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 datastore::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.
- 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) const

  Read one block from server to image.
- template<cnpts::Scalar T>
  bool 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) 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) const

  Read blocks from server and return them.
- template<cnpts::Scalar T>
  bool 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) const

  Read blocks from server and saves them into prealocated image.
- template<cnpts::Scalar T>
  i3d::Image3d< T > read_image (int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version) const

  Read full image.

- template<cnpts::Scalar T>
  bool 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) const

  Write block to server.
- template<cnpts::Scalar T>
  bool 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) const

  Write blocks to server.
- template<cnpts::Scalar T>
  bool write_image (const i3d::Image3d< T > &img, int channel, int timepoint, int angle, i3d::Vector3d< int > resolution, const std::string &version) const

  Write image to server.

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()

```
datastore::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_view()

```
ImageView datastore::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

3.1.3.2 read_block() [1/2]

```
template<cnpts::Scalar T>
bool datastore::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 ) 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

Parameters

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")

Returns

true At read success

false At read failiure

3.1.3.3 read_block() [2/2]

```
template<cnpts::Scalar T>
i3d::Image3d< T > datastore::Connection::read_block (
            i3d::Vector3d< int > coord,
            int channel,
            int timepoint,
            int angle,
            i3d::Vector3d< int > resolution,
            const std::string & version ) 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")

Returns

Image containing selected block

3.1.3.4 read_blocks() [1/2]

```
template<cnpts::Scalar T>
bool datastore::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 ) 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")

Returns

true At read success

false At read failure

3.1.3.5 read_blocks() [2/2]

```
template<cnpts::Scalar T>
std::vector< i3d::Image3d< T > > datastore::Connection::read_blocks (
```

```
            const std::vector< i3d::Vector3d< int > > & coords,
            int channel,
            int timepoint,
            int angle,
            i3d::Vector3d< int > resolution,
            const std::string & version ) 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
resolution	Resolution, at which the image is located
version	Version, at which the image is located (integer identifier or "latest")

Returns

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

3.1.3.6 read_image()

```
template<cnpts::Scalar T>
i3d::Image3d< T > datastore::Connection::read_image (
            int channel,
            int timepoint,
            int angle,
            i3d::Vector3d< int > resolution,
            const std::string & version ) 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")

Returns

i3d::Image3d<T> etched image

3.1.3.7 write_block()

```
template<cnpts::Scalar T>
bool datastore::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 ) 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

Parameters

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

Returns

true At write success

false At write failiure

3.1.3.8 write_blocks()

```
template<cnpts::Scalar T>
bool datastore::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 ) 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
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

true At write success

false At write failiure

3.1.3.9 write_image()

```
template<cnpts::Scalar T>
bool datastore::Connection::write_image (
            const i3d::Image3d< T > & img,
            int channel,
            int timepoint,
            int angle,
            i3d::Vector3d< int > resolution,
            const std::string & version ) 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")

Returns

true true At write success

false At write failiure

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

- /home/somik/CBIA/hpc-datastore-cpp/src/hpc_ds_api.hpp

3.2 datastore::DatasetProperties Class Reference

Class representing dataset properties.

`#include <hpc_ds_structs.hpp>`

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 datastore::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.
- template<cnpts::Scalar T>
  i3d::Image3d< T > read_block (i3d::Vector3d< int > coord) const

  Read one block from server.
- template<cnpts::Scalar T>
  bool read_block (i3d::Vector3d< int > coord, i3d::Image3d< T > &dest, i3d::Vector3d< int > dest_offset={0, 0, 0}) 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) const

  Read blocks from server and return them.
- template<cnpts::Scalar T>
  bool read_blocks (const std::vector< i3d::Vector3d< int > > &coords, i3d::Image3d< T > &dest, const std::vector< i3d::Vector3d< int > > &offsets) const

  Read blocks from server and saves them into prealocated image.
- template<cnpts::Scalar T>
  i3d::Image3d< T > read_image () const

  Read full image.
- template<cnpts::Scalar T>
  bool write_block (const i3d::Image3d< T > &src, i3d::Vector3d< int > coord, i3d::Vector3d< int > src_↩ offset={0, 0, 0}) const

  Write block to server.
- template<cnpts::Scalar T>
  bool write_blocks (const i3d::Image3d< T > &src, const std::vector< i3d::Vector3d< int > > &coords, const std::vector< i3d::Vector3d< int > > &src_offsets) const

  Write blocks to server.
- template<cnpts::Scalar T>
  bool write_image (const i3d::Image3d< T > &img) 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()

```
datastore::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")

3.3.3 Member Function Documentation

3.3.3.1 read_block() [1/2]

```
template<cnpts::Scalar T>
i3d::Image3d< T > datastore::ImageView::read_block (
            i3d::Vector3d< int > coord ) 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

Returns

Image containing selected block

3.3.3.2 read_block() [2/2]

```
template<cnpts::Scalar T>
bool datastore::ImageView::read_block (
            i3d::Vector3d< int > coord,
            i3d::Image3d< T > & dest,
            i3d::Vector3d< int > dest_offset = {0, 0, 0} ) 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

Returns

true At read success

false At read failiure

3.3.3.3 read_blocks() [1/2]

```
template<cnpts::Scalar T>
std::vector< i3d::Image3d< T > > datastore::ImageView::read_blocks (
            const std::vector< i3d::Vector3d< int > > & coords ) 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

Returns

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

3.3.3.4 read_blocks() [2/2]

```
template<cnpts::Scalar T>
bool datastore::ImageView::read_blocks (
            const std::vector< i3d::Vector3d< int > > & coords,
            i3d::Image3d< T > & dest,
            const std::vector< i3d::Vector3d< int > > & offsets ) 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
offsets	Offsets at wich the corresponding blocks should be saved

Returns

true At read success

false At read failure

3.3.3.5 read_image()

```
template<cnpts::Scalar T>
i3d::Image3d< T > datastore::ImageView::read_image
```

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

Returns

i3d::Image3d<T> Fetched image

3.3.3.6 write_block()

```
template<cnpts::Scalar T>
bool datastore::ImageView::write_block (
            const i3d::Image3d< T > & src,
            i3d::Vector3d< int > coord,
            i3d::Vector3d< int > src_offset = {0, 0, 0} ) 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

Returns

true At write success

false At write failiure

3.3.3.7 write_blocks()

```
template<cnpts::Scalar T>
bool datastore::ImageView::write_blocks (
            const i3d::Image3d< T > & src,
            const std::vector< i3d::Vector3d< int > > & coords,
            const std::vector< i3d::Vector3d< int > > & src_offsets ) 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

Returns

true At write success

false At write failiure

3.3.3.8 write_image()

```
template<cnpts::Scalar T>
bool datastore::ImageView::write_image (
            const i3d::Image3d< T > & img ) 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

Returns

true At write success

false At write failiure

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

- /home/somik/CBIA/hpc-datastore-cpp/src/hpc_ds_api.hpp

3.4 datastore::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 <string>
7 #include <type_traits>
8 #include <vector>
9
10 namespace datastore {
23 inline DatasetProperties get_dataset_properties(const std::string& ip,
24                                                 int port,
25                                                 const std::string& uuid);
26
48 template <cnpts::Scalar T>
49 i3d::Image3d<T> read_image(const std::string& ip,
50                            int port,
51                            const std::string& uuid,
52                            int channel = 0,
53                            int timepoint = 0,
54                            int angle = 0,
55                            i3d::Vector3d<int> resolution = {1, 1, 1},
56                            const std::string& version = "latest");
57
87 template <cnpts::Scalar T>
88 bool write_image(const i3d::Image3d<T>& img,
89                  const std::string& ip,
90                  int port,
91                  const std::string& uuid,
92                  int channel = 0,
93                  int timepoint = 0,
94                  int angle = 0,
95                  i3d::Vector3d<int> resolution = {1, 1, 1},
96                  const std::string version = "latest");
97
107 class ImageView {
108   public:
123     ImageView(std::string ip,
124               int port,
125               std::string uuid,
126               int channel,
127               int timepoint,
128               int angle,
129               i3d::Vector3d<int> resolution,
130               std::string version);
131
148     template <cnpts::Scalar T>
149     i3d::Image3d<T> read_block(i3d::Vector3d<int> coord) const;
150
169     template <cnpts::Scalar T>
170     bool read_block(i3d::Vector3d<int> coord,
171                     i3d::Image3d<T>& dest,
172                     i3d::Vector3d<int> dest_offset = {0, 0, 0}) const;
173
199     template <cnpts::Scalar T>
200     std::vector<i3d::Image3d<T»
201     read_blocks(const std::vector<i3d::Vector3d<int»& coords) const;
202
```

```
220     template <cnpts::Scalar T>
221     bool read_blocks(const std::vector<i3d::Vector3d<int»& coords,
222                      i3d::Image3d<T>& dest,
223                      const std::vector<i3d::Vector3d<int»& offsets) const;
224
237     template <cnpts::Scalar T>
238     i3d::Image3d<T> read_image() const;
239
257     template <cnpts::Scalar T>
258     bool write_block(const i3d::Image3d<T>& src,
259                      i3d::Vector3d<int> coord,
260                      i3d::Vector3d<int> src_offset = {0, 0, 0}) const;
278     template <cnpts::Scalar T>
279     bool write_blocks(const i3d::Image3d<T>& src,
280                       const std::vector<i3d::Vector3d<int»& coords,
281                       const std::vector<i3d::Vector3d<int»& src_offsets) const;
298     template <cnpts::Scalar T>
299     bool write_image(const i3d::Image3d<T>& img) const;
300
301   private:
302     std::string _ip;
303     int _port;
304     std::string _uuid;
305     int _channel;
306     int _timepoint;
307     int _angle;
308     i3d::Vector3d<int> _resolution;
309     std::string _version;
310 };
311
324 class Connection {
325   public:
334     Connection(std::string ip, int port, std::string uuid);
335
347     ImageView get_view(int channel,
348                        int timepoint,
349                        int angle,
350                        i3d::Vector3d<int> resolution,
351                        const std::string& version) const;
352
373     template <cnpts::Scalar T>
374     i3d::Image3d<T> read_block(i3d::Vector3d<int> coord,
375                                int channel,
376                                int timepoint,
377                                int angle,
378                                i3d::Vector3d<int> resolution,
379                                const std::string& version) const;
404     template <cnpts::Scalar T>
405     bool read_block(i3d::Vector3d<int> coord,
406                     i3d::Image3d<T>& dest,
407                     i3d::Vector3d<int> dest_offset,
408                     int channel,
409                     int timepoint,
410                     int angle,
411                     i3d::Vector3d<int> resolution,
412                     const std::string& version) const;
413
445     template <cnpts::Scalar T>
446     std::vector<i3d::Image3d<T»
447     read_blocks(const std::vector<i3d::Vector3d<int»& coords,
448                 int channel,
449                 int timepoint,
450                 int angle,
451                 i3d::Vector3d<int> resolution,
452                 const std::string& version) const;
453
478     template <cnpts::Scalar T>
479     bool read_blocks(const std::vector<i3d::Vector3d<int»& coords,
480                      i3d::Image3d<T>& dest,
481                      const std::vector<i3d::Vector3d<int»& dest_offsets,
482                      int channel,
483                      int timepoint,
484                      int angle,
485                      i3d::Vector3d<int> resolution,
486                      const std::string& version) const;
505     template <cnpts::Scalar T>
506     i3d::Image3d<T> read_image(int channel,
507                                int timepoint,
508                                int angle,
509                                i3d::Vector3d<int> resolution,
510                                const std::string& version) const;
511
535     template <cnpts::Scalar T>
536     bool write_block(const i3d::Image3d<T>& src,
537                      i3d::Vector3d<int> coord,
538                      i3d::Vector3d<int> src_offset,
539                      int channel,
```

```
540                     int timepoint,
541                     int angle,
542                     i3d::Vector3d<int> resolution,
543                     const std::string& version) const;
544
568     template <cnpts::Scalar T>
569     bool write_blocks(const i3d::Image3d<T>& src,
570                       const std::vector<i3d::Vector3d<int>>& coords,
571                       const std::vector<i3d::Vector3d<int>>& src_offsets,
572                       int channel,
573                       int timepoint,
574                       int angle,
575                       i3d::Vector3d<int> resolution,
576                       const std::string& version) const;
577
600     template <cnpts::Scalar T>
601     bool write_image(const i3d::Image3d<T>& img,
602                     int channel,
603                     int timepoint,
604                     int angle,
605                     i3d::Vector3d<int> resolution,
606                     const std::string& version) const;
607
608   private:
609     std::string _ip;
610     int _port;
611     std::string _uuid;
612 };
613
614 } // namespace datastore
615
616 /* ================= IMPLEMENTATION FOLLOWS ======================== */
617
618 namespace datastore {
619 /* ================================== Global space */
620 /* inline */ DatasetProperties get_dataset_properties(const std::string& ip,
621                                                       int port,
622                                                       const std::string& uuid) {
623     std::string dataset_url = details::get_dataset_url(ip, port, uuid);
624     return details::get_dataset_properties(dataset_url);
625 }
626
627 template <cnpts::Scalar T>
628 i3d::Image3d<T> read_image(const std::string& ip,
629                            int port,
630                            const std::string& uuid,
631                            int channel /* = 0 */,
632                            int timepoint /*  = 0 */,
633                            int angle /* = 0 */,
634                            i3d::Vector3d<int> resolution /* = {1, 1, 1} */,
635                            const std::string& version /* = "latest" */) {
636     return ImageView(ip, port, uuid, channel, timepoint, angle, resolution,
637                      version)
638         .read_image<T>();
639 }
640
641 template <cnpts::Scalar T>
642 bool write_image(const i3d::Image3d<T>& img,
643                  const std::string& ip,
644                  int port,
645                  const std::string& uuid,
646                  int channel /* = 0 */,
647                  int timepoint /* = 0 */,
648                  int angle /* = 0 */,
649                  i3d::Vector3d<int> resolution /*  = {1, 1, 1} */,
650                  const std::string version /* = "latest" */) {
651     return ImageView(ip, port, uuid, channel, timepoint, angle, resolution,
652                      version)
653         .write_image(img);
654 }
655
656 /* ================================== ImageView */
657
658 ImageView::ImageView(std::string ip,
659                      int port,
660                      std::string uuid,
661                      int channel,
662                      int timepoint,
663                      int angle,
664                      i3d::Vector3d<int> resolution,
665                      std::string version)
666     :  _ip(std::move(ip)), _port(port), _uuid(std::move(uuid)),
667       _channel(channel), _timepoint(timepoint), _angle(angle),
668       _resolution(resolution), _version(std::move(version)) {}
669
670 template <cnpts::Scalar T>
671 i3d::Image3d<T> ImageView::read_block(i3d::Vector3d<int> coord)const {
```

```
672     /* Fetch properties from server */
673     DatasetProperties props = get_dataset_properties(_ip, _port, _uuid);
674     i3d::Vector3d<int> block_dim =
675         details::get_block_dimensions(props, _resolution);
676
677     /* Prepare output image */
678     i3d::Image3d<T> img;
679     i3d::Vector3d<int> block_size = details::data_manip::get_block_size(
680         coord, block_dim, props.dimensions / _resolution);
681     img.MakeRoom(block_size);
682
683     /* Fetch and return */
684     read_block(coord, img);
685     return img;
686 }
687
688 template <cnpts::Scalar T>
689 bool ImageView::read_block(
690     i3d::Vector3d<int> coord,
691     i3d::Image3d<T>& dest,
692     i3d::Vector3d<int> dest_offset /*  = {0, 0, 0} */)const {
693     return read_blocks({coord}, dest, {dest_offset});
694 }
695
696 template <cnpts::Scalar T>
697 std::vector<i3d::Image3d<T>>
698 ImageView::read_blocks(const std::vector<i3d::Vector3d<int>>& coords)const {
699     /* Process blocks one by one */
700     std::vector<i3d::Image3d<T>> out;
701     for (auto coord :  coords)
702         out.push_back(read_block<T>(coord));
703
704     return out;
705 }
706
707 // TODO optimise
708 template <cnpts::Scalar T>
709 bool ImageView::read_blocks(
710     const std::vector<i3d::Vector3d<int>>& coords,
711     i3d::Image3d<T>& dest,
712     const std::vector<i3d::Vector3d<int>>& offsets)const {
713
714     /* Fetched properties from server */
715     std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);
716     DatasetProperties props = details::get_dataset_properties(dataset_url);
717     i3d::Vector3d<int> block_dim =
718         details::get_block_dimensions(props, _resolution);
719
720     i3d::Vector3d<int> img_dim = props.dimensions / _resolution;
721
722     /* Error checking (when not in debug, all checks automatically return true)*/
723     if (coords.size() != offsets.size()) {
724         details::log::error("Count of coordinates != count of offsets");
725         return false;
726     }
727
728     if (!details::check_block_coords(coords, img_dim, block_dim))
729         return false;
730
731     if (!details::check_offset_coords(offsets, coords, dest, block_dim,
732                                       img_dim))
733         return false;
734
735     /* prepare request url */
736     std::string session_url = details::requests::session_url_request(
737         dataset_url, _resolution, _version);
738
739     if (session_url.ends_with('/'))
740         session_url.pop_back();
741
742     /* Fetch blocks one by one */
743     for (std::size_t i = 0; i < coords.size(); ++i) {
744         auto& coord = coords[i];
745         auto& offset = offsets[i];
746
747         std::string url =
748             fmt::format("{}/{}/{}/{}/{}/{}/{}", session_url, coord.x, coord.y,
749                         coord.z, _timepoint, _channel, _angle);
750         auto [data, response] = details::requests::make_request(url);
751
752         details::data_manip::read_data(data, props.voxel_type, dest, offset);
753     }
754     return true;
755 }
756
757 template <cnpts::Scalar T>
758 i3d::Image3d<T> ImageView::read_image()const {
```

```
759     /* Fetch properties from server */
760     DatasetProperties props = get_dataset_properties(_ip, _port, _uuid);
761     i3d::Vector3d<int> block_dim =
762         details::get_block_dimensions(props, _resolution);
763     i3d::Vector3d<int> img_dim = props.dimensions / _resolution;
764
765     i3d::Vector3d<int> block_count =
766         (img_dim + block_dim - 1) / block_dim; // Ceiling
767
768     /* Prepare output image */
769     i3d::Image3d<T> out;
770     out.MakeRoom(img_dim);
771
772     /* Prepare coordinates of blocks and offsets to fetch whole image */
773     std::vector<i3d::Vector3d<int» blocks;
774     std::vector<i3d::Vector3d<int» offsets;
775
776     for (int x = 0; x < block_count.x; ++x)
777         for (int y = 0; y < block_count.y; ++y)
778             for (int z = 0; z < block_count.z; ++z) {
779                 blocks.emplace_back(x, y, z);
780                 offsets.emplace_back(x * block_dim.x, y * block_dim.y,
781                                      z * block_dim.z);
782             }
783
784     /* Fetch whole image and return */
785     read_blocks(blocks, out, offsets);
786     return out;
787 }
788
789 template <cnpts::Scalar T>
790 bool ImageView::write_block(
791     const i3d::Image3d<T>& src,
792     i3d::Vector3d<int> coord,
793     i3d::Vector3d<int> src_offset /*  = {0, 0, 0} */)const {
794     return write_blocks(src, {coord}, {src_offset});
795 }
796
797 // TODO optimise
798 template <cnpts::Scalar T>
799 bool ImageView::write_blocks(
800     const i3d::Image3d<T>& src,
801     const std::vector<i3d::Vector3d<int»& coords,
802     const std::vector<i3d::Vector3d<int»& src_offsets)const {
803
804     /* Fetch server properties */
805     std::string dataset_url = details::get_dataset_url(_ip, _port, _uuid);
806     DatasetProperties props = details::get_dataset_properties(dataset_url);
807     i3d::Vector3d<int> block_dim =
808         details::get_block_dimensions(props, _resolution);
809     i3d::Vector3d<int> img_dim = props.dimensions / _resolution;
810
811     /* Error checking (when not in debug, all checks automatically return true)*/
812     if (coords.size() != src_offsets.size()) {
813         details::log::error("Count of coordinates != count of offsets");
814         return false;
815     }
816
817     if (!details::check_block_coords(coords, img_dim, block_dim))
818         return false;
819
820     if (!details::check_offset_coords(src_offsets, coords, src, block_dim,
821                                       img_dim))
822         return false;
823
824     /* prepare request url */
825     std::string session_url = details::requests::session_url_request(
826         dataset_url, _resolution, _version);
827
828     if (session_url.ends_with('/'))
829         session_url.pop_back();
830
831     /* Write blocks to server one by one */
832     for (std::size_t i = 0; i < coords.size(); ++i) {
833         auto& coord = coords[i];
834         auto& offset = src_offsets[i];
835
836         i3d::Vector3d<int> block_size =
837             details::data_manip::get_block_size(coord, block_dim, img_dim);
838
839         /* Prepare vector representing octet-data (will be send to server) */
840         std::vector<char> data(details::data_manip::get_block_data_size(
841             block_size, props.voxel_type));
842
843         /* Transform image to octet-data */
844         details::data_manip::write_data(src, offset, data, props.voxel_type,
845                                         block_size);
```

```
846
847         std::string url =
848             fmt::format("{}/{}/{}/{}/{}/{}/{}", session_url, coord.x, coord.y,
849                         coord.z, _timepoint, _channel, _angle);
850
851         auto [_, response] = details::requests::make_request(
852             url, Poco::Net::HTTPRequest::HTTP_POST, data,
853             {{"Content-Type", "application/octet-stream"}});
854     }
855
856     return true;
857 }
858
859 template <cnpts::Scalar T>
860 bool ImageView::write_image(const i3d::Image3d<T>& img)const {
861
862     /* Fetch image properties from server */
863     DatasetProperties props = get_dataset_properties(_ip, _port, _uuid);
864     i3d::Vector3d<int> block_dim =
865         details::get_block_dimensions(props, _resolution);
866     i3d::Vector3d<int> img_dim = props.dimensions / _resolution;
867     i3d::Vector3d<int> block_count =
868         (img_dim + block_dim - 1) / block_dim; // Ceiling
869
870     /* Prepare coordinates of blocks and offsets to write whole image */
871     std::vector<i3d::Vector3d<int>> blocks;
872     std::vector<i3d::Vector3d<int>> offsets;
873
874     for (int x = 0; x < block_count.x; ++x)
875         for (int y = 0; y < block_count.y; ++y)
876             for (int z = 0; z < block_count.z; ++z) {
877                 blocks.emplace_back(x, y, z);
878                 offsets.emplace_back(x * block_dim.x, y * block_dim.y,
879                                      z * block_dim.z);
880             }
881
882     /* write whole image */
883     return write_blocks(img, blocks, offsets);
884 }
885
886 /* ==================================== Connection */
887
888 Connection::Connection(std::string ip, int port, std::string uuid)
889     :  _ip(std::move(ip)), _port(port), _uuid(std::move(uuid)) {}
890
891 ImageView Connection::get_view(int channel,
892                                int timepoint,
893                                int angle,
894                                i3d::Vector3d<int> resolution,
895                                const std::string& version)const {
896     return ImageView(_ip, _port, _uuid, channel, timepoint, angle, resolution,
897                      version);
898 }
899
900 template <cnpts::Scalar T>
901 i3d::Image3d<T> Connection::read_block(i3d::Vector3d<int> coord,
902                                        int channel,
903                                        int timepoint,
904                                        int angle,
905                                        i3d::Vector3d<int> resolution,
906                                        const std::string& version)const {
907     return get_view(channel, timepoint, angle, resolution, version)
908         .read_block<T>(coord);
909 }
910
911 template <cnpts::Scalar T>
912 bool Connection::read_block(i3d::Vector3d<int> coord,
913                             i3d::Image3d<T>& dest,
914                             i3d::Vector3d<int> dest_offset,
915                             int channel,
916                             int timepoint,
917                             int angle,
918                             i3d::Vector3d<int> resolution,
919                             const std::string& version)const {
920     return get_view(channel, timepoint, angle, resolution, version)
921         .read_block(coord, dest, dest_offset);
922 }
923
924 template <cnpts::Scalar T>
925 std::vector<i3d::Image3d<T>>
926 Connection::read_blocks(const std::vector<i3d::Vector3d<int>>& coords,
927                         int channel,
928                         int timepoint,
929                         int angle,
930                         i3d::Vector3d<int> resolution,
931                         const std::string& version)const {
932     return get_view(channel, timepoint, angle, resolution, version)
```

```
933         .read_blocks<T>(coords);
934 }
935
936 template <cnpts::Scalar T>
937 bool Connection::read_blocks(
938     const std::vector<i3d::Vector3d<int>& coords,
939     i3d::Image3d<T>& dest,
940     const std::vector<i3d::Vector3d<int>& dest_offsets,
941     int channel,
942     int timepoint,
943     int angle,
944     i3d::Vector3d<int> resolution,
945     const std::string& version)const {
946     return get_view(channel, timepoint, angle, resolution, version)
947         .read_blocks(coords, dest, dest_offsets);
948 }
949
950 template <cnpts::Scalar T>
951 i3d::Image3d<T> Connection::read_image(int channel,
952                                        int timepoint,
953                                        int angle,
954                                        i3d::Vector3d<int> resolution,
955                                        const std::string& version)const {
956     return get_view(channel, timepoint, angle, resolution, version)
957         .read_image<T>();
958 }
959
960 template <cnpts::Scalar T>
961 bool Connection::write_block(const i3d::Image3d<T>& src,
962                              i3d::Vector3d<int> coord,
963                              i3d::Vector3d<int> src_offset,
964                              int channel,
965                              int timepoint,
966                              int angle,
967                              i3d::Vector3d<int> resolution,
968                              const std::string& version)const {
969     return get_view(channel, timepoint, angle, resolution, version)
970         .write_block(src, coord, src_offset);
971 }
972
973 template <cnpts::Scalar T>
974 bool Connection::write_blocks(
975     const i3d::Image3d<T>& src,
976     const std::vector<i3d::Vector3d<int>& coords,
977     const std::vector<i3d::Vector3d<int>& src_offsets,
978     int channel,
979     int timepoint,
980     int angle,
981     i3d::Vector3d<int> resolution,
982     const std::string& version)const {
983     return get_view(channel, timepoint, angle, resolution, version)
984         .write_blocks(src, coords, src_offsets);
985 }
986
987 template <cnpts::Scalar T>
988 bool Connection::write_image(const i3d::Image3d<T>& img,
989                              int channel,
990                              int timepoint,
991                              int angle,
992                              i3d::Vector3d<int> resolution,
993                              const std::string& version)const {
994     return get_view(channel, timepoint, angle, resolution, version)
995         .write_image(img);
996 }
997
998 } // namespace datastore
```

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 datastore {
20 namespace details {
21 #ifdef DATASTORE_NDEBUG
22 constexpr inline bool debug = false;
23 #else
24 #ifdef NDEBUG
25 constexpr inline bool debug = false;
26 #else
27 constexpr inline bool debug = true;
28 #endif
29 #endif
30
39 inline std::string
40 get_dataset_url(const std::string& ip, int port, const std::string& uuid);
41
48 inline DatasetProperties get_dataset_properties(const std::string& dataset_url);
49
57 inline i3d::Vector3d<int> get_block_dimensions(const DatasetProperties& props,
58                                                i3d::Vector3d<int> resolution);
59
71 inline bool check_block_coords(const std::vector<i3d::Vector3d<int>>& coords,
72                                i3d::Vector3d<int> img_dim,
73                                i3d::Vector3d<int> block_dim);
74
89 template <typename T>
90 bool check_offset_coords(const std::vector<i3d::Vector3d<int>>& offsets,
91                          const std::vector<i3d::Vector3d<int>>& coords,
92                          const i3d::Image3d<T>& img,
93                          i3d::Vector3d<int> block_dim,
94                          i3d::Vector3d<int> img_dim);
95 namespace data_manip {
96 inline int get_block_data_size(i3d::Vector3d<int> block_size,
97                                const std::string& voxel_type);
98
99 inline i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
100                                          i3d::Vector3d<int> block_dim,
101                                          i3d::Vector3d<int> img_dim);
102
103 inline int get_linear_index(i3d::Vector3d<int> coord,
104                             i3d::Vector3d<int> block_dim,
105                             const std::string& voxel_type);
106
107 template <typename T>
108 T get_elem_at(std::span<const char> data,
109               const std::string& voxel_type,
110               int index);
111
112 template <typename T>
113 T get_elem_at(std::span<const char> data,
114               const std::string& voxel_type,
115               i3d::Vector3d<int> coord,
116               i3d::Vector3d<int> block_dim);
117
118 template <typename T>
119 void set_elem_at(std::span<char> data,
120                  const std::string& voxel_type,
121                  int index,
122                  T elem);
123
124 template <typename T>
125 void set_elem_at(std::span<char> data,
126                  const std::string& voxel_type,
127                  i3d::Vector3d<int> coord,
128                  i3d::Vector3d<int> block_dim,
129                  T elem);
130
140 template <typename T>
141 void read_data(std::span<const char> data,
142                const std::string& voxel_type,
143                i3d::Image3d<T>& dest,
144                i3d::Vector3d<int> offset);
145
156 template <typename T>
157 void write_data(const i3d::Image3d<T>& src,
158                 i3d::Vector3d<int> offset,
159                 std::span<char> data,
160                 const std::string& voxel_type,
161                 i3d::Vector3d<int> block_size);
162 } // namespace data_manip
163
164 namespace log {
171 inline void _log(const std::string& msg, const std::source_location& location);
172
```

```
179 inline void
180 info(const std::string& msg,
181      const std::source_location& location = std::source_location::current());
182
189 inline void
190 warning(const std::string& msg,
191         const std::source_location& location = std::source_location::current());
192
199 inline void
200 error(const std::string& msg,
201       const std::source_location& location = std::source_location::current());
202 } // namespace log
203
204 /* Helpers to parse Dataset Properties from JSON */
205 namespace props_parser {
206 using namespace Poco::JSON;
207
208 template <cnpts::Basic T>
209 T get_elem(Object::Ptr root, const std::string& name);
210
211 template <cnpts::Vector3d T>
212 T get_elem(Object::Ptr root, const std::string& name);
213
214 template <cnpts::Vector T>
215 T get_elem(Object::Ptr root, const std::string& name);
216
217 template <cnpts::ResolutionUnit T>
218 T get_elem(Object::Ptr root, const std::string& name);
219
220 template <cnpts::Optional T>
221 T get_elem(Object::Ptr root, const std::string& name);
222
223 inline std::vector<std::map<std::string, i3d::Vector3d<int>>>
224 get_resolution_levels(Object::Ptr root);
225
226 } // namespace props_parser
227
228 /* Helpers providing requests functionality */
229 namespace requests {
230 inline std::string session_url_request(const std::string& ds_url,
231                                        i3d::Vector3d<int> resolution,
232                                        const std::string& version);
233
234 inline std::pair<std::vector<char>, Poco::Net::HTTPResponse>
235 make_request(const std::string& url,
236              const std::string& type = Poco::Net::HTTPRequest::HTTP_GET,
237              const std::vector<char>& data = {},
238              const std::map<std::string, std::string>& headers = {});
239 } // namespace requests
240 } // namespace details
241 } // namespace datastore
242
243 /* ================ IMPLEMENTATION FOLLOWS ======================== */
244 namespace datastore {
245 namespace details {
246
247 /* inline */ std::string
248 get_dataset_url(const std::string& ip, int port, const std::string& uuid) {
249     std::string out;
250     if (!ip.starts_with("http://"))
251         out = "https://";
252     return out + fmt::format("{}:{}/datasets/{}", ip, port, uuid);
253 }
254
255 inline DatasetProperties
256 get_dataset_properties(const std::string& dataset_url) {
257     using namespace Poco::JSON;
258
259     /* Fetch JSON from server */
260     auto [data, response] = requests::make_request(dataset_url);
261     std::string json_str(data.begin(), data.end());
262
263     int res_code = response.getStatus();
264     if (res_code != 200)
265         log::warning(fmt::format(
266             "Request ended with code:  {}.  json may not be valid", res_code));
267
268     log::info("Parsing dataset properties from JSON string");
269     Parser parser;
270     Poco::Dynamic::Var result = parser.parse(json_str);
271
272     DatasetProperties props;
273     auto root = result.extract<Object::Ptr>();
274
275     using namespace props_parser;
276
277     /* Parse elements from JSON */
```

```
278
279     props.uuid = get_elem<std::string>(root, "uuid");
280     props.voxel_type = get_elem<std::string>(root, "voxelType");
281     props.dimensions = get_elem<i3d::Vector3d<int>>(root, "dimensions");
282     props.channels = get_elem<int>(root, "channels");
283     props.angles = get_elem<int>(root, "angles");
284     props.transformations =
285         get_elem<std::optional<std::string>>(root, "transformations");
286     props.voxel_unit = get_elem<std::string>(root, "voxelUnit");
287     props.voxel_resolution =
288         get_elem<std::optional<i3d::Vector3d<double>>>(root, "voxelResolution");
289     props.timepoint_resolution =
290         get_elem<std::optional<ResolutionUnit>>(root, "timepointResolution");
291     props.channel_resolution =
292         get_elem<std::optional<ResolutionUnit>>(root, "channelResolution");
293     props.angle_resolution =
294         get_elem<std::optional<ResolutionUnit>>(root, "angleResolution");
295     props.compression = get_elem<std::string>(root, "compression");
296     props.resolution_levels = get_resolution_levels(root);
297     props.versions = get_elem<std::vector<int>>(root, "versions");
298     props.label = get_elem<std::string>(root, "label");
299     props.view_registrations =
300         get_elem<std::optional<std::string>>(root, "viewRegistrations");
301     props.timepoint_ids = get_elem<std::vector<int>>(root, "timepointIds");
302
303     log::info("Parsing has finished");
304     return props;
305 }
306
307 /* inline */ i3d::Vector3d<int>
308 get_block_dimensions(const DatasetProperties& props,
309                      i3d::Vector3d<int> resolution) {
310     for (const auto& res_level :  props.resolution_levels)
311         if (res_level.at("resolutions") == resolution)
312             return res_level.at("blockDimensions");
313
314     log::error(fmt::format("Dimensions for resolution {} not found",
315                            to_string(resolution)));
316     return {-1, -1, -1};
317 }
318
319 /* inline */ bool
320 check_block_coords(const std::vector<i3d::Vector3d<int>>& coords,
321                    i3d::Vector3d<int> img_dim,
322                    i3d::Vector3d<int> block_dim) {
323     /* Act as NOOP if not in debug */
324     if constexpr (!debug)
325         return true;
326
327     log::info("Checking validity of given block coordinates");
328
329     for (i3d::Vector3d<int> coord :  coords)
330         if (data_manip::get_block_size(coord, block_dim, img_dim) ==
331             i3d::Vector3d(0, 0, 0)) {
332             log::error(fmt::format("Block coordinate {} is out of valid range",
333                                    to_string(coord)));
334
335             return false;
336         }
337     log::info("Check successfullly finished");
338     return true;
339 }
340
341 template <typename T>
342 bool check_offset_coords(const std::vector<i3d::Vector3d<int>>& offsets,
343                          const std::vector<i3d::Vector3d<int>>& coords,
344                          const i3d::Image3d<T>& img,
345                          i3d::Vector3d<int> block_dim,
346                          i3d::Vector3d<int> img_dim) {
347     /* Act as NOOP if not in debug */
348     if constexpr (!debug)
349         return true;
350
351     if (offsets.size() != coords.size())
352         return false;
353
354     log::info("Checking validity of given offset coordinates");
355
356     for (std::size_t i = 0; i < coords.size(); ++i) {
357         auto& coord = coords[i];
358         auto& offset = offsets[i];
359
360         i3d::Vector3d<int> block_size =
361             data_manip::get_block_size(coord, block_dim, img_dim);
362         for (int i = 0; i < 3; ++i)
363             if (!(0 <= coord[i] &&
364                   std::size_t(offset[i] + block_size[i]) <= img.GetSize()[i])) {
```

```
365                 log::error(
366                     fmt::format("Offset coordinate {} is out of valid range",
367                                 to_string(coord)));
368 
369                 return false;
370             }
371     }
372     log::info("Check successfullly finished");
373     return true;
374 }
375 
376 namespace data_manip {
377 /* inline */ int get_block_data_size(i3d::Vector3d<int> block_size,
378                                      const std::string& voxel_type) {
379 
380     int elem_size = type_byte_size.at(voxel_type);
381     return block_size.x * block_size.y * block_size.z * elem_size + 12;
382 }
383 
384 /* inline */ i3d::Vector3d<int> get_block_size(i3d::Vector3d<int> coord,
385                                                i3d::Vector3d<int> block_dim,
386                                                i3d::Vector3d<int> img_dim) {
387     i3d::Vector3d<int> start = (coord * block_dim);
388     i3d::Vector3d<int> end = (coord + 1) * block_dim;
389 
390     i3d::Vector3d<int> out;
391     for (int i = 0; i < 3; ++i) {
392         out[i] =
393             std::max(0, std::min(img_dim[i], end[i]) - std::max(start[i], 0));
394     }
395     return out;
396 }
397 
398 /* inline */ int get_linear_index(i3d::Vector3d<int> coord,
399                                   i3d::Vector3d<int> block_dim,
400                                   const std::string& voxel_type) {
401     int elem_size = type_byte_size.at(voxel_type);
402 
403     return 12 +                                     // header_offset
404            (coord.z * block_dim.x * block_dim.y +   // Main axis
405             coord.y * block_dim.x +                // secondary axis
406             coord.x) *                             // last axis
407                elem_size;                          // byte size
408 }
409 
410 template <typename T>
411 T get_elem_at(std::span<const char> data,
412               const std::string& voxel_type,
413               int index) {
414     int elem_size = type_byte_size.at(voxel_type);
415 
416     std::array<char, sizeof(T)> buffer{};
417     std::copy_n(data.begin() + index,      // source start
418                 elem_size,                 // count
419                 buffer.end() - elem_size); // dest start
420 
421     std::ranges::reverse(buffer);
422 
423     return *reinterpret_cast<T*>(&buffer[0]);
424 }
425 
426 template <typename T>
427 T get_elem_at(std::span<const char> data,
428               const std::string& voxel_type,
429               i3d::Vector3d<int> coord,
430               i3d::Vector3d<int> block_dim) {
431 
432     int index = get_linear_index(coord, block_dim, voxel_type);
433     return get_elem_at<T>(data, voxel_type, index);
434 }
435 
436 template <typename T>
437 void set_elem_at(std::span<char> data,
438                  const std::string& voxel_type,
439                  int index,
440                  T elem) {
441     int elem_size = type_byte_size.at(voxel_type);
442 
443     auto buffer = *reinterpret_cast<std::array<char, sizeof(T)>*>(&elem);
444     std::ranges::reverse(buffer);
445 
446     std::copy_n(buffer.end() - elem_size, // source start
447                 elem_size,                // count
448                 data.begin() + index);    // dest start
449 }
450 
451 template <typename T>
```

```
452 void set_elem_at(std::span<char> data,
453                  const std::string& voxel_type,
454                  i3d::Vector3d<int> coord,
455                  i3d::Vector3d<int> block_dim,
456                  T elem) {
457     int index = get_linear_index(coord, block_dim, voxel_type);
458     set_elem_at(data, voxel_type, index, elem);
459 }
460
461 template <typename T>
462 void read_data(std::span<const char> data,
463                const std::string& voxel_type,
464                i3d::Image3d<T>& dest,
465                i3d::Vector3d<int> offset) {
466     i3d::Vector3d<int> block_dim;
467     for (int i = 0; i < 3; ++i)
468         block_dim[i] = get_elem_at<int>(data, "uint32", i * 4);
469
470     for (int x = 0; x < block_dim.x; ++x)
471         for (int y = 0; y < block_dim.y; ++y)
472             for (int z = 0; z < block_dim.z; ++z)
473                 dest.SetVoxel(
474                     x + offset.x, y + offset.y, z + offset.z,
475                     get_elem_at<T>(data, voxel_type, {x, y, z}, block_dim));
476 }
477
478 template <typename T>
479 void write_data(const i3d::Image3d<T>& src,
480                 i3d::Vector3d<int> offset,
481                 std::span<char> data,
482                 const std::string& voxel_type,
483                 i3d::Vector3d<int> block_size) {
484     set_elem_at(data, "uint32", 0, block_size.x);
485     set_elem_at(data, "uint32", 4, block_size.y);
486     set_elem_at(data, "uint32", 8, block_size.z);
487
488     for (int x = 0; x < block_size.x; ++x)
489         for (int y = 0; y < block_size.y; ++y)
490             for (int z = 0; z < block_size.z; ++z)
491                 set_elem_at(
492                     data, voxel_type, {x, y, z}, block_size,
493                     src.GetVoxel(x + offset.x, y + offset.y, z + offset.z));
494 }
495 } // namespace data_manip
496
497 namespace log {
498 /* inline */ void _log(const std::string& msg,
499                        const std::string& type,
500                        const std::source_location& location) {
501     if constexpr (!debug)
502         return;
503
504     std::cout « fmt::format("[{}] {} at row {}:\n{} \n\n", type,
505                             location.function_name(), location.line(), msg);
506     if (type.find("ERROR") != std::string::npos)
507         std::cout « std::flush;
508 }
509
510 /* inline */ void info(const std::string& msg,
511                        const std::source_location&
512                            location /* = std::source_location::current() */) {
513     _log(msg, "INFO", location);
514 }
515
516 /* inline */ void
517 warning(const std::string& msg,
518         const std::source_location&
519             location /* = std::source_location::current() */) {
520     _log(msg, "WARNING", location);
521 }
522
523 /* inline */ void error(const std::string& msg,
524                         const std::source_location&
525                             location /* = std::source_location::current() */) {
526     _log(msg, "ERROR", location);
527 }
528 } // namespace log
529
530 namespace props_parser {
531
532 template <cnpts::Basic T>
533 T get_elem(Object::Ptr root, const std::string& name) {
534     if (!root->has(name)) {
535         log::warning(fmt::format("{} was not found", name));
536         return {};
537     }
538     return root->getValue<T>(name);
```

```
539 }
540
541 template <cnpts::Vector3d T>
542 T get_elem(Object::Ptr root, const std::string& name) {
543     using V = decltype(T{}.x);
544     if (!root->has(name)) {
545         log::warning(fmt::format("{} were not found", name));
546         return {};
547     }
548
549     Array::Ptr values = root->getArray(name);
550     if (values->size() != 3) {
551         log::warning("Incorrect number of dimensions");
552         return {};
553     }
554
555     T out;
556     for (unsigned i = 0; i < 3; ++i)
557         out[i] = values->getElement<V>(i);
558
559     return out;
560 }
561
562 template <cnpts::Vector T>
563 T get_elem(Object::Ptr root, const std::string& name) {
564     using V = typename T::value_type;
565     if (!root->has(name)) {
566         log::warning(fmt::format("{} were not found", name));
567         return {};
568     }
569
570     Array::Ptr values = root->getArray(name);
571     std::size_t count = values->size();
572
573     T out(count);
574     for (unsigned i = 0; i < count; ++i)
575         out[i] = values->getElement<V>(i);
576
577     return out;
578 }
579
580 template <cnpts::ResolutionUnit T>
581 T get_elem(Object::Ptr root, const std::string& name) {
582     if (!root->has(name)) {
583         log::warning(fmt::format("{} was not found", name));
584         return {};
585     }
586
587     Object::Ptr res_ptr = root->getObject(name);
588     ResolutionUnit res;
589
590     if (res_ptr->has("value")) {
591         res.value = res_ptr->getValue<double>("value");
592     }
593
594     if (res_ptr->has("unit")) {
595         res.unit = res_ptr->getValue<std::string>("unit");
596     }
597
598     return res;
599 }
600
601 template <cnpts::Optional T>
602 T get_elem(Object::Ptr root, const std::string& name) {
603     if (!root->has(name)) {
604         log::warning(fmt::format("{} were not found", name));
605         return {};
606     }
607
608     if (root->isNull(name))
609         return {};
610
611     T out;
612     out = get_elem<typename T::value_type>(root, name);
613     return out;
614 }
615
616 /* inline */ std::vector<std::map<std::string, i3d::Vector3d<int>>>
617 get_resolution_levels(Object::Ptr root) {
618     std::string name = "resolutionLevels";
619
620     if (!root->has(name)) {
621         log::warning("resolutionLevels were not found");
622         return {};
623     }
624
625     std::vector<std::map<std::string, i3d::Vector3d<int>>> out;
```

```
626
627     Array::Ptr array = root->getArray(name);
628     for (unsigned i = 0; i < array->size(); ++i) {
629         std::map<std::string, i3d::Vector3d<int» map;
630         Object::Ptr map_ptr = array->getObject(i);
631
632         for (const auto& name :  map_ptr->getNames()) {
633             map[name] = get_elem<i3d::Vector3d<int»(map_ptr, name);
634         }
635
636         out.push_back(map);
637     }
638
639     return out;
640 }
641
642 } // namespace props_parser
643
644 namespace requests {
645 /* inline */ std::string session_url_request(const std::string& ds_url,
646                                              i3d::Vector3d<int> resolution,
647                                              const std::string& version) {
648
649     log::info(
650         fmt::format("Obtaining session url for resolution:  {}, version:  {}",
651                     to_string(resolution), version));
652     std::string req_url =
653         fmt::format("{}/{}/{}/{}/{}/read-write", ds_url, resolution.x,
654                     resolution.y, resolution.z, version);
655
656     auto [_, response] = make_request(req_url);
657
658     int res_code = response.getStatus();
659     if (res_code != 307)
660         log::warning(fmt::format(
661             "Request ended with status:  {}, redirection may be incorrect",
662             res_code));
663
664     return response.get("Location");
665 }
666
667 /* inline */ std::pair<std::vector<char>, Poco::Net::HTTPResponse>
668 make_request(const std::string& url,
669              const std::string& type /*  = Poco::Net::HTTPRequest::HTTP_GET */,
670              const std::vector<char>& data /*  = {} */,
671              const std::map<std::string, std::string>& headers /* = {} */) {
672     Poco::URI uri(url);
673     std::string path(uri.getPathAndQuery());
674
675     Poco::Net::HTTPClientSession session(uri.getHost(), uri.getPort());
676
677     Poco::Net::HTTPRequest request(type, path,
678                                    Poco::Net::HTTPMessage::HTTP_1_1);
679
680     for (auto& [key, value] :  headers)
681         request.set(key, value);
682
683     request.setContentLength(data.size());
684
685     log::info(fmt::format("Sending {} request to url:  {}", type, url));
686     std::ostream& os = session.sendRequest(request);
687     for (char ch :  data)
688         os « ch;
689
690     Poco::Net::HTTPResponse response;
691     std::istream& rs = session.receiveResponse(response);
692
693     std::vector<char> out{std::istreambuf_iterator<char>(rs),
694                           std::istreambuf_iterator<char>()};
695
696     log::info(fmt::format(
697         "Fetched response with status:  {}, reason:  {}, content size:  {}",
698         response.getStatus(), response.getReason(), out.size()));
699
700     return {out, response};
701 }
702
703 } // namespace requests
704 } // namespace details
705 } // namespace datastore
```

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/vector3d.h>
7 #include <map>
8 #include <optional>
9 #include <ostream>
10 #include <sstream>
11 #include <string>
12 #include <vector>
13
14 namespace datastore {
15
16 /* dataset 'voxel_type' to 'byte_size' map*/
17 const inline std::map<std::string, int> type_byte_size{
18     {"uint8", 1}, {"uint16", 2}, {"uint32", 4}, {"uint64", 8},  {"int8", 1},
19     {"int16", 2}, {"int32", 4},  {"int64", 8},  {"float32", 4}, {"float64", 8}};
20
21 /* Maximal legal URL length */
22 constexpr inline std::size_t MAX_URL_LENGTH = 2048;
23
28 class ResolutionUnit {
29   public:
30     double value = 0.0;
31     std::string unit = "";
32
33     friend std::ostream& operator«(std::ostream& stream,
34                                    const ResolutionUnit& res) {
35         stream « fmt::format("{} {}", res.value, res.unit);
36         return stream;
37     }
38 };
39
40 /* Concepts definitions to make templates more readable */
41 namespace cnpts {
42 template <typename T>
43 concept Scalar = requires(T) {
44     requires std::is_scalar_v<T>;
45 };
46
47 template <typename T>
48 concept Basic = requires(T) {
49     requires Scalar<T> || std::is_same_v<T, std::string>;
50 };
51
52 template <typename T>
53 concept Vector = requires(T) {
54     requires std::is_same_v<std::vector<typename T::value_type>, T>;
55 };
56
57 template <typename T>
58 concept Optional = requires(T) {
59     requires std::is_same_v<std::optional<typename T::value_type>, T>;
60 };
61
62 template <typename T>
63 concept Vector3d = requires(T a) {
64     requires std::is_same_v<i3d::Vector3d<decltype(a.x)>, T>;
65     requires Basic<decltype(a.x)>;
66 };
67
68 template <typename T>
69 concept Streamable = requires(T a) {
70     {std::cout « a};
71 };
72
73 template <typename T>
74 concept Map = requires(T) {
75     requires std::is_same_v<
76         std::map<typename T::key_type, typename T::mapped_type>, T>;
77 };
78
79 template <typename T>
80 concept ResolutionUnit = requires(T) {
81     requires std::is_same_v<T, datastore::ResolutionUnit>;
82 };
83 } // namespace cnpts
84
85 namespace details {
86
89 template <cnpts::Streamable T>
90 std::string to_string(const T&);
91
92 template <cnpts::Vector T>
93 std::string to_string(const T&);
94
```

```
95 template <cnpts::Map T>
96 std::string to_string(const T&);
97
98 template <cnpts::Optional T>
99 std::string to_string(const T&);
100
102 template <cnpts::Streamable T>
103 std::string to_string(const T& val) {
104     std::stringstream ss;
105     ss « val;
106     return ss.str();
107 }
108
109 template <cnpts::Vector T>
110 std::string to_string(const T& vec) {
111     std::stringstream ss;
112     ss « "(";
113
114     const char* delim = "";
115     for (auto& v :  vec) {
116         ss « delim « to_string(v);
117         delim = ", ";
118     }
119
120     ss « ")";
121     return ss.str();
122 }
123
124 template <cnpts::Map T>
125 std::string to_string(const T& map) {
126     std::stringstream ss;
127     ss « "{\n";
128
129     for (const auto& [k, v] :  map) {
130         ss « to_string(k) « ":  " « to_string(v) « '\n';
131     }
132     ss « "}\n";
133     return ss.str();
134 }
135
136 template <cnpts::Optional T>
137 std::string to_string(const T& val) {
138     if (!val)
139         return "null";
140     return to_string(val.value());
141 }
142
143 } // namespace details
144
149 class DatasetProperties {
150   public:
151     std::string uuid;
152     std::string voxel_type;
153     i3d::Vector3d<int> dimensions;
154     int channels;
155     int angles;
156     std::optional<std::string> transformations;
157     std::string voxel_unit;
158     std::optional<i3d::Vector3d<double» voxel_resolution;
159     std::optional<ResolutionUnit> timepoint_resolution;
160     std::optional<ResolutionUnit> channel_resolution;
161     std::optional<ResolutionUnit> angle_resolution;
162     std::string compression;
163     std::vector<std::map<std::string, i3d::Vector3d<int»> resolution_levels;
164     std::vector<int> versions;
165     std::string label;
166     std::optional<std::string> view_registrations;
167     std::vector<int> timepoint_ids;
168
169     friend std::ostream& operator«(std::ostream& stream,
170                                    const DatasetProperties& ds) {
171         using details::to_string;
172
173         stream « "UUID: " « ds.uuid « '\n';
174         stream « "voxelType:  " « ds.voxel_type « '\n';
175         stream « "dimensions:  " « ds.dimensions « '\n';
176         stream « "channels:  " « ds.channels « '\n';
177         stream « "angles:  " « ds.angles « '\n';
178         stream « "transformations:  " « to_string(ds.transformations) « '\n';
179         stream « "voxelUnit:  " « ds.voxel_unit « '\n';
180         stream « "voxelResolution:  " « to_string(ds.voxel_resolution) « '\n';
181         stream « "timepointResolution:  " « to_string(ds.timepoint_resolution)
182                « '\n';
183         stream « "channelResolution:  " « to_string(ds.channel_resolution)
184                « '\n';
185         stream « "angleResolution:  " « to_string(ds.angle_resolution) « '\n';
186         stream « "comprestreamion:  " « ds.compression « '\n';
```

```
187         stream « "resolutionLevels:  " « to_string(ds.resolution_levels)
188                « '\n';
189         stream « "versions:  " « to_string(ds.versions) « '\n';
190         stream « "label:  " « ds.label « '\n';
191         stream « "viewRegistrations:  " « to_string(ds.view_registrations)
192                « '\n';
193         stream « "timepointIds:  " « to_string(ds.timepoint_ids) « '\n';
194
195         return stream;
196     }
197 };
198 } // namespace datastore
```

Index