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

```
#pragma once
#include "hpc_ds_details.hpp"
#include "hpc_ds_structs.hpp"
#include <fmt/core.h>
#include <i3d/image3d.h>
#include <i3d/transform.h>
#include <memory>
#include <string>
#include <type_traits>
#include <vector>

namespace ds {
inline dataset_props_ptr get_dataset_properties(const std::string& ip,
                                                int port,
                                                const std::string& uuid);

template <cnpts::Scalar T>
i3d::Image3d<T> read_image(const std::string& ip,
                           int port,
                           const std::string& uuid,
                           int channel = 0,
                           int timepoint = 0,
                           int angle = 0,
                           i3d::Vector3d<int> resolution = {1, 1, 1},
                           const std::string& version = "latest",
                           dataset_props_ptr props = nullptr);

template <cnpts::Scalar T>
void write_image(const i3d::Image3d<T>& img,
                 const std::string& ip,
                 int port,
                 const std::string& uuid,
                 int channel = 0,
                 int timepoint = 0,
                 int angle = 0,
                 i3d::Vector3d<int> resolution = {1, 1, 1},
                 const std::string version = "latest",
                 dataset_props_ptr props = nullptr);

template <cnpts::Scalar T>
void write_with_pyramids(const i3d::Image3d<T>& img,
                         const std::string& ip,
                         int port,
                         const std::string& uuid,
                         int channel = 0,
                         int timepoint = 0,
                         int angle = 0,
                         const std::string& version = "latest",
                         SamplingMode m = SamplingMode::NEAREST_NEIGHBOUR,
                         dataset_props_ptr props = nullptr);

class ImageView {
  public:
    ImageView(std::string ip,
              int port,
              std::string uuid,
              int channel,
              int timepoint,
              int angle,
              i3d::Vector3d<int> resolution,
              std::string version);

    dataset_props_ptr get_properties() const;

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

    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;

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

    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;

    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;

    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;

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

    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;
    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;

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

  private:
    std::string _ip;
    int _port;
    std::string _uuid;
    int _channel;
    int _timepoint;
    int _angle;
    i3d::Vector3d<int> _resolution;
    std::string _version;
};

class Connection {
  public:
    Connection(std::string ip, int port, std::string uuid);

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

    dataset_props_ptr get_properties() const;

    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;
    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;

    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;

    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;
    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;

    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;

    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;

    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;

    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;

    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;

    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;

  private:
    std::string _ip;
    int _port;
    std::string _uuid;
};

} // namespace ds

/* ================= IMPLEMENTATION FOLLOWS ======================== */

namespace ds {
/* ===================================== Global space */
/* inline */ dataset_props_ptr get_dataset_properties(const std::string& ip,
                                                       int port,
                                                       const std::string& uuid) {
    std::string dataset_url = details::get_dataset_url(ip, port, uuid);
    return std::make_shared<DatasetProperties>(
        details::get_dataset_properties(dataset_url));
}

template <cnpts::Scalar T>
i3d::Image3d<T> read_image(const std::string& ip,
                           int port,
                           const std::string& uuid,
                           int channel /* = 0 */,
                           int timepoint /*  = 0 */,
                           int angle /* = 0 */,
                           i3d::Vector3d<int> resolution /* = {1, 1, 1} */,
                           const std::string& version /* = "latest" */,
                           dataset_props_ptr props /* = nullptr */) {
    return ImageView(ip, port, uuid, channel, timepoint, angle, resolution,
                     version)
        .read_image<T>(props);
}

template <cnpts::Scalar T>
void write_image(const i3d::Image3d<T>& img,
                 const std::string& ip,
                 int port,
                 const std::string& uuid,
                 int channel /* = 0 */,
                 int timepoint /* = 0 */,
                 int angle /* = 0 */,
                 i3d::Vector3d<int> resolution /*  = {1, 1, 1} */,
                 const std::string version /* = "latest" */,
                 dataset_props_ptr props /* = nullptr */) {
    ImageView(ip, port, uuid, channel, timepoint, angle, resolution, version)
        .write_image(img, props);
}

template <cnpts::Scalar T>
void write_with_pyramids(const i3d::Image3d<T>& img,
                         const std::string& ip,
                         int port,
                         const std::string& uuid,
                         int channel /* = 0 */,
                         int timepoint /* = 0 */,
                         int angle /* = 0 */,
                         const std::string& version /* = "latest"*/,
                         SamplingMode m /* = SamplingMode::NEAREST_NEIGHBOUR */,
                         dataset_props_ptr props /* = nullptr */) {
    Connection(ip, port, uuid)
        .write_with_pyramids(img, channel, timepoint, angle, version, m, props);
}

/* ===================================== ImageView */

ImageView::ImageView(std::string ip,
                     int port,
                     std::string uuid,
                     int channel,
                     int timepoint,
                     int angle,
                     i3d::Vector3d<int> resolution,
                     std::string version)
    :  _ip(std::move(ip)), _port(port), _uuid(std::move(uuid)),
      _channel(channel), _timepoint(timepoint), _angle(angle),
      _resolution(resolution), _version(std::move(version)) {}

dataset_props_ptr ImageView::get_properties()const {
    return get_dataset_properties(_ip, _port, _uuid);
}

template <cnpts::Scalar T>
i3d::Image3d<T>
ImageView::read_block(i3d::Vector3d<int> coord,
                      dataset_props_ptr props /* = nullptr */)const {
    /* Fetch properties from server */
    if (!props)
        props = get_properties();

    i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);

    /* Prepare output image */
    i3d::Image3d<T> img;
    i3d::Vector3d<int> block_size = details::data_manip::get_block_size(
        coord, block_dim, props->get_img_dimensions(_resolution));
    img.MakeRoom(block_size);

    /* Fetch and return */
    read_block(coord, img);
    return img;
}

template <cnpts::Scalar T>
void 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_blocks({coord}, dest, {dest_offset}, props);
}

template <cnpts::Scalar T>
std::vector<i3d::Image3d<T»
ImageView::read_blocks(const std::vector<i3d::Vector3d<int»& coords,
                       dataset_props_ptr props /* = nullptr */)const {
    if (!props)
        props = get_properties();

    /* Process blocks one by one */
    std::vector<i3d::Image3d<T» out;
    for (auto coord :  coords)
        out.push_back(read_block<T>(coord));

    return out;
}

// TODO optimise
template <cnpts::Scalar T>
void 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 {

    if (!props)
        props = get_properties();

    /* Fetched properties from server */
    std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);
    i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);

    i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);

    if (coords.size() != offsets.size())
        throw std::logic_error("Count of coordinates != count of offsets");

    if (!details::check_block_coords(coords, img_dim, block_dim))
        throw std::out_of_range("Blocks out of range");

    /* prepare request url */
    std::string session_url = details::requests::session_url_request(
        dataset_url, _resolution, _version);

    if (session_url.ends_with('/'))
        session_url.pop_back();

    /* Fetch blocks one by one */
    for (std::size_t i = 0; i < coords.size(); ++i) {
        auto& coord = coords[i];
        auto& offset = offsets[i];

        std::string url =
            fmt::format("{}/{}/{}/{}/{}/{}/{}", session_url, coord.x, coord.y,
                        coord.z, _timepoint, _channel, _angle);
        auto [data, response] = details::requests::make_request(url);

        details::data_manip::read_data(data, props->voxel_type, dest, offset);
    }
}

template <cnpts::Scalar T>
i3d::Image3d<T>
ImageView::read_region(i3d::Vector3d<int> start_point,
                       i3d::Vector3d<int> end_point,
                       dataset_props_ptr props /* = nullptr */)const {
    if (!props)
        props = get_properties();

    i3d::Vector3d img_dim = props->get_img_dimensions(_resolution);
    i3d::Vector3d block_dim = props->get_block_dimensions(_resolution);

    std::vector<i3d::Vector3d<int>> coords = details::get_intercepted_blocks(
        start_point, end_point, img_dim, block_dim);

    std::vector<i3d::Vector3d<int>> offsets;
    for (auto coord :  coords)
        offsets.emplace_back(coord * block_dim - start_point);

    i3d::Image3d<T> out_img;
    out_img.MakeRoom(end_point - start_point);

    read_blocks(coords, out_img, offsets, props);
    return out_img;
}

template <cnpts::Scalar T>
void 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 {
    auto temp_img = read_region<T>(start_point, end_point, props);

    // Copy to desired location
    for (std::size_t x = 0; x < temp_img.GetSizeX(); ++x)
        for (std::size_t y = 0; y < temp_img.GetSizeY(); ++y)
            for (std::size_t z = 0; z < temp_img.GetSizeZ(); ++z)
                dest.SetVoxel(x + offset.x, y + offset.y, z + offset.z,
                              temp_img.GetVoxel({x, y, z}));
}

template <cnpts::Scalar T>
i3d::Image3d<T>
ImageView::read_image(dataset_props_ptr props /* = nullptr */)const {
    if (!props)
        props = get_properties();

    i3d::Vector3d img_dim = props->get_img_dimensions(_resolution);
    return read_region<T>(0, img_dim, props);
}

template <cnpts::Scalar T>
void 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_blocks(src, {coord}, {src_offset}, props);
}

// TODO optimise
template <cnpts::Scalar T>
void 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 {

    if (!props)
        props = get_properties();

    /* Fetch server properties */
    std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);

    i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
    i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);

    /* Error checking (when not in debug, all checks automatically return
* true)*/
    if (coords.size() != src_offsets.size())
        throw std::logic_error("Count of coordinates != count of offsets");

    if (!details::check_block_coords(coords, img_dim, block_dim))
        throw std::out_of_range("Blocks out of range");

    /* prepare request url */
    std::string session_url = details::requests::session_url_request(
        dataset_url, _resolution, _version);

    if (session_url.ends_with('/'))
        session_url.pop_back();

    /* Write blocks to server one by one */
    for (std::size_t i = 0; i < coords.size(); ++i) {
        auto& coord = coords[i];
        auto& offset = src_offsets[i];

        i3d::Vector3d<int> block_size =
            details::data_manip::get_block_size(coord, block_dim, img_dim);

        /* Prepare vector representing octet-data (will be send to server) */
        std::vector<char> data(details::data_manip::get_block_data_size(
            block_size, props->voxel_type));

        /* Transform image to octet-data */
        details::data_manip::write_data(src, offset, data, props->voxel_type,
                                        block_size);

        std::string url =
            fmt::format("{}/{}/{}/{}/{}/{}/{}", session_url, coord.x, coord.y,
                        coord.z, _timepoint, _channel, _angle);

        auto [_, response] = details::requests::make_request(
            url, Poco::Net::HTTPRequest::HTTP_POST, data,
            {{"Content-Type", "application/octet-stream"}});
    }
}

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

    /* Fetch image properties from server */
    if (!props)
        props = get_properties();

    i3d::Vector3d<int> block_dim = props->get_block_dimensions(_resolution);
    i3d::Vector3d<int> img_dim = props->get_img_dimensions(_resolution);
    i3d::Vector3d<int> block_count =
        (img_dim + block_dim - 1) / block_dim; // Ceiling

    /* Prepare coordinates of blocks and offsets to write whole image */
    std::vector<i3d::Vector3d<int» blocks;
    std::vector<i3d::Vector3d<int» offsets;

    for (int x = 0; x < block_count.x; ++x)
        for (int y = 0; y < block_count.y; ++y)
            for (int z = 0; z < block_count.z; ++z) {
                blocks.emplace_back(x, y, z);
                offsets.emplace_back(x * block_dim.x, y * block_dim.y,
                                     z * block_dim.z);
            }

    /* write whole image */
    write_blocks(img, blocks, offsets, props);
}

/* ===================================== Connection */

Connection::Connection(std::string ip, int port, std::string uuid)
    :  _ip(std::move(ip)), _port(port), _uuid(std::move(uuid)) {}

ImageView Connection::get_view(int channel,
                               int timepoint,
                               int angle,
                               i3d::Vector3d<int> resolution,
                               const std::string& version)const {
    return ImageView(_ip, _port, _uuid, channel, timepoint, angle, resolution,
                     version);
}

dataset_props_ptr Connection::get_properties()const {
    return get_dataset_properties(_ip, _port, _uuid);
}
```

```

template <cnpts::Scalar T>
i3d::Image3d<T>
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 {
    return get_view(channel, timepoint, angle, resolution, version)
        .read_block<T>(coord, props);
}

template <cnpts::Scalar T>
void 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 {
    return get_view(channel, timepoint, angle, resolution, version)
        .read_block(coord, dest, dest_offset, props);
}

template <cnpts::Scalar T>
std::vector<i3d::Image3d<T>»
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 {
    return get_view(channel, timepoint, angle, resolution, version)
        .read_blocks<T>(coords, props);
}

template <cnpts::Scalar T>
void 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 {
    get_view(channel, timepoint, angle, resolution, version)
        .read_blocks(coords, dest, dest_offsets, props);
}

template <cnpts::Scalar T>
i3d::Image3d<T>
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 {
    return get_view(channel, timepoint, angle, resolution, version)
        .read_region<T>(start_point, end_point, props);
}

template <cnpts::Scalar T>
void 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 {
    get_view(channel, timepoint, angle, resolution, version)
        .read_region<T>(start_point, end_point, dest, offset, props);
}

template <cnpts::Scalar T>
i3d::Image3d<T>
```

```
Connection::read_image(int channel,
                       int timepoint,
                       int angle,
                       i3d::Vector3d<int> resolution,
                       const std::string& version,
                       dataset_props_ptr props /* = nullptr */)const {
    return get_view(channel, timepoint, angle, resolution, version)
        .read_image<T>(props);
}

template <cnpts::Scalar T>
void 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 {
    get_view(channel, timepoint, angle, resolution, version)
        .write_block(src, coord, src_offset, props);
}

template <cnpts::Scalar T>
void 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 {
    get_view(channel, timepoint, angle, resolution, version)
        .write_blocks(src, coords, src_offsets, props);
}

template <cnpts::Scalar T>
void 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 {
    get_view(channel, timepoint, angle, resolution, version)
        .write_image(img, props);
}

template <cnpts::Scalar T>
void 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 {
    if (!props)
        props = get_properties();
    write_image(img, channel, timepoint, angle, {1, 1, 1}, version, props);

    for (const auto& res :  props->get_all_resolutions()) {
        if (res == i3d::Vector3d<int>{1, 1, 1})
            continue;

        i3d::Vector3d<int> new_dim = props->get_img_dimensions(res);
        i3d::Image3d<T> cpy;
        i3d::Resample(img, cpy, new_dim, m);
        write_image(cpy, channel, timepoint, angle, res, version, props);
    }
}

} // namespace ds
```

4.2 hpc_ds_details.hpp

```
#pragma once
#include "hpc_ds_structs.hpp"
#include <Poco/JSON/Object.h>
#include <Poco/JSON/Parser.h>
```

```
#include <Poco/Net/HTTPClientSession.h>
#include <Poco/Net/HTTPMessage.h>
#include <Poco/Net/HTTPRequest.h>
#include <Poco/Net/HTTPResponse.h>
#include <Poco/URI.h>
#include <i3d/image3d.h>
#include <i3d/vector3d.h>
#include <optional>
#include <source_location>
#include <span>
#include <string>
#include <type_traits>
/* ==================== DETAILS HEADERS ============================ */

namespace ds {
namespace details {
/* Definition of compile settings */

#ifdef DATASTORE_NDEBUG
constexpr inline bool _DEBUG_ = false;
#else
#ifdef NDEBUG
constexpr inline bool _DEBUG_ = false;
#else
constexpr inline bool _DEBUG_ = true;
#endif
#endif

#ifdef DATASTORE_NLOG
constexpr inline bool _LOG_ = false;
#else
constexpr inline bool _LOG_ = _DEBUG_;
#endif

#ifdef DATASTORE_NINFO
constexpr inline bool _INFO_ = false;
#else
constexpr inline bool _INFO_ = _LOG_;
#endif

#ifdef DATASTORE_NWARNING
constexpr inline bool _WARNING_ = false;
#else
constexpr inline bool _WARNING_ = _LOG_;
#endif

inline std::string
get_dataset_url(const std::string& ip, int port, const std::string& uuid);

inline DatasetProperties get_dataset_properties(const std::string& dataset_url);

inline bool check_block_coords(const std::vector<i3d::Vector3d<int>>& coords,
                               i3d::Vector3d<int> img_dim,
                               i3d::Vector3d<int> block_dim);

inline std::vector<i3d::Vector3d<int>>
get_intercepted_blocks(i3d::Vector3d<int> start_point,
                       i3d::Vector3d<int> end_point,
                       i3d::Vector3d<int> img_dim,
                       i3d::Vector3d<int> block_dim);

namespace data_manip {
inline int get_block_data_size(i3d::Vector3d<int> block_size,
                               const std::string& voxel_type);

inline i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
                                         i3d::Vector3d<int> block_dim,
                                         i3d::Vector3d<int> img_dim);

inline int get_linear_index(i3d::Vector3d<int> coord,
                             i3d::Vector3d<int> block_dim,
                             const std::string& voxel_type);

template <typename T>
T get_elem_at(std::span<const char> data,
              const std::string& voxel_type,
              int index);

template <typename T>
T get_elem_at(std::span<const char> data,
              const std::string& voxel_type,
              i3d::Vector3d<int> coord,
              i3d::Vector3d<int> block_dim);

template <typename T>
void set_elem_at(std::span<char> data,
                 const std::string& voxel_type,
                 int index,
                 T elem);

template <typename T>
void set_elem_at(std::span<char> data,
                 const std::string& voxel_type,
                 i3d::Vector3d<int> coord,
                 i3d::Vector3d<int> block_dim,
                 T elem);

template <typename T>
void read_data(std::span<const char> data,
               const std::string& voxel_type,
               i3d::Image3d<T>& dest,
               i3d::Vector3d<int> offset);

template <typename T>
void write_data(const i3d::Image3d<T>& src,
                i3d::Vector3d<int> offset,
                std::span<char> data,
                const std::string& voxel_type,
                i3d::Vector3d<int> block_size);
} // namespace data_manip

namespace log {

inline void _log(const std::string& msg,
                 const std::string& type,
                 const std::source_location& location);

inline void
info(const std::string& msg,
     const std::source_location& location = std::source_location::current());

inline void
warning(const std::string& msg,
        const std::source_location& location = std::source_location::current());

} // namespace log
/* Helpers to parse Dataset Properties from JSON */
namespace props_parser {
using namespace Poco::JSON;

template <cnpts::Basic T>
T get_elem(Object::Ptr root, const std::string& name);

template <cnpts::Vector3d T>
T get_elem(Object::Ptr root, const std::string& name);

template <cnpts::Vector T>
T get_elem(Object::Ptr root, const std::string& name);

template <cnpts::ResolutionUnit T>
T get_elem(Object::Ptr root, const std::string& name);

template <cnpts::Optional T>
T get_elem(Object::Ptr root, const std::string& name);

inline std::vector<std::map<std::string, i3d::Vector3d<int»>
get_resolution_levels(Object::Ptr root);

} // namespace props_parser

/* Helpers providing requests functionality */
namespace requests {
inline std::string session_url_request(const std::string& ds_url,
                                       i3d::Vector3d<int> resolution,
                                       const std::string& version);

inline std::pair<std::vector<char>, Poco::Net::HTTPResponse>
make_request(const std::string& url,
             const std::string& type = Poco::Net::HTTPRequest::HTTP_GET,
             const std::vector<char>& data = {},
             const std::map<std::string, std::string>& headers = {});
} // namespace requests
} // namespace details
} // namespace ds

/* ================= IMPLEMENTATION FOLLOWS ======================== */
namespace ds {
namespace details {

/* inline */ std::string
get_dataset_url(const std::string& ip, int port, const std::string& uuid) {
    std::string out;
    if (!ip.starts_with("http://"))
        out = "https://";
    return out + fmt::format("{}:{}/datasets/{}", ip, port, uuid);
}

inline DatasetProperties
get_dataset_properties(const std::string& dataset_url) {
    using namespace Poco::JSON;

    /* Fetch JSON from server */
    auto [data, response] = requests::make_request(dataset_url);
    std::string json_str(data.begin(), data.end());

    int res_code = response.getStatus();
    if (res_code != 200)
        log::warning(fmt::format(
            "Request ended with code:  {}.  json may not be valid", res_code));

    log::info("Parsing dataset properties from JSON string");
    Parser parser;
    Poco::Dynamic::Var result = parser.parse(json_str);

    DatasetProperties props;
    auto root = result.extract<Object::Ptr>();

    using namespace props_parser;

    /* Parse elements from JSON */

    props.uuid = get_elem<std::string>(root, "uuid");
    props.voxel_type = get_elem<std::string>(root, "voxelType");
    props.dimensions = get_elem<i3d::Vector3d<int>>(root, "dimensions");
    props.channels = get_elem<int>(root, "channels");
    props.angles = get_elem<int>(root, "angles");
    props.transformations =
        get_elem<std::optional<std::string>>(root, "transformations");
    props.voxel_unit = get_elem<std::string>(root, "voxelUnit");
    props.voxel_resolution =
        get_elem<std::optional<i3d::Vector3d<double>>>(root, "voxelResolution");
    props.timepoint_resolution =
        get_elem<std::optional<ResolutionUnit>>(root, "timepointResolution");
    props.channel_resolution =
        get_elem<std::optional<ResolutionUnit>>(root, "channelResolution");
    props.angle_resolution =
        get_elem<std::optional<ResolutionUnit>>(root, "angleResolution");
    props.compression = get_elem<std::string>(root, "compression");
    props.resolution_levels = get_resolution_levels(root);
    props.versions = get_elem<std::vector<int>>(root, "versions");
    props.label = get_elem<std::string>(root, "label");
    props.view_registrations =
        get_elem<std::optional<std::string>>(root, "viewRegistrations");
    props.timepoint_ids = get_elem<std::vector<int>>(root, "timepointIds");

    log::info("Parsing has finished");
    return props;
}

/* inline */ bool
check_block_coords(const std::vector<i3d::Vector3d<int>>& coords,
                   i3d::Vector3d<int> img_dim,
                   i3d::Vector3d<int> block_dim) {
    /* Act as NOOP if not in debug */
    if constexpr (!_DEBUG_)
        return true;

    log::info("Checking validity of given block coordinates");

    for (i3d::Vector3d<int> coord :  coords)
        if (data_manip::get_block_size(coord, block_dim, img_dim) ==
            i3d::Vector3d(0, 0, 0)) {
            log::warning(fmt::format(
                "Block coordinate {} is out of valid range", to_string(coord)));

            return false;
        }
    log::info("Check successfullly finished");
    return true;
}

/* inline */ std::vector<i3d::Vector3d<int>>
get_intercepted_blocks(i3d::Vector3d<int> start_point,
                       i3d::Vector3d<int> end_point,
                       i3d::Vector3d<int> img_dim,
                       i3d::Vector3d<int> block_dim) {
    assert(lt(start_point, end_point));

    i3d::Vector3d<int> block_count = (img_dim + block_dim - 1) / block_dim;
    std::vector<i3d::Vector3d<int>> out;

    for (int x = 0; x < block_count.x; ++x)
        for (int y = 0; y < block_count.y; ++y)
            for (int z = 0; z < block_count.z; ++z) {
                i3d::Vector3d<int> coord = {x, y, z};
                if (lt(start_point, (coord + 1) * block_dim) &&
                    lt(coord * block_dim, end_point))
                    out.push_back(coord);
            }

    return out;
}

namespace data_manip {
/* inline */ int get_block_data_size(i3d::Vector3d<int> block_size,
                                     const std::string& voxel_type) {

    int elem_size = type_byte_size.at(voxel_type);
    return block_size.x * block_size.y * block_size.z * elem_size + 12;
}

/* inline */ i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
                                               i3d::Vector3d<int> block_dim,
                                               i3d::Vector3d<int> img_dim) {
    i3d::Vector3d<int> start = (coord * block_dim);
    i3d::Vector3d<int> end = (coord + 1) * block_dim;

    i3d::Vector3d<int> out;
    for (int i = 0; i < 3; ++i) {
        out[i] =
            std::max(0, std::min(img_dim[i], end[i]) - std::max(start[i], 0));
    }
    return out;
}

/* inline */ int get_linear_index(i3d::Vector3d<int> coord,
                                  i3d::Vector3d<int> block_dim,
                                  const std::string& voxel_type) {
    int elem_size = type_byte_size.at(voxel_type);

    return 12 +                                    // header_offset
           (coord.z * block_dim.x * block_dim.y + // Main axis
            coord.y * block_dim.x +                // secondary axis
            coord.x) *                             // last axis
               elem_size;                          // byte size
}

template <typename T>
T get_elem_at(std::span<const char> data,
              const std::string& voxel_type,
              int index) {
    int elem_size = type_byte_size.at(voxel_type);

    std::array<char, sizeof(T)> buffer{};
    std::copy_n(data.begin() + index,      // source start
                elem_size,                 // count
                buffer.end() - elem_size); // dest start

    std::ranges::reverse(buffer);

    return *reinterpret_cast<T*>(&buffer[0]);
}

template <typename T>
T get_elem_at(std::span<const char> data,
              const std::string& voxel_type,
              i3d::Vector3d<int> coord,
              i3d::Vector3d<int> block_dim) {

    int index = get_linear_index(coord, block_dim, voxel_type);
    return get_elem_at<T>(data, voxel_type, index);
}

template <typename T>
void set_elem_at(std::span<char> data,
                 const std::string& voxel_type,
                 int index,
                 T elem) {
    int elem_size = type_byte_size.at(voxel_type);

    auto buffer = *reinterpret_cast<std::array<char, sizeof(T)>*>(&elem);
    std::ranges::reverse(buffer);

    std::copy_n(buffer.end() - elem_size, // source start
                elem_size,                // count
                data.begin() + index);    // dest start
}

```

```
template <typename T>
void set_elem_at(std::span<char> data,
                 const std::string& voxel_type,
                 i3d::Vector3d<int> coord,
                 i3d::Vector3d<int> block_dim,
                 T elem) {
    int index = get_linear_index(coord, block_dim, voxel_type);
    set_elem_at(data, voxel_type, index, elem);
}

template <typename T>
void read_data(std::span<const char> data,
               const std::string& voxel_type,
               i3d::Image3d<T>& dest,
               i3d::Vector3d<int> offset) {
    i3d::Vector3d<int> block_dim;
    for (int i = 0; i < 3; ++i)
        block_dim[i] = get_elem_at<int>(data, "uint32", i * 4);

    for (int x = 0; x < block_dim.x; ++x)
        for (int y = 0; y < block_dim.y; ++y)
            for (int z = 0; z < block_dim.z; ++z) {
                i3d::Vector3d<int> coord{x + offset.x, y + offset.y,
                                         z + offset.z};

                if (!(0 <= coord.x && coord.x < int(dest.GetSizeX())) ||
                    !(0 <= coord.y && coord.y < int(dest.GetSizeY())) ||
                    !(0 <= coord.z && coord.z < int(dest.GetSizeZ())))
                    continue;

                dest.SetVoxel(coord, get_elem_at<T>(data, voxel_type, {x, y, z},
                                                    block_dim));
            }
}

template <typename T>
void write_data(const i3d::Image3d<T>& src,
                i3d::Vector3d<int> offset,
                std::span<char> data,
                const std::string& voxel_type,
                i3d::Vector3d<int> block_size) {
    set_elem_at(data, "uint32", 0, block_size.x);
    set_elem_at(data, "uint32", 4, block_size.y);
    set_elem_at(data, "uint32", 8, block_size.z);

    for (int x = 0; x < block_size.x; ++x)
        for (int y = 0; y < block_size.y; ++y)
            for (int z = 0; z < block_size.z; ++z)
                set_elem_at(
                    data, voxel_type, {x, y, z}, block_size,
                    src.GetVoxel(x + offset.x, y + offset.y, z + offset.z));
}
} // namespace data_manip

namespace log {
/* inline */ void _log(const std::string& msg,
                       const std::string& type,
                       const std::source_location& location) {
    if constexpr (!_LOG_)
        return;

    std::cout « fmt::format("[{}] {} at row {}:\n{} \n\n", type,
                            location.function_name(), location.line(), msg);
}

/* inline */ void info(const std::string& msg,
                       const std::source_location&
                           location /* = std::source_location::current() */) {
    if constexpr (!_INFO_)
        return;
    _log(msg, "INFO", location);
}

/* inline */ void
warning(const std::string& msg,
        const std::source_location&
            location /* = std::source_location::current() */) {
    if constexpr (!_WARNING_)
        return;
    _log(msg, "WARNING", location);
}
} // namespace log

namespace props_parser {

template <cnpts::Basic T>
T get_elem(Object::Ptr root, const std::string& name) {
    if (!root->has(name)) {
        log::warning(fmt::format("{} was not found", name));
        return {};
    }
    return root->getValue<T>(name);
}

template <cnpts::Vector3d T>
T get_elem(Object::Ptr root, const std::string& name) {
    using V = decltype(T{}.x);
    if (!root->has(name)) {
        log::warning(fmt::format("{} were not found", name));
        return {};
    }

    Array::Ptr values = root->getArray(name);
    if (values->size() != 3) {
        log::warning("Incorrect number of dimensions");
        return {};
    }

    T out;
    for (unsigned i = 0; i < 3; ++i)
        out[i] = values->getElement<V>(i);

    return out;
}

template <cnpts::Vector T>
T get_elem(Object::Ptr root, const std::string& name) {
    using V = typename T::value_type;
    if (!root->has(name)) {
        log::warning(fmt::format("{} were not found", name));
        return {};
    }

    Array::Ptr values = root->getArray(name);
    std::size_t count = values->size();

    T out(count);
    for (unsigned i = 0; i < count; ++i)
        out[i] = values->getElement<V>(i);

    return out;
}

template <cnpts::ResolutionUnit T>
T get_elem(Object::Ptr root, const std::string& name) {
    if (!root->has(name)) {
        log::warning(fmt::format("{} was not found", name));
        return {};
    }

    Object::Ptr res_ptr = root->getObject(name);
    ResolutionUnit res;

    if (res_ptr->has("value")) {
        res.value = res_ptr->getValue<double>("value");
    }

    if (res_ptr->has("unit")) {
        res.unit = res_ptr->getValue<std::string>("unit");
    }

    return res;
}

template <cnpts::Optional T>
T get_elem(Object::Ptr root, const std::string& name) {
    if (!root->has(name)) {
        log::warning(fmt::format("{} were not found", name));
        return {};
    }

    if (root->isNull(name))
        return {};

    T out;
    out = get_elem<typename T::value_type>(root, name);
    return out;
}

/* inline */ std::vector<std::map<std::string, i3d::Vector3d<int»>
get_resolution_levels(Object::Ptr root) {
    std::string name = "resolutionLevels";

    if (!root->has(name)) {
        log::warning("resolutionLevels were not found");
        return {};
    }

    std::vector<std::map<std::string, i3d::Vector3d<int»> out;

    Array::Ptr array = root->getArray(name);
    for (unsigned i = 0; i < array->size(); ++i) {
        std::map<std::string, i3d::Vector3d<int» map;
        Object::Ptr map_ptr = array->getObject(i);

        for (const auto& name :  map_ptr->getNames()) {
            map[name] = get_elem<i3d::Vector3d<int»(map_ptr, name);
        }

        out.push_back(map);
    }

    return out;
}

} // namespace props_parser

namespace requests {
/* inline */ std::string session_url_request(const std::string& ds_url,
                                             i3d::Vector3d<int> resolution,
                                             const std::string& version) {

    log::info(
        fmt::format("Obtaining session url for resolution:  {}, version:  {}",
                    to_string(resolution), version));
    std::string req_url =
        fmt::format("{}/{}/{}/{}/{}/read-write", ds_url, resolution.x,
                    resolution.y, resolution.z, version);

    auto [_, response] = make_request(req_url);

    int res_code = response.getStatus();
    if (res_code != 307)
        log::warning(fmt::format(
            "Request ended with status:  {}, redirection may be incorrect",
            res_code));

    return response.get("Location");
}

/* inline */ std::pair<std::vector<char>, Poco::Net::HTTPResponse>
make_request(const std::string& url,
             const std::string& type /*  = Poco::Net::HTTPRequest::HTTP_GET */,
             const std::vector<char>& data /*  = {} */,
             const std::map<std::string, std::string>& headers /* = {} */) {
    Poco::URI uri(url);
    std::string path(uri.getPathAndQuery());

    Poco::Net::HTTPClientSession session(uri.getHost(), uri.getPort());

    Poco::Net::HTTPRequest request(type, path,
                                   Poco::Net::HTTPMessage::HTTP_1_1);

    for (auto& [key, value] :  headers)
        request.set(key, value);

    request.setContentLength(data.size());

    log::info(fmt::format("Sending {} request to url:  {}", type, url));
    std::ostream& os = session.sendRequest(request);
    for (char ch :  data)
        os « ch;

    Poco::Net::HTTPResponse response;
    std::istream& rs = session.receiveResponse(response);

    std::vector<char> out{std::istreambuf_iterator<char>(rs),
                          std::istreambuf_iterator<char>()};

    log::info(fmt::format(
        "Fetched response with status:  {}, reason:  {}, content size:  {}",
        response.getStatus(), response.getReason(), out.size()));

    return {out, response};
}

} // namespace requests
} // namespace details
} // namespace ds
```

4.3 hpc_ds_structs.hpp

```
#pragma once
#include <array>
#include <cassert>
#include <fmt/core.h>
#include <i3d/image3d.h>
#include <i3d/transform.h>
#include <i3d/vector3d.h>
#include <map>
#include <optional>
#include <ostream>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>
#include <memory>

template <typename T, typename U>
bool lt(i3d::Vector3d<T> lhs, i3d::Vector3d<U> rhs) {
    return lhs.x < rhs.x && lhs.y < rhs.y && lhs.z < rhs.z;
}

template <typename T, typename U>
requires std::is_integral_v<T> && std::is_integral_v<U>
bool eq(i3d::Vector3d<T> lhs, i3d::Vector3d<U> rhs) {
    for (int i = 0; i < 3; ++i)
        if (static_cast<long long>(lhs[i]) != static_cast<long long>(rhs[i]))
            return false;
    return true;
}

template <typename T, typename U>
requires std::is_floating_point_v<T> && std::is_floating_point_v<U>
bool eq(i3d::Vector3d<T> lhs, i3d::Vector3d<U> rhs) {
    for (int i = 0; i < 3; ++i)
        if (static_cast<long double>(lhs[i]) != static_cast<long double>(rhs[i]))
            return false;
    return true;
}

namespace ds {

using i3d::SamplingMode;

/* dataset 'voxel_type' to 'byte_size' map*/
const inline std::map<std::string, int> type_byte_size{
    {"uint8", 1}, {"uint16", 2}, {"uint32", 4}, {"uint64", 8},  {"int8", 1},
    {"int16", 2}, {"int32", 4},  {"int64", 8},  {"float32", 4}, {"float64", 8}};

/* Maximal legal URL length */
constexpr inline std::size_t MAX_URL_LENGTH = 2048;

class ResolutionUnit {
  public:
    double value = 0.0;
    std::string unit = "";

    friend std::ostream& operator«(std::ostream& stream,
                                    const ResolutionUnit& res) {
        stream « fmt::format("{} {}", res.value, res.unit);
        return stream;
    }
};

/* Concepts definitions to make templates more readable */
namespace cnpts {
template <typename T>
concept Scalar = requires(T) {
    requires std::is_scalar_v<T>;
};

template <typename T>
concept Basic = requires(T) {
    requires Scalar<T> || std::is_same_v<T, std::string>;
};

template <typename T>
concept Vector = requires(T) {
    requires std::is_same_v<std::vector<typename T::value_type>, T>;
};

template <typename T>
concept Optional = requires(T) {
    requires std::is_same_v<std::optional<typename T::value_type>, T>;
};

```

```
template <typename T>
concept Vector3d = requires(T a) {
    requires std::is_same_v<i3d::Vector3d<decltype(a.x)>, T>;
    requires Basic<decltype(a.x)>;
};

template <typename T>
concept Streamable = requires(T a) {
    {std::cout « a};
};

template <typename T>
concept Map = requires(T) {
    requires std::is_same_v<
        std::map<typename T::key_type, typename T::mapped_type>, T>;
};

template <typename T>
concept ResolutionUnit = requires(T) {
    requires std::is_same_v<T, ds::ResolutionUnit>;
};
} // namespace cnpts

namespace details {

template <cnpts::Streamable T>
std::string to_string(const T&);

template <cnpts::Vector T>
std::string to_string(const T&);

template <cnpts::Map T>
std::string to_string(const T&);

template <cnpts::Optional T>
std::string to_string(const T&);

template <cnpts::Streamable T>
std::string to_string(const T& val) {
    std::stringstream ss;
    ss « val;
    return ss.str();
}

template <cnpts::Vector T>
std::string to_string(const T& vec) {
    std::stringstream ss;
    ss « "(";

    const char* delim = "";
    for (auto& v :  vec) {
        ss « delim « to_string(v);
        delim = ", ";
    }

    ss « ")";
    return ss.str();
}

template <cnpts::Map T>
std::string to_string(const T& map) {
    std::stringstream ss;
    ss « "{\n";

    for (const auto& [k, v] :  map) {
        ss « to_string(k) « ":  " « to_string(v) « '\n';
    }
    ss « "}";
    return ss.str();
}

template <cnpts::Optional T>
std::string to_string(const T& val) {
    if (!val)
        return "null";
    return to_string(val.value());
}

} // namespace details

class DatasetProperties {
  public:
    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;

    i3d::Vector3d<int>
    get_block_dimensions(i3d::Vector3d<int> resolution)const {
        for (const auto& map :  resolution_levels)
            if (map.at("resolutions") == resolution)
                return map.at("blockDimensions");

        std::string msg = fmt::format("Resolution {} not found in properties",
                                      details::to_string(resolution));
        throw std::out_of_range(msg.c_str());
    }

    i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
                                      i3d::Vector3d<int> resolution)const {
        i3d::Vector3d<int> block_dim = get_block_dimensions(resolution);
        i3d::Vector3d<int> start = (coord * block_dim);
        i3d::Vector3d<int> end = (coord + 1) * block_dim;

        i3d::Vector3d<int> out;
        for (int i = 0; i < 3; ++i) {
            out[i] = std::max(0, std::min(dimensions[i], end[i]) -
                                     std::max(start[i], 0));
        }
        return out;
    }

    i3d::Vector3d<int> get_block_count(i3d::Vector3d<int> resolution)const {
        i3d::Vector3d<int> block_dim = get_block_dimensions(resolution);

        return (dimensions + block_dim - 1) / block_dim;
    }

    i3d::Vector3d<int> get_img_dimensions(i3d::Vector3d<int> resolution)const {
        return dimensions / resolution;
    }

    std::vector<i3d::Vector3d<int» get_all_resolutions()const {
        std::vector<i3d::Vector3d<int» out;
        for (const auto& map :  resolution_levels)
            out.push_back(map.at("resolutions"));
        return out;
    }

    friend std::ostream& operator«(std::ostream& stream,
                                    const DatasetProperties& ds) {
        using details::to_string;

        stream « "UUID: " « ds.uuid « '\n';
        stream « "voxelType:  " « ds.voxel_type « '\n';
        stream « "dimensions:  " « ds.dimensions « '\n';
        stream « "channels:  " « ds.channels « '\n';
        stream « "angles:  " « ds.angles « '\n';
        stream « "transformations:  " « to_string(ds.transformations) « '\n';
        stream « "voxelUnit:  " « ds.voxel_unit « '\n';
        stream « "voxelResolution:  " « to_string(ds.voxel_resolution) « '\n';
        stream « "timepointResolution:  " « to_string(ds.timepoint_resolution)
               « '\n';
        stream « "channelResolution:  " « to_string(ds.channel_resolution)
               « '\n';
        stream « "angleResolution:  " « to_string(ds.angle_resolution) « '\n';
        stream « "comprestreamion:  " « ds.compression « '\n';
        stream « "resolutionLevels:  " « to_string(ds.resolution_levels)
               « '\n';
        stream « "versions:  " « to_string(ds.versions) « '\n';
        stream « "label:  " « ds.label « '\n';
        stream « "viewRegistrations:  " « to_string(ds.view_registrations)
               « '\n';
        stream « "timepointIds:  " « to_string(ds.timepoint_ids) « '\n';

        return stream;
    }
};
using dataset_props_ptr = std::shared_ptr<DatasetProperties>;

} // namespace ds
```

Index