

## HPC-datastore-cpp

Generated by Doxygen 1.9.3



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 ds::Connection Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 Connection()	6
3.1.3 Member Function Documentation	7
3.1.3.1 get_properties()	7
3.1.3.2 get_view()	7
3.1.3.3 read_block() [1/2]	8
3.1.3.4 read_block() [2/2]	8
3.1.3.5 read_blocks() [1/2]	9
3.1.3.6 read_blocks() [2/2]	10
3.1.3.7 read_image()	11
3.1.3.8 read_region() [1/2]	12
3.1.3.9 read_region() [2/2]	12
3.1.3.10 write_block()	13
3.1.3.11 write_blocks()	14
3.1.3.12 write_image()	15
3.1.3.13 write_with_pyramids()	15
3.2 ds::DatasetProperties Class Reference	16
3.2.1 Detailed Description	17
3.3 ds::ImageView Class Reference	17
3.3.1 Detailed Description	18
3.3.2 Constructor & Destructor Documentation	19
3.3.2.1 ImageView()	19
3.3.3 Member Function Documentation	19
3.3.3.1 get_properties()	19
3.3.3.2 read_block() [1/2]	20
3.3.3.3 read_block() [2/2]	20
3.3.3.4 read_blocks() [1/2]	21
3.3.3.5 read_blocks() [2/2]	21
3.3.3.6 read_image()	22
3.3.3.7 read_region() [1/2]	23
3.3.3.8 read_region() [2/2]	23
3.3.3.9 write_block()	24
3.3.3.10 write_blocks()	24
3.3.3.11 write_image()	25

3.4 ds::ResolutionUnit Class Reference . . . . .	26
3.4.1 Detailed Description . . . . .	26
<b>4 File Documentation</b>	<b>27</b>
4.1 hpc_ds_api.hpp . . . . .	27
4.2 hpc_ds_details.hpp . . . . .	35
4.3 hpc_ds_structs.hpp . . . . .	43
<b>Index</b>	<b>47</b>

# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">ds::Connection</a>	
Representation of connection to dataset . . . . .	5
<a href="#">ds::DatasetProperties</a>	
Class representing dataset properties . . . . .	16
<a href="#">ds::ImageView</a>	
Representation of connection to specific image . . . . .	17
<a href="#">ds::ResolutionUnit</a>	
Class representing resolution unit (in <a href="#">DatasetProperties</a> ) . . . . .	26



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

/home/somik/CBIA/hpc-datastore-cpp/src/ <a href="#">hpc_ds_api.hpp</a> . . . . .	27
/home/somik/CBIA/hpc-datastore-cpp/src/ <a href="#">hpc_ds_details.hpp</a> . . . . .	35
/home/somik/CBIA/hpc-datastore-cpp/src/ <a href="#">hpc_ds_structs.hpp</a> . . . . .	43





## Chapter 3

# Class Documentation

### 3.1 ds::Connection Class Reference

Representation of connection to dataset.

```
#include <hpc_ds_api.hpp>
```

#### Public Member Functions

- [Connection](#) (std::string ip, int port, std::string uuid)  
*Construct a new [Connection](#) object.*
- [ImageView](#) [get\\_view](#) (int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version) const  
*Get [ImageView](#) of specified image.*
- dataset\_props\_ptr [get\\_properties](#) () const  
*Get dataset properties.*
- template<cnpts::Scalar T>  
i3d::Image3d< T > [read\\_block](#) (i3d::Vector3d< int > coord, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset\_props\_ptr props=nullptr) const  
*Read one block from server to image.*
- template<cnpts::Scalar T>  
void [read\\_block](#) (i3d::Vector3d< int > coord, i3d::Image3d< T > &dest, i3d::Vector3d< int > dest\_offset, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset\_props\_ptr props=nullptr) const  
*Read one block from server to image.*
- template<cnpts::Scalar T>  
std::vector< i3d::Image3d< T > > [read\\_blocks](#) (const std::vector< i3d::Vector3d< int > > &coords, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset\_props\_ptr props=nullptr) const  
*Read blocks from server and return them.*
- template<cnpts::Scalar T>  
void [read\\_blocks](#) (const std::vector< i3d::Vector3d< int > > &coords, i3d::Image3d< T > &dest, const std::vector< i3d::Vector3d< int > > &dest\_offsets, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset\_props\_ptr props=nullptr) const  
*Read blocks from server and saves them into preallocated image.*

- `template<cnpts::Scalar T>`  
`i3d::Image3d< T > read_region (i3d::Vector3d< int > start_point, i3d::Vector3d< int > end_point, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset_props_ptr props=nullptr) const`  
*Read region of interest from the server.*
- `template<cnpts::Scalar T>`  
`void read_region (i3d::Vector3d< int > start_point, i3d::Vector3d< int > end_point, i3d::Image3d< T > &dest, i3d::Vector3d< int > offset, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset_props_ptr props=nullptr) const`  
*Read region of interest from the server.*
- `template<cnpts::Scalar T>`  
`i3d::Image3d< T > read_image (int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset_props_ptr props=nullptr) const`  
*Read full image.*
- `template<cnpts::Scalar T>`  
`void write_block (const i3d::Image3d< T > &src, i3d::Vector3d< int > coord, i3d::Vector3d< int > src_offset, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset_props_ptr props=nullptr) const`  
*Write block to server.*
- `template<cnpts::Scalar T>`  
`void write_blocks (const i3d::Image3d< T > &src, const std::vector< i3d::Vector3d< int > > &coords, const std::vector< i3d::Vector3d< int > > &src_offsets, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset_props_ptr props=nullptr) const`  
*Write blocks to server.*
- `template<cnpts::Scalar T>`  
`void write_image (const i3d::Image3d< T > &img, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version, dataset_props_ptr props=nullptr) const`  
*Write image to server.*
- `template<cnpts::Scalar T>`  
`void write_with_pyramids (const i3d::Image3d< T > &img, int channel, int timepoint, int angle, const std::string &version, SamplingMode m, dataset_props_ptr props=nullptr) const`

### 3.1.1 Detailed Description

Representation of connection to dataset.

Class representing connection to specific dataset on the server. It provides basic methods for read/write operations necessary to transfer images (in the dataset) from/to server. This class does not cache or precollect any data, so the first HTTP request will be sent only when corresponding function is called.

All of the methods accepts arguments that uniquely identifies requested image. At the backend, this class transfers commands into [ImageView](#) objects.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 Connection()

```
ds::Connection::Connection (
    std::string ip,
    int port,
    std::string uuid )
```

Construct a new [Connection](#) object.

## Parameters

<i>ip</i>	IP address of server ( <a href="#">http://</a> at the beginning is not necessary)
<i>port</i>	Port, where the server is listening for requests
<i>uuid</i>	Unique identifier of dataset

### 3.1.3 Member Function Documentation

#### 3.1.3.1 get\_properties()

```
dataset_props_ptr ds::Connection::get_properties ( ) const
```

Get dataset properties.

## Returns

[DatasetProperties](#)

#### 3.1.3.2 get\_view()

```
ImageView ds::Connection::get_view (
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version ) const
```

Get [ImageView](#) of specified image.

## Parameters

<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")

## Returns

[ImageView](#)

### 3.1.3.3 read\_block() [1/2]

```
template<cnpts::Scalar T>
void ds::Connection::read_block (
    i3d::Vector3d< int > coord,
    i3d::Image3d< T > & dest,
    i3d::Vector3d< int > dest_offset,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Read one block from server to image.

Reads one block of image located at <coord> and saves it to <dest> with offset <dest\_offset>.

If in DEBUG, function will check wheter given coordinate corresponds to valid block as well as wheter the block fits into the image (taking offset into account).

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>coord</i>	Block coordinate
<i>dest</i>	Image to write data to
<i>dest_offset</i>	Offset by which the corresponding write should be moved
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

### 3.1.3.4 read\_block() [2/2]

```
template<cnpts::Scalar T>
i3d::Image3d< T > ds::Connection::read_block (
    i3d::Vector3d< int > coord,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Read one block from server to image.

Reads one block of image located at `<coord>` and saves it to `<dest>` with offset `<dest_offset>`.

If in DEBUG, function will check wheter given coordinate corresponds to valid block as well as wheter the block fits into the image (taking offset into account).

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>coord</i>	Block coordinate
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

#### Returns

Image containing selected block

#### 3.1.3.5 read\_blocks() [1/2]

```
template<cnpts::Scalar T>
void ds::Connection::read_blocks (
    const std::vector< i3d::Vector3d< int > > & coords,
    i3d::Image3d< T > & dest,
    const std::vector< i3d::Vector3d< int > > & dest_offsets,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Read blocks from server and saves them into preallocated image.

Read blocks specified in `<coords>` and saves them into locations given in `<offsets>`.

If in DEBUG, the function checks if coordinates given in `<coords>` points to a valid blocks, as well as wheter the offsets specified for each block are within image boundaries.

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>coords</i>	Block coordinates
<i>dest</i>	Preallocated destination image
<i>dest_offsets</i>	Offsets at wich the corresponding blocks should be saved
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

## 3.1.3.6 read\_blocks() [2/2]

```
template<cnpts::Scalar T>
std::vector< i3d::Image3d< T > > ds::Connection::read_blocks (
    const std::vector< i3d::Vector3d< int > > & coords,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Read blocks from server and return them.

Reads blocks specified in <coords> and returns them. Corresponding sizes are collected from server and calculated specifically for each block.

This function is not optimized, meaning that for each coord in <coord>, one HTTP request will be sent out to the server. This can heavily slow down speed of the application as communication via network is not cheap. If you do not have specific needs, most of the time it will be faster to collect blocks into preallocated image (second overload of read\_blocks), however it will eat more RAM.

If in DEBUG, the function checks if coordinates given in <coords> points to a valid blocks.

As there is no (meaningfull) way for C++ to choose correct underlying type in runtime, make sure to specify correct template type.

## Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>coords</i>	Block coordinates
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located

## Parameters

<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

## Returns

Vector of fetched blocks (the order is the same as given in <coords>)

## 3.1.3.7 read\_image()

```
template<cnpts::Scalar T>
i3d::Image3d< T > ds::Connection::read_image (
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Read full image.

Read full image from the server and return it. The information about dimensions are fetched from the server.

As there is no (meaningfull) way for C++ to choose correct underlying type in runtime, make sure to specify correct template type.

## Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

## Returns

i3d::Image3d<T> etched image

### 3.1.3.8 read\_region() [1/2]

```
template<cnpts::Scalar T>
void ds::Connection::read_region (
    i3d::Vector3d< int > start_point,
    i3d::Vector3d< int > end_point,
    i3d::Image3d< T > & dest,
    i3d::Vector3d< int > offset,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Read region of interest from the server.

Read all necessary blocks intersecting with chosen region from the server and insert region into preallocated image <dest> at <offset>.

It is necessary, that *start\_point* < *end\_point* (elem-wise)..

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>start_point</i>	smallest point of the region
<i>end_point</i>	largest point of the region
<i>dest</i>	destination image
<i>offset</i>	offset to destination image
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

### 3.1.3.9 read\_region() [2/2]

```
template<cnpts::Scalar T>
i3d::Image3d< T > ds::Connection::read_region (
    i3d::Vector3d< int > start_point,
    i3d::Vector3d< int > end_point,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
```



```
const std::string & version,
dataset_props_ptr props = nullptr ) const
```

Read region of interest from the server.

Read all necessary blocks intersecting with chosen region from the server. It is necessary, that `start_point < end_point` (elem-wise)..

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>start_point</i>	smallest point of the region
<i>end_point</i>	largest point of the region
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

#### Returns

Image containing selected block

#### 3.1.3.10 write\_block()

```
template<cnpts::Scalar T>
void ds::Connection::write_block (
    const i3d::Image3d< T > & src,
    i3d::Vector3d< int > coord,
    i3d::Vector3d< int > src_offset,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Write block to server.

Write block from source image to server. The information about dimensions are fetched from the server.

If in DEBUG, the function checks if coordinate given in `<coord>` points to a valid block, as well as whether the offset specified for block is within image boundaries.

## Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>src</i>	Source image to collect block from
<i>coord</i>	Block coordinates
<i>src_offset</i>	Offset of given block in source image
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

## 3.1.3.11 write\_blocks()

```
template<cnpts::Scalar T>
void ds::Connection::write_blocks (
    const i3d::Image3d< T > & src,
    const std::vector< i3d::Vector3d< int > > & coords,
    const std::vector< i3d::Vector3d< int > > & src_offsets,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Write blocks to server.

Write blocks from source image to server. The information about dimensions are fetched from the server.

If in DEBUG, the function checks if coordinates given in <coords> points to a valid block, as well as wheter the offsets specified for each block is within image boundaries.

## Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>src</i>	Source image to collect blocks from
<i>coords</i>	Vector of block coordinates
<i>src_offsets</i>	Offsets of corresponding blocks in source image
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located

## Parameters

<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

## 3.1.3.12 write\_image()

```
template<cnpts::Scalar T>
void ds::Connection::write_image (
    const i3d::Image3d< T > & img,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    const std::string & version,
    dataset_props_ptr props = nullptr ) const
```

Write image to server.

Write full image to server.

It is recommended to make sure that the dimension of the source image is the same as the dimension of image at server side.

Mostly, given smaller source image will emit error and fail to upload. Given larger source image will result in cropping.

## Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>img</i>	Source image
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

## 3.1.3.13 write\_with\_pyramids()

```
template<cnpts::Scalar T>
void ds::Connection::write_with_pyramids (
```

```

const i3d::Image3d< T > & img,
int channel,
int timepoint,
int angle,
const std::string & version,
SamplingMode m,
dataset_props_ptr props = nullptr ) const

```

@brief Write full image and generate pyramids

Creates (several if needed) HTTP requests and sends whole image to

datastore. Input image is considered to be full-resolution (that is: {1, 1, 1}). All other resolutions will be generated with selected <ResamplingMode> and uploaded to server as well.

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>img</i>	Input image in original resolution
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>m</i>	Sampling mode used for image resampling
<i>props</i>	[Optional] cached dataset properties

The documentation for this class was generated from the following file:

- /home/somik/CBIA/hpc-datastore-cpp/src/hpc\_ds\_api.hpp

## 3.2 ds::DatasetProperties Class Reference

Class representing dataset properties.

```
#include <hpc_ds_structs.hpp>
```

### Public Member Functions

- i3d::Vector3d< int > **get\_block\_dimensions** (i3d::Vector3d< int > resolution) const
- i3d::Vector3d< int > **get\_block\_size** (i3d::Vector3d< int > coord, i3d::Vector3d< int > resolution) const
- i3d::Vector3d< int > **get\_block\_count** (i3d::Vector3d< int > resolution) const
- i3d::Vector3d< int > **get\_img\_dimensions** (i3d::Vector3d< int > resolution) const
- std::vector< i3d::Vector3d< int > > **get\_all\_resolutions** () const

## Public Attributes

- std::string **uuid**
- std::string **voxel\_type**
- i3d::Vector3d< int > **dimensions**
- int **channels**
- int **angles**
- std::optional< std::string > **transformations**
- std::string **voxel\_unit**
- std::optional< i3d::Vector3d< double > > **voxel\_resolution**
- std::optional< [ResolutionUnit](#) > **timepoint\_resolution**
- std::optional< [ResolutionUnit](#) > **channel\_resolution**
- std::optional< [ResolutionUnit](#) > **angle\_resolution**
- std::string **compression**
- std::vector< std::map< std::string, i3d::Vector3d< int > > > **resolution\_levels**
- std::vector< int > **versions**
- std::string **label**
- std::optional< std::string > **view\_registrations**
- std::vector< int > **timepoint\_ids**

## Friends

- std::ostream & **operator**<< (std::ostream &stream, const [DatasetProperties](#) &ds)

### 3.2.1 Detailed Description

Class representing dataset properties.

The documentation for this class was generated from the following file:

- /home/somik/CBIA/hpc-datastore-cpp/src/hpc\_ds\_structs.hpp

## 3.3 ds::ImageView Class Reference

Representation of connection to specific image.

```
#include <hpc_ds_api.hpp>
```

## Public Member Functions

- [ImageView](#) (std::string ip, int port, std::string uuid, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, std::string version)  
*Construct a new Image View object.*
- dataset\_props\_ptr [get\\_properties](#) () const  
*Get dataset properties.*
- template<cnpts::Scalar T>  
i3d::Image3d< T > [read\\_block](#) (i3d::Vector3d< int > coord, dataset\_props\_ptr props=nullptr) const  
*Read one block from server.*
- template<cnpts::Scalar T>  
void [read\\_block](#) (i3d::Vector3d< int > coord, i3d::Image3d< T > &dest, i3d::Vector3d< int > dest\_offset={0, 0, 0}, dataset\_props\_ptr props=nullptr) const  
*Read one block from server to image.*
- template<cnpts::Scalar T>  
std::vector< i3d::Image3d< T > > [read\\_blocks](#) (const std::vector< i3d::Vector3d< int > > &coords, dataset\_props\_ptr props=nullptr) const  
*Read blocks from server and return them.*
- template<cnpts::Scalar T>  
void [read\\_blocks](#) (const std::vector< i3d::Vector3d< int > > &coords, i3d::Image3d< T > &dest, const std::vector< i3d::Vector3d< int > > &offsets, dataset\_props\_ptr props=nullptr) const  
*Read blocks from server and saves them into preallocated image.*
- template<cnpts::Scalar T>  
i3d::Image3d< T > [read\\_region](#) (i3d::Vector3d< int > start\_point, i3d::Vector3d< int > end\_point, dataset\_props\_ptr props=nullptr) const  
*Read region of interest from the server.*
- template<cnpts::Scalar T>  
void [read\\_region](#) (i3d::Vector3d< int > start\_point, i3d::Vector3d< int > end\_point, i3d::Image3d< T > &dest, i3d::Vector3d< int > offset={0, 0, 0}, dataset\_props\_ptr props=nullptr) const  
*Read region of interest from the server.*
- template<cnpts::Scalar T>  
i3d::Image3d< T > [read\\_image](#) (dataset\_props\_ptr props=nullptr) const  
*Read full image.*
- template<cnpts::Scalar T>  
void [write\\_block](#) (const i3d::Image3d< T > &src, i3d::Vector3d< int > coord, i3d::Vector3d< int > src\_offset={0, 0, 0}, dataset\_props\_ptr props=nullptr) const  
*Write block to server.*
- template<cnpts::Scalar T>  
void [write\\_blocks](#) (const i3d::Image3d< T > &src, const std::vector< i3d::Vector3d< int > > &coords, const std::vector< i3d::Vector3d< int > > &src\_offsets, dataset\_props\_ptr props=nullptr) const  
*Write blocks to server.*
- template<cnpts::Scalar T>  
void [write\\_image](#) (const i3d::Image3d< T > &img, dataset\_props\_ptr props=nullptr) const  
*Write image to server.*

### 3.3.1 Detailed Description

Representation of connection to specific image.

Class representing connection to specific image on the server. This class provides basic methods for read/write operations necessary to transfer images from/to server. This class does not cache or precollect any data, so the first HTTP request will be send only when corresponding function is called.

## 3.3.2 Constructor & Destructor Documentation

### 3.3.2.1 ImageView()

```
ds::ImageView::ImageView (
    std::string ip,
    int port,
    std::string uuid,
    int channel,
    int timepoint,
    int angle,
    i3d::Vector3d< int > resolution,
    std::string version )
```

Construct a new Image View object.

#### Parameters

<i>ip</i>	IP address of server ( <a href="http://">http://</a> at the beginning is not necessary)
<i>port</i>	Port, where the server is listening for requests
<i>uuid</i>	Unique identifier of dataset
<i>channel</i>	Channel, at which the image is located
<i>timepoint</i>	Timepoint, at which the image is located
<i>angle</i>	Angle, at which the image is located
<i>resolution</i>	Resolution, at which the image is located
<i>version</i>	Version, at which the image is located (integer identifier or "latest")
<i>props</i>	[Optional] cached dataset properties

## 3.3.3 Member Function Documentation

### 3.3.3.1 get\_properties()

```
dataset_props_ptr ds::ImageView::get_properties ( ) const
```

Get dataset properties.

#### Returns

[DatasetProperties](#)

### 3.3.3.2 read\_block() [1/2]

```
template<cnpts::Scalar T>
i3d::Image3d< T > ds::ImageView::read_block (
    i3d::Vector3d< int > coord,
    dataset_props_ptr props = nullptr ) const
```

Read one block from server.

Reads one block of image located at <coord> and returns it. The information about size of the image is collected from the server.

If in DEBUG, function will check wheter given coordinate corresponds to valid block.

As there is no (meaningfull) way for C++ to choose correct underlying type in runtime, make sure to specify correct template type.

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>coord</i>	Block coordinate
<i>props</i>	[Optional] cached dataset properties

#### Returns

Image containing selected block

### 3.3.3.3 read\_block() [2/2]

```
template<cnpts::Scalar T>
void ds::ImageView::read_block (
    i3d::Vector3d< int > coord,
    i3d::Image3d< T > & dest,
    i3d::Vector3d< int > dest_offset = {0, 0, 0},
    dataset_props_ptr props = nullptr ) const
```

Read one block from server to image.

Reads one block of image located at <coord> and saves it to <dest> with offset <dest\_offset>.

If in DEBUG, function will check wheter given coordinate corresponds to valid block as well as wheter the block fits into the image (taking offset into account).

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---



## Parameters

<i>coord</i>	Block coordinate
<i>dest</i>	Image to write data to
<i>dest_offset</i>	Offset by which the corresponding write should be moved
<i>props</i>	[Optional] cached dataset properties

## 3.3.3.4 read\_blocks() [1/2]

```
template<cnpts::Scalar T>
std::vector< i3d::Image3d< T > > ds::ImageView::read_blocks (
    const std::vector< i3d::Vector3d< int > > & coords,
    dataset_props_ptr props = nullptr ) const
```

Read blocks from server and return them.

Reads blocks specified in <coords> and returns them. Corresponding sizes are collected from server and calculated specifically for each block.

This function is not optimized, meaning that for each coord in <coord>, one HTTP request will be sent out to the server. This can heavily slow down speed of the application as communication via network is not cheap. If you do not have specific needs, most of the time it will be faster to collect blocks into preallocated image (second overload of read\_blocks), however it will eat more RAM.

If in DEBUG, the function checks if coordinates given in <coords> points to a valid blocks.

As there is no (meaningfull) way for C++ to choose correct underlying type in runtime, make sure to specify correct template type.

## Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>coords</i>	Block coordinates
<i>props</i>	[Optional] cached dataset properties

## Returns

Vector of fetched blocks (the order is the same as given in <coords>)

## 3.3.3.5 read\_blocks() [2/2]

```
template<cnpts::Scalar T>
void ds::ImageView::read_blocks (
```

```
const std::vector< i3d::Vector3d< int > > & coords,
i3d::Image3d< T > & dest,
const std::vector< i3d::Vector3d< int > > & offsets,
dataset_props_ptr props = nullptr ) const
```

Read blocks from server and saves them into preallocated image.

Read blocks specified in <coords> and saves them into locations given in <offsets>.

If in DEBUG, the function checks if coordinates given in <coords> points to a valid blocks.

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>coords</i>	Block coordinates
<i>dest</i>	Preallocated destination image
<i>offsets</i>	Offsets at wich the corresponding blocks should be saved
<i>props</i>	[Optional] cached dataset properties

### 3.3.3.6 read\_image()

```
template<cnpts::Scalar T>
i3d::Image3d< T > ds::ImageView::read_image (
    dataset_props_ptr props = nullptr ) const
```

Read full image.

Read full image from the server and return it. The information about dimensions are fetched from the server.

As there is no (meaningfull) way for C++ to choose correct underlying type in runtime, make sure to specify correct template type.

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>props</i>	[Optional] cached dataset properties
--------------	--------------------------------------

#### Returns

i3d::Image3d<T> Fetched image

**3.3.3.7 read\_region() [1/2]**

```
template<cnpts::Scalar T>
i3d::Image3d< T > ds::ImageView::read_region (
    i3d::Vector3d< int > start_point,
    i3d::Vector3d< int > end_point,
    dataset_props_ptr props = nullptr ) const
```

Read region of interest from the server.

Read all necessary blocks intersecting with chosen region from the server. It is necessary, that `start_point < end_point` (elem-wise)..

**Template Parameters**

<i>T</i>	Scalar used as underlying type for image representation
----------	---

**Parameters**

<i>start_point</i>	smallest point of the region
<i>end_point</i>	largest point of the region
<i>props</i>	[Optional] cached dataset properties

**Returns**

i3d::Image3d<T> Selected region

**3.3.3.8 read\_region() [2/2]**

```
template<cnpts::Scalar T>
void ds::ImageView::read_region (
    i3d::Vector3d< int > start_point,
    i3d::Vector3d< int > end_point,
    i3d::Image3d< T > & dest,
    i3d::Vector3d< int > offset = {0, 0, 0},
    dataset_props_ptr props = nullptr ) const
```

Read region of interest from the server.

Read all necessary blocks intersecting with chosen region from the server and insert region into preallocated image <dest> at <offset>.

It is necessary, that `start_point < end_point` (elem-wise)..

**Template Parameters**

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>start_point</i>	smallest point of the region
<i>end_point</i>	largest point of the region
<i>dest</i>	destination image
<i>offset</i>	offset to destination image
<i>props</i>	[Optional] cached dataset properties

**3.3.3.9 write\_block()**

```
template<cnpts::Scalar T>
void ds::ImageView::write_block (
    const i3d::Image3d< T > & src,
    i3d::Vector3d< int > coord,
    i3d::Vector3d< int > src_offset = {0, 0, 0},
    dataset_props_ptr props = nullptr ) const
```

Write block to server.

Write block from source image to server. The information about dimensions are fetched from the server.

If in DEBUG, the function checks if coordinate given in <coord> points to a valid block, as well as wheter the offset specified for block is within image boundaries.

## Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

## Parameters

<i>src</i>	Source image to collect block from
<i>coord</i>	Block coordinates
<i>src_offset</i>	Offset of given block in source image
<i>props</i>	[Optional] cached dataset properties

**3.3.3.10 write\_blocks()**

```
template<cnpts::Scalar T>
void ds::ImageView::write_blocks (
    const i3d::Image3d< T > & src,
    const std::vector< i3d::Vector3d< int > > & coords,
    const std::vector< i3d::Vector3d< int > > & src_offsets,
    dataset_props_ptr props = nullptr ) const
```

Write blocks to server.

Write blocks from source image to server. The information about dimensions are fetched from the server.

If in DEBUG, the function checks if coordinates given in <coords> points to a valid block, as well as whether the offsets specified for each block is within image boundaries.

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>src</i>	Source image to collect blocks from
<i>coords</i>	Vector of block coordinates
<i>src_offsets</i>	Offsets of corresponding blocks in source image
<i>props</i>	[Optional] cached dataset properties

#### 3.3.3.11 write\_image()

```
template<cnpts::Scalar T>
void ds::ImageView::write_image (
    const i3d::Image3d< T > & img,
    dataset_props_ptr props = nullptr ) const
```

Write image to server.

Write full image to server.

It is recommended to make sure that the dimension of the source image is the same as the dimension of image at server side.

Mostly, given smaller source image will emit error and fail to upload. Given larger source image will result in cropping.

#### Template Parameters

<i>T</i>	Scalar used as underlying type for image representation
----------	---

#### Parameters

<i>img</i>	Source image
<i>props</i>	[Optional] cached dataset properties

The documentation for this class was generated from the following file:

- /home/somik/CBIA/hpc-datastore-cpp/src/hpc\_ds\_api.hpp

## 3.4 ds::ResolutionUnit Class Reference

Class representing resolution unit (in [DatasetProperties](#))

```
#include <hpc_ds_structs.hpp>
```

### Public Attributes

- double **value** = 0.0
- std::string **unit** = ""

### Friends

- std::ostream & **operator**<< (std::ostream &stream, const [ResolutionUnit](#) &res)

### 3.4.1 Detailed Description

Class representing resolution unit (in [DatasetProperties](#))

The documentation for this class was generated from the following file:

- /home/somik/CBIA/hpc-datastore-cpp/src/hpc\_ds\_structs.hpp

## Chapter 4

# File Documentation

### 4.1 hpc\_ds\_api.hpp

```
1 #pragma once
2 #include "hpc_ds_details.hpp"
3 #include "hpc_ds_structs.hpp"
4 #include <fmt/core.h>
5 #include <i3d/image3d.h>
6 #include <i3d/transform.h>
7 #include <memory>
8 #include <string>
9 #include <type_traits>
10 #include <vector>
11
12 namespace ds {
13     inline dataset_props_ptr get_dataset_properties(const std::string& ip,
14                                                    int port,
15                                                    const std::string& uuid);
16
17     template <cnpts::Scalar T>
18     i3d::Image3d<T> read_image(const std::string& ip,
19                               int port,
20                               const std::string& uuid,
21                               int channel = 0,
22                               int timepoint = 0,
23                               int angle = 0,
24                               i3d::Vector3d<int> resolution = {1, 1, 1},
25                               const std::string& version = "latest",
26                               dataset_props_ptr props = nullptr);
27
28     template <cnpts::Scalar T>
29     void write_image(const i3d::Image3d<T>& img,
30                     const std::string& ip,
31                     int port,
32                     const std::string& uuid,
33                     int channel = 0,
34                     int timepoint = 0,
35                     int angle = 0,
36                     i3d::Vector3d<int> resolution = {1, 1, 1},
37                     const std::string& version = "latest",
38                     dataset_props_ptr props = nullptr);
39
40     template <cnpts::Scalar T>
41     void write_with_pyramids(const i3d::Image3d<T>& img,
42                             const std::string& ip,
43                             int port,
44                             const std::string& uuid,
45                             int channel = 0,
46                             int timepoint = 0,
47                             int angle = 0,
48                             const std::string& version = "latest",
49                             SamplingMode m = SamplingMode::NEAREST_NEIGHBOUR,
50                             dataset_props_ptr props = nullptr);
51
52     class ImageView {
53     public:
54         ImageView(std::string ip,
55                  int port,
56                  std::string uuid,
57                  int channel,
58                  int timepoint,
```

```

156         int angle,
157         i3d::Vector3d<int> resolution,
158         std::string version);
159
160 dataset_props_ptr get_properties() const;
161
162 template <cnpts::Scalar T>
163 i3d::Image3d<T> read_block(i3d::Vector3d<int> coord,
164                             dataset_props_ptr props = nullptr) const;
165
166 template <cnpts::Scalar T>
167 void read_block(i3d::Vector3d<int> coord,
168                 i3d::Image3d<T>& dest,
169                 i3d::Vector3d<int> dest_offset = {0, 0, 0},
170                 dataset_props_ptr props = nullptr) const;
171
172 template <cnpts::Scalar T>
173 std::vector<i3d::Image3d<T>>
174 read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
175             dataset_props_ptr props = nullptr) const;
176
177 template <cnpts::Scalar T>
178 void read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
179                 i3d::Image3d<T>& dest,
180                 const std::vector<i3d::Vector3d<int>>& offsets,
181                 dataset_props_ptr props = nullptr) const;
182
183 template <cnpts::Scalar T>
184 i3d::Image3d<T> read_region(i3d::Vector3d<int> start_point,
185                             i3d::Vector3d<int> end_point,
186                             dataset_props_ptr props = nullptr) const;
187
188 template <cnpts::Scalar T>
189 void read_region(i3d::Vector3d<int> start_point,
190                 i3d::Vector3d<int> end_point,
191                 i3d::Image3d<T>& dest,
192                 i3d::Vector3d<int> offset = {0, 0, 0},
193                 dataset_props_ptr props = nullptr) const;
194
195 template <cnpts::Scalar T>
196 i3d::Image3d<T> read_image(dataset_props_ptr props = nullptr) const;
197
198 template <cnpts::Scalar T>
199 void write_block(const i3d::Image3d<T>& src,
200                 i3d::Vector3d<int> coord,
201                 i3d::Vector3d<int> src_offset = {0, 0, 0},
202                 dataset_props_ptr props = nullptr) const;
203
204 template <cnpts::Scalar T>
205 void write_blocks(const i3d::Image3d<T>& src,
206                  const std::vector<i3d::Vector3d<int>>& coords,
207                  const std::vector<i3d::Vector3d<int>>& src_offsets,
208                  dataset_props_ptr props = nullptr) const;
209
210 template <cnpts::Scalar T>
211 void write_image(const i3d::Image3d<T>& img,
212                 dataset_props_ptr props = nullptr) const;
213
214 private:
215     std::string _ip;
216     int _port;
217     std::string _uuid;
218     int _channel;
219     int _timepoint;
220     int _angle;
221     i3d::Vector3d<int> _resolution;
222     std::string _version;
223 };
224
225 class Connection {
226 public:
227     Connection(std::string ip, int port, std::string uuid);
228
229     ImageView get_view(int channel,
230                       int timepoint,
231                       int angle,
232                       i3d::Vector3d<int> resolution,
233                       const std::string& version) const;
234
235     dataset_props_ptr get_properties() const;
236
237     template <cnpts::Scalar T>
238     i3d::Image3d<T> read_block(i3d::Vector3d<int> coord,
239                               int channel,
240                               int timepoint,
241                               int angle,
242                               i3d::Vector3d<int> resolution,
243                               const std::string& version,

```



```

467         dataset_props_ptr props = nullptr) const;
491     template <cnpts::Scalar T>
492     void read_block(i3d::Vector3d<int> coord,
493                   i3d::Image3d<T>& dest,
494                   i3d::Vector3d<int> dest_offset,
495                   int channel,
496                   int timepoint,
497                   int angle,
498                   i3d::Vector3d<int> resolution,
499                   const std::string& version,
500                   dataset_props_ptr props = nullptr) const;
501
502     template <cnpts::Scalar T>
503     std::vector<i3d::Image3d<T>>
504     read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
505               int channel,
506               int timepoint,
507               int angle,
508               i3d::Vector3d<int> resolution,
509               const std::string& version,
510               dataset_props_ptr props = nullptr) const;
511
512     template <cnpts::Scalar T>
513     void read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
514                   i3d::Image3d<T>& dest,
515                   const std::vector<i3d::Vector3d<int>>& dest_offsets,
516                   int channel,
517                   int timepoint,
518                   int angle,
519                   i3d::Vector3d<int> resolution,
520                   const std::string& version,
521                   dataset_props_ptr props = nullptr) const;
522
523     template <cnpts::Scalar T>
524     i3d::Image3d<T> read_region(i3d::Vector3d<int> start_point,
525                               i3d::Vector3d<int> end_point,
526                               int channel,
527                               int timepoint,
528                               int angle,
529                               i3d::Vector3d<int> resolution,
530                               const std::string& version,
531                               dataset_props_ptr props = nullptr) const;
532
533     template <cnpts::Scalar T>
534     void read_region(i3d::Vector3d<int> start_point,
535                    i3d::Vector3d<int> end_point,
536                    i3d::Image3d<T>& dest,
537                    i3d::Vector3d<int> offset,
538                    int channel,
539                    int timepoint,
540                    int angle,
541                    i3d::Vector3d<int> resolution,
542                    const std::string& version,
543                    dataset_props_ptr props = nullptr) const;
544
545     template <cnpts::Scalar T>
546     i3d::Image3d<T> read_image(int channel,
547                               int timepoint,
548                               int angle,
549                               i3d::Vector3d<int> resolution,
550                               const std::string& version,
551                               dataset_props_ptr props = nullptr) const;
552
553     template <cnpts::Scalar T>
554     void write_block(const i3d::Image3d<T>& src,
555                   i3d::Vector3d<int> coord,
556                   i3d::Vector3d<int> src_offset,
557                   int channel,
558                   int timepoint,
559                   int angle,
560                   i3d::Vector3d<int> resolution,
561                   const std::string& version,
562                   dataset_props_ptr props = nullptr) const;
563
564     template <cnpts::Scalar T>
565     void write_blocks(const i3d::Image3d<T>& src,
566                    const std::vector<i3d::Vector3d<int>>& coords,
567                    const std::vector<i3d::Vector3d<int>>& src_offsets,
568                    int channel,
569                    int timepoint,
570                    int angle,
571                    i3d::Vector3d<int> resolution,
572                    const std::string& version,
573                    dataset_props_ptr props = nullptr) const;
574
575     template <cnpts::Scalar T>
576     void write_image(const i3d::Image3d<T>& img,
577                    int channel,

```

```

755         int timepoint,
756         int angle,
757         i3d::Vector3d<int> resolution,
758         const std::string& version,
759         dataset_props_ptr props = nullptr) const;
760
761     template <cnpts::Scalar T>
762     void write_with_pyramids(const i3d::Image3d<T>& img,
763                             int channel,
764                             int timepoint,
765                             int angle,
766                             const std::string& version,
767                             SamplingMode m,
768                             dataset_props_ptr props = nullptr) const;
769
770     private:
771         std::string _ip;
772         int _port;
773         std::string _uuid;
774 };
775
776 } // namespace ds
777
778 /* ===== IMPLEMENTATION FOLLOWS ===== */
779
780 namespace ds {
781     /* ===== Global space */
782     /* inline */ dataset_props_ptr get_dataset_properties(const std::string& ip,
783                                                         int port,
784                                                         const std::string& uuid) {
785         std::string dataset_url = details::get_dataset_url(ip, port, uuid);
786         return std::make_shared<DatasetProperties>(
787             details::get_dataset_properties(dataset_url));
788     }
789
790     template <cnpts::Scalar T>
791     i3d::Image3d<T> read_image(const std::string& ip,
792                               int port,
793                               const std::string& uuid,
794                               int channel /* = 0 */,
795                               int timepoint /* = 0 */,
796                               int angle /* = 0 */,
797                               i3d::Vector3d<int> resolution /* = {1, 1, 1} */,
798                               const std::string& version /* = "latest" */,
799                               dataset_props_ptr props /* = nullptr */) {
800         return ImageView(ip, port, uuid, channel, timepoint, angle, resolution,
801                         version)
802             .read_image<T>(props);
803     }
804
805     template <cnpts::Scalar T>
806     void write_image(const i3d::Image3d<T>& img,
807                     const std::string& ip,
808                     int port,
809                     const std::string& uuid,
810                     int channel /* = 0 */,
811                     int timepoint /* = 0 */,
812                     int angle /* = 0 */,
813                     i3d::Vector3d<int> resolution /* = {1, 1, 1} */,
814                     const std::string& version /* = "latest" */,
815                     dataset_props_ptr props /* = nullptr */) {
816         ImageView(ip, port, uuid, channel, timepoint, angle, resolution, version)
817             .write_image(img, props);
818     }
819
820     template <cnpts::Scalar T>
821     void write_with_pyramids(const i3d::Image3d<T>& img,
822                             const std::string& ip,
823                             int port,
824                             const std::string& uuid,
825                             int channel /* = 0 */,
826                             int timepoint /* = 0 */,
827                             int angle /* = 0 */,
828                             const std::string& version /* = "latest" */,
829                             SamplingMode m /* = SamplingMode::NEAREST_NEIGHBOUR */,
830                             dataset_props_ptr props /* = nullptr */) {
831         Connection(ip, port, uuid)
832             .write_with_pyramids(img, channel, timepoint, angle, version, m, props);
833     }
834
835     /* ===== ImageView */
836
837     ImageView::ImageView(std::string ip,
838                          int port,
839                          std::string uuid,
840                          int channel,
841                          int timepoint,

```

```

862         int angle,
863         i3d::Vector3d<int> resolution,
864         std::string version)
865     : _ip(std::move(ip)), _port(port), _uuid(std::move(uuid)),
866       _channel(channel), _timepoint(timepoint), _angle(angle),
867       _resolution(resolution), _version(std::move(version)) {}
868
869 dataset_props_ptr ImageView::get_properties() const {
870     return get_dataset_properties(_ip, _port, _uuid);
871 }
872
873 template <cnpts::Scalar T>
874 i3d::Image3d<T>
875 ImageView::read_block(i3d::Vector3d<int> coord,
876                      dataset_props_ptr props /* = nullptr */) const {
877     /* Fetch properties from server */
878     if (!props)
879         props = get_properties();
880
881     i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
882
883     /* Prepare output image */
884     i3d::Image3d<T> img;
885     i3d::Vector3d<int> block_size = details::data_manip::get_block_size(
886         coord, block_dim, props->get_img_dimensions(_resolution));
887     img.MakeRoom(block_size);
888
889     /* Fetch and return */
890     read_block(coord, img);
891     return img;
892 }
893
894 template <cnpts::Scalar T>
895 void ImageView::read_block(i3d::Vector3d<int> coord,
896                          i3d::Image3d<T>& dest,
897                          i3d::Vector3d<int> dest_offset /* = {0, 0, 0} */,
898                          dataset_props_ptr props /* = nullptr */) const {
899     read_blocks({coord}, dest, {dest_offset}, props);
900 }
901
902 template <cnpts::Scalar T>
903 std::vector<i3d::Image3d<T>>
904 ImageView::read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
905                      dataset_props_ptr props /* = nullptr */) const {
906     if (!props)
907         props = get_properties();
908
909     /* Process blocks one by one */
910     std::vector<i3d::Image3d<T>> out;
911     for (auto coord : coords)
912         out.push_back(read_block<T>(coord));
913
914     return out;
915 }
916
917 // TODO optimise
918 template <cnpts::Scalar T>
919 void ImageView::read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
920                          i3d::Image3d<T>& dest,
921                          const std::vector<i3d::Vector3d<int>>& offsets,
922                          dataset_props_ptr props /* = nullptr */)
923 const {
924
925     if (!props)
926         props = get_properties();
927
928     /* Fetched properties from server */
929     std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);
930     i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
931
932     i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);
933
934     if (coords.size() != offsets.size())
935         throw std::logic_error("Count of coordinates != count of offsets");
936
937     if (!details::check_block_coords(coords, img_dim, block_dim))
938         throw std::out_of_range("Blocks out of range");
939
940     /* prepare request url */
941     std::string session_url = details::requests::session_url_request(
942         dataset_url, _resolution, _version);
943
944     if (session_url.ends_with('/'))
945         session_url.pop_back();
946
947     /* Fetch blocks one by one */
948     for (std::size_t i = 0; i < coords.size(); ++i) {

```

```

949     auto& coord = coords[i];
950     auto& offset = offsets[i];
951
952     std::string url =
953         fmt::format("{}{}/{}/{}/{}/{}/{}/{}", session_url, coord.x, coord.y,
954             coord.z, _timepoint, _channel, _angle);
955     auto [data, response] = details::requests::make_request(url);
956
957     details::data_manip::read_data(data, props->voxel_type, dest, offset);
958 }
959 }
960
961 template <cnpts::Scalar T>
962 i3d::Image3d<T>
963 ImageView::read_region(i3d::Vector3d<int> start_point,
964     i3d::Vector3d<int> end_point,
965     dataset_props_ptr props /* = nullptr */)const {
966     if (!props)
967         props = get_properties();
968
969     i3d::Vector3d img_dim = props->get_img_dimensions(_resolution);
970     i3d::Vector3d block_dim = props->get_block_dimensions(_resolution);
971
972     std::vector<i3d::Vector3d<int>> coords = details::get_intercepted_blocks(
973         start_point, end_point, img_dim, block_dim);
974
975     std::vector<i3d::Vector3d<int>> offsets;
976     for (auto coord : coords)
977         offsets.emplace_back(coord * block_dim - start_point);
978
979     i3d::Image3d<T> out_img;
980     out_img.MakeRoom(end_point - start_point);
981
982     read_blocks(coords, out_img, offsets, props);
983     return out_img;
984 }
985
986 template <cnpts::Scalar T>
987 void ImageView::read_region(i3d::Vector3d<int> start_point,
988     i3d::Vector3d<int> end_point,
989     i3d::Image3d<T>& dest,
990     i3d::Vector3d<int> offset /* = {0, 0, 0} */,
991     dataset_props_ptr props /* = nullptr */)const {
992     auto temp_img = read_region<T>(start_point, end_point, props);
993
994     // Copy to desired location
995     for (std::size_t x = 0; x < temp_img.GetSizeX(); ++x)
996         for (std::size_t y = 0; y < temp_img.GetSizeY(); ++y)
997             for (std::size_t z = 0; z < temp_img.GetSizeZ(); ++z)
998                 dest.SetVoxel(x + offset.x, y + offset.y, z + offset.z,
999                     temp_img.GetVoxel({x, y, z}));
1000 }
1001
1002 template <cnpts::Scalar T>
1003 i3d::Image3d<T>
1004 ImageView::read_image(dataset_props_ptr props /* = nullptr */)const {
1005     if (!props)
1006         props = get_properties();
1007
1008     i3d::Vector3d img_dim = props->get_img_dimensions(_resolution);
1009     return read_region<T>(0, img_dim, props);
1010 }
1011
1012 template <cnpts::Scalar T>
1013 void ImageView::write_block(const i3d::Image3d<T>& src,
1014     i3d::Vector3d<int> coord,
1015     i3d::Vector3d<int> src_offset /* = {0, 0, 0} */,
1016     dataset_props_ptr props /* = nullptr */)const {
1017     write_blocks(src, {coord}, {src_offset}, props);
1018 }
1019
1020 // TODO optimise
1021 template <cnpts::Scalar T>
1022 void ImageView::write_blocks(const i3d::Image3d<T>& src,
1023     const std::vector<i3d::Vector3d<int>>& coords,
1024     const std::vector<i3d::Vector3d<int>>& src_offsets,
1025     dataset_props_ptr props /* = nullptr */)const {
1026
1027     if (!props)
1028         props = get_properties();
1029
1030     /* Fetch server properties */
1031     std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);
1032
1033     i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
1034     i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);
1035

```

Generated by Doxygen

```

1123
1124 template <cnpts::Scalar T>
1125 i3d::Image3d<T>
1126 Connection::read_block(i3d::Vector3d<int> coord,
1127                        int channel,
1128                        int timepoint,
1129                        int angle,
1130                        i3d::Vector3d<int> resolution,
1131                        const std::string& version,
1132                        dataset_props_ptr props /* = nullptr */)const {
1133     return get_view(channel, timepoint, angle, resolution, version)
1134         .read_block<T>(coord, props);
1135 }
1136
1137 template <cnpts::Scalar T>
1138 void Connection::read_block(i3d::Vector3d<int> coord,
1139                             i3d::Image3d<T>& dest,
1140                             i3d::Vector3d<int> dest_offset,
1141                             int channel,
1142                             int timepoint,
1143                             int angle,
1144                             i3d::Vector3d<int> resolution,
1145                             const std::string& version,
1146                             dataset_props_ptr props /* = nullptr */)const {
1147     return get_view(channel, timepoint, angle, resolution, version)
1148         .read_block(coord, dest, dest_offset, props);
1149 }
1150
1151 template <cnpts::Scalar T>
1152 std::vector<i3d::Image3d<T>>
1153 Connection::read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
1154                        int channel,
1155                        int timepoint,
1156                        int angle,
1157                        i3d::Vector3d<int> resolution,
1158                        const std::string& version,
1159                        dataset_props_ptr props /* = nullptr */)const {
1160     return get_view(channel, timepoint, angle, resolution, version)
1161         .read_blocks<T>(coords, props);
1162 }
1163
1164 template <cnpts::Scalar T>
1165 void Connection::read_blocks(
1166     const std::vector<i3d::Vector3d<int>>& coords,
1167     i3d::Image3d<T>& dest,
1168     const std::vector<i3d::Vector3d<int>>& dest_offsets,
1169     int channel,
1170     int timepoint,
1171     int angle,
1172     i3d::Vector3d<int> resolution,
1173     const std::string& version,
1174     dataset_props_ptr props /* = nullptr */)const {
1175     get_view(channel, timepoint, angle, resolution, version)
1176         .read_blocks(coords, dest, dest_offsets, props);
1177 }
1178
1179 template <cnpts::Scalar T>
1180 i3d::Image3d<T>
1181 Connection::read_region(i3d::Vector3d<int> start_point,
1182                        i3d::Vector3d<int> end_point,
1183                        int channel,
1184                        int timepoint,
1185                        int angle,
1186                        i3d::Vector3d<int> resolution,
1187                        const std::string& version,
1188                        dataset_props_ptr props /* = nullptr */)const {
1189     return get_view(channel, timepoint, angle, resolution, version)
1190         .read_region<T>(start_point, end_point, props);
1191 }
1192
1193 template <cnpts::Scalar T>
1194 void Connection::read_region(i3d::Vector3d<int> start_point,
1195                             i3d::Vector3d<int> end_point,
1196                             i3d::Image3d<T>& dest,
1197                             i3d::Vector3d<int> offset,
1198                             int channel,
1199                             int timepoint,
1200                             int angle,
1201                             i3d::Vector3d<int> resolution,
1202                             const std::string& version,
1203                             dataset_props_ptr props /* = nullptr */)const {
1204     get_view(channel, timepoint, angle, resolution, version)
1205         .read_region<T>(start_point, end_point, dest, offset, props);
1206 }
1207
1208 template <cnpts::Scalar T>
1209 i3d::Image3d<T>

```

```

1210 Connection::read_image(int channel,
1211                        int timepoint,
1212                        int angle,
1213                        i3d::Vector3d<int> resolution,
1214                        const std::string& version,
1215                        dataset_props_ptr props /* = nullptr */)const {
1216     return get_view(channel, timepoint, angle, resolution, version)
1217         .read_image<T>(props);
1218 }
1219
1220 template <cnpts::Scalar T>
1221 void Connection::write_block(const i3d::Image3d<T>& src,
1222                             i3d::Vector3d<int> coord,
1223                             i3d::Vector3d<int> src_offset,
1224                             int channel,
1225                             int timepoint,
1226                             int angle,
1227                             i3d::Vector3d<int> resolution,
1228                             const std::string& version,
1229                             dataset_props_ptr props /* = nullptr */)const {
1230     get_view(channel, timepoint, angle, resolution, version)
1231         .write_block(src, coord, src_offset, props);
1232 }
1233
1234 template <cnpts::Scalar T>
1235 void Connection::write_blocks(
1236     const i3d::Image3d<T>& src,
1237     const std::vector<i3d::Vector3d<int>>& coords,
1238     const std::vector<i3d::Vector3d<int>>& src_offsets,
1239     int channel,
1240     int timepoint,
1241     int angle,
1242     i3d::Vector3d<int> resolution,
1243     const std::string& version,
1244     dataset_props_ptr props /* = nullptr */)const {
1245     get_view(channel, timepoint, angle, resolution, version)
1246         .write_blocks(src, coords, src_offsets, props);
1247 }
1248
1249 template <cnpts::Scalar T>
1250 void Connection::write_image(const i3d::Image3d<T>& img,
1251                             int channel,
1252                             int timepoint,
1253                             int angle,
1254                             i3d::Vector3d<int> resolution,
1255                             const std::string& version,
1256                             dataset_props_ptr props /* = nullptr */)const {
1257     get_view(channel, timepoint, angle, resolution, version)
1258         .write_image(img, props);
1259 }
1260
1261 template <cnpts::Scalar T>
1262 void Connection::write_with_pyramids(
1263     const i3d::Image3d<T>& img,
1264     int channel,
1265     int timepoint,
1266     int angle,
1267     const std::string& version,
1268     SamplingMode m,
1269     dataset_props_ptr props /* = nullptr */)const {
1270     if (!props)
1271         props = get_properties();
1272     write_image(img, channel, timepoint, angle, {1, 1, 1}, version, props);
1273
1274     for (const auto& res : props->get_all_resolutions()) {
1275         if (res == i3d::Vector3d<int>{1, 1, 1})
1276             continue;
1277
1278         i3d::Vector3d<int> new_dim = props->get_img_dimensions(res);
1279         i3d::Image3d<T> cpy;
1280         i3d::Resample(img, cpy, new_dim, m);
1281         write_image(cpy, channel, timepoint, angle, res, version, props);
1282     }
1283 }
1284
1285 } // namespace ds

```

## 4.2 hpc\_ds\_details.hpp

```

1 #pragma once
2 #include "hpc_ds_structs.hpp"
3 #include <Poco/JSON/Object.h>
4 #include <Poco/JSON/Parser.h>

```

```

5 #include <Poco/Net/HTTPClientSession.h>
6 #include <Poco/Net/HTTPMessage.h>
7 #include <Poco/Net/HTTPRequest.h>
8 #include <Poco/Net/HTTPResponse.h>
9 #include <Poco/URI.h>
10 #include <i3d/image3d.h>
11 #include <i3d/vector3d.h>
12 #include <optional>
13 #include <source_location>
14 #include <span>
15 #include <string>
16 #include <type_traits>
17 /* ===== DETAILS HEADERS ===== */
18
19 namespace ds {
20 namespace details {
21 /* Definition of compile settings */
22
23 #ifdef DATASTORE_NDEBUG
24 constexpr inline bool _DEBUG_ = false;
25 #else
26 #ifdef NDEBUG
27 constexpr inline bool _DEBUG_ = false;
28 #else
29 constexpr inline bool _DEBUG_ = true;
30 #endif
31 #endif
32
33 #ifdef DATASTORE_NLOG
34 constexpr inline bool _LOG_ = false;
35 #else
36 constexpr inline bool _LOG_ = _DEBUG_;
37 #endif
38
39 #ifdef DATASTORE_NINFO
40 constexpr inline bool _INFO_ = false;
41 #else
42 constexpr inline bool _INFO_ = _LOG_;
43 #endif
44
45 #ifdef DATASTORE_NWARNING
46 constexpr inline bool _WARNING_ = false;
47 #else
48 constexpr inline bool _WARNING_ = _LOG_;
49 #endif
50
51 inline std::string
52 get_dataset_url(const std::string& ip, int port, const std::string& uuid);
53
54 inline DatasetProperties get_dataset_properties(const std::string& dataset_url);
55
56 inline bool check_block_coords(const std::vector<i3d::Vector3d<int>& coords,
57                               i3d::Vector3d<int> img_dim,
58                               i3d::Vector3d<int> block_dim);
59
60 inline std::vector<i3d::Vector3d<int>>
61 get_intercepted_blocks(i3d::Vector3d<int> start_point,
62                       i3d::Vector3d<int> end_point,
63                       i3d::Vector3d<int> img_dim,
64                       i3d::Vector3d<int> block_dim);
65
66 namespace data_manip {
67 inline int get_block_data_size(i3d::Vector3d<int> block_size,
68                               const std::string& voxel_type);
69
70 inline i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
71                                         i3d::Vector3d<int> block_dim,
72                                         i3d::Vector3d<int> img_dim);
73
74 inline int get_linear_index(i3d::Vector3d<int> coord,
75                             i3d::Vector3d<int> block_dim,
76                             const std::string& voxel_type);
77
78 template <typename T>
79 T get_elem_at(std::span<const char> data,
80              const std::string& voxel_type,
81              int index);
82
83 template <typename T>
84 T get_elem_at(std::span<const char> data,
85              const std::string& voxel_type,
86              i3d::Vector3d<int> coord,
87              i3d::Vector3d<int> block_dim);
88
89 template <typename T>
90 void set_elem_at(std::span<char> data,
91                 const std::string& voxel_type,

```



```

117             int index,
118             T elem);
119
120 template <typename T>
121 void set_elem_at(std::span<char> data,
122                 const std::string& voxel_type,
123                 i3d::Vector3d<int> coord,
124                 i3d::Vector3d<int> block_dim,
125                 T elem);
126
127 template <typename T>
128 void read_data(std::span<const char> data,
129               const std::string& voxel_type,
130               i3d::Image3d<T>& dest,
131               i3d::Vector3d<int> offset);
132
133 template <typename T>
134 void write_data(const i3d::Image3d<T>& src,
135                i3d::Vector3d<int> offset,
136                std::span<char> data,
137                const std::string& voxel_type,
138                i3d::Vector3d<int> block_size);
139 } // namespace data_manip
140
141 namespace log {
142
143 inline void _log(const std::string& msg,
144                 const std::string& type,
145                 const std::source_location& location);
146
147 inline void
148 info(const std::string& msg,
149      const std::source_location& location = std::source_location::current());
150
151 inline void
152 warning(const std::string& msg,
153         const std::source_location& location = std::source_location::current());
154
155 } // namespace log
156
157 /* Helpers to parse Dataset Properties from JSON */
158 namespace props_parser {
159 using namespace Poco::JSON;
160
161 template <cnpts::Basic T>
162 T get_elem(Object::Ptr root, const std::string& name);
163
164 template <cnpts::Vector3d T>
165 T get_elem(Object::Ptr root, const std::string& name);
166
167 template <cnpts::Vector T>
168 T get_elem(Object::Ptr root, const std::string& name);
169
170 template <cnpts::ResolutionUnit T>
171 T get_elem(Object::Ptr root, const std::string& name);
172
173 template <cnpts::Optional T>
174 T get_elem(Object::Ptr root, const std::string& name);
175
176 inline std::vector<std::map<std::string, i3d::Vector3d<int>>>
177 get_resolution_levels(Object::Ptr root);
178
179 } // namespace props_parser
180
181 /* Helpers providing requests functionality */
182 namespace requests {
183
184 inline std::string session_url_request(const std::string& ds_url,
185                                       i3d::Vector3d<int> resolution,
186                                       const std::string& version);
187
188 inline std::pair<std::vector<char>, Poco::Net::HTTPResponse>
189 make_request(const std::string& url,
190             const std::string& type = Poco::Net::HTTPRequest::HTTP_GET,
191             const std::vector<char>& data = {},
192             const std::map<std::string, std::string>& headers = {});
193
194 } // namespace requests
195
196 } // namespace details
197
198 } // namespace ds
199
200 /* ===== IMPLEMENTATION FOLLOWS ===== */
201 namespace ds {
202 namespace details {
203
204 /* inline */ std::string
205 get_dataset_url(const std::string& ip, int port, const std::string& uuid) {
206     std::string out;
207     if (!ip.starts_with("http://"))
208         out = "https://";
209
210 }
211
212 }

```

```

242     return out + fmt::format("{}:{}/datasets/{}", ip, port, uuid);
243 }
244
245 inline DatasetProperties
246 get_dataset_properties(const std::string& dataset_url) {
247     using namespace Poco::JSON;
248
249     /* Fetch JSON from server */
250     auto [data, response] = requests::make_request(dataset_url);
251     std::string json_str(data.begin(), data.end());
252
253     int res_code = response.getStatus();
254     if (res_code != 200)
255         log::warning(fmt::format(
256             "Request ended with code: {}. json may not be valid", res_code));
257
258     log::info("Parsing dataset properties from JSON string");
259     Parser parser;
260     Poco::Dynamic::Var result = parser.parse(json_str);
261
262     DatasetProperties props;
263     auto root = result.extract<Object::Ptr>();
264
265     using namespace props_parser;
266
267     /* Parse elements from JSON */
268
269     props.uuid = get_elem<std::string>(root, "uuid");
270     props.voxel_type = get_elem<std::string>(root, "voxelType");
271     props.dimensions = get_elem<i3d::Vector3d<int>>(root, "dimensions");
272     props.channels = get_elem<int>(root, "channels");
273     props.angles = get_elem<int>(root, "angles");
274     props.transformations =
275         get_elem<std::optional<std::string>>(root, "transformations");
276     props.voxel_unit = get_elem<std::string>(root, "voxelUnit");
277     props.voxel_resolution =
278         get_elem<std::optional<i3d::Vector3d<double>>>(root, "voxelResolution");
279     props.timepoint_resolution =
280         get_elem<std::optional<ResolutionUnit>>(root, "timepointResolution");
281     props.channel_resolution =
282         get_elem<std::optional<ResolutionUnit>>(root, "channelResolution");
283     props.angle_resolution =
284         get_elem<std::optional<ResolutionUnit>>(root, "angleResolution");
285     props.compression = get_elem<std::string>(root, "compression");
286     props.resolution_levels = get_resolution_levels(root);
287     props.versions = get_elem<std::vector<int>>(root, "versions");
288     props.label = get_elem<std::string>(root, "label");
289     props.view_registrations =
290         get_elem<std::optional<std::string>>(root, "viewRegistrations");
291     props.timepoint_ids = get_elem<std::vector<int>>(root, "timepointIds");
292
293     log::info("Parsing has finished");
294     return props;
295 }
296
297 /* inline */ bool
298 check_block_coords(const std::vector<i3d::Vector3d<int>& coords,
299                   i3d::Vector3d<int> img_dim,
300                   i3d::Vector3d<int> block_dim) {
301     /* Act as NOOP if not in debug */
302     if constexpr (!_DEBUG_)
303         return true;
304
305     log::info("Checking validity of given block coordinates");
306
307     for (i3d::Vector3d<int> coord : coords)
308         if (data_manip::get_block_size(coord, block_dim, img_dim) ==
309             i3d::Vector3d(0, 0, 0)) {
310             log::warning(fmt::format(
311                 "Block coordinate {} is out of valid range", to_string(coord)));
312
313             return false;
314         }
315     log::info("Check successfullly finished");
316     return true;
317 }
318
319 /* inline */ std::vector<i3d::Vector3d<int>>
320 get_intercepted_blocks(i3d::Vector3d<int> start_point,
321                       i3d::Vector3d<int> end_point,
322                       i3d::Vector3d<int> img_dim,
323                       i3d::Vector3d<int> block_dim) {
324     assert(!lt(start_point, end_point));
325
326     i3d::Vector3d<int> block_count = (img_dim + block_dim - 1) / block_dim;
327     std::vector<i3d::Vector3d<int>> out;
328

```

```

329     for (int x = 0; x < block_count.x; ++x)
330         for (int y = 0; y < block_count.y; ++y)
331             for (int z = 0; z < block_count.z; ++z) {
332                 i3d::Vector3d<int> coord = {x, y, z};
333                 if (lt(start_point, (coord + 1) * block_dim) &&
334                     lt(coord * block_dim, end_point))
335                     out.push_back(coord);
336             }
337
338     return out;
339 }
340
341 namespace data_manip {
342 /* inline */ int get_block_data_size(i3d::Vector3d<int> block_size,
343                                     const std::string& voxel_type) {
344
345     int elem_size = type_byte_size.at(voxel_type);
346     return block_size.x * block_size.y * block_size.z * elem_size + 12;
347 }
348
349 /* inline */ i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
350                                                i3d::Vector3d<int> block_dim,
351                                                i3d::Vector3d<int> img_dim) {
352     i3d::Vector3d<int> start = (coord * block_dim);
353     i3d::Vector3d<int> end = (coord + 1) * block_dim;
354
355     i3d::Vector3d<int> out;
356     for (int i = 0; i < 3; ++i) {
357         out[i] =
358             std::max(0, std::min(img_dim[i], end[i]) - std::max(start[i], 0));
359     }
360     return out;
361 }
362
363 /* inline */ int get_linear_index(i3d::Vector3d<int> coord,
364                                  i3d::Vector3d<int> block_dim,
365                                  const std::string& voxel_type) {
366     int elem_size = type_byte_size.at(voxel_type);
367
368     return 12 +
369         (coord.z * block_dim.x * block_dim.y + // header_offset
370          coord.y * block_dim.x + // Main axis
371          coord.x) * // secondary axis
372         elem_size; // last axis
373 }
374
375 template <typename T>
376 T get_elem_at(std::span<const char> data,
377               const std::string& voxel_type,
378               int index) {
379     int elem_size = type_byte_size.at(voxel_type);
380
381     std::array<char, sizeof(T)> buffer{};
382     std::copy_n(data.begin() + index, // source start
383                elem_size, // count
384                buffer.end() - elem_size); // dest start
385
386     std::ranges::reverse(buffer);
387
388     return *reinterpret_cast<T*>(&buffer[0]);
389 }
390
391 template <typename T>
392 T get_elem_at(std::span<const char> data,
393               const std::string& voxel_type,
394               i3d::Vector3d<int> coord,
395               i3d::Vector3d<int> block_dim) {
396
397     int index = get_linear_index(coord, block_dim, voxel_type);
398     return get_elem_at<T>(data, voxel_type, index);
399 }
400
401 template <typename T>
402 void set_elem_at(std::span<char> data,
403                  const std::string& voxel_type,
404                  int index,
405                  T elem) {
406     int elem_size = type_byte_size.at(voxel_type);
407
408     auto buffer = *reinterpret_cast<std::array<char, sizeof(T)>*>(&elem);
409     std::ranges::reverse(buffer);
410
411     std::copy_n(buffer.end() - elem_size, // source start
412                elem_size, // count
413                data.begin() + index); // dest start
414 }
415

```

```

416 template <typename T>
417 void set_elem_at(std::span<char> data,
418                 const std::string& voxel_type,
419                 i3d::Vector3d<int> coord,
420                 i3d::Vector3d<int> block_dim,
421                 T elem) {
422     int index = get_linear_index(coord, block_dim, voxel_type);
423     set_elem_at(data, voxel_type, index, elem);
424 }
425
426 template <typename T>
427 void read_data(std::span<const char> data,
428               const std::string& voxel_type,
429               i3d::Image3d<T>& dest,
430               i3d::Vector3d<int> offset) {
431     i3d::Vector3d<int> block_dim;
432     for (int i = 0; i < 3; ++i)
433         block_dim[i] = get_elem_at<int>(data, "uint32", i * 4);
434
435     for (int x = 0; x < block_dim.x; ++x)
436         for (int y = 0; y < block_dim.y; ++y)
437             for (int z = 0; z < block_dim.z; ++z) {
438                 i3d::Vector3d<int> coord{x + offset.x, y + offset.y,
439                                         z + offset.z};
440
441                 if (!(0 <= coord.x && coord.x < int(dest.GetSizeX()) ||
442                     !(0 <= coord.y && coord.y < int(dest.GetSizeY()) ||
443                     !(0 <= coord.z && coord.z < int(dest.GetSizeZ()))))
444                     continue;
445
446                 dest.SetVoxel(coord, get_elem_at<T>(data, voxel_type, {x, y, z},
447                                                         block_dim));
448             }
449 }
450
451 template <typename T>
452 void write_data(const i3d::Image3d<T>& src,
453                i3d::Vector3d<int> offset,
454                std::span<char> data,
455                const std::string& voxel_type,
456                i3d::Vector3d<int> block_size) {
457     set_elem_at(data, "uint32", 0, block_size.x);
458     set_elem_at(data, "uint32", 4, block_size.y);
459     set_elem_at(data, "uint32", 8, block_size.z);
460
461     for (int x = 0; x < block_size.x; ++x)
462         for (int y = 0; y < block_size.y; ++y)
463             for (int z = 0; z < block_size.z; ++z)
464                 set_elem_at(
465                     data, voxel_type, {x, y, z}, block_size,
466                     src.GetVoxel(x + offset.x, y + offset.y, z + offset.z));
467 }
468 } // namespace data_manip
469
470 namespace log {
471     /* inline */ void _log(const std::string& msg,
472                          const std::string& type,
473                          const std::source_location& location) {
474         if constexpr (!_LOG_)
475             return;
476
477         std::cout << fmt::format("{} {} at row {}:{}\n\n", type,
478                                 location.function_name(), location.line());
479     }
480
481     /* inline */ void info(const std::string& msg,
482                          const std::source_location&
483                          location /* = std::source_location::current() */) {
484         if constexpr (!_INFO_)
485             return;
486         _log(msg, "INFO", location);
487     }
488
489     /* inline */ void
490     warning(const std::string& msg,
491            const std::source_location&
492            location /* = std::source_location::current() */) {
493         if constexpr (!_WARNING_)
494             return;
495         _log(msg, "WARNING", location);
496     }
497 } // namespace log
498
499 namespace props_parser {
500
501     template <cnpts::Basic T>
502     T get_elem(Object::Ptr root, const std::string& name) {

```

```

503     if (!root->has(name)) {
504         log::warning(fmt::format("{} was not found", name));
505         return {};
506     }
507     return root->getValue<T>(name);
508 }
509
510 template <cnpts::Vector3d T>
511 T get_elem(Object::Ptr root, const std::string& name) {
512     using V = decltype(T{}.x);
513     if (!root->has(name)) {
514         log::warning(fmt::format("{} were not found", name));
515         return {};
516     }
517
518     Array::Ptr values = root->getArray(name);
519     if (values->size() != 3) {
520         log::warning("Incorrect number of dimensions");
521         return {};
522     }
523
524     T out;
525     for (unsigned i = 0; i < 3; ++i)
526         out[i] = values->getElement<V>(i);
527
528     return out;
529 }
530
531 template <cnpts::Vector T>
532 T get_elem(Object::Ptr root, const std::string& name) {
533     using V = typename T::value_type;
534     if (!root->has(name)) {
535         log::warning(fmt::format("{} were not found", name));
536         return {};
537     }
538
539     Array::Ptr values = root->getArray(name);
540     std::size_t count = values->size();
541
542     T out(count);
543     for (unsigned i = 0; i < count; ++i)
544         out[i] = values->getElement<V>(i);
545
546     return out;
547 }
548
549 template <cnpts::ResolutionUnit T>
550 T get_elem(Object::Ptr root, const std::string& name) {
551     if (!root->has(name)) {
552         log::warning(fmt::format("{} was not found", name));
553         return {};
554     }
555
556     Object::Ptr res_ptr = root->getObject(name);
557     ResolutionUnit res;
558
559     if (res_ptr->has("value")) {
560         res.value = res_ptr->getValue<double>("value");
561     }
562
563     if (res_ptr->has("unit")) {
564         res.unit = res_ptr->getValue<std::string>("unit");
565     }
566
567     return res;
568 }
569
570 template <cnpts::Optional T>
571 T get_elem(Object::Ptr root, const std::string& name) {
572     if (!root->has(name)) {
573         log::warning(fmt::format("{} were not found", name));
574         return {};
575     }
576
577     if (root->isNull(name))
578         return {};
579
580     T out;
581     out = get_elem<typename T::value_type>(root, name);
582     return out;
583 }
584
585 /* inline */ std::vector<std::map<std::string, i3d::Vector3d<int>>>
586 get_resolution_levels(Object::Ptr root) {
587     std::string name = "resolutionLevels";
588
589     if (!root->has(name)) {

```



## 4.3 hpc\_ds\_structs.hpp

```

1 #pragma once
2 #include <array>
3 #include <cassert>
4 #include <fmt/core.h>
5 #include <i3d/image3d.h>
6 #include <i3d/transform.h>
7 #include <i3d/vector3d.h>
8 #include <map>
9 #include <optional>
10 #include <ostream>
11 #include <sstream>
12 #include <stdexcept>
13 #include <string>
14 #include <vector>
15 #include <memory>
16
17 template <typename T, typename U>
18 bool lt(i3d::Vector3d<T> lhs, i3d::Vector3d<U> rhs) {
19     return lhs.x < rhs.x && lhs.y < rhs.y && lhs.z < rhs.z;
20 }
21
22 template <typename T, typename U>
23 requires std::is_integral_v<T> && std::is_integral_v<U>
24 bool eq(i3d::Vector3d<T> lhs, i3d::Vector3d<U> rhs) {
25     for (int i = 0; i < 3; ++i)
26         if (static_cast<long long>(lhs[i]) != static_cast<long long>(rhs[i]))
27             return false;
28     return true;
29 }
30
31 template <typename T, typename U>
32 requires std::is_floating_point_v<T> && std::is_floating_point_v<U>
33 bool eq(i3d::Vector3d<T> lhs, i3d::Vector3d<U> rhs) {
34     for (int i = 0; i < 3; ++i)
35         if (static_cast<long double>(lhs[i]) != static_cast<long double>(rhs[i]))
36             return false;
37     return true;
38 }
39
40 namespace ds {
41
42 using i3d::SamplingMode;
43
44 /* dataset 'voxel_type' to 'byte_size' map*/
45 const inline std::map<std::string, int> type_byte_size{
46     {"uint8", 1}, {"uint16", 2}, {"uint32", 4}, {"uint64", 8}, {"int8", 1},
47     {"int16", 2}, {"int32", 4}, {"int64", 8}, {"float32", 4}, {"float64", 8}};
48
49 /* Maximal legal URL length */
50 constexpr inline std::size_t MAX_URL_LENGTH = 2048;
51
52 class ResolutionUnit {
53 public:
54     double value = 0.0;
55     std::string unit = "";
56
57     friend std::ostream& operator<<(std::ostream& stream,
58                                     const ResolutionUnit& res) {
59         stream << fmt::format("{} {}", res.value, res.unit);
60         return stream;
61     }
62 };
63
64 /* Concepts definitions to make templates more readable */
65 namespace cnpts {
66 template <typename T>
67 concept Scalar = requires(T) {
68     requires std::is_scalar_v<T>;
69 };
70
71 template <typename T>
72 concept Basic = requires(T) {
73     requires Scalar<T> || std::is_same_v<T, std::string>;
74 };
75
76 template <typename T>
77 concept Vector = requires(T) {
78     requires std::is_same_v<std::vector<typename T::value_type>, T>;
79 };
80
81 template <typename T>
82 concept Optional = requires(T) {
83     requires std::is_same_v<std::optional<typename T::value_type>, T>;
84 };
85
86 }
87
88
89

```

```

90 template <typename T>
91 concept Vector3d = requires(T a) {
92     requires std::is_same_v<i3d::Vector3d<decltype(a.x)>, T>;
93     requires Basic<decltype(a.x)>;
94 };
95
96 template <typename T>
97 concept Streamable = requires(T a) {
98     {std::cout << a};
99 };
100
101 template <typename T>
102 concept Map = requires(T) {
103     requires std::is_same_v<
104         std::map<typename T::key_type, typename T::mapped_type>, T>;
105 };
106
107 template <typename T>
108 concept ResolutionUnit = requires(T) {
109     requires std::is_same_v<T, ds::ResolutionUnit>;
110 };
111 } // namespace cnpts
112
113 namespace details {
114
115 template <cnpts::Streamable T>
116 std::string to_string(const T&);
117
118 template <cnpts::Vector T>
119 std::string to_string(const T&);
120
121 template <cnpts::Map T>
122 std::string to_string(const T&);
123
124 template <cnpts::Optional T>
125 std::string to_string(const T&);
126
127 template <cnpts::Streamable T>
128 std::string to_string(const T& val) {
129     std::stringstream ss;
130     ss << val;
131     return ss.str();
132 }
133
134 template <cnpts::Vector T>
135 std::string to_string(const T& vec) {
136     std::stringstream ss;
137     ss << "[";
138
139     const char* delim = ",";
140     for (auto& v : vec) {
141         ss << delim << to_string(v);
142         delim = ", ";
143     }
144
145     ss << "];";
146     return ss.str();
147 }
148
149 template <cnpts::Map T>
150 std::string to_string(const T& map) {
151     std::stringstream ss;
152     ss << "{\n";
153
154     for (const auto& [k, v] : map) {
155         ss << to_string(k) << ": " << to_string(v) << '\n';
156     }
157
158     ss << "}";
159     return ss.str();
160 }
161
162 template <cnpts::Optional T>
163 std::string to_string(const T& val) {
164     if (!val)
165         return "null";
166     return to_string(val.value());
167 }
168
169 } // namespace details
170
171 class DatasetProperties {
172 public:
173     std::string uuid;
174     std::string voxel_type;
175     i3d::Vector3d<int> dimensions;
176     int channels;
177     int angles;

```



```

184     std::optional<std::string> transformations;
185     std::string voxel_unit;
186     std::optional<i3d::Vector3d<double>> voxel_resolution;
187     std::optional<ResolutionUnit> timepoint_resolution;
188     std::optional<ResolutionUnit> channel_resolution;
189     std::optional<ResolutionUnit> angle_resolution;
190     std::string compression;
191     std::vector<std::map<std::string, i3d::Vector3d<int>>> resolution_levels;
192     std::vector<int> versions;
193     std::string label;
194     std::optional<std::string> view_registrations;
195     std::vector<int> timepoint_ids;
196
197     i3d::Vector3d<int>
198     get_block_dimensions(i3d::Vector3d<int> resolution) const {
199         for (const auto& map : resolution_levels)
200             if (map.at("resolutions") == resolution)
201                 return map.at("blockDimensions");
202
203         std::string msg = fmt::format("Resolution {} not found in properties",
204                                     details::to_string(resolution));
205         throw std::out_of_range(msg.c_str());
206     }
207
208     i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
209                                     i3d::Vector3d<int> resolution) const {
210         i3d::Vector3d<int> block_dim = get_block_dimensions(resolution);
211         i3d::Vector3d<int> start = (coord * block_dim);
212         i3d::Vector3d<int> end = (coord + 1) * block_dim;
213
214         i3d::Vector3d<int> out;
215         for (int i = 0; i < 3; ++i) {
216             out[i] = std::max(0, std::min(dimensions[i], end[i]) -
217                                   std::max(start[i], 0));
218         }
219         return out;
220     }
221
222     i3d::Vector3d<int> get_block_count(i3d::Vector3d<int> resolution) const {
223         i3d::Vector3d<int> block_dim = get_block_dimensions(resolution);
224
225         return (dimensions + block_dim - 1) / block_dim;
226     }
227
228     i3d::Vector3d<int> get_img_dimensions(i3d::Vector3d<int> resolution) const {
229         return dimensions / resolution;
230     }
231
232     std::vector<i3d::Vector3d<int>> get_all_resolutions() const {
233         std::vector<i3d::Vector3d<int>> out;
234         for (const auto& map : resolution_levels)
235             out.push_back(map.at("resolutions"));
236         return out;
237     }
238
239     friend std::ostream& operator<<(std::ostream& stream,
240                                    const DatasetProperties& ds) {
241         using details::to_string;
242
243         stream << "UUID: " << ds.uuid << '\n';
244         stream << "voxelType: " << ds.voxel_type << '\n';
245         stream << "dimensions: " << ds.dimensions << '\n';
246         stream << "channels: " << ds.channels << '\n';
247         stream << "angles: " << ds.angles << '\n';
248         stream << "transformations: " << to_string(ds.transformations) << '\n';
249         stream << "voxelUnit: " << ds.voxel_unit << '\n';
250         stream << "voxelResolution: " << to_string(ds.voxel_resolution) << '\n';
251         stream << "timepointResolution: " << to_string(ds.timepoint_resolution)
252             << '\n';
253         stream << "channelResolution: " << to_string(ds.channel_resolution)
254             << '\n';
255         stream << "angleResolution: " << to_string(ds.angle_resolution) << '\n';
256         stream << "compression: " << ds.compression << '\n';
257         stream << "resolutionLevels: " << to_string(ds.resolution_levels)
258             << '\n';
259         stream << "versions: " << to_string(ds.versions) << '\n';
260         stream << "label: " << ds.label << '\n';
261         stream << "viewRegistrations: " << to_string(ds.view_registrations)
262             << '\n';
263         stream << "timepointIds: " << to_string(ds.timepoint_ids) << '\n';
264
265         return stream;
266     }
267 };
268 using dataset_props_ptr = std::shared_ptr<DatasetProperties>;
269
270 } // namespace ds

```



# Index

/home/somik/CBIA/hpc-datastore-cpp/src/hpc\_ds\_api.hpp, [ds::Connection](#), [11](#)  
[27](#) [ds::ImageView](#), [22](#)  
/home/somik/CBIA/hpc-datastore-cpp/src/hpc\_ds\_details.hpp, [read\\_region](#)  
[35](#) [ds::Connection](#), [11](#), [12](#)  
/home/somik/CBIA/hpc-datastore-cpp/src/hpc\_ds\_structs.hpp, [ds::ImageView](#), [22](#), [23](#)  
[43](#)

Connection  
    [ds::Connection](#), [6](#)

[ds::Connection](#), [5](#)  
    [Connection](#), [6](#)  
    [get\\_properties](#), [7](#)  
    [get\\_view](#), [7](#)  
    [read\\_block](#), [7](#), [8](#)  
    [read\\_blocks](#), [9](#), [10](#)  
    [read\\_image](#), [11](#)  
    [read\\_region](#), [11](#), [12](#)  
    [write\\_block](#), [13](#)  
    [write\\_blocks](#), [14](#)  
    [write\\_image](#), [15](#)  
    [write\\_with\\_pyramids](#), [15](#)

[ds::DatasetProperties](#), [16](#)

[ds::ImageView](#), [17](#)  
    [get\\_properties](#), [19](#)  
    [ImageView](#), [19](#)  
    [read\\_block](#), [19](#), [20](#)  
    [read\\_blocks](#), [21](#)  
    [read\\_image](#), [22](#)  
    [read\\_region](#), [22](#), [23](#)  
    [write\\_block](#), [24](#)  
    [write\\_blocks](#), [24](#)  
    [write\\_image](#), [25](#)

[ds::ResolutionUnit](#), [26](#)

[get\\_properties](#)  
    [ds::Connection](#), [7](#)  
    [ds::ImageView](#), [19](#)

[get\\_view](#)  
    [ds::Connection](#), [7](#)

[ImageView](#)  
    [ds::ImageView](#), [19](#)

[read\\_block](#)  
    [ds::Connection](#), [7](#), [8](#)  
    [ds::ImageView](#), [19](#), [20](#)

[read\\_blocks](#)  
    [ds::Connection](#), [9](#), [10](#)  
    [ds::ImageView](#), [21](#)

[read\\_image](#)