# Caterva

***Release 0.4.0***

**Mar 30, 2021**

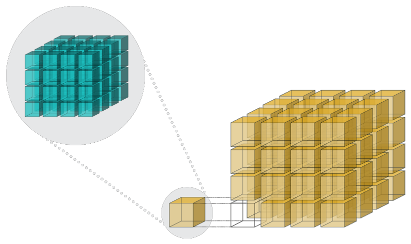
## Getting Started

Caterva is an open source C library that allows to store large multidimensional, chunked, compressed datasets. Data can be stored either in-memory or on-disk, but the API to handle both versions is the same.

CHAPTER 1

# Package overview

Caterva is a container for multidimensional data that is specially designed to read, in an incredibly efficient way, slices of it. To achieve this, a new chunking-based data layout has been created.



Like other libraries like zarr, HDF5 or TileDB, Caterva stores the data into multidimensional chunks (yellow cubes). These chunks can then be read individually, improving performance when reading slices of the dataset. But also, Caterva introduces a new level of chunking. Within each chunk, the data is re-partitioned into smaller multidimensional sets called blocks (green cubes). In this way, Caterva can read blocks individually (and also in parallel) instead of chunks.

These partition levels allow to access data efficiently with a larger set of data access patterns. This is due to obtain the desired slice, instead of reading the data using the chunks, data are obtained using the blocks.

## 1.1 Blosc

In Caterva the compression is handled transparently for the user by leveraging the Blosc2 library. Blosc is an extremely fast compressor specially designed for binary data. It uses the blocking technique to reduce activity on the memory bus as much as possible. It also leverages SIMD (SSE2, AVX2 for Intel, NEON for ARM, Altivec for Power) and multi-threading capabilities present in multi-core processors so as to accelerate the compression/decompression process to a maximum.

Being able to store in an in-memory data container does not mean that data cannot be persisted. It is critical to find a way to store and retrieve data efficiently. Also, it is important to adopt open formats for reducing the maintenance burden and facilitate its adoption more quickly. Blosc2 brings such an efficient and open format for persistency.

An aditional feature that introduces Blosc2 is the concept of metalayers. They are small metadata for informing about the kind of data that is stored on a Blosc2 container. They are handy for defining layers with different specs: data types, geo-spatial…

### 1.1.1 Caterva metalayer

Caterva is created by specifying a metalayer on top of a Blosc2 container for storing multidimensional information. This metalayer can be modified so that the shapes can be updated (e.g. an array can grow or shrink). Specifically, Caterva metalayer follows the msgpack format:

```
|-0-|-1-|-2-|-3-|~~~~~~~~~~~~~~~~|---|~~~~~~~~~~~~~~~~|---|~~~~~~~~~~~~~~~~|
| 9X| n | n | 9X| shape          | 9X| chunkshape     | 9X| blockshape     |
|---|---|---|---|~~~~~~~~~~~~~~~~|---|~~~~~~~~~~~~~~~~|---|~~~~~~~~~~~~~~~~|
  ^   ^   ^   ^                    ^                    ^
  |   |   |   |                    |                    |
  |   |   |   |                    |                    +--[msgpack] positive fixnum␣
↪for n
  |   |   |   |                    +--[msgpack] positive fixnum for n
  |   |   |   +--[msgpack] fixarray with X=nd elements
  |   |   +--[msgpack] positive fixnum for the number of dimensions (n, up to 127)
  |   +--[msgpack] positive fixnum for the metalayer format version (up to 127)
  +---[msgpack] fixarray with X=5 elements
```

In this format, the shape section is meant to store the actual shape info:

```
|---|--8 bytes---|---|--8 bytes---|~~~~~|---|--8 bytes---|
| d3| first_dim  | d3| second_dim | ... | d3| nth_dim    |
|---|------------|---|------------|~~~~~|---|------------|
  ^                ^                        ^
  |                |                        |
  |                |                        +--[msgpack] int64
  |                +--[msgpack] int64
  +--[msgpack] int64
```

Next, the chunkshape section is meant to store the actual chunk shape info:

```
|---|--4 bytes---|---|--4 bytes---|~~~~~|---|--4 bytes---|
| d2| first_dim  | d2| second_dim | ... | d2| nth_dim    |
|---|------------|---|------------|~~~~~|---|------------|
  ^                ^                        ^
  |                |                        |
  |                |                        +--[msgpack] int32
  |                +--[msgpack] int32
  +--[msgpack] int32
```

Finally, the blockshape section is meant to store the actual block shape info:

```
|---|--4 bytes---|---|--4 bytes---|~~~~~|---|--4 bytes---|
| d2| first_dim  | d2| second_dim | ... | d2| nth_dim    |
|---|------------|---|------------|~~~~~|---|------------|
  ^                ^                        ^
  |                |                        |
  |                |                        +--[msgpack] int32
  |                +--[msgpack] int32
  +--[msgpack] int32
```

CHAPTER 2

# Installation

Caterva can be built, tested and installed using CMake. The following procedure describes a typical CMake build. In order to install Caterva, you need to have the library c-blosc2 builded or installed.

## 2.1 Unix

1. Create the build directory inside the sources and move into it:

```
cd caterva-sources
mkdir build/
cd build/
```

2. Now run CMake configuration and, if necessary, specify the directory where the Blosc build is and the directory where the Blosc headers are:

```
cmake -DBLOSC_DIR='blosc_build_dir' -DBLOSC_INCLUDE='blosc_headers_dir' -DCMAKE_
↪BUILD_TYPE='Debug/Release' ..
```

3. Build and test Caterva:

```
cmake --build .
ctest
```

4. If desired, install Caterva:

```
cmake --build . --target install
```

## 2.2 Windows

1. Create the build directory inside the sources and move into it:

```
cd caterva-sources
mkdir build/
cd build/
```

2. Now run CMake configuration and, if necessary, specify the directory where the Blosc build is and the directory where the Blosc headers are:

```
cmake -DBLOSC_DIR='blosc_build_dir' -DBLOSC_INCLUDE='blosc_headers_dir' ..
```

3. Build and test Caterva:

```
cmake --build . --config 'Debug/Release'
ctest --build-config 'Debug/Release'
```

4. If desired, install Caterva:

```
cmake --build . --target install --config 'Debug/Release'
```

That's all folks!

# CHAPTER 3

# C API

The entire development of the Caterva core has been done with the C programming language. This choice allows Caterva to obtain:

- High performance in the treatment of multidimensional arrays.
- Better interaction with other languages like Python, R. . .
- Portability to a wide variety of computing platforms and operating systems.

On the following pages you can find a detailed description of the entire C Caterva API.

## 3.1 Context

**`struct caterva_ctx_t`**

Context for caterva arrays that specifies the functions used to manage memory and the compression/decompression parameters used in Blosc.

### 3.1.1 Configuration parameters

**`struct caterva_config_t`**

Configuration parameters used to create a caterva context.

#### Public Members

void *(***`alloc`**) (size_t)

The memory allocation function used internally.

void (***`free`**) (void *)

The memory release function used internally.

int **`compcodec`**

Defines the codec used in compression.

int **`complevel`**
Determines the compression level used in Blosc.

int **`usedict`**
Indicates whether a dict is used to compress data or not.

int **`nthreads`**
Determines the maximum number of threads that can be used.

uint8_t **`filters`**[**`BLOSC2_MAX_FILTERS`**]
Defines the filters used in compression.

uint8_t **`filtersmeta`**[**`BLOSC2_MAX_FILTERS`**]
Indicates the meta filters used in Blosc.

blosc2_prefilter_fn **`prefilter`**
Defines the function that is applied to the data before compressing it.

blosc2_prefilter_params ***`pparams`**
Indicates the parameters of the prefilter function.

blosc2_btune ***`udbtune`**
Indicates user-defined parameters for btune.

### 3.1.2 Creation

int **`caterva_ctx_new`**(*caterva_config_t *cfg*, *caterva_ctx_t **ctx*)
Create a context for caterva.

**Return** An error code.

**Parameters**

- `cfg`: The configuration parameters needed for the context creation.
- `ctx`: Pointer to the memory pointer where the context will be created.

### 3.1.3 Destruction

int **`caterva_ctx_free`**(*caterva_ctx_t **ctx*)
Free a context.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the pointer to the context to be freed.

## 3.2 Array

**`struct caterva_array_t`**
A multidimensional array of data that can be compressed data.

### 3.2.1 Parameters

#### General parameters

**`struct caterva_params_t`**

General parameters needed for the creation of a caterva array.

##### Public Members

uint8_t **`itemsize`**

The size of each item of the array.

int64_t **`shape`**[**`CATERVA_MAX_DIM`**]

The array shape.

uint8_t **`ndim`**

The array dimensions.

#### Storage parameters

**`struct caterva_storage_t`**

Storage parameters needed for the creation of a caterva array.

##### Public Members

*caterva_storage_backend_t* **`backend`**

The backend storage.

*caterva_storage_properties_t* **`properties`**

The specific properties for the selected `backend`.

**`enum caterva_storage_backend_t`**

The backends available to store the data of the caterva array.

*Values:*

**`CATERVA_STORAGE_BLOSC`**

Indicates that the data is stored using a Blosc super-chunk.

**`CATERVA_STORAGE_PLAINBUFFER`**

Indicates that the data is stored using a plain buffer.

**`union caterva_storage_properties_t`**

*#include <caterva.h>* The storage properties for an array.

##### Public Members

*caterva_storage_properties_blosc_t* **`blosc`**

The storage properties when the array is backed by a Blosc super-chunk.

*caterva_storage_properties_plainbuffer_t* **`plainbuffer`**

The storage properties when the array is backed by a plain buffer.

**`struct caterva_storage_properties_blosc_t`**

The storage properties for an array backed by a Blosc super-chunk.

#### Public Members

int32_t **chunkshape**[**CATERVA_MAX_DIM**]
: The shape of each chunk of Blosc.

int32_t **blockshape**[**CATERVA_MAX_DIM**]
: The shape of each block of Blosc.

bool **sequencial**
: Flag to indicate if the super-chunk is stored sequentially or sparsely.

char ***urlpath**
: The super-chunk name.

  If `urlpath` is not `NULL`, the super-chunk will be stored on disk.

*caterva_metalayer_t* **metalayers**[**CATERVA_MAX_METALAYERS**]
: List with the metalayers desired.

int32_t **nmetalayers**
: The number of metalayers.

**struct caterva_metalayer_t**
: The metalayer data needed to store it on an array.

#### Public Members

char ***name**
: The name of the metalater.

uint8_t ***sdata**
: The serialized data to store.

int32_t **size**
: The size of the serialized data.

**struct caterva_storage_properties_plainbuffer_t**
: The storage properties that have a caterva array backed by a plain buffer.

#### Public Members

char ***urlpath**
: The plain buffer name.

  If `urlpath` is not `NULL`, the plain buffer will be stored on disk. (Not implemented yet).

## 3.2.2 Creation

### Chunk by chunk

int **caterva_empty** (*caterva_ctx_t *ctx*, *caterva_params_t *params*, *caterva_storage_t *storage*, *caterva_array_t **array*)
: Create an empty array.

  **Return** An error code.

  **Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `params`: Pointer to the general params of the array desired.
- `storage`: Pointer to the storage params of the array desired.
- `array`: Pointer to the memory pointer where the array will be created.

int **`caterva_append`**(*caterva_ctx_t *ctx*, *caterva_array_t *array*, void **chunk*, int64_t *chunksize*)

Append a chunk to a caterva array (until it is completely filled).

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `array`: Pointer to the caterva array.
- `chunk`: Pointer to the buffer where the chunk data is stored.
- `chunksize`: Size (in bytes) of the buffer.

### From/To buffer

int **`caterva_from_buffer`**(*caterva_ctx_t *ctx*, void **buffer*, int64_t *buffersize*, *caterva_params_t *params*, *caterva_storage_t *storage*, *caterva_array_t **array*)

Create a caterva array from the data stored in a buffer.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `buffer`: Pointer to the buffer where source data is stored.
- `buffersize`: The size (in bytes) of the buffer.
- `params`: Pointer to the general params of the array desired.
- `storage`: Pointer to the storage params of the array desired.
- `array`: Pointer to the memory pointer where the array will be created.

int **`caterva_to_buffer`**(*caterva_ctx_t *ctx*, *caterva_array_t *array*, void **buffer*, int64_t *buffersize*)

Extract the data into a C buffer from a caterva array.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `array`: Pointer to the caterva array.
- `buffer`: Pointer to the buffer where the data will be stored.
- `buffersize`: Size (in bytes) of the buffer.

### From/To frame

int **caterva_from_schunk** (*caterva_ctx_t* **ctx*, blosc2_schunk **schunk*, *caterva_array_t* ***array*)

Create a caterva array from a super-chunk.

It can only be used if the array is backed by a blosc super-chunk.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `schunk`: The blosc super-chunk where the caterva array is stored.
- `array`: Pointer to the memory pointer where the array will be created.

int **caterva_from_serial_schunk** (*caterva_ctx_t* **ctx*, uint8_t **serial_schunk*, int64_t *len*, *caterva_array_t* ***array*)

Create a caterva array from a serialized super-chunk.

It can only be used if the array is backed by a blosc super-chunk.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `serial_schunk`: The serialized super-chunk where the caterva array is stored.
- `len`: The size (in bytes) of the serialized super-chunk.
- `array`: Pointer to the memory pointer where the array will be created.

### From/To file

int **caterva_open** (*caterva_ctx_t* **ctx*, **const** char **urlpath*, *caterva_array_t* ***array*)

Read a caterva array from disk.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `urlpath`: The urlpath of the caterva array on disk.
- `array`: Pointer to the memory pointer where the array will be created.

## 3.2.3 Copying

int **caterva_copy** (*caterva_ctx_t* **ctx*, *caterva_array_t* **src*, *caterva_storage_t* **storage*, *caterva_array_t* ***array*)

Make a copy of the array data.

The copy is done into a new caterva array.

**Return** An error code

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `src`: Pointer to the array from which data is copied.
- `storage`: Pointer to the storage params of the array desired.
- `array`: Pointer to the memory pointer where the array will be created.

### 3.2.4 Slicing

int **`caterva_get_slice`**(*caterva_ctx_t* **ctx*, *caterva_array_t* **src*, int64_t **start*, int64_t **stop*, *caterva_storage_t* **storage*, *caterva_array_t* ***array*)

Get a slice from an array and store it into a new array.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `src`: Pointer to the array from which the slice will be extracted
- `start`: The coordinates where the slice will begin.
- `stop`: The coordinates where the slice will end.
- `storage`: Pointer to the storage params of the array desired.
- `array`: Pointer to the memory pointer where the array will be created.

int **`caterva_get_slice_buffer`**(*caterva_ctx_t* **ctx*, *caterva_array_t* **src*, int64_t **start*, int64_t **stop*, int64_t **shape*, void **buffer*, int64_t *buffersize*)

Get a slice from an array and store it into a C buffer.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `src`: Pointer to the array from which the slice will be extracted.
- `start`: The coordinates where the slice will begin.
- `stop`: The coordinates where the slice will end.
- `shape`: The shape of the buffer.
- `buffer`: Pointer to the buffer where the data will be stored.
- `buffersize`: The size (in bytes) of the buffer.

int **`caterva_set_slice_buffer`**(*caterva_ctx_t* **ctx*, void **buffer*, int64_t *buffersize*, int64_t **start*, int64_t **stop*, *caterva_array_t* **array*)

Set a slice into a caterva array from a C buffer.

It can only be used if the array is backed by a plainbuffer.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `buffer`: Pointer to the buffer where the slice data is.

- `buffersize`: The size (in bytes) of the buffer.
- `start`: The coordinates where the slice will begin.
- `stop`: The coordinates where the slice will end.
- `array`: Pointer to the caterva array where the slice will be set

int **`caterva_squeeze`** (*caterva_ctx_t *ctx*, *caterva_array_t *array*)

Squeeze a caterva array.

This function remove single-dimensional entries from the shape of a caterva array.

**Return** An error code

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `array`: Pointer to the caterva array.

### 3.2.5 Destruction

int **`caterva_free`** (*caterva_ctx_t *ctx*, *caterva_array_t **array*)

Free an array.

**Return** An error code.

**Parameters**

- `ctx`: Pointer to the caterva context to be used.
- `array`: Pointer to the memory pointer where the array is placed.

# CHAPTER 4

# High-level APIs

Here you can find a list with the wrappers available for Caterva:

- cat4py: a Pythonic wrapper of Caterva.

# CHAPTER 5

# Contributing to Caterva

Caterva is a community maintained project. We want to make contributing to this project as easy and transparent as possible.

## 5.1 Asking for help

If you have a question about how to use Caterva, please post your question on StackOverflow using the "caterva" tag.

## 5.2 Bug reports

We use GitHub issues to track public bugs. Please ensure your description is clear and has sufficient instructions to be able to reproduce the issue. The ideal report should contain the following:

1. Summarize the problem: Include details about your goal, describe expected and actual results and include any error messages.

2. Describe what you've tried: Show what you've tried, tell us what you found and why it didn't meet your needs.

3. Minimum reproducible example: Share the minimum amount of code needed to reproduce your issue. You can format the code nicely using markdown:

~~~
```C
#import <caterva.h>

int main() {
    ...
}
```
~~~

4. Determine the environment: Indicates the Caterva version and the operating system the code is running on.

## 5.3 Contributing to code

We actively welcome your code contributions. By contributing to Caterva, you agree that your contributions will be licensed under the LICENSE file of the project.

### 5.3.1 Fork the repo

Make a fork of the Caterva repository and clone it:

```
git clone https://github.com/<your-github-username>/caterva
```

### 5.3.2 Create your branch

Before you do any new work or submit a pull request, please open an issue on GitHub to report the bug or propose the feature you'd like to add.

Then create a new, separate branch for each piece of work you want to do.

### 5.3.3 Update docstrings

If you've changed APIs, update the involved docstrings using the doxygen format.

### 5.3.4 Run the test suite

If you have added code that needs to be tested, add the necessary tests and verify that all tests pass successfully.

# CHAPTER 6

# Roadmap

Caterva is a multidimensional container for binary data. It is a pure C library, allowing for better interoperatibility between different languages (although Python stands out high in the list).

This document lists the goals for a production release of Caterva.

## 6.1 Existing features

- **Built on top of Blosc2:** besides transparent compression, this allows to store large compressed datasets either in-memory or on-disk. In addition, Caterva inherits all the improvements that are being introduced in Blosc2 (see https://github.com/Blosc/c-blosc2/blob/master/ROADMAP.md).
- **Two-level multidimensional chunking:** like other libraries, Caterva stores the data into multidimensional chunks for efficient slicing. But in addition, Caterva introduces a new level of chunking. Within each chunk, the data is re-chunked into smaller multidimensional sets called blocks, leading to more fine-grained, and hence, to even more efficient slicing capabilities.
- **Plainbuffer support:** Caterva also allows to store data in a contiguous buffer. In this way, it facilitates the interoperability with other libraries like NumPy.

## 6.2 Actions to be done

- **Update values:** this will make array creation more flexible, since the process of populating an array will not necessarily need to follow a row-wise order (it can actually be any order).
- **Resize array dimensions:** this will allow to increase or decrease in size any dimension of the arrays.
- **Improve slicing capabilities:** currently Caterva only supports basic slicing based on *start:stop* ranges; we would like to extend this to *start:stop:step* as well as selections based on an array of booleans (similar to NumPy).

- **Add support for DLPack:** support for DLPack would be nice for being able to share data between different frameworks and devices. This should complement (or even replace in the long term) the existing plainbuffer support. See this dicussion for more insight on what advantages could the support for DLPack bring for Caterva.
- **Support for multidimensional filters:** this will improve the in-memory spatial locally for data that is **n-dim closer** in the array; by n-dim closer we mean that the multidimensional norm (in an Euclidean space) between two different positions of elements is shorter. This may led to better compression opportunities when spatial locality (Euclidean space) is high.
- **Support for multidimensional codecs:** this is the equivalent for multidim filters, but for codecs. Multidim codecs can leverage n-dim spatial locality in order to compress better/faster. Such codecs could be used in combination with others, uni-dim codecs (e.g. LZ4), so as to get better ratios.
- **Provide wheels:** this will make the installation much more easier for the user.

## 6.3 Outreaching

- **Improve the main Caterva README:** this should allow a better overview at first glance of all the features that Caterva offers right now.
- **Attend to meetings and conferences:** it is very important to plan going to conferences for advertising Caterva and meeting people in-person. We need to decide which meetings to attend. As there are not that much conferences about C libraries, it is important to leverage the cat4py wrapper so as to try to promote Caterva on Python conferences too.
- Other outreaching activities would be to produce videos of the kind 'Caterva in 10 minutes', or producing compelling tutorials (preferably based on Jupyter notebook, and using services like binder that allows a low entry level for quick trials).

# CHAPTER 7

# Release notes

## 7.1 Changes from 0.4.0 to 0.4.1

XXX version-specific blurb XXX

## 7.2 Changes from 0.3.3 to 0.4.0

- API renaming. The function names simulate a *namespace* and some variable names have been changed to those used by the general community (*part* -> *chunk*).
- Add a new level of multi-dimensionality. As a result, unlike other libraries, Caterva supports two levels of multi-dimensional chunking (chunks and blocks).
- Improve library compilation to allow users to avoid building tests and examples.
- Simplify the test's suite for a proper integration in Windows.
- Update documentation to improve the library description and to add the *Release notes* and the *Roadmap* in a new section.

## 7.3 Changes from 0.3.0 to 0.3.3

- Fixing that 0.3.1 and 0.3.2 tags were not made in master :-/

## 7.4 Changes from 0.2.2 to 0.3.0

- Big code and API refactorization. As result, the API is more consistent and hopefully more intuitive to use. For more info on the new API, see https://caterva.readthedocs.io.

## 7.5 Changes from 0.2.1 to 0.2.2

- Add a *caterva_from_sframe()* function.

## 7.6 Changes from 0.2.0 to 0.2.1

- Both static and dynamic libraries are created by default now. If you want to disable the creation of one of them, just set the cmake options *SHARED_LIB=OFF* or *STATIC_LIB=OFF*.
- Add a *copy* parameter to *caterva_from_file()*.

# Index