HPC-datastore-cpp

Generated by Doxygen 1.9.3

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

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 prealocated 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 tranfser images (in the dataset) 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.

All of the methods accepts arguments that uniquely identifies requested image. At the backend, this class tranfsers 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

ip	IP address of server (`http://` at the beginning is not necessary)
port	Port, where the server is listening for requests
uuid	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

channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	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

T	Scalar used as underlying type for image representation

Parameters

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

T	Scalar used as underlying type for image representation

Parameters

coord	Block coordinate
channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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 prealocated 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

T	Scalar used as underlying type for image representation

Parameters

coords	Block coordinates
dest	Prealocated destination image
dest_offsets	Offsets at wich the corresponding blocks should be saved
channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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 specificaly 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 prealocated image (second overload of read_blocks), however it will eat more RAM.

If in DEBUG, the fucntion 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

T	Scalar used as underlying type for image representation

Parameters

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

Parameters

resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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

T	Scalar used as underlying type for image representation

Parameters

channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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 neccessary blocks intersecting with chosen region from the server and insert region into preallocated image <dest> at <offset>.

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

Template Parameters

T	Scalar used as underlying type for image representation

Parameters

start_point	smallest point of the region
end_point	largest point of the region
dest	destination image
offset	offset to destination image
channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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 neccessary blocks intersecting with chosen region from the server. It is neccessary, that start_point < end_point (elem-wise)..

Template Parameters

T	Scalar used as underlying type for image representation

Parameters

start_point	smallest point of the region
end_point	largest point of the region
channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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 wheter the offset specified for block is within image boundaries.

Template Parameters

T	Scalar used as underlying type for image representation

Parameters

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

T	Scalar used as underlying type for image representation

Parameters

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

Parameters

angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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

T	Scalar used as underlying type for image representation

Parameters

img	Source image
channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
props	[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

T	Scalar used as underlying type for image representation

Parameters

img	Input image in original resolution
channel	Channel, at which the image is located
timepoint	Timepoint, at which the image is located
angle	Angle, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")
m	Sampling mode used for image resampling
props	[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 prealocated 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

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

T	Scalar used as underlying type for image representation

Parameters

coord	Block coordinate
props	[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

T	Scalar used as underlying type for image representation

Parameters

coord	Block coordinate
dest	Image to write data to
dest_offset	Offset by which the corresponding write should be moved
props	[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 specificaly 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 prealocated image (second overload of read_blocks), however it will eat more RAM.

If in DEBUG, the fucntion 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

T	Scalar used as underlying type for image representation

Parameters

coords	Block coordinates
props	[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 prealocated 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

T	Scalar used as underlying type for image representation

Parameters

coords	Block coordinates
dest	Prealocated destination image
offsets	Offsets at wich the corresponding blocks should be saved
props	[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

T	Scalar used as underlying type for image representation

Parameters

props	[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 neccessary blocks intersecting with chosen region from the server. It is neccessary, that start_point < end_point (elem-wise)..

Template Parameters

T	Scalar used as underlying type for image representation

Parameters

start_point	smallest point of the region
end_point	largest point of the region
props	[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 neccessary blocks intersecting with chosen region from the server and insert region into preallocated image <dest> at <offset>.

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

Template Parameters

T	Scalar used as underlying type for image representation

Parameters

start_point	smallest point of the region
end_point	largest point of the region
dest	destination image
offset	offset to destination image
props	[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

T	Scalar used as underlying type for image representation

Parameters

src	Source image to collect block from
coord	Block coordinates
src_offset	Offset of given block in source image
props	[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 wheter the offsets specified for each block is within image boundaries.

Template Parameters

T	Scalar used as underlying type for image representation

Parameters

src	Source image to collect blocks from
coords	Vector of block coordinates
src_offsets	Offsets of corresponding blocks in source image
props	[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

T	Scalar used as underlying type for image representation

Parameters

img	Source image
props	[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 {
25 inline dataset_props_ptr get_dataset_properties(const std::string& ip,
26                                                 int port,
27                                                 const std::string& uuid);
28
51 template <cnpts::Scalar T>
52 i3d::Image3d<T> read_image(const std::string& ip,
53                            int port,
54                            const std::string& uuid,
55                            int channel = 0,
56                            int timepoint = 0,
57                            int angle = 0,
58                            i3d::Vector3d<int> resolution = {1, 1, 1},
59                            const std::string& version = "latest",
60                            dataset_props_ptr props = nullptr);
61
80 template <cnpts::Scalar T>
81 void write_image(const i3d::Image3d<T>& img,
82                  const std::string& ip,
83                  int port,
84                  const std::string& uuid,
85                  int channel = 0,
86                  int timepoint = 0,
87                  int angle = 0,
88                  i3d::Vector3d<int> resolution = {1, 1, 1},
89                  const std::string version = "latest",
90                  dataset_props_ptr props = nullptr);
91
113 template <cnpts::Scalar T>
114 void write_with_pyramids(const i3d::Image3d<T>& img,
115                          const std::string& ip,
116                          int port,
117                          const std::string& uuid,
118                          int channel = 0,
119                          int timepoint = 0,
120                          int angle = 0,
121                          const std::string& version = "latest",
122                          SamplingMode m = SamplingMode::NEAREST_NEIGHBOUR,
123                          dataset_props_ptr props = nullptr);
124
134 class ImageView {
135   public:
151     ImageView(std::string ip,
152               int port,
153               std::string uuid,
154               int channel,
155               int timepoint,
```

```
156             int angle,
157             i3d::Vector3d<int> resolution,
158             std::string version);
159
165     dataset_props_ptr get_properties() const;
166
184     template <cnpts::Scalar T>
185     i3d::Image3d<T> read_block(i3d::Vector3d<int> coord,
186                                dataset_props_ptr props = nullptr) const;
187
205     template <cnpts::Scalar T>
206     void read_block(i3d::Vector3d<int> coord,
207                     i3d::Image3d<T>& dest,
208                     i3d::Vector3d<int> dest_offset = {0, 0, 0},
209                     dataset_props_ptr props = nullptr) const;
210
237     template <cnpts::Scalar T>
238     std::vector<i3d::Image3d<T>>
239     read_blocks(const std::vector<i3d::Vector3d<int>& coords,
240                 dataset_props_ptr props = nullptr) const;
241
257     template <cnpts::Scalar T>
258     void read_blocks(const std::vector<i3d::Vector3d<int>& coords,
259                      i3d::Image3d<T>& dest,
260                      const std::vector<i3d::Vector3d<int>& offsets,
261                      dataset_props_ptr props = nullptr) const;
262
275     template <cnpts::Scalar T>
276     i3d::Image3d<T> read_region(i3d::Vector3d<int> start_point,
277                                 i3d::Vector3d<int> end_point,
278                                 dataset_props_ptr props = nullptr) const;
279
295     template <cnpts::Scalar T>
296     void read_region(i3d::Vector3d<int> start_point,
297                      i3d::Vector3d<int> end_point,
298                      i3d::Image3d<T>& dest,
299                      i3d::Vector3d<int> offset = {0, 0, 0},
300                      dataset_props_ptr props = nullptr) const;
301
315     template <cnpts::Scalar T>
316     i3d::Image3d<T> read_image(dataset_props_ptr props = nullptr) const;
317
334     template <cnpts::Scalar T>
335     void write_block(const i3d::Image3d<T>& src,
336                      i3d::Vector3d<int> coord,
337                      i3d::Vector3d<int> src_offset = {0, 0, 0},
338                      dataset_props_ptr props = nullptr) const;
355     template <cnpts::Scalar T>
356     void write_blocks(const i3d::Image3d<T>& src,
357                       const std::vector<i3d::Vector3d<int>& coords,
358                       const std::vector<i3d::Vector3d<int>& src_offsets,
359                       dataset_props_ptr props = nullptr) const;
360
376     template <cnpts::Scalar T>
377     void write_image(const i3d::Image3d<T>& img,
378                      dataset_props_ptr props = nullptr) const;
379
380   private:
381     std::string _ip;
382     int _port;
383     std::string _uuid;
384     int _channel;
385     int _timepoint;
386     int _angle;
387     i3d::Vector3d<int> _resolution;
388     std::string _version;
389 };
390
403 class Connection {
404   public:
413     Connection(std::string ip, int port, std::string uuid);
414
426     ImageView get_view(int channel,
427                        int timepoint,
428                        int angle,
429                        i3d::Vector3d<int> resolution,
430                        const std::string& version) const;
431
437     dataset_props_ptr get_properties() const;
438
460     template <cnpts::Scalar T>
461     i3d::Image3d<T> read_block(i3d::Vector3d<int> coord,
462                                int channel,
463                                int timepoint,
464                                int angle,
465                                i3d::Vector3d<int> resolution,
466                                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
534     template <cnpts::Scalar T>
535     std::vector<i3d::Image3d<T>>
536     read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
537                 int channel,
538                 int timepoint,
539                 int angle,
540                 i3d::Vector3d<int> resolution,
541                 const std::string& version,
542                 dataset_props_ptr props = nullptr) const;
543
567     template <cnpts::Scalar T>
568     void read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
569                      i3d::Image3d<T>& dest,
570                      const std::vector<i3d::Vector3d<int>>& dest_offsets,
571                      int channel,
572                      int timepoint,
573                      int angle,
574                      i3d::Vector3d<int> resolution,
575                      const std::string& version,
576                      dataset_props_ptr props = nullptr) const;
595     template <cnpts::Scalar T>
596     i3d::Image3d<T> read_region(i3d::Vector3d<int> start_point,
597                                 i3d::Vector3d<int> end_point,
598                                 int channel,
599                                 int timepoint,
600                                 int angle,
601                                 i3d::Vector3d<int> resolution,
602                                 const std::string& version,
603                                 dataset_props_ptr props = nullptr) const;
604
626     template <cnpts::Scalar T>
627     void read_region(i3d::Vector3d<int> start_point,
628                      i3d::Vector3d<int> end_point,
629                      i3d::Image3d<T>& dest,
630                      i3d::Vector3d<int> offset,
631                      int channel,
632                      int timepoint,
633                      int angle,
634                      i3d::Vector3d<int> resolution,
635                      const std::string& version,
636                      dataset_props_ptr props = nullptr) const;
637
657     template <cnpts::Scalar T>
658     i3d::Image3d<T> read_image(int channel,
659                                int timepoint,
660                                int angle,
661                                i3d::Vector3d<int> resolution,
662                                const std::string& version,
663                                dataset_props_ptr props = nullptr) const;
664
687     template <cnpts::Scalar T>
688     void write_block(const i3d::Image3d<T>& src,
689                      i3d::Vector3d<int> coord,
690                      i3d::Vector3d<int> src_offset,
691                      int channel,
692                      int timepoint,
693                      int angle,
694                      i3d::Vector3d<int> resolution,
695                      const std::string& version,
696                      dataset_props_ptr props = nullptr) const;
697
720     template <cnpts::Scalar T>
721     void write_blocks(const i3d::Image3d<T>& src,
722                       const std::vector<i3d::Vector3d<int>>& coords,
723                       const std::vector<i3d::Vector3d<int>>& src_offsets,
724                       int channel,
725                       int timepoint,
726                       int angle,
727                       i3d::Vector3d<int> resolution,
728                       const std::string& version,
729                       dataset_props_ptr props = nullptr) const;
730
752     template <cnpts::Scalar T>
753     void write_image(const i3d::Image3d<T>& img,
754                      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
781     template <cnpts::Scalar T>
782     void write_with_pyramids(const i3d::Image3d<T>& img,
783                              int channel,
784                              int timepoint,
785                              int angle,
786                              const std::string& version,
787                              SamplingMode m,
788                              dataset_props_ptr props = nullptr) const;
789
790   private:
791     std::string _ip;
792     int _port;
793     std::string _uuid;
794 };
795
796 } // namespace ds
797
798 /* ================ IMPLEMENTATION FOLLOWS ======================== */
799
800 namespace ds {
801 /* ==================================== Global space */
802 /* inline */ dataset_props_ptr get_dataset_properties(const std::string& ip,
803                                                       int port,
804                                                       const std::string& uuid) {
805     std::string dataset_url = details::get_dataset_url(ip, port, uuid);
806     return std::make_shared<DatasetProperties>(
807         details::get_dataset_properties(dataset_url));
808 }
809
810 template <cnpts::Scalar T>
811 i3d::Image3d<T> read_image(const std::string& ip,
812                            int port,
813                            const std::string& uuid,
814                            int channel /* = 0 */,
815                            int timepoint /*  = 0 */,
816                            int angle /* = 0 */,
817                            i3d::Vector3d<int> resolution /* = {1, 1, 1} */,
818                            const std::string& version /* = "latest" */,
819                            dataset_props_ptr props /* = nullptr */) {
820     return ImageView(ip, port, uuid, channel, timepoint, angle, resolution,
821                      version)
822         .read_image<T>(props);
823 }
824
825 template <cnpts::Scalar T>
826 void write_image(const i3d::Image3d<T>& img,
827                  const std::string& ip,
828                  int port,
829                  const std::string& uuid,
830                  int channel /* = 0 */,
831                  int timepoint /* = 0 */,
832                  int angle /* = 0 */,
833                  i3d::Vector3d<int> resolution /*  = {1, 1, 1} */,
834                  const std::string version /* = "latest" */,
835                  dataset_props_ptr props /* = nullptr */) {
836     ImageView(ip, port, uuid, channel, timepoint, angle, resolution, version)
837         .write_image(img, props);
838 }
839
840 template <cnpts::Scalar T>
841 void write_with_pyramids(const i3d::Image3d<T>& img,
842                          const std::string& ip,
843                          int port,
844                          const std::string& uuid,
845                          int channel /* = 0 */,
846                          int timepoint /* = 0 */,
847                          int angle /* = 0 */,
848                          const std::string& version /* = "latest"*/,
849                          SamplingMode m /* = SamplingMode::NEAREST_NEIGHBOUR */,
850                          dataset_props_ptr props /* = nullptr */) {
851     Connection(ip, port, uuid)
852         .write_with_pyramids(img, channel, timepoint, angle, version, m, props);
853 }
854
855 /* ==================================== ImageView */
856
857 ImageView::ImageView(std::string ip,
858                      int port,
859                      std::string uuid,
860                      int channel,
861                      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 template <cnpts::Scalar T>
918 void ImageView::read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
919                             i3d::Image3d<T>& dest,
920                             const std::vector<i3d::Vector3d<int>>& offsets,
921                             dataset_props_ptr props /* = nullptr */
922 )const {
923
924     if (!props)
925         props = get_properties();
926
927     /* Fetched properties from server */
928     std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);
929     i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
930
931     i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);
932
933     if (coords.size() != offsets.size())
934         throw std::logic_error("Count of coordinates != count of offsets");
935
936     if (!details::check_block_coords(coords, img_dim, block_dim))
937         throw std::out_of_range("Blocks out of range");
938
939     /* prepare request url */
940     std::string session_url = details::requests::session_url_request(
941         dataset_url, _resolution, _version);
942
943     if (session_url.ends_with('/'))
944         session_url.pop_back();
945
946     std::vector<std::pair<std::string, std::vector<std::size_t>>> requests =
947         details::create_requests(coords, session_url, _timepoint, _channel,
948                                  _angle);
```

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

```
1036     /* Fetch server properties */
1037     std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);
1038
1039     i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
1040     i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);
1041
1042     /* Error checking (when not in debug, all checks automatically return
1043 * true)*/
1044     if (coords.size() != src_offsets.size())
1045         throw std::logic_error("Count of coordinates != count of offsets");
1046
1047     if (!details::check_block_coords(coords, img_dim, block_dim))
1048         throw std::out_of_range("Blocks out of range");
1049
1050     /* prepare request url */
1051     std::string session_url = details::requests::session_url_request(
1052         dataset_url, _resolution, _version);
1053
1054     if (session_url.ends_with('/'))
1055         session_url.pop_back();
1056
1057     std::vector<std::pair<std::string, std::vector<std::size_t>> requests =
1058         details::create_requests(coords, session_url, _timepoint, _channel,
1059                                  _angle, 256);
1060
1061     for (const auto& [req, idxs] :  requests) {
1062         std::size_t full_size = 0;
1063         for (std::size_t i :  idxs)
1064             full_size += details::data_manip::get_block_data_size(
1065                 props->get_block_size(coords[i], _resolution),
1066                 props->voxel_type);
1067
1068         /* Prepare vector representing octet-data (will be send to server) */
1069         std::vector<char> data(full_size);
1070
1071         /* Transform image to octet-data */
1072         std::size_t start_i = 0;
1073         for (std::size_t i :  idxs) {
1074             i3d::Vector3d<int> block_size =
1075                 props->get_block_size(coords[i], _resolution);
1076             std::size_t data_size = details::data_manip::get_block_data_size(
1077                 block_size, props->voxel_type);
1078
1079             details::data_manip::write_data(
1080                 src, src_offsets[i],
1081                 std::span(data.begin() + start_i, data_size),
1082                 props->voxel_type, block_size);
1083
1084             start_i += data_size;
1085         }
1086
1087         auto [_, response] = details::requests::make_request(
1088             req, Poco::Net::HTTPRequest::HTTP_POST, data,
1089             {{"Content-Type", "application/octet-stream"}});
1090     }
1091 }
1092
1093 template <cnpts::Scalar T>
1094 void ImageView::write_image(const i3d::Image3d<T>& img,
1095                             dataset_props_ptr props /* = nullptr */)const {
1096
1097     /* Fetch image properties from server */
1098     if (!props)
1099         props = get_properties();
1100
1101     i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
1102     i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);
1103     i3d::Vector3d<int> block_count =
1104         (img_dim + block_dim - 1) / block_dim; // Ceiling
1105
1106     /* Prepare coordinates of blocks and offsets to write whole image */
1107     std::vector<i3d::Vector3d<int>> blocks;
1108     std::vector<i3d::Vector3d<int>> offsets;
1109
1110     for (int x = 0; x < block_count.x; ++x)
1111         for (int y = 0; y < block_count.y; ++y)
1112             for (int z = 0; z < block_count.z; ++z) {
1113                 blocks.emplace_back(x, y, z);
1114                 offsets.emplace_back(x * block_dim.x, y * block_dim.y,
1115                                      z * block_dim.z);
1116             }
1117
1118     /* write whole image */
1119     write_blocks(img, blocks, offsets, props);
1120 }
1121
1122 /* ================================== Connection */
```

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

```
1210 void Connection::read_region(i3d::Vector3d<int> start_point,
1211                             i3d::Vector3d<int> end_point,
1212                             i3d::Image3d<T>& dest,
1213                             i3d::Vector3d<int> offset,
1214                             int channel,
1215                             int timepoint,
1216                             int angle,
1217                             i3d::Vector3d<int> resolution,
1218                             const std::string& version,
1219                             dataset_props_ptr props /* = nullptr */)const {
1220     get_view(channel, timepoint, angle, resolution, version)
1221         .read_region<T>(start_point, end_point, dest, offset, props);
1222 }
1223
1224 template <cnpts::Scalar T>
1225 i3d::Image3d<T>
1226 Connection::read_image(int channel,
1227                        int timepoint,
1228                        int angle,
1229                        i3d::Vector3d<int> resolution,
1230                        const std::string& version,
1231                        dataset_props_ptr props /* = nullptr */)const {
1232     return get_view(channel, timepoint, angle, resolution, version)
1233         .read_image<T>(props);
1234 }
1235
1236 template <cnpts::Scalar T>
1237 void Connection::write_block(const i3d::Image3d<T>& src,
1238                             i3d::Vector3d<int> coord,
1239                             i3d::Vector3d<int> src_offset,
1240                             int channel,
1241                             int timepoint,
1242                             int angle,
1243                             i3d::Vector3d<int> resolution,
1244                             const std::string& version,
1245                             dataset_props_ptr props /* = nullptr */)const {
1246     get_view(channel, timepoint, angle, resolution, version)
1247         .write_block(src, coord, src_offset, props);
1248 }
1249
1250 template <cnpts::Scalar T>
1251 void Connection::write_blocks(
1252     const i3d::Image3d<T>& src,
1253     const std::vector<i3d::Vector3d<int>>& coords,
1254     const std::vector<i3d::Vector3d<int>>& src_offsets,
1255     int channel,
1256     int timepoint,
1257     int angle,
1258     i3d::Vector3d<int> resolution,
1259     const std::string& version,
1260     dataset_props_ptr props /* = nullptr */)const {
1261     get_view(channel, timepoint, angle, resolution, version)
1262         .write_blocks(src, coords, src_offsets, props);
1263 }
1264
1265 template <cnpts::Scalar T>
1266 void Connection::write_image(const i3d::Image3d<T>& img,
1267                             int channel,
1268                             int timepoint,
1269                             int angle,
1270                             i3d::Vector3d<int> resolution,
1271                             const std::string& version,
1272                             dataset_props_ptr props /* = nullptr */)const {
1273     get_view(channel, timepoint, angle, resolution, version)
1274         .write_image(img, props);
1275 }
1276
1277 template <cnpts::Scalar T>
1278 void Connection::write_with_pyramids(
1279     const i3d::Image3d<T>& img,
1280     int channel,
1281     int timepoint,
1282     int angle,
1283     const std::string& version,
1284     SamplingMode m,
1285     dataset_props_ptr props /* = nullptr */)const {
1286     if (!props)
1287         props = get_properties();
1288     write_image(img, channel, timepoint, angle, {1, 1, 1}, version, props);
1289
1290     for (const auto& res :  props->get_all_resolutions()) {
1291         if (res == i3d::Vector3d<int>{1, 1, 1})
1292             continue;
1293
1294         i3d::Vector3d<int> new_dim = props->get_img_dimensions(res);
1295         i3d::Image3d<T> cpy;
1296         i3d::Resample(img, cpy, new_dim, m);
```

```
1297         write_image(cpy, channel, timepoint, angle, res, version, props);
1298     }
1299 }
1300
1301 } // 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
59 inline std::string
60 get_dataset_url(const std::string& ip, int port, const std::string& uuid);
61
68 inline DatasetProperties get_dataset_properties(const std::string& dataset_url);
69
81 inline bool check_block_coords(const std::vector<i3d::Vector3d<int>& coords,
82                                i3d::Vector3d<int> img_dim,
83                                i3d::Vector3d<int> block_dim);
84
85 inline std::vector<i3d::Vector3d<int>
86 get_intercepted_blocks(i3d::Vector3d<int> start_point,
87                        i3d::Vector3d<int> end_point,
88                        i3d::Vector3d<int> img_dim,
89                        i3d::Vector3d<int> block_dim);
90
101 inline std::vector<std::pair<std::string, std::vector<std::size_t>>>
102 create_requests(const std::vector<i3d::Vector3d<int>& coords,
103                 const std::string& session_url,
104                 int timepoint,
105                 int channel,
106                 int angle,
107                 std::size_t max_request_size = MAX_URL_LENGTH);
108
109 namespace data_manip {
110 inline int get_block_data_size(i3d::Vector3d<int> block_size,
```

```
111                                const std::string& voxel_type);
112
113 inline i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
114                                          i3d::Vector3d<int> block_dim,
115                                          i3d::Vector3d<int> img_dim);
116
117 inline int get_linear_index(i3d::Vector3d<int> coord,
118                             i3d::Vector3d<int> block_dim,
119                             const std::string& voxel_type);
120
121 template <typename T>
122 T get_elem_at(std::span<const char> data,
123               const std::string& voxel_type,
124               int index);
125
126 template <typename T>
127 T get_elem_at(std::span<const char> data,
128               const std::string& voxel_type,
129               i3d::Vector3d<int> coord,
130               i3d::Vector3d<int> block_dim);
131
132 template <typename T>
133 void set_elem_at(std::span<char> data,
134                  const std::string& voxel_type,
135                  int index,
136                  T elem);
137
138 template <typename T>
139 void set_elem_at(std::span<char> data,
140                  const std::string& voxel_type,
141                  i3d::Vector3d<int> coord,
142                  i3d::Vector3d<int> block_dim,
143                  T elem);
144
154 template <typename T>
155 void read_data(std::span<const char> data,
156                const std::string& voxel_type,
157                i3d::Image3d<T>& dest,
158                i3d::Vector3d<int> offset);
159
170 template <typename T>
171 void write_data(const i3d::Image3d<T>& src,
172                 i3d::Vector3d<int> offset,
173                 std::span<char> data,
174                 const std::string& voxel_type,
175                 i3d::Vector3d<int> block_size);
176 } // namespace data_manip
177
178 namespace log {
179
187 inline void _log(const std::string& msg,
188                  const std::string& type,
189                  const std::source_location& location);
190
197 inline void
198 info(const std::string& msg,
199      const std::source_location& location = std::source_location::current());
200
207 inline void
208 warning(const std::string& msg,
209         const std::source_location& location = std::source_location::current());
210
211 } // namespace log
212 /* Helpers to parse Dataset Properties from JSON */
213 namespace props_parser {
214 using namespace Poco::JSON;
215
216 template <cnpts::Basic T>
217 T get_elem(Object::Ptr root, const std::string& name);
218
219 template <cnpts::Vector3d T>
220 T get_elem(Object::Ptr root, const std::string& name);
221
222 template <cnpts::Vector T>
223 T get_elem(Object::Ptr root, const std::string& name);
224
225 template <cnpts::ResolutionUnit T>
226 T get_elem(Object::Ptr root, const std::string& name);
227
228 template <cnpts::Optional T>
229 T get_elem(Object::Ptr root, const std::string& name);
230
231 inline std::vector<std::map<std::string, i3d::Vector3d<int>>>
232 get_resolution_levels(Object::Ptr root);
233
234 } // namespace props_parser
235
```

```
236 /* Helpers providing requests functionality */
237 namespace requests {
238 inline std::string session_url_request(const std::string& ds_url,
239                                        i3d::Vector3d<int> resolution,
240                                        const std::string& version);
241
242 inline std::pair<std::vector<char>, Poco::Net::HTTPResponse>
243 make_request(const std::string& url,
244              const std::string& type = Poco::Net::HTTPRequest::HTTP_GET,
245              const std::vector<char>& data = {},
246              const std::map<std::string, std::string>& headers = {});
247 } // namespace requests
248 } // namespace details
249 } // namespace ds
250
251 /* ================ IMPLEMENTATION FOLLOWS ======================== */
252 namespace ds {
253 namespace details {
254
255 /* inline */ std::string
256 get_dataset_url(const std::string& ip, int port, const std::string& uuid) {
257     std::string out;
258     if (!ip.starts_with("http://"))
259         out = "https://";
260     return out + fmt::format("{}:{}/datasets/{}", ip, port, uuid);
261 }
262
263 inline DatasetProperties
264 get_dataset_properties(const std::string& dataset_url) {
265     using namespace Poco::JSON;
266
267     /* Fetch JSON from server */
268     auto [data, response] = requests::make_request(dataset_url);
269     std::string json_str(data.begin(), data.end());
270
271     int res_code = response.getStatus();
272     if (res_code != 200)
273         log::warning(fmt::format(
274             "Request ended with code:  {}.  json may not be valid", res_code));
275
276     log::info("Parsing dataset properties from JSON string");
277     Parser parser;
278     Poco::Dynamic::Var result = parser.parse(json_str);
279
280     DatasetProperties props;
281     auto root = result.extract<Object::Ptr>();
282
283     using namespace props_parser;
284
285     /* Parse elements from JSON */
286
287     props.uuid = get_elem<std::string>(root, "uuid");
288     props.voxel_type = get_elem<std::string>(root, "voxelType");
289     props.dimensions = get_elem<i3d::Vector3d<int»(root, "dimensions");
290     props.channels = get_elem<int>(root, "channels");
291     props.angles = get_elem<int>(root, "angles");
292     props.transformations =
293         get_elem<std::optional<std::string»(root, "transformations");
294     props.voxel_unit = get_elem<std::string>(root, "voxelUnit");
295     props.voxel_resolution =
296         get_elem<std::optional<i3d::Vector3d<double»>(root, "voxelResolution");
297     props.timepoint_resolution =
298         get_elem<std::optional<ResolutionUnit»(root, "timepointResolution");
299     props.channel_resolution =
300         get_elem<std::optional<ResolutionUnit»(root, "channelResolution");
301     props.angle_resolution =
302         get_elem<std::optional<ResolutionUnit»(root, "angleResolution");
303     props.compression = get_elem<std::string>(root, "compression");
304     props.resolution_levels = get_resolution_levels(root);
305     props.versions = get_elem<std::vector<int»(root, "versions");
306     props.label = get_elem<std::string>(root, "label");
307     props.view_registrations =
308         get_elem<std::optional<std::string»(root, "viewRegistrations");
309     props.timepoint_ids = get_elem<std::vector<int»(root, "timepointIds");
310
311     log::info("Parsing has finished");
312     return props;
313 }
314
315 /* inline */ bool
316 check_block_coords(const std::vector<i3d::Vector3d<int»& coords,
317                    i3d::Vector3d<int> img_dim,
318                    i3d::Vector3d<int> block_dim) {
319     /* Act as NOOP if not in debug */
320     if constexpr (!_DEBUG_)
321         return true;
322
```

```
323     log::info("Checking validity of given block coordinates");
324
325     for (i3d::Vector3d<int> coord :  coords)
326         if (data_manip::get_block_size(coord, block_dim, img_dim) ==
327             i3d::Vector3d(0, 0, 0)) {
328             log::warning(fmt::format(
329                 "Block coordinate {} is out of valid range", to_string(coord)));
330
331             return false;
332         }
333     log::info("Check successfullly finished");
334     return true;
335 }
336
337 /* inline */ std::vector<i3d::Vector3d<int>>
338 get_intercepted_blocks(i3d::Vector3d<int> start_point,
339                        i3d::Vector3d<int> end_point,
340                        i3d::Vector3d<int> img_dim,
341                        i3d::Vector3d<int> block_dim) {
342     assert(lt(start_point, end_point));
343
344     i3d::Vector3d<int> block_count = (img_dim + block_dim - 1) / block_dim;
345     std::vector<i3d::Vector3d<int>> out;
346
347     for (int x = 0; x < block_count.x; ++x)
348         for (int y = 0; y < block_count.y; ++y)
349             for (int z = 0; z < block_count.z; ++z) {
350                 i3d::Vector3d<int> coord = {x, y, z};
351                 if (lt(start_point, (coord + 1) * block_dim) &&
352                     lt(coord * block_dim, end_point))
353                     out.push_back(coord);
354             }
355
356     return out;
357 }
358
359 /* inline */ std::vector<std::pair<std::string, std::vector<std::size_t>>>
360 create_requests(const std::vector<i3d::Vector3d<int>>& coords,
361                 const std::string& session_url,
362                 int timepoint,
363                 int channel,
364                 int angle,
365                 std::size_t max_request_size /* = MAX_URL_LENGTH*/) {
366     std::vector<std::pair<std::string, std::vector<std::size_t>>> out;
367
368     std::string final_url = session_url;
369     std::vector<std::size_t> indexes;
370
371     for (std::size_t i = 0; i < coords.size(); ++i) {
372         const auto& coord = coords[i];
373         std::string to_append =
374             fmt::format("/{}/{}/{}/{}/{}/{}", coord.x, coord.y, coord.z,
375                         timepoint, channel, angle);
376
377         if (final_url.size() + to_append.size() > max_request_size) {
378             out.emplace_back(final_url, indexes);
379             final_url = session_url;
380             indexes.clear();
381         }
382
383         final_url += to_append;
384         indexes.push_back(i);
385     }
386
387     if (!indexes.empty()) {
388         out.emplace_back(final_url, indexes);
389     }
390
391     return out;
392 }
393
394 namespace data_manip {
395 /* inline */ int get_block_data_size(i3d::Vector3d<int> block_size,
396                                      const std::string& voxel_type) {
397
398     int elem_size = type_byte_size.at(voxel_type);
399     return block_size.x * block_size.y * block_size.z * elem_size + 12;
400 }
401
402 /* inline */ i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
403                                                i3d::Vector3d<int> block_dim,
404                                                i3d::Vector3d<int> img_dim) {
405     i3d::Vector3d<int> start = (coord * block_dim);
406     i3d::Vector3d<int> end = (coord + 1) * block_dim;
407
408     i3d::Vector3d<int> out;
409     for (int i = 0; i < 3; ++i) {
```

```
410         out[i] =
411             std::max(0, std::min(img_dim[i], end[i]) - std::max(start[i], 0));
412     }
413     return out;
414 }
415
416 /* inline */ int get_linear_index(i3d::Vector3d<int> coord,
417                                   i3d::Vector3d<int> block_dim,
418                                   const std::string& voxel_type) {
419     int elem_size = type_byte_size.at(voxel_type);
420
421     return 12 +                                     // header_offset
422            (coord.z * block_dim.x * block_dim.y + // Main axis
423             coord.y * block_dim.x +               // secondary axis
424             coord.x) *                            // last axis
425                elem_size;                         // byte size
426 }
427
428 template <typename T>
429 T get_elem_at(std::span<const char> data,
430               const std::string& voxel_type,
431               int index) {
432     int elem_size = type_byte_size.at(voxel_type);
433
434     std::array<char, sizeof(T)> buffer{};
435     std::copy_n(data.begin() + index,      // source start
436                 elem_size,                 // count
437                 buffer.end() - elem_size); // dest start
438
439     std::ranges::reverse(buffer);
440
441     return *reinterpret_cast<T*>(&buffer[0]);
442 }
443
444 template <typename T>
445 T get_elem_at(std::span<const char> data,
446               const std::string& voxel_type,
447               i3d::Vector3d<int> coord,
448               i3d::Vector3d<int> block_dim) {
449
450     int index = get_linear_index(coord, block_dim, voxel_type);
451     return get_elem_at<T>(data, voxel_type, index);
452 }
453
454 template <typename T>
455 void set_elem_at(std::span<char> data,
456                  const std::string& voxel_type,
457                  int index,
458                  T elem) {
459     int elem_size = type_byte_size.at(voxel_type);
460
461     auto buffer = *reinterpret_cast<std::array<char, sizeof(T)>*>(&elem);
462     std::ranges::reverse(buffer);
463
464     std::copy_n(buffer.end() - elem_size, // source start
465                 elem_size,                // count
466                 data.begin() + index);    // dest start
467 }
468
469 template <typename T>
470 void set_elem_at(std::span<char> data,
471                  const std::string& voxel_type,
472                  i3d::Vector3d<int> coord,
473                  i3d::Vector3d<int> block_dim,
474                  T elem) {
475     int index = get_linear_index(coord, block_dim, voxel_type);
476     set_elem_at(data, voxel_type, index, elem);
477 }
478
479 template <typename T>
480 void read_data(std::span<const char> data,
481                const std::string& voxel_type,
482                i3d::Image3d<T>& dest,
483                i3d::Vector3d<int> offset) {
484     i3d::Vector3d<int> block_dim;
485     for (int i = 0; i < 3; ++i)
486         block_dim[i] = get_elem_at<int>(data, "uint32", i * 4);
487
488     for (int x = 0; x < block_dim.x; ++x)
489         for (int y = 0; y < block_dim.y; ++y)
490             for (int z = 0; z < block_dim.z; ++z) {
491                 i3d::Vector3d<int> coord{x + offset.x, y + offset.y,
492                                          z + offset.z};
493
494                 if (!(0 <= coord.x && coord.x < int(dest.GetSizeX())) ||
495                     !(0 <= coord.y && coord.y < int(dest.GetSizeY())) ||
496                     !(0 <= coord.z && coord.z < int(dest.GetSizeZ())))
```

```
497                     continue;
498
499                 dest.SetVoxel(coord, get_elem_at<T>(data, voxel_type, {x, y, z},
500                                                     block_dim));
501             }
502 }
503
504 template <typename T>
505 void write_data(const i3d::Image3d<T>& src,
506                 i3d::Vector3d<int> offset,
507                 std::span<char> data,
508                 const std::string& voxel_type,
509                 i3d::Vector3d<int> block_size) {
510     set_elem_at(data, "uint32", 0, block_size.x);
511     set_elem_at(data, "uint32", 4, block_size.y);
512     set_elem_at(data, "uint32", 8, block_size.z);
513
514     for (int x = 0; x < block_size.x; ++x)
515         for (int y = 0; y < block_size.y; ++y)
516             for (int z = 0; z < block_size.z; ++z)
517                 set_elem_at(
518                     data, voxel_type, {x, y, z}, block_size,
519                     src.GetVoxel(x + offset.x, y + offset.y, z + offset.z));
520 }
521 } // namespace data_manip
522
523 namespace log {
524 /* inline */ void _log(const std::string& msg,
525                        const std::string& type,
526                        const std::source_location& location) {
527     if constexpr (!_LOG_)
528         return;
529
530     std::cout << fmt::format("[{}] {} at row {}:\n{} \n\n", type,
531                              location.function_name(), location.line(), msg);
532 }
533
534 /* inline */ void info(const std::string& msg,
535                        const std::source_location&
536                            location /* = std::source_location::current() */) {
537     if constexpr (!_INFO_)
538         return;
539     _log(msg, "INFO", location);
540 }
541
542 /* inline */ void
543 warning(const std::string& msg,
544         const std::source_location&
545             location /* = std::source_location::current() */) {
546     if constexpr (!_WARNING_)
547         return;
548     _log(msg, "WARNING", location);
549 }
550 } // namespace log
551
552 namespace props_parser {
553
554 template <cnpts::Basic T>
555 T get_elem(Object::Ptr root, const std::string& name) {
556     if (!root->has(name)) {
557         log::warning(fmt::format("{} was not found", name));
558         return {};
559     }
560     return root->getValue<T>(name);
561 }
562
563 template <cnpts::Vector3d T>
564 T get_elem(Object::Ptr root, const std::string& name) {
565     using V = decltype(T{}.x);
566     if (!root->has(name)) {
567         log::warning(fmt::format("{} were not found", name));
568         return {};
569     }
570
571     Array::Ptr values = root->getArray(name);
572     if (values->size() != 3) {
573         log::warning("Incorrect number of dimensions");
574         return {};
575     }
576
577     T out;
578     for (unsigned i = 0; i < 3; ++i)
579         out[i] = values->getElement<V>(i);
580
581     return out;
582 }
583
```

```
584 template <cnpts::Vector T>
585 T get_elem(Object::Ptr root, const std::string& name) {
586     using V = typename T::value_type;
587     if (!root->has(name)) {
588         log::warning(fmt::format("{} were not found", name));
589         return {};
590     }
591
592     Array::Ptr values = root->getArray(name);
593     std::size_t count = values->size();
594
595     T out(count);
596     for (unsigned i = 0; i < count; ++i)
597         out[i] = values->getElement<V>(i);
598
599     return out;
600 }
601
602 template <cnpts::ResolutionUnit T>
603 T get_elem(Object::Ptr root, const std::string& name) {
604     if (!root->has(name)) {
605         log::warning(fmt::format("{} was not found", name));
606         return {};
607     }
608
609     Object::Ptr res_ptr = root->getObject(name);
610     ResolutionUnit res;
611
612     if (res_ptr->has("value")) {
613         res.value = res_ptr->getValue<double>("value");
614     }
615
616     if (res_ptr->has("unit")) {
617         res.unit = res_ptr->getValue<std::string>("unit");
618     }
619
620     return res;
621 }
622
623 template <cnpts::Optional T>
624 T get_elem(Object::Ptr root, const std::string& name) {
625     if (!root->has(name)) {
626         log::warning(fmt::format("{} were not found", name));
627         return {};
628     }
629
630     if (root->isNull(name))
631         return {};
632
633     T out;
634     out = get_elem<typename T::value_type>(root, name);
635     return out;
636 }
637
638 /* inline */ std::vector<std::map<std::string, i3d::Vector3d<int>>>
639 get_resolution_levels(Object::Ptr root) {
640     std::string name = "resolutionLevels";
641
642     if (!root->has(name)) {
643         log::warning("resolutionLevels were not found");
644         return {};
645     }
646
647     std::vector<std::map<std::string, i3d::Vector3d<int>>> out;
648
649     Array::Ptr array = root->getArray(name);
650     for (unsigned i = 0; i < array->size(); ++i) {
651         std::map<std::string, i3d::Vector3d<int>> map;
652         Object::Ptr map_ptr = array->getObject(i);
653
654         for (const auto& name :  map_ptr->getNames()) {
655             map[name] = get_elem<i3d::Vector3d<int>>(map_ptr, name);
656         }
657
658         out.push_back(map);
659     }
660
661     return out;
662 }
663
664 } // namespace props_parser
665
666 namespace requests {
667 /* inline */ std::string session_url_request(const std::string& ds_url,
668                                              i3d::Vector3d<int> resolution,
669                                              const std::string& version) {
670
```

```
671     log::info(
672         fmt::format("Obtaining session url for resolution:  {}, version:  {}",
673                     to_string(resolution), version));
674     std::string req_url =
675         fmt::format("{}/{}/{}/{}/{}/read-write", ds_url, resolution.x,
676                     resolution.y, resolution.z, version);
677
678     auto [_, response] = make_request(req_url);
679
680     int res_code = response.getStatus();
681     if (res_code != 307)
682         log::warning(fmt::format(
683             "Request ended with status:  {}, redirection may be incorrect",
684             res_code));
685
686     return response.get("Location");
687 }
688
689 /* inline */ std::pair<std::vector<char>, Poco::Net::HTTPResponse>
690 make_request(const std::string& url,
691              const std::string& type /*  = Poco::Net::HTTPRequest::HTTP_GET */,
692              const std::vector<char>& data /*  = {} */,
693              const std::map<std::string, std::string>& headers /* = {} */) {
694     Poco::URI uri(url);
695     std::string path(uri.getPathAndQuery());
696
697     Poco::Net::HTTPClientSession session(uri.getHost(), uri.getPort());
698
699     Poco::Net::HTTPRequest request(type, path,
700                                    Poco::Net::HTTPMessage::HTTP_1_1);
701
702     for (auto& [key, value] :  headers)
703         request.set(key, value);
704
705     request.setContentLength(data.size());
706
707     log::info(fmt::format("Sending {} request to url:  {}", type, url));
708     std::ostream& os = session.sendRequest(request);
709     for (char ch :  data)
710         os « ch;
711
712     Poco::Net::HTTPResponse response;
713     std::istream& rs = session.receiveResponse(response);
714
715     std::vector<char> out{std::istreambuf_iterator<char>(rs),
716                           std::istreambuf_iterator<char>()};
717
718     log::info(fmt::format(
719         "Fetched response with status:  {}, reason:  {}, content size:  {}",
720         response.getStatus(), response.getReason(), out.size()));
721
722     return {out, response};
723 }
724
725 } // namespace requests
726 } // namespace details
727 } // namespace ds
```

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
56 class ResolutionUnit {
57   public:
58     double value = 0.0;
59     std::string unit = "";
60
61     friend std::ostream& operator«(std::ostream& stream,
62                                    const ResolutionUnit& res) {
63         stream « fmt::format("{} {}", res.value, res.unit);
64         return stream;
65     }
66 };
67
68 /* Concepts definitions to make templates more readable */
69 namespace cnpts {
70 template <typename T>
71 concept Scalar = requires(T) {
72     requires std::is_scalar_v<T>;
73 };
74
75 template <typename T>
76 concept Basic = requires(T) {
77     requires Scalar<T> || std::is_same_v<T, std::string>;
78 };
79
80 template <typename T>
81 concept Vector = requires(T) {
82     requires std::is_same_v<std::vector<typename T::value_type>, T>;
83 };
84
85 template <typename T>
86 concept Optional = requires(T) {
87     requires std::is_same_v<std::optional<typename T::value_type>, T>;
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
```

```
117 template <cnpts::Streamable T>
118 std::string to_string(const T&);
119
120 template <cnpts::Vector T>
121 std::string to_string(const T&);
122
123 template <cnpts::Map T>
124 std::string to_string(const T&);
125
126 template <cnpts::Optional T>
127 std::string to_string(const T&);
128
130 template <cnpts::Streamable T>
131 std::string to_string(const T& val) {
132     std::stringstream ss;
133     ss « val;
134     return ss.str();
135 }
136
137 template <cnpts::Vector T>
138 std::string to_string(const T& vec) {
139     std::stringstream ss;
140     ss « "(";
141
142     const char* delim = "";
143     for (auto& v :  vec) {
144         ss « delim « to_string(v);
145         delim = ", ";
146     }
147
148     ss « ")";
149     return ss.str();
150 }
151
152 template <cnpts::Map T>
153 std::string to_string(const T& map) {
154     std::stringstream ss;
155     ss « "{\n";
156
157     for (const auto& [k, v] :  map) {
158         ss « to_string(k) « ":  " « to_string(v) « '\n';
159     }
160     ss « "}";
161     return ss.str();
162 }
163
164 template <cnpts::Optional T>
165 std::string to_string(const T& val) {
166     if (!val)
167         return "null";
168     return to_string(val.value());
169 }
170
171 } // namespace details
172
177 class DatasetProperties {
178   public:
179     std::string uuid;
180     std::string voxel_type;
181     i3d::Vector3d<int> dimensions;
182     int channels;
183     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 « "comprestreamion:  " « 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