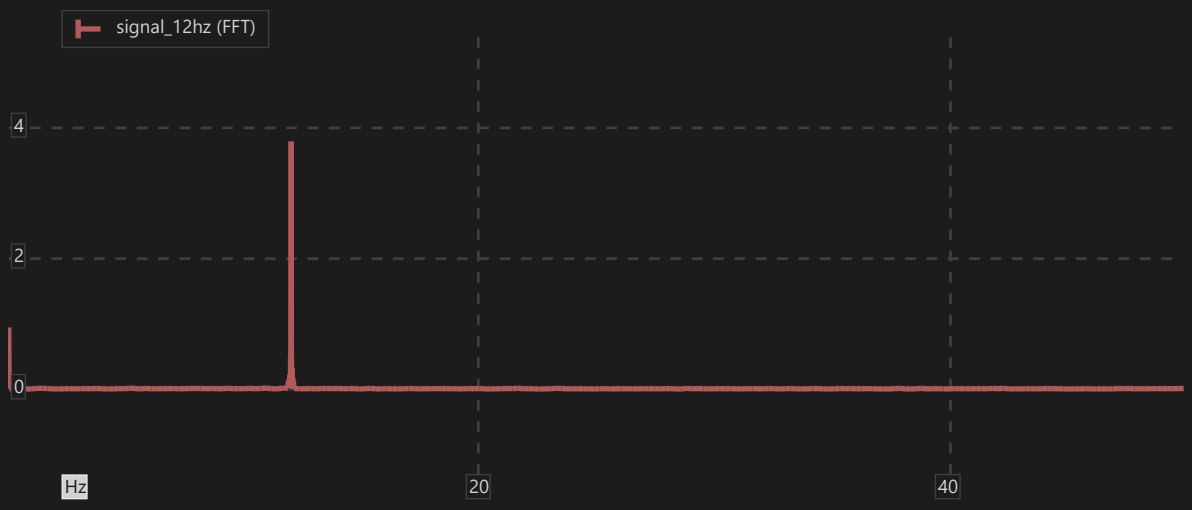
If you only need the FFT data, use the `clab.fftdata` function (refer to 6.2.20).

```
out = clab.fft(id=0, obj=null, name='', side='left')
```

out	Chart's unique identifier.
id	[optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
obj	[optional] `clab.TimeSeries` object.
name	[optional] Variable's name.
side	[optional] Y-axis side. It can be either 'left' or 'right'.

Examples

```
chart_id1 = clab.fft(obj=f['signal_12hz'])
```



7.1.7 Spectrogram Plot

Plot the spectrogram of a `clab.TimeSeries` object. The y-axis will display frequency in hertz.

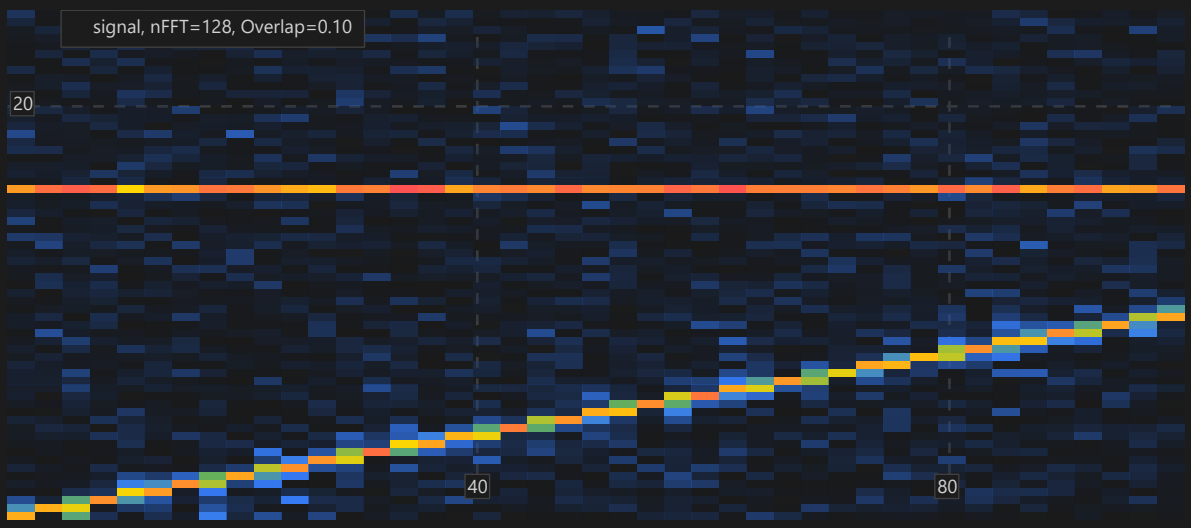
This function computes the Fast Fourier Transform (FFT) of consecutive time windows (spectrogram). The window size is determined by the `nfft` parameter, and the overlap between adjacent windows is controlled by the `overlap` parameter.

```
out = clab.spectrogram(id=0, obj=null, name='', nfft=256, overlap=0.1)
```

out	Chart's unique identifier.
id	[optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
obj	[optional] `clab.TimeSeries` object.
name	[optional] Variable's name.
nfft	[optional] Number of samples in the FFT window.
overlap	[optional] Fraction of overlap between two windows. It can vary between 0.0 and 0.9.

Examples

```
var1 = clab.create_random(xdiff=0.02, dist='normal') * 0.5
var2 = clab.create_sine(xdiff=0.02, freq=16)
var3 = clab.create_chirp(xdiff=0.02)
signal = var1 + var2 + var3
clab.spectrogram(obj=signal, nfft=128)
```



7.1.8 HeatMap Plot

Plot the heatmap between two `clab.TimeSeries` objects.

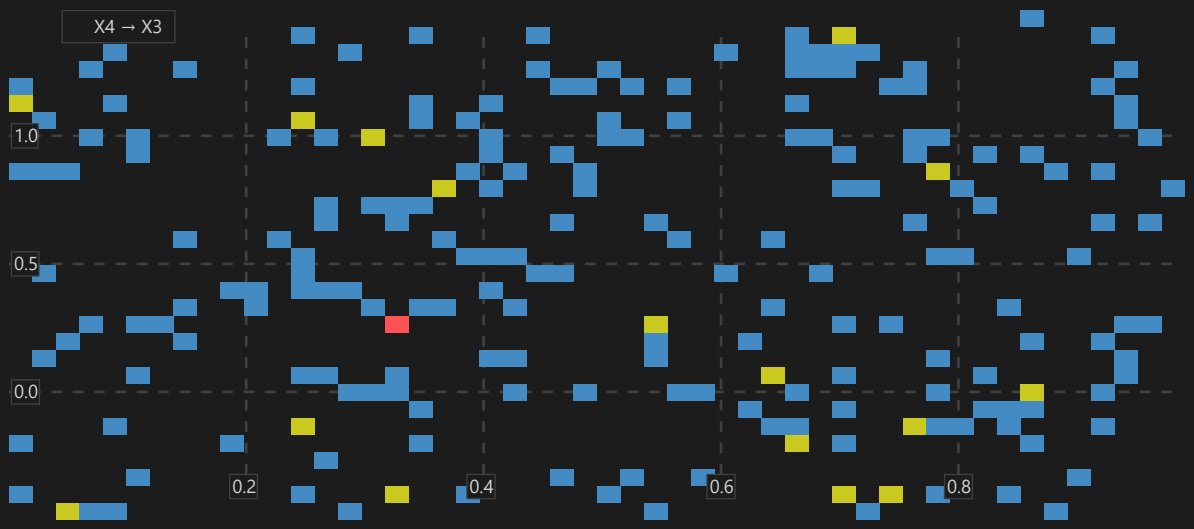
If you only need the underlying data matrix, use the `clab.heatmapdata` function (refer to 6.2.21).

```
out = clab.heatmap(id=0, obj=null, name='')
```

out	Chart's unique identifier.
id	[optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
obj	[optional] `clab.TimeSeries` object.
name	[optional] Variable's name.

Examples

```
chart_id1 = clab.heatmap()
clab.heatmap(id=chart_id1, obj=f['X3'])
clab.heatmap(id=chart_id1, obj=f['X4'])
```



7.1.9 Histogram Plot

Plot the histogram of a `clab.TimeSeries` object.

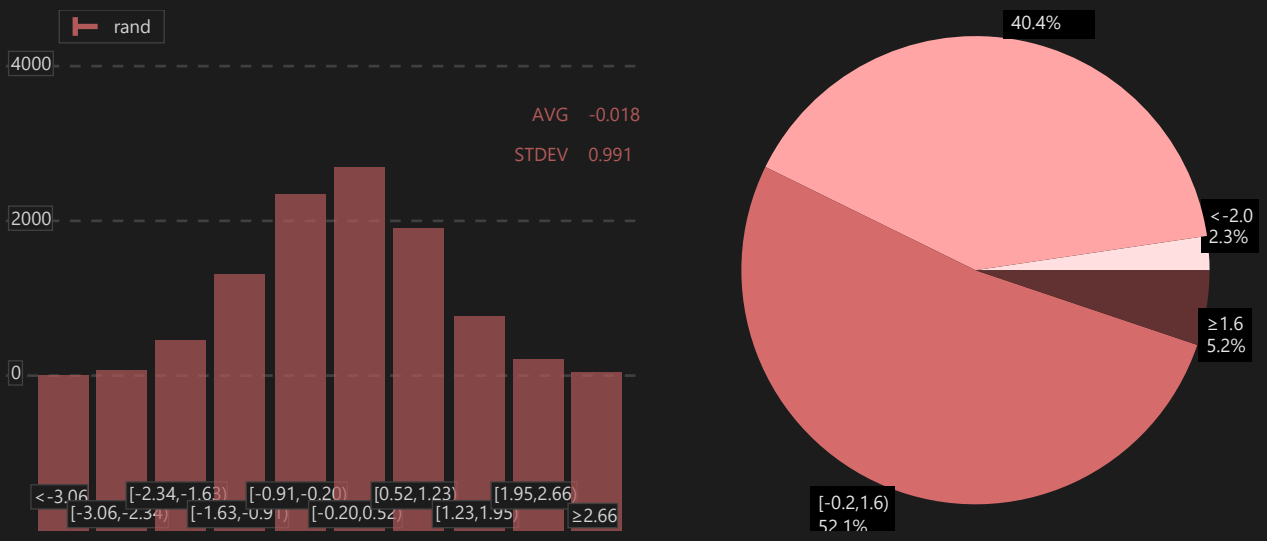
If you only need the histogram data, use the `clab.histdata` function (refer to 6.2.22).

```
out = clab.hist(id=0, obj=null, name='', type='bar', nbins=8, bins=[])
```

- **out** Chart's unique identifier.
- **id** [optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
- **obj** [optional] `clab.TimeSeries` object.
- **name** [optional] Variable's name.
- **type** [optional] Chart's type. It can be either 'bar' or 'pie'.
- **nbins** [optional] Number of bars (for the 'bar' type) or number of slices (for the 'pie' type). Value cannot be less than 3.
- **bins** [optional] List with the bins limits. Its length cannot be less than 2. If this argument is provided along with `nbins`, the latter will be ignored.

Examples

```
rand = clab.create_random(dist='normal')
chart_id1 = clab.hist(obj=rand, nbins=10)
chart_id2 = clab.hist(obj=rand, type='pie', nbins=4)
```



7.1.10 Violin Plot

Plot the violin diagram of a `clab.TimeSeries` object.

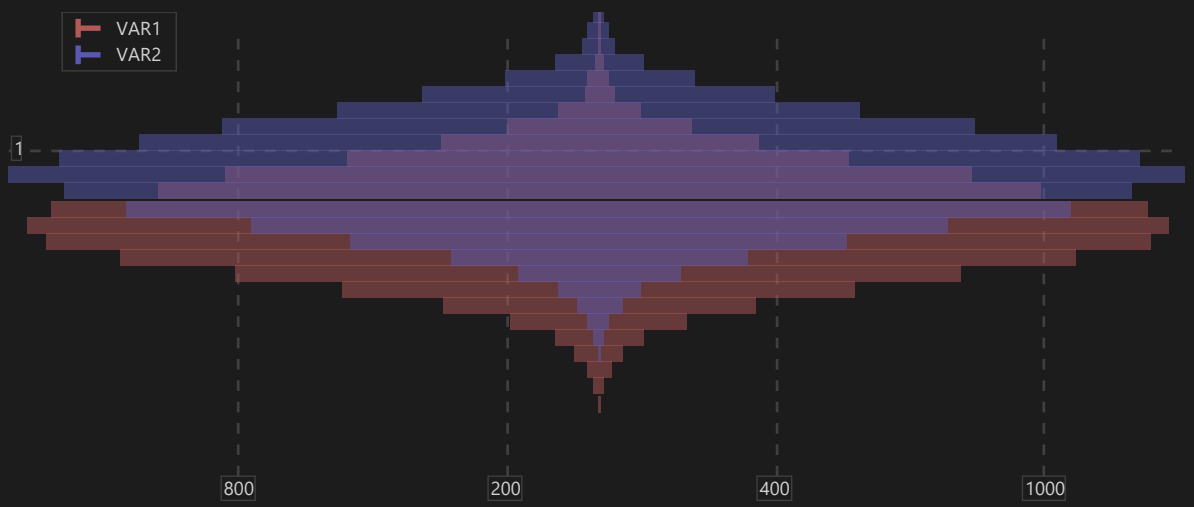
If you only need the violin data, use the `clab.histdata` function (refer to 6.2.22).

```
out = clab.violinplot(id=0, obj=null, name='', nbins=100)
```

- **out** Chart's unique identifier.
- **id** [optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
- **obj** [optional] `clab.TimeSeries` object.
- **name** [optional] Variable's name.
- **nbins** [optional] Number of horizontal bars. Value cannot be less than 10.

Examples

```
var1 = clab.create_random(dist='normal', mean=0.0, stddev=1.0, seed=1234)
var2 = clab.create_random(dist='normal', mean=1.0, stddev=1.0, seed=1234)
chart_id = clab.violinplot()
clab.violinplot(id=chart_id, obj=var1, name='VAR1', nbins=25)
clab.violinplot(id=chart_id, obj=var2, name='VAR2', nbins=25)
```



7.2 Functions to Create 3D Charts

7.2.1 Time Series Plot With 3 Axis

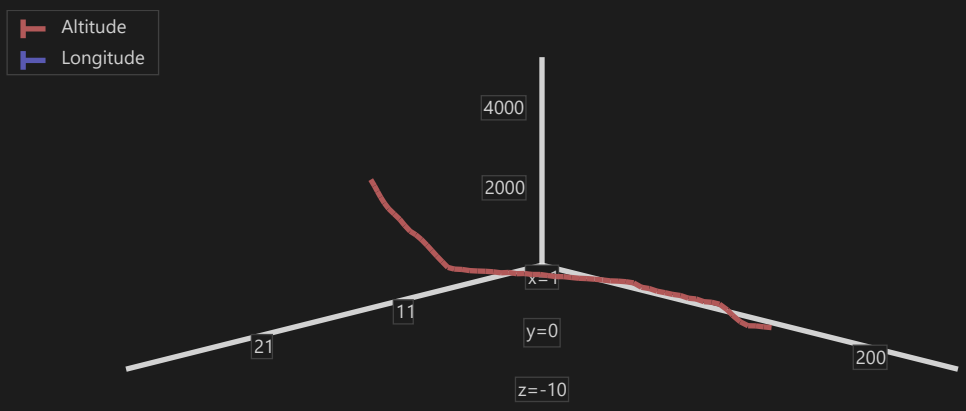
Plot two `clab.TimeSeries` objects to display engineering values on the y-axis and z-axis, with time on the x-axis. If the objects were created using default loaders, the x-axis will display time in seconds. Only two variables can be added to a single chart: one for the y-axis and one for the z-axis.

```
out = clab.plot3d(id=0, obj=null, name='')
```

out	Chart's unique identifier.
id	[optional] Chart's unique identifier. This argument is used to add an object to an existent chart.
obj	[optional] `clab.TimeSeries` object.
name	[optional] Variable's name.

Examples

```
chart_id1 = clab.plot3d()
clab.plot3d(id=chart_id1, name='Altitude')
clab.plot3d(id=chart_id1, name='Longitude')
```



7.2.2 Mesh Plot

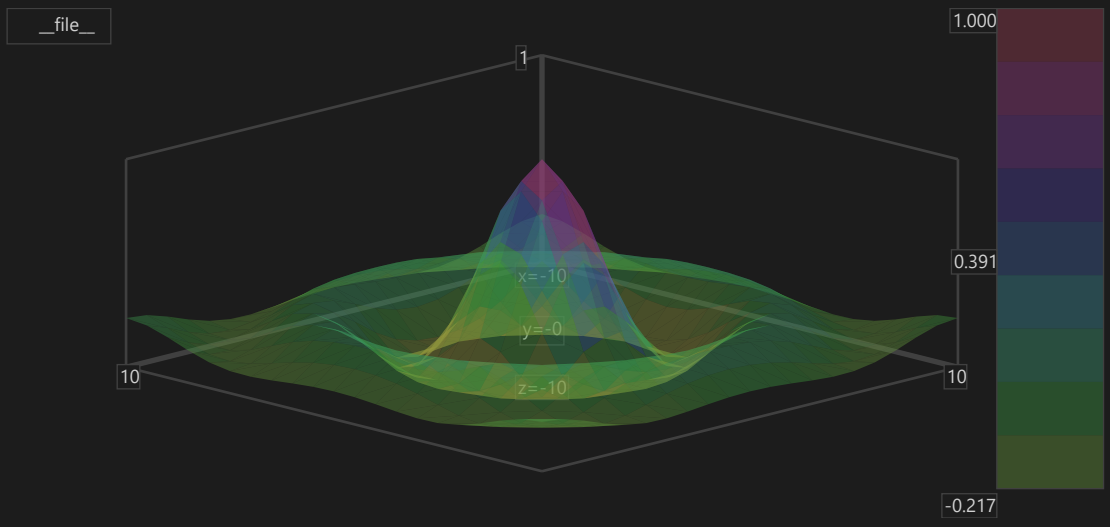
Plot a `clab.Mesh` object in a 3D surface chart.

```
out = clab.mesh(obj)
```

out	Chart's unique identifier.
obj	[mandatory] clab.Mesh object.

Examples

```
mesh1 = clab.create_mesh_sinc(xdiff=1.0, zdiff=1.0)
chart_id1 = clab.mesh(mesh1)
```



7.3 Functions to Create Synoptic Charts

7.3.1 Create a Gauge with Time Series values

Create a chart with a gauge that visualizes a single `clab.TimeSeries` value. The gauge value dynamically updates based on the input time series. To update the gauge value manually, hold **CTRL** while moving the mouse cursor along a standard 2D chart.

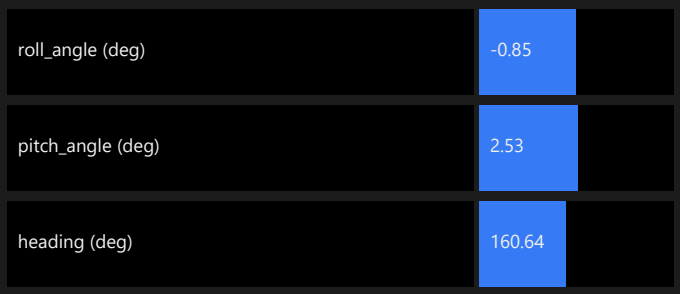
```
out = clab.synoptic(id=0, obj=null, name='', x=-1, y=-1, width=250, min=0.0, max=10.0, red=0x36, green=0x7b, blue=0xf5)
```

- **out** Chart's unique identifier.
- **chart** [optional] Chart's unique identifier. Use this argument to add an object (the gauge) to an existing chart.
- **obj** [optional] The`clab.TimeSeries` object whose value will be displayed on the gauge.
- **name** [optional] The display name for the variable being visualized.
- **x** [optional] Object's position in the x-axis, in pixels. If this value is negative, the object will be automatically positioned in the chart.
- **y** [optional] Object's position in the y-axis, in pixels. If this value is negative, the object will be automatically positioned in the chart.
- **width** [optional] Object's width, in pixels. If this value is negative, the object will be automatically sized.
- **min** [optional] Minimum value in the scale (the gauge's lower bound).
- **max** [optional] Maximum value in the scale (the gauge's upper bound).
- **red** [optional] Red channel intensity (0 to 255).
- **green** [optional] Green channel intensity (0 to 255).
- **blue** [optional] Blue channel intensity (0 to 255).

Examples

```
chart_id1 = clab.synoptic()
clab.synoptic(id=chart_id1, name='roll_angle (deg)', min=-180.0, max=180.0)
clab.synoptic(id=chart_id1, name='pitch_angle (deg)', min=-180.0, max=180.0)
```

```
clab.synoptic(id=chart_id1, name='heading (deg)', min=0.0, max=360.0)
```



7.4 Functions to Export Charts

7.4.1 Save Charts to File

Save charts to file. Currently, only PDF and PNG formats are supported.

```
clab.save(path, id=0, size='')
```

path [mandatory] File's full path. The path must include the desired extension (e.g., .pdf, .png).

id [optional] Chart's or page's unique identifier. If not provided, all pages will be exported to the file.

size [optional] Page size specification. If no size is provided, the system automatically selects a default based on the file extension. Supported values are:

- a3 A3 Paper Size.
- a4 A4 Paper Size.
- a5 A5 Paper Size.
- a6 A6 Paper Size.
- letter Letter Paper Size.
- <width>x<height> Custom size specified in pixels (e.g., 200x100).

Examples

```
clab.save(path='c:/user/out.pdf')
clab.save(path='c:/user/out.png', id=1234, size='256x128')
```

7.5 Functions to Modify/Configure Charts

7.5.1 Set Chart Parameters

Set parameters for a single chart.

```
clab.set_chart(id, ...)
```

id	[mandatory] Chart's unique identifier.
lmax	[optional] Maximum value (in engineering units) of the left axis. Setting this parameter disables auto-zoom for the left y-axis.
lmin	[optional] Minimum value (in engineering units) of the left axis. Setting this parameter disables auto-zoom for the left y-axis.
rmax	[optional] Maximum value (in engineering units) of the right axis. Setting this parameter disables auto-zoom for the right y-axis.
rmin	[optional] Minimum value (in engineering units) of the right axis. Setting this parameter disables auto-zoom for the right y-axis.
ltick	[optional] Tick step value (in engineering units) for the left axis. Setting this parameter disables auto-tick for the left y-axis.
rtick	[optional] Tick step value (in engineering units) for the right axis. Setting this parameter disables auto-tick for the right y-axis.
fit	[optional] Show fitted curve information. Options are `'exp'`, `'linear'`, or `'poly'`.
xlabel	[optional] Text to be displayed near the x-axis ticks.

Examples

```
clab.set_chart(id=1234, lmax=10, lmin=0)
clab.set_chart(id=1234, rtick=2.5)
clab.set_chart(id=1234, fit='linear')
```

7.6 Functions to Create Objects on 2D Charts

7.6.1 Create a Circle

Create a circle in a single chart.

```
clab.circle(chart, xcenter, ycenter, diameter, red, green, blue)
```

chart	[mandatory] Chart's unique identifier.
xcenter	[mandatory] Circle center coordinate in the x-axis. Units match the target chart's x-axis.
ycenter	[mandatory] Circle center coordinate in the y-axis. Units match the target chart's y-axis.
diameter	[mandatory] Circle diameter. Units match the target chart's x-axis units.
red	[optional] Red channel intensity (0 to 255).
green	[optional] Green channel intensity (0 to 255).
blue	[optional] Blue channel intensity (0 to 255).

Examples

```
clab.circle(chart=1234, xcenter=10, ycenter=20, diameter=5, red=128, green=128, blue=128)
```

7.6.2 Create a DataTip

Create a datatip in a single chart.

```
clab.datatip(chart, name, x, red, green, blue)
```

chart	[mandatory] Chart's unique identifier.
name	[mandatory] Parameter's name.
x	[mandatory] Position in the x-axis (in the target chart's coordinate units).
red	[optional] Red channel intensity (0 to 255).
green	[optional] Green channel intensity (0 to 255).
blue	[optional] Blue channel intensity (0 to 255).

Examples

```
clab.datatip(chart=1234, name='value1', x=10, red=128, green=128, blue=128)
```

7.6.3 Create a Flag

Create a vertical flag across all charts (if only x is provided) or a horizontal flag in a single chart (if only y is provided).

```
clab.flag(chart, x, y, text, red, green, blue)
```

chart	[optional] Chart's unique identifier.
x	[optional] Position in the x-axis. If y is omitted, it creates a vertical flag.
y	[optional] Position in the y-axis. If x is omitted, it creates a horizontal flag.
text	[optional] Flag's text.
red	[optional] Red channel intensity (0 to 255).
green	[optional] Green channel intensity (0 to 255).
blue	[optional] Blue channel intensity (0 to 255).

Examples

```
clab.flag(x=20.0, text='event', red=128, green=128, blue=128)
clab.flag(chart=1234, y=100.0, text='limit')
```

7.6.4 Create Marks in All Charts

Create marks in all charts.

```
clab.mark(x, width, text, red, green, blue)
```

x	[mandatory] Starting position in the x-axis (in the target chart's coordinate units).
width	[mandatory] Width of the mark (in the target chart's x-axis units).
text	[optional] Text to display on the mark.
red	[optional] Red channel intensity (0 to 255).
green	[optional] Green channel intensity (0 to 255).
blue	[optional] Blue channel intensity (0 to 255).

Examples

```
clab.mark(x=10, width=12.5, text='event', red=128, green=128, blue=128)
```

7.6.5 Create a Rectangle

Create a rectangle in a single chart.

```
clab.rectangle(chart, xcenter, ycenter, width, height, red, green, blue)
```

chart	[mandatory] Chart's unique identifier.
xcenter	[mandatory] Rectangle center coordinate in the x-axis (in the target chart's coordinate units).
ycenter	[mandatory] Rectangle center coordinate in the y-axis (in the target chart's coordinate units).
width	[mandatory] Rectangle width (in the target chart's x-axis units).
height	[mandatory] Rectangle height (in the target chart's y-axis units).
red	[optional] Red channel intensity (0 to 255).
green	[optional] Green channel intensity (0 to 255).
blue	[optional] Blue channel intensity (0 to 255).

Examples

```
clab.rectangle(chart=1234, xcenter=10, ycenter=20, width=5, height=5, red=128, green=128,
blue=128)
```

7.6.6 Create a Text

Create a text label in a single chart.

```
clab.text(chart, x, y, text, red, green, blue)
```

chart	[mandatory] Chart's unique identifier.
x	[mandatory] Text coordinate in the x-axis (in the target chart's coordinate units).
y	[mandatory] Text coordinate in the y-axis (in the target chart's coordinate units).
text	[mandatory] Text to be displayed.
red	[optional] Red channel intensity (0 to 255).
green	[optional] Green channel intensity (0 to 255).

blue [optional] Blue channel intensity (0 to 255).

Examples

```
clab.text(chart=1234, x=10, y=20, text='this is a text', red=128, green=128, blue=128)
```

7.6.7 Setup Legend

Setup a legend for a single chart.

```
clab.legend(chart, x, y, visible, full)
```

chart [mandatory] Chart's unique identifier.

x [optional] Position (in pixels) on the chart for the legend.

y [optional] Position (in pixels) on the chart for the legend.

visible [optional] If `False`, the legend will not be displayed.

full [optional] If `True`, the legend will display full variable names.

Examples

```
clab.legend(chart=1234, visible=False)
clab.legend(chart=1234, x=50)
```

8 Page Functions

8.1 Create Page

Creates a new empty page.

```
out = clab.new_page(id=0, name)
```

- **out** Unique identifier of the created page.
- **id** [optional] Unique identifier to assign to the page. If not provided, C-LAB **MUST** generate a unique identifier.
- **name** [mandatory] Page name. This value **MUST** be displayed in the page tab within the user interface.

Examples

```
clab.new_page(id=1234, name='page 1')
```

8.2 Delete Page

Deletes an existing page.

```
clab.del_page(id=0, name="")
```

- **id** [optional] Unique identifier of the page to delete.
- **name** [optional] Name of the page to delete.

  At least one of `id` or `name` **MUST** be provided. If both are provided, `id` will take precedence.

Examples

```
clab.del_page(id=1234)
```

8.3 Set Page Properties

Modifies properties of an existing page.

```
clab.set_page(id=0, cols=-1)
```

- **id** [mandatory] Unique identifier of the page.
- **cols** [optional] Number of columns in the page layout. If the value is negative, the column count will not be modified.

  The page identified by `id` **MUST** exist. If `cols` is provided and non-negative, the layout will be updated accordingly.

Examples

```
clab.set_page(id=1234, cols=3)
```

9 Miscellaneous Functions

This section defines functions for interacting with global front-end state, including charts, pages, and navigation controls.

9.1 Set Global Parameters

Sets global parameters affecting all charts and pages.

`clab.set(key, args...)`

- `key` [mandatory] Identifier specifying the parameter to modify.
- `args...` [mandatory] Values assigned to the parameter identified by `key`. Argument types and count depend on the selected key.

- `'tab'` Set the active tab. Argument must be in the range 1–4.
- `'globaltime'` Set the current global time.

  Used by time-dependent features such as `clab.synoptic` (see 7.3.1).
- `'timelimits'` Set the global time limits.
- `'timeformat'` Set the time display format used by charts. Argument is of the following values:
  - `'default'` Display raw time values as loaded (no formatting).
  - `'hhmmss'` Interpret values as seconds and format as HH:MM:SS (hours wrap at 24).
  - `'unixtime'` Interpret values as seconds since 00:00:00 UTC, January 1, 1970.
  - `'gpstime'` Interpret values as seconds since 00:00:00 UTC, January 5, 1980 (no leap second adjustment).
- `'zoom'` Equivalent to `'timelimits'`.

Examples

```
clab.set('timeformat', 'hhmmss')
clab.set('timelimits', 10.0, 20.0)
```

9.2 Close Charts and Pages

Closes charts or pages by identifier.

```
clab.close(type, id=0)
```

- `type` [mandatory] Identifier specifying the target to close. Valid values:
  - `'pages'` Close all pages.
  - `'page'` Close a single page.
  - `'chart'` Close a single chart.
- `id` [optional] Unique identifier of the target. Required when `type` is `page` or `chart`. Ignored when `type` is `pages`.

Examples

```
clab.close('pages')
clab.close('chart', id=1234)
```

9.3 Reset Parameters

Resets global parameters affecting charts and pages.

```
clab.reset(key)
```

- `key` [mandatory] Identifier specifying the parameter to reset. Valid values:
  - `'bookmarks'` Remove all bookmarks.
  - `'flags'` Clear the cache of *flags* objects (see 7.6.3), preventing reuse in newly created charts.
  - `'marks'` Clear the cache of *marks* objects (see 7.6.4), preventing reuse in newly created charts.
  - `'zoom'` Reset zoom levels for all pages.

Examples

```
clab.reset('zoom')
```

9.4 Create Bookmark

Creates a bookmark representing a time interval in the front-end interface.

Bookmarks allow navigation between predefined time ranges and can also be created via the horizontal scrollbar context menu.

```
clab.bookmark(tini, tend)
```

- `tini` [mandatory] Start time of the bookmark.
- `tend` [mandatory] End time of the bookmark.

Examples

```
clab.bookmark(tini=1.0, tend=5.0)
```

10 COM Server Functions

C-LAB provides support for COM servers (IDispatch-compatible) through an out-of-process architecture. It exposes an API that allows external applications to interact with C-LAB from a separate process.

The examples in the following subsections are written in Python; however, the same principles can be applied to other programming languages.

NOTE: This feature is currently experimental and may be subject to change in future releases.

10.1 Create and Update C-LAB Objects

10.1.1 Create `clab.TimeSeries`

Creates a `clab.TimeSeries` object from two arrays.

```
HRESULT CreateTimeSeries([in] BSTR name, [in] SAFEARRAY(VARIANT) xvalues, [in] SAFEARRAY(VARIANT
yvalues)
```

name	Name of the object.
xvalues	X-axis values. Must have the same length as `yvalues`.
yvalues	Y-axis values. Must have the same length as `xvalues`.

Examples

```
import win32com.client
obj = win32com.client.Dispatch('ClabApp.ClabObject')
obj.CreateTimeSeries('parameter1', [1,2,3,4,5,6,7,8], [10,20,30,40,50,60,70,80])
```

10.1.2 Update `clab.TimeSeries`

Updates a `clab.TimeSeries` object. If the object does not exist, it is created.

The length of an existing object is not modified:

- If input arrays are shorter, remaining values are preserved.
- If input arrays are longer, excess elements are ignored.

Passing an empty array leaves the corresponding data unchanged.

```
HRESULT UpdateTimeSeries([in] BSTR name, [in] SAFEARRAY(VARIANT) xvalues, [in] SAFEARRAY(VARIANT
yvalues)
```

name	Name of the object.
xvalues	X-axis values. Must match the length of `yvalues` or be empty.
yvalues	Y-axis values. Must match the length of `xvalues` or be empty.

Examples

```
import win32com.client
obj = win32com.client.Dispatch('ClabApp.ClabObject')
obj.UpdateTimeSeries('parameter1', [1,2,3,4,5,6,7,8], [10,20,30,40,50,60,70,80])
```

10.2 Functions to Create 2D Charts

10.2.1 Time Series Plot

Plots a `clab.TimeSeries` object. The x-axis represents time, and the y-axis represents the corresponding values. If the object was created using built-in loaders, time is expressed in seconds.

See Section 7.1.1 for the Python API equivalent.

```
HRESULT Plot([in] ULONGLONG chartId, [in] BSTR name, [in] BSTR side, [in] BSTR style, [out, retval] ULONGLONG* result)
```

- chartId: Unique chart identifier. If nonzero, the object is added to the specified chart. If 0, a new chart is created.
- name: Name of the object.
- side: Y-axis side. Valid values: 'left', 'right'.
- style: Rendering style. Valid values: 'bool', 'dots', 'line'.
- result: Returns the chart identifier used for the operation. In scripting languages (e.g., Python, VBScript), this value is returned directly by the function.

Examples

```
import win32com.client
obj = win32com.client.Dispatch('ClabApp.ClabObject')
obj.Plot(1234, 'parameter1', 'left', 'line') # returns 1234
```

10.3 Functions to Export Charts

10.3.1 Save Charts to File

Exports charts to a file. Supported formats: PDF and PNG.

See Section 7.4.1 for the Python API equivalent.

```
HRESULT Save([in] BSTR path, [in] ULONGLONG id, [in] BSTR size)
```

- path: Full file path, including extension.
- id: Unique identifier of the chart or page. If 0, all pages are exported.

size Output page size. Supported values:

a3 A3 paper size.

a4 A4 paper size.

a5 A5 paper size.

a6 A6 paper size.

letter Letter paper size.

<width>x<height> Custom size in pixels (e.g., 200x100).

Examples

```
import win32com.client
obj = win32com.client.Dispatch('ClabApp.ClabObject')
obj.Save('c:\file.pdf', 0, '200x100')
```

11 Third-Party Libraries

11.1 Apache Arrow

Source code available at https://github.com/apache/arrow.

11.1.1 License

```
Apache Arrow
Copyright 2016-2024 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

This product includes software from the SFrame project (BSD, 3-clause).
* Copyright (C) 2015 Dato, Inc.
* Copyright (c) 2009 Carnegie Mellon University.

This product includes software from the Feather project (Apache 2.0)
https://github.com/wesm/feather

This product includes software from the DyND project (BSD 2-clause)
https://github.com/libdynd

This product includes software from the LLVM project
 * distributed under the University of Illinois Open Source

This product includes software from the google-lint project
 * Copyright (c) 2009 Google Inc. All rights reserved.

This product includes software from the mman-win32 project
 * Copyright https://code.google.com/p/mman-win32/
 * Licensed under the MIT License;

This product includes software from the LevelDB project
 * Copyright (c) 2011 The LevelDB Authors. All rights reserved.
 * Use of this source code is governed by a BSD-style license that can be
 * Moved from Kudu http://github.com/cloudera/kudu

This product includes software from the CMake project
 * Copyright 2001-2009 Kitware, Inc.
 * Copyright 2012-2014 Continuum Analytics, Inc.
 * All rights reserved.

This product includes software from https://github.com/matthew-brett/multibuild (BSD 2-clause)
 * Copyright (c) 2013-2016, Matt Terry and Matthew Brett; all rights reserved.

This product includes software from the Ibis project (Apache 2.0)
 * Copyright (c) 2015 Cloudera, Inc.
```

```
 * https://github.com/cloudera/ibis

This product includes software from Dremio (Apache 2.0)
  * Copyright (C) 2017-2018 Dremio Corporation
  * https://github.com/dremio/dremio-oss

This product includes software from Google Guava (Apache 2.0)
  * Copyright (C) 2007 The Guava Authors
  * https://github.com/google/guava

This product include software from CMake (BSD 3-Clause)
  * CMake - Cross Platform Makefile Generator
  * Copyright 2000-2019 Kitware, Inc. and Contributors

The web site includes files generated by Jekyll.

--------------------------------------------------------------------------------

This product includes code from Apache Kudu, which includes the following in
its NOTICE file:

  Apache Kudu
  Copyright 2016 The Apache Software Foundation

  This product includes software developed at
  The Apache Software Foundation (http://www.apache.org/).

  Portions of this software were developed at
  Cloudera, Inc (http://www.cloudera.com/).

--------------------------------------------------------------------------------

This product includes code from Apache ORC, which includes the following in
its NOTICE file:

  Apache ORC
  Copyright 2013-2019 The Apache Software Foundation

  This product includes software developed by The Apache Software
  Foundation (http://www.apache.org/).

  This product includes software developed by Hewlett-Packard:
  (c) Copyright [2014-2015] Hewlett-Packard Development Company, L.P
```

11.2 Cairo Graphics

Source code available at https://gitlab.freedesktop.org/cairo/cairo.git.

Distributed under the Mozilla Public License (MPL) version 1.1.

11.3 CURL

Source code available at https://github.com/curl/curl.

Copyright (c) Daniel Stenberg (daniel@haxx.se).

11.4 Dear ImGui

Source code available at https://github.com/ocornut/imgui.

11.4.1 License

```
The MIT License (MIT)

Copyright (c) 2014-2026 Omar Cornut

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

11.5 FreeType

Source code available at https://gitlab.freedesktop.org/freetype/freetype.

Portions of this software are copyright ©2024 The FreeType Project (www.freetype.org). All rights reserved.

11.6 GLFW

Source code available at https://github.com/glfw/glfw.

Distributed under the zlib License.

11.7 HDF5

Source code available at https://github.com/HDFGroup/hdf5.

11.7.1 License

```
Copyright Notice and License Terms for
HDF5 (Hierarchical Data Format 5) Software Library and Utilities
-----------------------------------------------------------------------------

HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 2006 by The HDF Group.

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 1998-2006 by The Board of Trustees of the University of Illinois.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted for any purpose (including commercial purposes)
provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions, and the following disclaimer in the documentation
   and/or materials provided with the distribution.

3. Neither the name of The HDF Group, the name of the University, nor the
   name of any Contributor may be used to endorse or promote products derived
   from this software without specific prior written permission from
   The HDF Group, the University, or the Contributor, respectively.

DISCLAIMER:
THIS SOFTWARE IS PROVIDED BY THE HDF GROUP AND THE CONTRIBUTORS
"AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. IN NO
```

EVENT SHALL THE HDF GROUP OR THE CONTRIBUTORS BE LIABLE FOR ANY DAMAGES SUFFERED BY THE USERS ARISING OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code ("Enhancements") to anyone; however, if you choose to make your Enhancements available either publicly, or directly to The HDF Group, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.

Limited portions of HDF5 were developed by Lawrence Berkeley National Laboratory (LBNL). LBNL's Copyright Notice and Licensing Terms can be found here: COPYING_LBNL_HDF5 file in this directory or at https://raw.githubusercontent.com/hdfgroup/hdf5/develop/COPYING_LBNL_HDF5.

Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois, Fortner Software, Unidata Program Center (netCDF), The Independent JPEG Group (JPEG), Jean-loup Gailly and Mark Adler (gzip), and Digital Equipment Corporation (DEC).

Portions of HDF5 were developed with support from the Lawrence Berkeley National Laboratory (LBNL) and the United States Department of Energy under Prime Contract No. DE-AC02-05CH11231.

Portions of HDF5 were developed with support from Lawrence Livermore National Laboratory and the United States Department of Energy under Prime Contract No. DE-AC52-07NA27344.

Portions of HDF5 were developed with support from the University of California, Lawrence Livermore National Laboratory (UC LLNL).
The following statement applies to those portions of the product and must be retained in any redistribution of source code, binaries, documentation, and/or accompanying materials:

This work was partially produced at the University of California,
Lawrence Livermore National Laboratory (UC LLNL) under contract
no. W-7405-ENG-48 (Contract 48) between the U.S. Department of Energy
(DOE) and The Regents of the University of California (University)
for the operation of UC LLNL.

DISCLAIMER:
THIS WORK WAS PREPARED AS AN ACCOUNT OF WORK SPONSORED BY AN AGENCY OF
THE UNITED STATES GOVERNMENT. NEITHER THE UNITED STATES GOVERNMENT NOR
THE UNIVERSITY OF CALIFORNIA NOR ANY OF THEIR EMPLOYEES, MAKES ANY
WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY LIABILITY OR RESPONSIBILITY
FOR THE ACCURACY, COMPLETENESS, OR USEFULNESS OF ANY INFORMATION,
APPARATUS, PRODUCT, OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE
WOULD NOT INFRINGE PRIVATELY- OWNED RIGHTS. REFERENCE HEREIN TO ANY
SPECIFIC COMMERCIAL PRODUCTS, PROCESS, OR SERVICE BY TRADE NAME,
TRADEMARK, MANUFACTURER, OR OTHERWISE, DOES NOT NECESSARILY CONSTITUTE
OR IMPLY ITS ENDORSEMENT, RECOMMENDATION, OR FAVORING BY THE UNITED
STATES GOVERNMENT OR THE UNIVERSITY OF CALIFORNIA. THE VIEWS AND
OPINIONS OF AUTHORS EXPRESSED HEREIN DO NOT NECESSARILY STATE OR REFLECT
THOSE OF THE UNITED STATES GOVERNMENT OR THE UNIVERSITY OF CALIFORNIA,
AND SHALL NOT BE USED FOR ADVERTISING OR PRODUCT ENDORSEMENT PURPOSES.

11.8 LZMA SDK

Source code available at https://7-zip.org/sdk.html.

LZMA SDK is placed in the public domain.

11.9 matio

Source code available at https://github.com/tbeu/matio.

11.9.1 License

Copyright (c) 2015-2025, The matio contributors
Copyright (c) 2011-2014, Christopher C. Hulbert
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11.10 pugixml

Source code available at https://github.com/zeux/pugixml.

11.10.1 License

MIT License

Copyright (c) 2006-2025 Arseny Kapoulkine

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

11.11 zlib

Source code available at https://github.com/madler/zlib.git.

11.12 zlib-ng

Source code available at https://github.com/zlib-ng/zlib-ng.