highdicom Documentation

Release 0.20.0

Markus D. Herrmann

Nov 09, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION

The `highdicom` build distribution provides an application programming interface (API) for creating DICOM objects for image-derived information, focusing on Information Object Definitions (IODs) relevant for quantitative imaging, computer vision and machine learning such as Segmentation (SEG) images and Structured Reporting (SR) documents.

The *highdicom* Python package contains several subpackages for different modalities and SOP classes, for example *highdicom.seg* for SEG images and *highdicom.sr* for SR documents.

1.1 Motivation and goals

The DICOM standard is crucial for achieving interoperability between image analysis applications and image storage and communication systems during both development and clinical deployment. However, the standard is vast and complex and implementing it correctly can be challenging - even for DICOM experts. The main goal of *highdicom* is to abstract the complexity of the standard and allow developers of image analysis applications to focus on the algorithm and the data analysis rather than low-level data encoding. To this end, *highdicom* provides a high-level, intuitive application programming interface (API) that enables developers to create high-quality DICOM objects in a few lines of Python code. Importantly, the API is compatible with digital pathology and radiology imaging modalities, including Slide Microscopy (SM), Computed Tomography (CT) and Magnetic Resonance (MR) imaging.

1.2 Design

The *highdicom* Python package exposes an object-orientated application programming interface (API) for the construction of DICOM Service Object Pair (SOP) instances according to the corresponding IODs. Each SOP class is implemented in form of a Python class that inherits from *pydicom.dataset.Dataset*. The class constructor accepts the image-derived information (e.g. pixel data as a *numpy.ndarray*) as well as required contextual information (e.g. patient identifier) as specified by the corresponding IOD. DICOM validation is built-in and is automatically performed upon object construction to ensure created SOP instances are compliant with the standard.

CHAPTER

TWO

INSTALLATION GUIDE

2.1 Requirements

- Python (version 3.6 or higher)
- Python package manager pip

2.2 Installation

Pre-build package available at PyPi:

```
pip install highdicom
```

The library relies on the underlying `pydicom` package for decoding of pixel data, which internally delegates the task to either the `pillow` or the `pylibjpeg` packages. Since `pillow` is a dependency of *highdicom* and will automatically be installed, some transfer syntax can thus be readily decoded and encoded (baseline JPEG, JPEG-2000, JPEG-LS). Support for additional transfer syntaxes (e.g., lossless JPEG) requires installation of the `pylibjpeg` package as well as the `pylibjpeg-libjpeg` and `pylibjpeg-openjpeg` packages. Since `pylibjpeg-libjpeg` is licensed under a copyleft GPL v3 license, it is not installed by default when you install *highdicom*. To install the `pylibjpeg` packages along with *highdicom*, use

```
pip install highdicom[libjpeg]
```

Install directly from source code (available on Github):

```
git clone https://github.com/herrmannlab/highdicom ~/highdicom
pip install ~/highdicom
```

CHAPTER

THREE

USER GUIDE

Creating and parsing DICOM objects using the *highdicom* package.

3.1 Creating Segmentation (SEG) images

Derive a Segmentation image from a series of single-frame Computed Tomography (CT) images:

```
from pathlib import Path

import highdicom as hd
import numpy as np
from pydicom.sr.codedict import codes
from pydicom.filereader import dcmread

# Path to directory containing single-frame legacy CT Image instances
# stored as PS3.10 files
series_dir = Path('path/to/series/directory')
image_files = series_dir.glob('*.dcm')

# Read CT Image data sets from PS3.10 files on disk
image_datasets = [dcmread(str(f)) for f in image_files]

# Create a binary segmentation mask
mask = np.zeros(
    shape=(
        len(image_datasets),
        image_datasets[0].Rows,
        image_datasets[0].Columns
    ),
    dtype=np.bool
)
mask[1:-1, 10:-10, 100:-100] = True

# Describe the algorithm that created the segmentation
algorithm_identification = hd.AlgorithmIdentificationSequence(
    name='test',
    version='v1.0',
    family=codes.cid7162.ArtificialIntelligence
)
```

(continues on next page)

(continued from previous page)

```
# Describe the segment
description_segment_1 = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='first segment',
    segmented_property_category=codes.cid7150.Tissue,
    segmented_property_type=codes.cid7166.ConnectiveTissue,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.AUTOMATIC,
    algorithm_identification=algorithm_identification,
    tracking_uid=hd.UID(),
    tracking_id='test segmentation of computed tomography image'
)

# Create the Segmentation instance
seg_dataset = hd.seg.Segmentation(
    source_images=image_datasets,
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=[description_segment_1],
    series_instance_uid=hd.UID(),
    series_number=2,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Manufacturer',
    manufacturer_model_name='Model',
    software_versions='v1',
    device_serial_number='Device XYZ',
)

print(seg_dataset)

seg_dataset.save_as("seg.dcm")
```

Derive a Segmentation image from a multi-frame Slide Microscopy (SM) image:

```
from pathlib import Path

import highdicom as hd
import numpy as np
from pydicom.sr.codedict import codes
from pydicom.filereader import dcmread

# Path to multi-frame SM image instance stored as PS3.10 file
image_file = Path('/path/to/image/file')

# Read SM Image data set from PS3.10 files on disk
image_dataset = dcmread(str(image_file))

# Create a binary segmentation mask
mask = np.max(image_dataset.pixel_array, axis=3) > 1

# Describe the algorithm that created the segmentation
algorithm_identification = hd.AlgorithmIdentificationSequence(
```

(continues on next page)

(continued from previous page)

```
    name='test',
    version='v1.0',
    family=codes.cid7162.ArtificialIntelligence
)

# Describe the segment
description_segment_1 = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='first segment',
    segmented_property_category=codes.cid7150.Tissue,
    segmented_property_type=codes.cid7166.ConnectiveTissue,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.AUTOMATIC,
    algorithm_identification=algorithm_identification,
    tracking_uid=hd.UID(),
    tracking_id='test segmentation of slide microscopy image'
)

# Create the Segmentation instance
seg_dataset = hd.seg.Segmentation(
    source_images=[image_dataset],
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=[description_segment_1],
    series_instance_uid=hd.UID(),
    series_number=2,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Manufacturer',
    manufacturer_model_name='Model',
    software_versions='v1',
    device_serial_number='Device XYZ'
)

print(seg_dataset)
```

3.2 Parsing Segmentation (SEG) images

Iterating over segments in a segmentation image instance:

```
from pathlib import Path

import highdicom as hd
from pydicom.filereader import dcmread

# Path to multi-frame SEG image instance stored as PS3.10 file
seg_file = Path('/path/to/seg/file')

# Read SEG Image data set from PS3.10 files on disk
seg_dataset = dcmread(str(seg_file))
```

(continues on next page)

(continued from previous page)

```
# Iterate over segments and print the information about the frames
# that encode the segment across different image positions
for frames, frame_descriptions, description in hd.seg.utils.iter_segments(seg_dataset):
    print(frames.shape)
    print(
        set([
            item.SegmentIdentificationSequence[0].ReferencedSegmentNumber
            for item in frame_descriptions
        ])
    )
    print(description.SegmentNumber)
```

3.3 Creating Structured Report (SR) documents

Create a Structured Report document that contains a numeric area measurement for a planar region of interest (ROI) in a single-frame computed tomography (CT) image:

```
from pathlib import Path

import highdicom as hd
import numpy as np
from pydicom.filereader import dcmread
from pydicom.sr.codedict import codes
from pydicom.uid import generate_uid
from highdicom.sr.content import FindingSite
from highdicom.sr.templates import Measurement, TrackingIdentifier

# Path to single-frame CT image instance stored as PS3.10 file
image_file = Path('/path/to/image/file')

# Read CT Image data set from PS3.10 files on disk
image_dataset = dcmread(str(image_file))

# Describe the context of reported observations: the person that reported
# the observations and the device that was used to make the observations
observer_person_context = hd.sr.ObserverContext(
    observer_type=codes.DCM.Person,
    observer_identifying_attributes=hd.sr.PersonObserverIdentifyingAttributes(
        name='Foo'
    )
)
observer_device_context = hd.sr.ObserverContext(
    observer_type=codes.DCM.Device,
    observer_identifying_attributes=hd.sr.DeviceObserverIdentifyingAttributes(
        uid=hd.UID()
    )
)
observation_context = hd.sr.ObservationContext(
    observer_person_context=observer_person_context,
    observer_device_context=observer_device_context,
```

(continues on next page)

(continued from previous page)

```
)

# Describe the image region for which observations were made
# (in physical space based on the frame of reference)
referenced_region = hd.sr.ImageRegion3D(
    graphic_type=hd.sr.GraphicTypeValues3D.POLYGON,
    graphic_data=np.array([
        (165.0, 200.0, 134.0),
        (170.0, 200.0, 134.0),
        (170.0, 220.0, 134.0),
        (165.0, 220.0, 134.0),
        (165.0, 200.0, 134.0),
    ]),
    frame_of_reference_uid=image_dataset.FrameOfReferenceUID
)

# Describe the anatomic site at which observations were made
finding_sites = [
    FindingSite(
        anatomic_location=codes.SCT.CervicoThoracicSpine,
        topographical_modifier=codes.SCT.VertebralForamen
    ),
]

# Describe the imaging measurements for the image region defined above
measurements = [
    Measurement(
        name=codes.SCT.AreaOfDefinedRegion,
        tracking_identifier=hd.sr.TrackingIdentifier(uid=generate_uid()),
        value=1.7,
        unit=codes.UCUM.SquareMillimeter,
        properties=hd.sr.MeasurementProperties(
            normality=hd.sr.CodedConcept(
                value="17621005",
                meaning="Normal",
                scheme_designator="SCT"
            ),
            level_of_significance=codes.SCT.NotSignificant
        )
    )
]
imaging_measurements = [
    hd.sr.PlanarROIMeasurementsAndQualitativeEvaluations(
        tracking_identifier=TrackingIdentifier(
            uid=hd.UID(),
            identifier='Planar ROI Measurements'
        ),
        referenced_region=referenced_region,
        finding_type=codes.SCT.SpinalCord,
        measurements=measurements,
        finding_sites=finding_sites
    )
```

(continues on next page)

(continued from previous page)

```
]

# Create the report content
measurement_report = hd.sr.MeasurementReport(
    observation_context=observation_context,
    procedure_reported=codes.LN.CTUnspecifiedBodyRegion,
    imaging_measurements=imaging_measurements
)

# Create the Structured Report instance
sr_dataset = hd.sr.Comprehensive3DSR(
    evidence=[image_dataset],
    content=measurement_report[0],
    series_number=1,
    series_instance_uid=hd.UID(),
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Manufacturer'
)

print(sr_dataset)
```

3.4 Parsing Structured Report (SR) documents

Finding relevant content in the nested SR content tree:

```
from pathlib import Path

import highdicom as hd
from pydicom.filereader import dcmread
from pydicom.sr.codedict import codes

# Path to SR document instance stored as PS3.10 file
document_file = Path('/path/to/document/file')

# Load document from file on disk
sr_dataset = dcmread(str(document_file))

# Find all content items that may contain other content items.
containers = hd.sr.utils.find_content_items(
    dataset=sr_dataset,
    relationship_type=RelationshipTypeValues.CONTAINS
)
print(containers)

# Query content of SR document, where content is structured according
# to TID 1500 "Measurment Report"
if sr_dataset.ContentTemplateSequence[0].TemplateIdentifier == 'TID1500':
    # Determine who made the observations reported in the document
    observers = hd.sr.utils.find_content_items(
```

(continues on next page)

(continued from previous page)

```
        dataset=sr_dataset,
        name=codes.DCM.PersonObserverName
    )
    print(observers)

    # Find all imaging measurements reported in the document
    measurements = hd.sr.utils.find_content_items(
        dataset=sr_dataset,
        name=codes.DCM.ImagingMeasurements,
        recursive=True
    )
    print(measurements)

    # Find all findings reported in the document
    findings = hd.sr.utils.find_content_items(
        dataset=sr_dataset,
        name=codes.DCM.Finding,
        recursive=True
    )
    print(findings)

    # Find regions of interest (ROI) described in the document
    # in form of spatial coordinates (SCOORD)
    regions = hd.sr.utils.find_content_items(
        dataset=sr_dataset,
        value_type=ValueTypeValues.SCOORD,
        recursive=True
    )
    print(regions)
```

3.5 Creating Secondary Capture (SC) images

Secondary captures are a way to store images that were not created directly by an imaging modality within a DICOM file. They are often used to store screenshots or overlays, and are widely supported by viewers. However other methods of displaying image derived information, such as segmentation images and structured reports should be preferred if they are supported because they can capture more detail about how the derived information was obtained and what it represents.

In this example, we use a secondary capture to store an image containing a labeled bounding box region drawn over a CT image.

```
import highdicom as hd
import numpy as np
from pydicom import dcmread
from pydicom.uid import RLELossless
from PIL import Image, ImageDraw

# Read in the source CT image
image_dataset = dcmread('/path/to/image.dcm')
```

(continues on next page)

(continued from previous page)

```
# Create an image for display by windowing the original image and drawing a
# bounding box over it using Pillow's ImageDraw module
slope = getattr(image_dataset, 'RescaleSlope', 1)
intercept = getattr(image_dataset, 'RescaleIntercept', 0)
original_image = image_dataset.pixel_array * slope + intercept

# Window the image to a soft tissue window (center 40, width 400)
# and rescale to the range 0 to 255
lower = -160
upper = 240
windowed_image = np.clip(original_image, lower, upper)
windowed_image = (windowed_image - lower) * 255 / (upper - lower)
windowed_image = windowed_image.astype(np.uint8)

# Create RGB channels
windowed_image = np.tile(windowed_image[:, :, np.newaxis], [1, 1, 3])

# Cast to a PIL image for easy drawing of boxes and text
pil_image = Image.fromarray(windowed_image)

# Draw a red bounding box over part of the image
x0 = 10
y0 = 10
x1 = 60
y1 = 60
draw_obj = ImageDraw.Draw(pil_image)
draw_obj.rectangle(
    [x0, y0, x1, y1],
    outline='red',
    fill=None,
    width=3
)

# Add some text
draw_obj.text(xy=[10, 70], text='Region of Interest', fill='red')

# Convert to numpy array
pixel_array = np.array(pil_image)

# The patient orientation defines the directions of the rows and columns of the
# image, relative to the anatomy of the patient.  In a standard CT axial image,
# the rows are oriented leftwards and the columns are oriented posteriorly, so
# the patient orientation is ['L', 'P']
patient_orientation=['L', 'P']

# Create the secondary capture image. By using the `from_ref_dataset`
# constructor, all the patient and study information will be copied from the
# original image dataset
sc_image = hd.sc.SCImage.from_ref_dataset(
    ref_dataset=image_dataset,
    pixel_array=pixel_array,
    photometric_interpretation=hd.PhotometricInterpretationValues.RGB,
```

(continues on next page)

(continued from previous page)

```
    bits_allocated=8,
    coordinate_system=hd.CoordinateSystemNames.PATIENT,
    series_instance_uid=hd.UID(),
    sop_instance_uid=hd.UID(),
    series_number=100,
    instance_number=1,
    manufacturer='Manufacturer',
    pixel_spacing=image_dataset.PixelSpacing,
    patient_orientation=patient_orientation,
    transfer_syntax_uid=RLELossless
)

# Save the file
sc_image.save_as('sc_output.dcm')
```

To save a 3D image as a series of output slices, simply loop over the 2D slices and ensure that the individual output instances share a common series instance UID. Here is an example for a CT scan that is in a NumPy array called "ct_to_save" where we do not have the original DICOM files on hand. We want to overlay a segmentation that is stored in a NumPy array called "seg_out".

```
import highdicom as hd
import numpy as np
import os

pixel_spacing = [1.0, 1.0]
sz = ct_to_save.shape[2]
series_instance_uid = hd.UID()
study_instance_uid = hd.UID()

for iz in range(sz):
    this_slice = ct_to_save[:, :, iz]

    # Window the image to a soft tissue window (center 40, width 400)
    # and rescale to the range 0 to 255
    lower = -160
    upper = 240
    windowed_image = np.clip(this_slice, lower, upper)
    windowed_image = (windowed_image - lower) * 255 / (upper - lower)

    # Create RGB channels
    pixel_array = np.tile(windowed_image[:, :, np.newaxis], [1, 1, 3])

    # transparency level
    alpha = 0.1

    pixel_array[:, :, 0] = 255 * (1 - alpha) * seg_out[:, :, iz] + alpha * pixel_array[:,
↪ :, 0]
    pixel_array[:, :, 1] = alpha * pixel_array[:, :, 1]
    pixel_array[:, :, 2] = alpha * pixel_array[:, :, 2]

    patient_orientation = ['L', 'P']
```

(continues on next page)

(continued from previous page)

```
    # Create the secondary capture image
    sc_image = hd.sc.SCImage(
        pixel_array=pixel_array.astype(np.uint8),
        photometric_interpretation=hd.PhotometricInterpretationValues.RGB,
        bits_allocated=8,
        coordinate_system=hd.CoordinateSystemNames.PATIENT,
        study_instance_uid=study_instance_uid,
        series_instance_uid=series_instance_uid,
        sop_instance_uid=hd.UID(),
        series_number=100,
        instance_number=iz + 1,
        manufacturer='Manufacturer',
        pixel_spacing=pixel_spacing,
        patient_orientation=patient_orientation,
    )

    sc_image.save_as(os.path.join("output", 'sc_output_' + str(iz) + '.dcm'))
```

3.6 Creating Grayscale Softcopy Presentation State (GSPS) Objects

A presentation state contains information about how another image should be rendered, and may include "annotations" in the form of basic shapes, polylines, and text overlays. Note that a GSPS is not recommended for storing annotations for any purpose except visualization. A structured report would usually be preferred for storing annotations for clinical or research purposes.

```
import highdicom as hd

import numpy as np
from pydicom import dcmread
from pydicom.valuerep import PersonName


# Read in an example CT image
image_dataset = dcmread('path/to/image.dcm')

# Create an annotation containing a polyline
polyline = hd.pr.GraphicObject(
    graphic_type=hd.pr.GraphicTypeValues.POLYLINE,
    graphic_data=np.array([
        [10.0, 10.0],
        [20.0, 10.0],
        [20.0, 20.0],
        [10.0, 20.0]]
    ),  # coordinates of polyline vertices
    units=hd.pr.AnnotationUnitsValues.PIXEL,  # units for graphic data
    tracking_id='Finding1',  # site-specific ID
    tracking_uid=hd.UID()  # highdicom will generate a unique ID
)

# Create a text object annotation
```

(continues on next page)

(continued from previous page)

```
text = hd.pr.TextObject(
    text_value='Important Finding!',
    bounding_box=np.array(
        [30.0, 30.0, 40.0, 40.0]  # left, top, right, bottom
    ),
    units=hd.pr.AnnotationUnitsValues.PIXEL,  # units for bounding box
    tracking_id='Finding1Text',  # site-specific ID
    tracking_uid=hd.UID()  # highdicom will generate a unique ID
)

# Create a single layer that will contain both graphics
# There may be multiple layers, and each GraphicAnnotation object
# belongs to a single layer
layer = hd.pr.GraphicLayer(
    layer_name='LAYER1',
    order=1,  # order in which layers are displayed (lower first)
    description='Simple Annotation Layer',
)

# A GraphicAnnotation may contain multiple text and/or graphic objects
# and is rendered over all referenced images
annotation = hd.pr.GraphicAnnotation(
    referenced_images=[image_dataset],
    graphic_layer=layer,
    graphic_objects=[polyline],
    text_objects=[text]
)

# Assemble the components into a GSPS object
gsps = hd.pr.GrayscaleSoftcopyPresentationState(
    referenced_images=[image_dataset],
    series_instance_uid=hd.UID(),
    series_number=123,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Manufacturer',
    manufacturer_model_name='Model',
    software_versions='v1',
    device_serial_number='Device XYZ',
    content_label='ANNOTATIONS',
    graphic_layers=[layer],
    graphic_annotations=[annotation],
    institution_name='MGH',
    institutional_department_name='Radiology',
    content_creator_name=PersonName.from_named_components(
        family_name='Doe',
        given_name='John'
    ),
)

# Save the GSPS file
gsps.save_as('gsps.dcm')
```

CHAPTER

FOUR

DEVELOPER GUIDE

Source code is available at Github and can be cloned via git:

```
git clone https://github.com/herrmannlab/highdicom ~/highdicom
```

The *highdicom* package can be installed in *develop* mode for local development:

```
pip install -e ~/highdicom
```

4.1 Pull requests

Don't commit code changes to the `master` branch. New features should be implemented in a separate branch called `feature/*` and bug fixes should be applied in separate branch called `bugfix/*`.

Before creating a pull request on Github, read the coding style guideline, run the tests and check PEP8 compliance.

4.2 Coding style

Code must comply with PEP 8. The flake8 package is used to enforce compliance.

The project uses numpydoc for documenting code according to PEP 257 docstring conventions. Further information and examples for the NumPy style can be found at the NumPy Github repository and the website of the Napoleon sphinx extension.

All API classes, functions and modules must be documented (including "private" functions and methods). Each docstring must describe input parameters and return values. Types must be specified using type hints as specified by PEP 484 (see typing module) in both the function definition as well as the docstring.

4.3 Running tests

The project uses pytest to write and runs unit tests. Tests should be placed in a separate `tests` folder within the package root folder. Files containing actual test code should follow the pattern `test_*.py`.

Install requirements:

```
pip install -r ~/highdicom/requirements_test.txt
```

Run tests (including checks for PEP8 compliance):

```
cd ~/highdicom
pytest --flake8
```

4.4 Building documentation

Install requirements:

```
pip install -r ~/highdicom/requirements_docs.txt
```

Build documentation in *HTML* format:

```
cd ~/highdicom
sphinx-build -b html docs/ docs/build/
```

The built `index.html` file will be located in `docs/build`.

4.5 Design principles

Interoperability with Pydicom - Highdicom is built on the pydicom library. Highdicom types are typically derived from the `pydicom.dataset.Dataset` or `pydicom.sequence.Sequence` classes and should remain interoperable with them as far as possible such that experienced users can use the lower-level pydicom API to inspect or change the object if needed.

Standard DICOM Terminology - Where possible, highdicom types, functions, parameters, enums, etc map onto concepts within the DICOM standard and should follow the same terminology to ensure that the meaning is unambiguous. Where the terminology used in the standard may not be easily understood by those unfamiliar with it, this should be addressed via documentation rather than using alternative terminology.

Standard Compliance on Encoding - Highdicom should not allow users to create DICOM objects that are not in compliance with the standard. The library should validate all parameters passed to it and should raise an exception if they would result in the creation of an invalid object, and give a clear explanation to the user why the parameters passed are invalid. Furthermore, highdicom objects should always exist in a state of standards compliance, without any intermediate invalid states. Once a constructor has completed, the user should be confident that they have a valid object.

Standard Compliance on Decoding - Unfortunately, many DICOM objects found in the real world have minor deviations from the standard. When decoding DICOM objects, highdicom should tolerate minor deviations as far as they do not interfere with its functionality. When highdicom needs to assume that objects are standard compliant in order to function, it should check this assumption first and raise an exception explaining the issue to the user if it finds an error. Unless there are exceptional circumstances, highdicom should not attempt to work around issues in non-compliant files produced by other implementations.

The Decoding API - Highdicom classes implement functionality for conveniently accessing information contained within the relevant dataset. To use this functionality with existing pydicom dataset, such as those read in from file or received over network, the dataset must first be converted to the relevant highdicom type. This is implemented by the alternative `from_dataset()` or `from_sequence()` constructors on highdicom types. These methods should perform "eager" type conversion of the dataset and all datasets contained within it into the relevant highdicom types, where they exist. This way, objects created from scratch by users and those converted from pydicom datasets using `from_dataset()` or `from_sequence()` should appear identical to users and developers as far as possible.

CHAPTER

FIVE

CONFORMANCE STATEMENT

CHAPTER

SIX

CITATION

The following article describes in detail the motivation for creating the *highdicom* library, the design goals of the library, and experiments demonstrating the library's capabilities. If you use *highdicom* in research, please cite this article.

> Highdicom: A Python library for standardized encoding of image annotations and machine learning model outputs in pathology and radiology. C.P. Bridge, C. Gorman, S. Pieper, S.W. Doyle, J.K. Lennerz, J. Kalpathy-Cramer, D.A. Clunie, A.Y. Fedorov, and M.D. Herrmann.

CHAPTER

SEVEN

LICENSE

highdicom is free and open source software licensed under the permissive MIT license.

CHAPTER

EIGHT

API DOCUMENTATION

8.1 highdicom package

class highdicom.**AlgorithmIdentificationSequence**(*name*, *family*, *version*, *source=None*, *parameters=None*)

Bases: Sequence

Sequence of data elements describing information useful for identification of an algorithm.

Parameters

- **name** (*str*) – Name of the algorithm
- **family** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Kind of algorithm family
- **version** (*str*) – Version of the algorithm
- **source** (*str, optional*) – Source of the algorithm, e.g. name of the algorithm manufacturer
- **parameters** (*Dict[str, str], optional*) – Name and actual value of the parameters with which the algorithm was invoked

property family: *CodedConcept*

Kind of the algorithm family.

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

classmethod from_sequence(*sequence*)

Construct instance from an existing data element sequence.

Parameters
sequence (*pydicom.sequence.Sequence*) – Data element sequence representing the Algorithm Identification Sequence

Returns
Algorithm Identification Sequence

Return type
highdicom.seg.content.AlgorithmIdentificationSequence

property name: str

Name of the algorithm.

Type
str

Return type
str

property parameters: Optional[Dict[str, str]]

Union[Dict[str, str], None]: Dictionary mapping algorithm parameter names to values, if any

Return type
typing.Optional[typing.Dict[str, str]]

property source: Optional[str]

Union[str, None]: Source of the algorithm, e.g. name of the algorithm manufacturer, if any

Return type
typing.Optional[str]

property version: str

Version of the algorithm.

Type
str

Return type
str

class highdicom.AnatomicalOrientationTypeValues(*value*)

Bases: Enum

Enumerated values for Anatomical Orientation Type attribute.

BIPED = 'BIPED'

QUADRUPED = 'QUADRUPED'

class highdicom.ContentCreatorIdentificationCodeSequence(*person_identification_codes*, *institution_name*, *person_address=None*, *person_telephone_numbers=None*, *person_telecom_information=None*, *institution_code=None*, *institution_address=None*, *institutional_department_name=None*, *institutional_department_type_code=None*)

Bases: Sequence

Sequence of data elements for identifying the person who created content.

Parameters

- **person_identification_codes** (*Sequence[Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]]*) – Coded description(s) identifying the person.
- **institution_name** (*str*) – Name of the to which the identified individual is responsible or accountable.
- **person_address** (*Union[str, None]*) – Mailing address of the person.

- **person_telephone_numbers** (*Union[Sequence[str], None], optional*) – Person's telephone number(s).
- **person_telecom_information** (*Union[str, None], optional*) – The person's telecommunication contact information, including email or other addresses.
- **institution_code** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Coded concept identifying the institution.
- **institution_address** (*Union[str, None], optional*) – Mailing address of the institution.
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department, unit or service within the healthcare facility.
- **institutional_department_type_code** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – A coded description of the type of Department or Service.

class highdicom.**ContentQualificationValues**(*value*)

Bases: Enum

Enumerated values for Content Qualification attribute.

PRODUCT = 'PRODUCT'

RESEARCH = 'RESEARCH'

SERVICE = 'SERVICE'

class highdicom.**CoordinateSystemNames**(*value*)

Bases: Enum

Enumerated values for coordinate system names.

PATIENT = 'PATIENT'

SLIDE = 'SLIDE'

class highdicom.**DimensionOrganizationTypeValues**(*value*)

Bases: Enum

Enumerated values for Dimension Organization Type attribute.

THREE_DIMENSIONAL = '3D'

THREE_DIMENSIONAL_TEMPORAL = '3D_TEMPORAL'

TILED_FULL = 'TILED_FULL'

TILED_SPARSE = 'TILED_SPARSE'

class highdicom.**IssuerOfIdentifier**(*issuer_of_identifier*, *issuer_of_identifier_type=None*)

Bases: Dataset

Dataset describing the issuer or a specimen or container identifier.

Parameters

- **issuer_of_identifier** (*str*) – Identifier of the entity that created the examined specimen

- **issuer_of_identifier_type** (*Union[str, highdicom.enum.UniversalEntityIDTypeValues], optional*) – Type of identifier of the entity that created the examined specimen (required if *issuer_of_specimen_id* is a Unique Entity ID)

class highdicom.**LUT**(*first_mapped_value*, *lut_data*, *lut_explanation=None*)

Bases: Dataset

Dataset describing a lookup table (LUT).

Parameters

- **first_mapped_value** (*int*) – Pixel value that will be mapped to the first value in the lookup-table.
- **lut_data** (*numpy.ndarray*) – Lookup table data. Must be of type uint16.
- **lut_explanation** (*Union[str, None], optional*) – Free-form text explanation of the meaning of the LUT.

Note: After the LUT is applied, a pixel in the image with value equal to `first_mapped_value` is mapped to an output value of `lut_data[0]`, an input value of `first_mapped_value + 1` is mapped to `lut_data[1]`, and so on.

---

property **bits_per_entry: int**

Bits allocated for the lookup table data. 8 or 16.

Type
int

Return type
int

property **first_mapped_value: int**

Pixel value that will be mapped to the first value in the lookup table.

Type
int

Return type
int

property **lut_data: ndarray**

LUT data

Type
numpy.ndarray

Return type
numpy.ndarray

property **number_of_entries: int**

Number of entries in the lookup table.

Type
int

Return type
int

class highdicom.**LateralityValues**(*value*)

Bases: Enum

Enumerated values for Laterality attribute.

L = 'L'

Left

R = 'R'

Right

class highdicom.**ModalityLUT**(*lut_type*, *first_mapped_value*, *lut_data*, *lut_explanation=None*)

Bases: *LUT*

Dataset describing an item of the Modality LUT Sequence.

Parameters

- **lut_type** (*Union[highdicom.RescaleTypeValues, str]*) – String or enumerated value specifying the units of the output of the LUT operation.
- **first_mapped_value** (*int*) – Pixel value that will be mapped to the first value in the lookup-table.
- **lut_data** (*numpy.ndarray*) – Lookup table data. Must be of type uint16.
- **lut_explanation** (*Union[str, None], optional*) – Free-form text explanation of the meaning of the LUT.

class highdicom.**ModalityLUTTransformation**(*rescale_intercept=None*, *rescale_slope=None*, *rescale_type=None*, *modality_lut=None*)

Bases: Dataset

Dataset describing the Modality LUT Transformation as part of the Pixel Transformation Sequence to transform the manufacturer dependent pixel values into pixel values that are meaningful for the modality and are manufacturer independent.

Parameters

- **rescale_intercept** (*Union[int, float, None], optional*) – Intercept of linear function used for rescaling pixel values.
- **rescale_slope** (*Union[int, float, None], optional*) – Slope of linear function used for rescaling pixel values.
- **rescale_type** (*Union[highdicom.RescaleTypeValues, str, None], optional*) – String or enumerated value specifying the units of the output of the Modality LUT or rescale operation.
- **modality_lut** (*Union[highdicom.ModalityLUT, None], optional*) – Lookup table specifying a pixel rescaling operation to apply to the stored values to give modality values.

Note: Either *modality_lut* may be specified or all three of *rescale_slope*, *rescale_intercept*, and *rescale_type* may be specified. All four parameters should not be specified simultaneously.

---

class highdicom.**PaletteColorLUT**(*first_mapped_value*, *lut_data*, *color*)

Bases: Dataset

Dataset describing a palette color lookup table (LUT).

Parameters

- **first_mapped_value** (*int*) – Pixel value that will be mapped to the first value in the lookup table.
- **lut_data** (*numpy.ndarray*) – Lookup table data. Must be of type uint16.
- **color** (*str*) – Text representing the color (`red`, `green`, or `blue`).

Note: After the LUT is applied, a pixel in the image with value equal to `first_mapped_value` is mapped to an output value of `lut_data[0]`, an input value of `first_mapped_value + 1` is mapped to `lut_data[1]`, and so on.

---

property bits_per_entry: int

Bits allocated for the lookup table data. 8 or 16.

Type
int

Return type
int

property first_mapped_value: int

Pixel value that will be mapped to the first value in the lookup table.

Type
int

Return type
int

property lut_data: ndarray

lookup table data

Type
numpy.ndarray

Return type
numpy.ndarray

property number_of_entries: int

Number of entries in the lookup table.

Type
int

Return type
int

class highdicom.**PaletteColorLUTTransformation**(*red_lut*, *green_lut*, *blue_lut*, *palette_color_lut_uid=None*)

Bases: `Dataset`

Dataset describing the Palette Color LUT Transformation as part of the Pixel Transformation Sequence to transform grayscale into RGB color pixel values.

Parameters

- **red_lut** (*Union[highdicom.PaletteColorLUT, highdicom.SegmentedPaletteColorLUT]*) – Lookup table for the red output color channel.
- **green** (*Union[highdicom.PaletteColorLUT, highdicom.SegmentedPaletteColorLUT]*) – Lookup table for the green output color channel.

- **blue_lut** (*Union[highdicom.PaletteColorLUT, highdicom.SegmentedPaletteColorLUT]*) – Lookup table for the blue output color channel.
- **palette_color_lut_uid** (*Union[highdicom.UID, str, None], optional*) – Unique identifier for the palette color lookup table.

property blue_lut: Union[*PaletteColorLUT*, *SegmentedPaletteColorLUT*]

Union[highdicom.PaletteColorLUT, highdicom.SegmentedPaletteColorLUT]: Lookup table for the blue output color channel

Return type
typing.Union[*highdicom.content.PaletteColorLUT*, *highdicom.content.SegmentedPaletteColorLUT*]

property green_lut: Union[*PaletteColorLUT*, *SegmentedPaletteColorLUT*]

Union[highdicom.PaletteColorLUT, highdicom.SegmentedPaletteColorLUT]: Lookup table for the green output color channel

Return type
typing.Union[*highdicom.content.PaletteColorLUT*, *highdicom.content.SegmentedPaletteColorLUT*]

property red_lut: Union[*PaletteColorLUT*, *SegmentedPaletteColorLUT*]

Union[highdicom.PaletteColorLUT, highdicom.SegmentedPaletteColorLUT]: Lookup table for the red output color channel

Return type
typing.Union[*highdicom.content.PaletteColorLUT*, *highdicom.content.SegmentedPaletteColorLUT*]

class highdicom.**PatientOrientationValuesBiped**(*value*)

Bases: Enum

Enumerated values for Patient Orientation attribute if Anatomical Orientation Type attribute has value "BIPED".

A = 'A'
Anterior

F = 'F'
Foot

H = 'H'
Head

L = 'L'
Left

P = 'P'
Posterior

R = 'R'
Right

class highdicom.**PatientOrientationValuesQuadruped**(*value*)

Bases: Enum

Enumerated values for Patient Orientation attribute if Anatomical Orientation Type attribute has value "QUADRUPED".

CD = 'CD'
Caudal

CR = 'CR'
Cranial

D = 'D'
Dorsal

DI = 'DI'
Distal

L = 'L'
Lateral

LE = 'LE'
Left

M = 'M'
Medial

PA = 'PA'
Palmar

PL = 'PL'
Plantar

PR = 'PR'
Proximal

R = 'R'
Rostral

RT = 'RT'
Right

V = 'V'
Ventral

class highdicom.**PatientSexValues**(*value*)

Bases: Enum

Enumerated values for Patient's Sex attribute.

F = 'F'
Female

M = 'M'
Male

O = 'O'
Other

class highdicom.**PhotometricInterpretationValues**(*value*)

Bases: Enum

Enumerated values for Photometric Interpretation attribute.

See Section C.7.6.3.1.2 for more information.

MONOCHROME1 = 'MONOCHROME1'

MONOCHROME2 = 'MONOCHROME2'

PALETTE_COLOR = 'PALETTE COLOR'

RGB = 'RGB'

YBR_FULL = 'YBR_FULL'

YBR_FULL_422 = 'YBR_FULL_422'

YBR_ICT = 'YBR_ICT'

YBR_PARTIAL_420 = 'YBR_PARTIAL_420'

YBR_RCT = 'YBR_RCT'

class highdicom.**PixelMeasuresSequence**(*pixel_spacing*, *slice_thickness*, *spacing_between_slices=None*)

Bases: Sequence

Sequence of data elements describing physical spacing of an image based on the Pixel Measures functional group macro.

Parameters

- **pixel_spacing** (*Sequence[float]*) – Distance in physical space between neighboring pixels in millimeters along the row and column dimension of the image. First value represents the spacing between rows (vertical) and second value represents the spacing between columns (horizontal).
- **slice_thickness** (*Union[float, None]*) – Depth of physical space volume the image represents in millimeter.
- **spacing_between_slices** (*Union[float, None], optional*) – Distance in physical space between two consecutive images in millimeters. Only required for certain modalities, such as MR.

classmethod from_sequence(*sequence*)

Create a PixelMeasuresSequence from an existing Sequence.

Parameters

sequence (*pydicom.sequence.Sequence*) – Sequence to be converted.

Returns

Plane Measures Sequence.

Return type

highdicom.PixelMeasuresSequence

Raises

- **TypeError:** – If sequence is not of the correct type.
- **ValueError:** – If sequence does not contain exactly one item.
- **AttributeError:** – If sequence does not contain the attributes required for a pixel measures sequence.

class highdicom.**PixelRepresentationValues**(*value*)

Bases: Enum

Enumerated values for Planar Representation attribute.

COMPLEMENT = 1

UNSIGNED_INTEGER = 0

class highdicom.**PlanarConfigurationValues**(*value*)

Bases: Enum

Enumerated values for Planar Representation attribute.

COLOR_BY_PIXEL = 0

COLOR_BY_PLANE = 1

class highdicom.**PlaneOrientationSequence**(*coordinate_system*, *image_orientation*)

Bases: Sequence

Sequence of data elements describing the image position in the patient or slide coordinate system based on either the Plane Orientation (Patient) or the Plane Orientation (Slide) functional group macro, respectively.

Parameters

- **coordinate_system** (*Union[str, highdicom.CoordinateSystemNames]*) – Frame of reference coordinate system
- **image_orientation** (*Sequence[float]*) – Direction cosines for the first row (first triplet) and the first column (second triplet) of an image with respect to the X, Y, and Z axis of the three-dimensional coordinate system

classmethod from_sequence(*sequence*)

Create a PlaneOrientationSequence from an existing Sequence.

The coordinate system is inferred from the attributes in the sequence.

Parameters
sequence (*pydicom.sequence.Sequence*) – Sequence to be converted.

Returns
Plane Orientation Sequence.

Return type
highdicom.PlaneOrientationSequence

Raises

- **TypeError:** – If sequence is not of the correct type.
- **ValueError:** – If sequence does not contain exactly one item.
- **AttributeError:** – If sequence does not contain the attributes required for a plane orientation sequence.

class highdicom.**PlanePositionSequence**(*coordinate_system*, *image_position*, *pixel_matrix_position=None*)

Bases: Sequence

Sequence of data elements describing the position of an individual plane (frame) in the patient coordinate system based on the Plane Position (Patient) functional group macro or in the slide coordinate system based on the Plane Position (Slide) functional group macro.

Parameters

- **coordinate_system** (*Union[str, highdicom.CoordinateSystemNames]*) – Frame of reference coordinate system

- **image_position** (*Sequence[float]*) – Offset of the first row and first column of the plane (frame) in millimeter along the x, y, and z axis of the three-dimensional patient or slide coordinate system
- **pixel_matrix_position** (*Tuple[int, int], optional*) – Offset of the first column and first row of the plane (frame) in pixels along the row and column direction of the total pixel matrix (only required if *coordinate_system* is "SLIDE")

Note: The values of both *image_position* and *pixel_matrix_position* are one-based.

classmethod from_sequence(*sequence*)

Create a PlanePositionSequence from an existing Sequence.

The coordinate system is inferred from the attributes in the sequence.

Parameters
sequence (*pydicom.sequence.Sequence*) – Sequence to be converted.

Returns
Plane Position Sequence.

Return type
highdicom.PlanePositionSequence

Raises

- **TypeError:** – If sequence is not of the correct type.
- **ValueError:** – If sequence does not contain exactly one item.
- **AttributeError:** – If sequence does not contain the attributes required for a plane position sequence.

class highdicom.**PresentationLUT**(*first_mapped_value*, *lut_data*, *lut_explanation=None*)

Bases: *LUT*

Dataset describing an item of the Presentation LUT Sequence.

Parameters

- **first_mapped_value** (*int*) – Pixel value that will be mapped to the first value in the lookup-table.
- **lut_data** (*numpy.ndarray*) – Lookup table data. Must be of type uint16.
- **lut_explanation** (*Union[str, None], optional*) – Free-form text explanation of the meaning of the LUT.

class highdicom.**PresentationLUTShapeValues**(*value*)

Bases: Enum

Enumerated values for the Presentation LUT Shape attribute.

IDENTITY = 'IDENTITY'

No further translation of values is performed.

INVERSE = 'INVERSE'

A value of INVERSE shall mean the same as a value of IDENTITY, except that the minimum output value shall convey the meaning of the maximum available luminance, and the maximum value shall convey the minimum available luminance.

class highdicom.**PresentationLUTTransformation**(*presentation_lut_shape=None*, *presentation_lut=None*)

Bases: Dataset

Dataset describing the Presentation LUT Transformation as part of the Pixel Transformation Sequence to transform polarity pixel values into device-indendent presentation values (P-Values).

Parameters

- **presentation_lut_shape** (*Union[highdicom.pr.PresentationLUTShapeValues, str, None], optional*) – Shape of the presentation LUT
- **presentation_lut** (*Optional[highdicom.PresentationLUT], optional*) – Presentation LUT

Note: Only one of `presentation_lut_shape` or `presentation_lut` should be provided.

class highdicom.**ReferencedImageSequence**(*referenced_images=None*, *referenced_frame_number=None*, *referenced_segment_number=None*)

Bases: Sequence

Sequence of data elements describing a set of referenced images.

Parameters

- **referenced_images** (*Union[Sequence[pydicom.Dataset], None], optional*) – Images to which the VOI LUT described in this dataset applies. Note that if unspecified, the VOI LUT applies to every image referenced in the presentation state object that this dataset is included in.
- **referenced_frame_number** (*Union[int, Sequence[int], None], optional*) – Frame number(s) within a referenced multiframe image to which this VOI LUT applies.
- **referenced_segment_number**(*Union[int, Sequence[int], None], optional*) – Segment number(s) within a referenced segmentation image to which this VOI LUT applies.

class highdicom.**RescaleTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Rescale Type.

This specifies the units of the result of the rescale operation. Other values may be used, but they are not defined by the DICOM standard.

ED = 'ED'

Electron density in 1023 electrons/ml.

EDW = 'EDW'

Electron density normalized to water.

Units are N/Nw where N is number of electrons per unit volume, and Nw is number of electrons in the same unit of water at standard temperature and pressure.

HU = 'HU'

Hounsfield Units (CT).

HU_MOD = 'HU_MOD'

Modified Hounsfield Unit.

MGML = 'MGML'

Milligrams per milliliter.

OD = 'OD'

The number in the LUT represents thousands of optical density.

That is, a value of 2140 represents an optical density of 2.140.

PCT = 'PCT'

Percentage (%)

US = 'US'

Unspecified.

Z_EFF = 'Z_EFF'

Effective Atomic Number (i.e., Effective-Z).

class highdicom.**SOPClass**(*study_instance_uid*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *sop_class_uid*, *instance_number*, *modality*, *manufacturer=None*, *transfer_syntax_uid=None*, *patient_id=None*, *patient_name=None*, *patient_birth_date=None*, *patient_sex=None*, *accession_number=None*, *study_id=None*, *study_date=None*, *study_time=None*, *referring_physician_name=None*, *content_qualification=None*, *coding_schemes=None*, *series_description=None*, *manufacturer_model_name=None*, *software_versions=None*, *device_serial_number=None*, *institution_name=None*, *institutional_department_name=None*)

Bases: `Dataset`

Base class for DICOM SOP Instances.

Parameters

- **study_instance_uid** (*str*) – UID of the study
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **modality** (*str*) – Name of the modality
- **manufacturer** (*Union[str, None], optional*) – Name of the manufacturer (developer) of the device (software) that creates the instance
- **transfer_syntax_uid** (*Union[str, None], optional*) – UID of transfer syntax that should be used for encoding of data elements. Defaults to Implicit VR Little Endian (UID `"1.2.840.10008.1.2"`)
- **patient_id** (*Union[str, None], optional*) – ID of the patient (medical record number)
- **patient_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the patient
- **patient_birth_date** (*Union[str, None], optional*) – Patient's birth date
- **patient_sex** (*Union[str, highdicom.PatientSexValues, None], optional*) – Patient's sex
- **study_id** (*Union[str, None], optional*) – ID of the study
- **accession_number** (*Union[str, None], optional*) – Accession number of the study

- **study_date** (*Union[str, datetime.date, None], optional*) – Date of study creation
- **study_time** (*Union[str, datetime.time, None], optional*) – Time of study creation
- **referring_physician_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the referring physician
- **content_qualification** (*Union[str, highdicom.ContentQualificationValues, None], optional*) – Indicator of content qualification
- **coding_schemes** (*Union[Sequence[highdicom.sr.CodingSchemeIdentificationItem], None], optional*) – private or public coding schemes that are not part of the DICOM standard
- **series_description** (*Union[str, None], optional*) – Human readable description of the series
- **manufacturer_model_name** (*Union[str, None], optional*) – Name of the device model (name of the software library or application) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Version(s) of the software that creates the instance
- **device_serial_number** (*str*) – Manufacturer's serial number of the device
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance.
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance.

Note: The constructor only provides attributes that are required by the standard (type 1 and 2) as part of the Patient, General Study, Patient Study, General Series, General Equipment and SOP Common modules. Derived classes are responsible for providing additional attributes required by the corresponding Information Object Definition (IOD). Additional optional attributes can subsequently be added to the dataset.

---

copy_patient_and_study_information(*dataset*)

Copies patient- and study-related metadata from *dataset* that are defined in the following modules: Patient, General Study, Patient Study, Clinical Trial Subject and Clinical Trial Study.

Parameters
dataset (*pydicom.dataset.Dataset*) – DICOM Data Set from which attributes should be copied

Return type
None

copy_specimen_information(*dataset*)

Copies specimen-related metadata from *dataset* that are defined in the Specimen module.

Parameters
dataset (*pydicom.dataset.Dataset*) – DICOM Data Set from which attributes should be copied

Return type
None

class highdicom.**SegmentedPaletteColorLUT**(*first_mapped_value*, *segmented_lut_data*, *color*)

Bases: Dataset

Dataset describing a segmented palette color lookup table (LUT).

Parameters

- **first_mapped_value** (*int*) – Pixel value that will be mapped to the first value in the lookup table.
- **segmented_lut_data** (*numpy.ndarray*) – Segmented lookup table data. Must be of type uint16.
- **color** (*str*) – Free-form text explanation of the color (red, green, or blue).

Note: After the LUT is applied, a pixel in the image with value equal to first_mapped_value is mapped to an output value of lut_data[0], an input value of first_mapped_value + 1 is mapped to lut_data[1], and so on.

See here for details of how the segmented LUT data is encoded. Highdicom may provide utilities to assist in creating these arrays in a future release.

---

property **bits_per_entry**: int

Bits allocated for the lookup table data. 8 or 16.

Type
int

Return type
int

property **first_mapped_value**: int

Pixel value that will be mapped to the first value in the lookup table.

Type
int

Return type
int

property **lut_data**: ndarray

expanded lookup table data

Type
numpy.ndarray

Return type
numpy.ndarray

property **number_of_entries**: int

Number of entries in the lookup table.

Type
int

Return type
int

property **segmented_lut_data**: ndarray

segmented lookup table data

Type
numpy.ndarray

Return type
numpy.ndarray

class highdicom.**SpecimenCollection**(*procedure*)

Bases: *ContentSequence*

Sequence of SR content items describing a specimen collection procedure.

Parameters
procedure (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Surgical procedure used to collect the examined specimen

property **procedure**: *CodedConcept*

Surgical procedure

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

class highdicom.**SpecimenDescription**(*specimen_id*, *specimen_uid*, *specimen_location=None*, *specimen_preparation_steps=None*, *issuer_of_specimen_id=None*, *primary_anatomic_structures=None*)

Bases: Dataset

Dataset describing a specimen.

Parameters

- **specimen_id** (*str*) – Identifier of the examined specimen
- **specimen_uid** (*str*) – Unique identifier of the examined specimen
- **specimen_location** (*Union[str, Tuple[float, float, float]], optional*) – Location of the examined specimen relative to the container provided either in form of text or in form of spatial X, Y, Z coordinates specifying the position (offset) relative to the three-dimensional slide coordinate system in millimeter (X, Y) and micrometer (Z) unit.
- **specimen_preparation_steps** (*Sequence[highdicom.SpecimenPreparationStep], optional*) – Steps that were applied during the preparation of the examined specimen in the laboratory prior to image acquisition
- **issuer_of_specimen_id** (*highdicom.IssuerOfIdentifier, optional*) – Description of the issuer of the specimen identifier
- **primary_anatomic_structures** (*Sequence[Union[pydicom.sr.Code, highdicom.sr.CodedConcept]]*) – Body site at which specimen was collected

classmethod **from_dataset**(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an item of Specimen Description Sequence

Returns
Constructed object

Return type
highdicom.SpecimenDescription

property specimen_id: str

Specimen identifier

Type
str

Return type
str

property specimen_preparation_steps: List[*SpecimenPreparationStep*]

Specimen preparation steps

Type
highdicom.SpecimenPreparationStep

Return type
typing.List[*highdicom.content.SpecimenPreparationStep*]

property specimen_uid: *UID*

Unique specimen identifier

Type
highdicom.UID

Return type
highdicom.uid.UID

class highdicom.SpecimenPreparationStep(*specimen_id*, *processing_procedure*, *processing_description=None*, *processing_datetime=None*, *issuer_of_specimen_id=None*, *fixative=None*, *embedding_medium=None*)

Bases: Dataset

Dataset describing a specimen preparation step according to structured reporting template TID 8001 Specimen Preparation.

Parameters

- **specimen_id** (*str*) – Identifier of the processed specimen
- **processing_procedure** (*Union[*highdicom.SpecimenCollection, highdicom.SpecimenSampling, highdicom.SpecimenStaining, highdicom.SpecimenProcessing*]*) – Procedure used during processing
- **processing_datetime** (*datetime.datetime, optional*) – Datetime of processing
- **processing_description** (*Union[str, pydicom.sr.coding.Code,* highdicom.sr.CodedConcept*], optional*) – Description of processing
- **issuer_of_specimen_id** (highdicom.IssuerOfIdentifier, *optional*) –
- **fixative** (*Union[pydicom.sr.coding.Code,* highdicom.sr.CodedConcept*], optional*) – Fixative used during processing
- **embedding_medium** (*Union[pydicom.sr.coding.Code,* highdicom.sr.CodedConcept*], optional*) – Embedding medium used during processing

property embedding_medium: Optional[*CodedConcept*]

Tissue embedding medium

Type
highdicom.sr.CodedConcept

Return type
typing.Optional[*highdicom.sr.coding.CodedConcept*]

property fixative: Optional[*CodedConcept*]

Tissue fixative

Type
highdicom.sr.CodedConcept

Return type
typing.Optional[*highdicom.sr.coding.CodedConcept*]

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset

Returns
Specimen Preparation Step

Return type
highdicom.SpecimenPreparationStep

property processing_procedure: Union[*SpecimenCollection*, *SpecimenSampling*, *SpecimenStaining*, *SpecimenProcessing*]

Union[highdicom.SpecimenCollection, highdicom.SpecimenSampling, highdicom.SpecimenStaining, highdicom.SpecimenProcessing]:

Procedure used during processing

Return type
typing.Union[*highdicom.content.SpecimenCollection*, *highdicom.content.SpecimenSampling*, *highdicom.content.SpecimenStaining*, *highdicom.content.SpecimenProcessing*]

property processing_type: *CodedConcept*

Processing type

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property specimen_id: str

Specimen identifier

Type
str

Return type
str

class highdicom.**SpecimenProcessing**(*description*)

Bases: *ContentSequence*

Sequence of SR content items describing a specimen processing procedure.

Parameters
description (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, str]*) – Description of the processing

property description: *CodedConcept*

Processing step description

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

class highdicom.SpecimenSampling(*method*, *parent_specimen_id*, *parent_specimen_type*, *issuer_of_parent_specimen_id=None*)

Bases: *ContentSequence*

Sequence of SR content items describing a specimen sampling procedure.

See SR template TID 8002 Specimen Sampling.

Parameters

- **method** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Method used to sample the examined specimen from a parent specimen
- **parent_specimen_id** (*str*) – Identifier of the parent specimen
- **parent_specimen_type** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Type of the parent specimen
- **issuer_of_parent_specimen_id** (*highdicom.IssuerOfIdentifier, optional*) – Issuer who created the parent specimen

property method: *CodedConcept*

Sampling method

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property parent_specimen_id: str

Parent specimen identifier

Type
str

Return type
str

property parent_specimen_type: *CodedConcept*

Parent specimen type

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

class highdicom.**SpecimenStaining**(*substances*)

Bases: *ContentSequence*

Sequence of SR content items describing a specimen staining procedure

See SR template TID 8003 Specimen Staining.

Parameters

substances (*Sequence[Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, str]]*) – Substances used to stain examined specimen(s)

property **substances**: List[*CodedConcept*]

Substances used for staining

Type

List[*highdicom.sr.CodedConcept*]

Return type

typing.List[*highdicom.sr.coding.CodedConcept*]

class highdicom.**UID**(*value: Optional[str] = None*)

Bases: UID

Unique DICOM identifier.

If an object is constructed without a value being provided, a value will be automatically generated using the highdicom-specific root.

Setup new instance of the class.

Parameters

- **val** (*str or pydicom.uid.UID*) – The UID string to use to create the UID object.
- **validation_mode** (*int*) – Defines if values are validated and how validation errors are handled.

Returns

The UID object.

Return type

pydicom.uid.UID

classmethod **from_uuid**(*uuid*)

Create a DICOM UID from a UUID using the 2.25 root.

Parameters

uuid (*str*) – UUID

Returns

UID

Return type

highdicom.UID

Examples

```
>>> from uuid import uuid4
>>> import highdicom as hd
>>> uuid = str(uuid4())
>>> uid = hd.UID.from_uuid(uuid)
```

class highdicom.**UniversalEntityIDTypeValues**(*value*)

Bases: Enum

Enumerated values for Universal Entity ID Type attribute.

DNS = 'DNS'

An Internet dotted name. Either in ASCII or as integers.

EUI64 = 'EUI64'

An IEEE Extended Unique Identifier.

ISO = 'ISO'

An International Standards Organization Object Identifier.

URI = 'URI'

Uniform Resource Identifier.

UUID = 'UUID'

The DCE Universal Unique Identifier.

X400 = 'X400'

An X.400 MHS identifier.

X500 = 'X500'

An X.500 directory name.

class highdicom.**VOILUT**(*first_mapped_value*, *lut_data*, *lut_explanation=None*)

Bases: *LUT*

Dataset describing an item of the VOI LUT Sequence.

Parameters

- **first_mapped_value** (*int*) – Pixel value that will be mapped to the first value in the lookup-table.
- **lut_data** (*numpy.ndarray*) – Lookup table data. Must be of type uint16.
- **lut_explanation** (*Union[str, None], optional*) – Free-form text explanation of the meaning of the LUT.

class highdicom.**VOILUTFunctionValues**(*value*)

Bases: Enum

Enumerated values for attribute VOI LUT Function.

LINEAR = 'LINEAR'

LINEAR_EXACT = 'LINEAR_EXACT'

SIGMOID = 'SIGMOID'

class highdicom.**VOILUTTransformation**(*window_center=None*, *window_width=None*, *window_explanation=None*, *voi_lut_function=None*, *voi_luts=None*)

Bases: `Dataset`

Dataset describing the VOI LUT Transformation as part of the Pixel Transformation Sequence to transform modality pixel values into pixel values that are of interest to a user or an application.

Parameters

- **window_center** (*Union[float, Sequence[float], None], optional*) – Center value of the intensity window used for display.
- **window_width** (*Union[float, Sequence[float], None], optional*) – Width of the intensity window used for display.
- **window_explanation** (*Union[str, Sequence[str], None], optional*) – Free-form explanation of the window center and width.
- **voi_lut_function** (*Union[*highdicom.VOILUTFunctionValues*, str, None], optional*) – Description of the LUT function parametrized by `window_center`. and `window_width`.
- **voi_luts** (*Union[Sequence[*highdicom.VOILUT*], None], optional*) – Intensity lookup tables used for display.

Note: Either `window_center` and `window_width` should be provided or `voi_luts` should be provided, or both. `window_explanation` should only be provided if `window_center` is provided.

8.1.1 highdicom.color module

class highdicom.color.**CIELabColor**(*l_star*, *a_star*, *b_star*)

Bases: `object`

Class to represent a color value in CIELab color space.

Parameters

- **l_star** (*float*) – Lightness value in the range 0.0 (black) to 100.0 (white).
- **a_star** (*float*) – Red-green value from -128.0 (red) to 127.0 (green).
- **b_star** (*float*) – Blue-yellow value from -128.0 (blue) to 127.0 (yellow).

property value: Tuple[int, int, int]

Tuple[int]: Value formatted as a triplet of 16 bit unsigned integers.

Return type
`typing.Tuple[int, int, int]`

class highdicom.color.**ColorManager**(*icc_profile*)

Bases: `object`

Class for color management using ICC profiles.

Parameters
icc_profile (*bytes*) – ICC profile

Raises
ValueError – When ICC Profile cannot be read.

transform_frame(*array*)

Transforms a frame by applying the ICC profile.

Parameters
array (*numpy.ndarray*) – Pixel data of a color image frame in form of an array with dimensions (Rows x Columns x SamplesPerPixel)

Returns
Color corrected pixel data of a image frame in form of an array with dimensions (Rows x Columns x SamplesPerPixel)

Return type
numpy.ndarray

Raises
ValueError – When *array* does not have 3 dimensions and thus does not represent a color image frame.

8.1.2 highdicom.frame module

highdicom.frame.**decode_frame**(*value*, *transfer_syntax_uid*, *rows*, *columns*, *samples_per_pixel*, *bits_allocated*, *bits_stored*, *photometric_interpretation*, *pixel_representation=0*, *planar_configuration=None*)

Decode pixel data of an individual frame.

Parameters

- **value** (*bytes*) – Pixel data of a frame (potentially compressed in case of encapsulated format encoding, depending on the transfer syntax)
- **transfer_syntax_uid** (*str*) – Transfer Syntax UID
- **rows** (*int*) – Number of pixel rows in the frame
- **columns** (*int*) – Number of pixel columns in the frame
- **samples_per_pixel** (*int*) – Number of (color) samples per pixel
- **bits_allocated** (*int*) – Number of bits that need to be allocated per pixel sample
- **bits_stored** (*int*) – Number of bits that are required to store a pixel sample
- **photometric_interpretation** (*Union[str, highdicom.PhotometricInterpretationValues]*) – Photometric interpretation
- **pixel_representation** (*Union[highdicom.PixelRepresentationValues, int, None], optional*) – Whether pixel samples are represented as unsigned integers or 2's complements
- **planar_configuration** (*Union[highdicom.PlanarConfigurationValues, int, None], optional*) – Whether color samples are encoded by pixel (`R1G1B1R2G2B2...`) or by plane (`R1R2...G1G2...B1B2...`).

Returns
Decoded pixel data

Return type
numpy.ndarray

Raises
ValueError – When transfer syntax is not supported.

Note: In case of color image frames, the *photometric_interpretation* parameter describes the color space of the **encoded** pixel data and data may be converted from the specified color space into RGB color space upon decoding. For example, the JPEG codec generally converts pixels from RGB into YBR color space prior to compression to take advantage of the correlation between RGB color bands and improve compression efficiency. In case of an image data set with an encapsulated Pixel Data element containing JPEG compressed image frames, the value of the Photometric Interpretation element specifies the color space in which image frames were compressed. If *photometric_interpretation* specifies a YBR color space, then this function assumes that pixels were converted from RGB to YBR color space during encoding prior to JPEG compression and need to be converted back into RGB color space after JPEG decompression during decoding. If *photometric_interpretation* specifies an RGB color space, then the function assumes that no color space conversion was performed during encoding and therefore no conversion needs to be performed during decoding either. In both case, the function is supposed to return decoded pixel data of color image frames in RGB color space.

highdicom.frame.**encode_frame**(*array*, *transfer_syntax_uid*, *bits_allocated*, *bits_stored*, *photometric_interpretation*, *pixel_representation=0*, *planar_configuration=None*)

Encode pixel data of an individual frame.

Parameters

- **array** (*numpy.ndarray*) – Pixel data in form of an array with dimensions (Rows x Columns x SamplesPerPixel) in case of a color image and (Rows x Columns) in case of a monochrome image
- **transfer_syntax_uid** (*int*) – Transfer Syntax UID
- **bits_allocated** (*int*) – Number of bits that need to be allocated per pixel sample
- **bits_stored** (*int*) – Number of bits that are required to store a pixel sample
- **photometric_interpretation** (*int*) – Photometric interpretation
- **pixel_representation** (*Union[highdicom.PixelRepresentationValues, int, None], optional*) – Whether pixel samples are represented as unsigned integers or 2's complements
- **planar_configuration** (*Union[highdicom.PlanarConfigurationValues, int, None], optional*) – Whether color samples are encoded by pixel (`R1G1B1R2G2B2...`) or by plane (`R1R2...G1G2...B1B2...`).

Returns

Encoded pixel data (potentially compressed in case of encapsulated format encoding, depending on the transfer snytax)

Return type

bytes

Raises

ValueError – When *transfer_syntax_uid* is not supported or when *planar_configuration* is missing in case of a color image frame.

Note: In case of color image frames, the *photometric_interpretation* parameter describes the color space of the **encoded** pixel data and data may be converted from RGB color space into the specified color space upon encoding. For example, the JPEG codec converts pixels from RGB into YBR color space prior to compression to take advantage of the correlation between RGB color bands and improve compression efficiency. Therefore,

pixels are supposed to be provided via *array* in RGB color space, but *photometric_interpretation* needs to specify a YBR color space.

8.1.3 highdicom.io module

Input/Output of datasets based on DICOM Part10 files.

class highdicom.io.**ImageFileReader**(*filename*)

Bases: object

Reader for DICOM datasets representing Image Information Entities.

It provides efficient access to individual Frame items contained in the Pixel Data element without loading the entire element into memory.

Examples

```
>>> from pydicom.data import get_testdata_file
>>> from highdicom.io import ImageFileReader
>>> test_filepath = get_testdata_file('eCT_Supplemental.dcm')
>>>
>>> with ImageFileReader(test_filepath) as image:
...     print(image.metadata.SOPInstanceUID)
...     for i in range(image.number_of_frames):
...         frame = image.read_frame(i)
...         print(frame.shape)
1.3.6.1.4.1.5962.1.1.10.3.1.1166562673.14401
(512, 512)
(512, 512)
```

Parameters
filename (*Union[str, pathlib.Path, pydicom.filebase.DicomfileLike]*) – DICOM Part10 file containing a dataset of an image SOP Instance

close()

Closes file.

Return type
None

property filename: str

Path to the image file

Type
str

Return type
str

property metadata: Dataset

Metadata

Type
pydicom.dataset.Dataset

Return type
pydicom.dataset.Dataset

property number_of_frames: int

Number of frames

Type
int

Return type
int

open()

Open file for reading.

Raises

- **FileNotFoundError** – When file cannot be found
- **OSError** – When file cannot be opened
- **IOError** – When DICOM metadata cannot be read from file
- **ValueError** – When DICOM dataset contained in file does not represent an image

---

Note: Builds a Basic Offset Table to speed up subsequent frame-level access.

Return type
None

read_frame(*index*, *correct_color=True*)

Reads and decodes the pixel data of an individual frame item.

Parameters

- **index** (*int*) – Zero-based frame index
- **correct_color** (*bool, optional*) – Whether colors should be corrected by applying an ICC transformation. Will only be performed if metadata contain an ICC Profile. Default = True.

Returns
Array of decoded pixels of the frame with shape (Rows x Columns) in case of a monochrome image or (Rows x Columns x SamplesPerPixel) in case of a color image.

Return type
numpy.ndarray

Raises
IOError – When frame could not be read

read_frame_raw(*index*)

Reads the raw pixel data of an individual frame item.

Parameters
index (*int*) – Zero-based frame index

Returns
Pixel data of a given frame item encoded in the transfer syntax.

Return type
bytes

Raises
IOError – When frame could not be read

8.1.4 highdicom.spatial module

class highdicom.spatial.**ImageToReferenceTransformer**(*image_position*, *image_orientation*, *pixel_spacing*)

Bases: `object`

Class for transforming coordinates from image to reference space.

This class facilitates the mapping of image coordinates in the pixel matrix of an image or an image frame (tile or plane) into the patient or slide coordinate system defined by the frame of reference. For example, this class may be used to map spatial coordinates (SCOORD) to 3D spatial coordinates (SCOORD3D).

Image coordinates are (column, row) pairs of floating-point values, where the (0.0, 0.0) point is located at the top left corner of the top left hand corner pixel of the pixel matrix. Image coordinates have pixel units at sub-pixel resolution.

Reference coordinates are (x, y, z) triplets of floating-point values, where the (0.0, 0.0) point is located at the origin of the frame of reference. Reference coordinates have millimeter units.

Examples

```
>>> transformer = ImageToReferenceTransformer(
...     image_position=[56.0, 34.2, 1.0],
...     image_orientation=[1.0, 0.0, 0.0, 0.0, 1.0, 0.0],
...     pixel_spacing=[0.5, 0.5]
... )
>>>
>>> image_coords = np.array([[0.0, 10.0], [5.0, 5.0]])
>>> ref_coords = transformer(image_coords)
>>> print(ref_coords)
[[55.75 38.95  1.  ]
 [58.25 36.45  1.  ]]
```

Warning: This class shall not be used for pixel indices. Use the class:*highdicom.spatial.PixelToReferenceTransformer* class instead.

Construct transformation object.

Parameters

- **image_position** (*Sequence[float]*) – Position of the slice (image or frame) in the frame of reference, i.e., the offset of the top left hand corner pixel in the pixel matrix from the origin of the reference coordinate system along the X, Y, and Z axis
- **image_orientation** (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing column index) and the column direction (second triplet: vertical, top to bottom, increasing row index) direction expressed in the three-dimensional patient or slide coordinate system defined by the frame of reference

- **pixel_spacing** (*Sequence[float]*) – Spacing between pixels in millimeter unit along the column direction (first value: spacing between rows, vertical, top to bottom, increasing row index) and the rows direction (second value: spacing between columns: horizontal, left to right, increasing column index)

Raises

- **TypeError** – When any of the arguments is not a sequence.
- **ValueError** – When any of the arguments has an incorrect length.

__call__(*coordinates*)

Transform image coordinates to frame of reference coordinates.

Parameters

coordinates (*numpy.ndarray*) – Array of (column, row) coordinates at sub-pixel resolution in the range [0, Columns] and [0, Rows], respectively. Array of floating-point values with shape `(n, 2)`, where *n* is the number of coordinates, the first column represents the *column* values and the second column represents the *row* values. The `(0.0, 0.0)` coordinate is located at the top left corner of the top left hand corner pixel in the total pixel matrix.

Returns

Array of (x, y, z) coordinates in the coordinate system defined by the frame of reference. Array has shape `(n, 3)`, where *n* is the number of coordinates, the first column represents the *X* offsets, the second column represents the *Y* offsets and the third column represents the *Z* offsets

Return type

numpy.ndarray

Raises

ValueError – When *coordinates* has incorrect shape.

property affine: ndarray

4x4 affine transformation matrix

Type

numpy.ndarray

Return type

numpy.ndarray

class highdicom.spatial.**PixelToReferenceTransformer**(*image_position*, *image_orientation*, *pixel_spacing*)

Bases: `object`

Class for transforming pixel indices to reference coordinates.

This class facilitates the mapping of pixel indices to the pixel matrix of an image or an image frame (tile or plane) into the patient or slide coordinate system defined by the frame of reference.

Pixel indices are (column, row) pairs of zero-based integer values, where the (0, 0) index is located at the **center** of the top left hand corner pixel of the pixel matrix.

Reference coordinates are (x, y, z) triplets of floating-point values, where the (0.0, 0.0) point is located at the origin of the frame of reference.

Examples

```
>>> import numpy as np
>>>
>>> # Create a transformer by specifying the reference space of
>>> # an image
>>> transformer = PixelToReferenceTransformer(
...     image_position=[56.0, 34.2, 1.0],
...     image_orientation=[1.0, 0.0, 0.0, 0.0, 1.0, 0.0],
...     pixel_spacing=[0.5, 0.5])
>>>
>>> # Use the transformer to convert coordinates
>>> pixel_indices = np.array([[0, 10], [5, 5]])
>>> ref_coords = transformer(pixel_indices)
>>> print(ref_coords)
[[56.  39.2  1. ]
 [58.5 36.7  1. ]]
```

Warning: This class shall not be used to map spatial coordinates (SCOORD) to 3D spatial coordinates (SCOORD3D). Use the *highdicom.spatial.ImageToReferenceTransformer* class instead.

Construct transformation object.

Parameters

- **image_position** (*Sequence[float]*) – Position of the slice (image or frame) in the frame of reference, i.e., the offset of the top left hand corner pixel in the pixel matrix from the origin of the reference coordinate system along the X, Y, and Z axis
- **image_orientation** (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing column index) and the column direction (second triplet: vertical, top to bottom, increasing row index) direction expressed in the three-dimensional patient or slide coordinate system defined by the frame of reference
- **pixel_spacing** (*Sequence[float]*) – Spacing between pixels in millimeter unit along the column direction (first value: spacing between rows, vertical, top to bottom, increasing row index) and the rows direction (second value: spacing between columns: horizontal, left to right, increasing column index)

Raises

- **TypeError** – When any of the arguments is not a sequence.
- **ValueError** – When any of the arguments has an incorrect length.

__call__(*indices*)

Transform image pixel indices to frame of reference coordinates.

Parameters

indices (*numpy.ndarray*) – Array of (column, row) zero-based pixel indices in the range [0, Columns - 1] and [0, Rows - 1], respectively. Array of integer values with shape `(n, 2)`, where *n* is the number of indices, the first column represents the *column* index and the second column represents the *row* index. The `(0, 0)` coordinate is located at the **center** of the top left pixel in the total pixel matrix.

Returns

Array of (x, y, z) coordinates in the coordinate system defined by the frame of reference.

Array has shape `(n, 3)`, where *n* is the number of coordinates, the first column represents the *x* offsets, the second column represents the *y* offsets and the third column represents the *z* offsets

Return type
numpy.ndarray

Raises

- **ValueError** – When *indices* has incorrect shape.
- **TypeError** – When *indices* don't have integer data type.

property affine: ndarray

4x4 affine transformation matrix

Type
numpy.ndarray

Return type
numpy.ndarray

class highdicom.spatial.**ReferenceToImageTransformer**(*image_position*, *image_orientation*, *pixel_spacing*, *spacing_between_slices=1.0*)

Bases: `object`

Class for transforming coordinates from reference to image space.

This class facilitates the mapping of coordinates in the patient or slide coordinate system defined by the frame of reference into the total pixel matrix. For example, this class may be used to map 3D spatial coordinates (SCOORD3D) to spatial coordinates (SCOORD).

Reference coordinates are (x, y, z) triplets of floating-point values, where the (0.0, 0.0) point is located at the origin of the frame of reference. Reference coordinates have millimeter units.

Image coordinates are (column, row) pairs of floating-point values, where the (0.0, 0.0) point is located at the top left corner of the top left hand corner pixel of the pixel matrix. Image coordinates have pixel units at sub-pixel resolution.

Examples

```
>>> # Create a transformer by specifying the reference space of
>>> # an image
>>> transformer = ReferenceToImageTransformer(
...     image_position=[56.0, 34.2, 1.0],
...     image_orientation=[1.0, 0.0, 0.0, 0.0, 1.0, 0.0],
...     pixel_spacing=[0.5, 0.5]
... )
>>>
>>> # Use the transformer to convert coordinates
>>> ref_coords = np.array([[56., 39.2,  1. ], [58.5, 36.7, 1.]])
>>> image_coords = transformer(ref_coords)
>>> print(image_coords)
[[ 0.5 10.5  0. ]
 [ 5.5  5.5  0. ]]
```

Warning: This class shall not be used for pixel indices. Use the *highdicom.spatial.ReferenceToPixelTransformer* class instead.

Construct transformation object.

Builds an inverse of an affine transformation matrix for mapping coordinates from the frame of reference into the two dimensional pixel matrix.

Parameters

- **image_position** (*Sequence[float]*) – Position of the slice (image or frame) in the frame of reference, i.e., the offset of the top left hand corner pixel in the pixel matrix from the origin of the reference coordinate system along the X, Y, and Z axis
- **image_orientation** (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing column index) and the column direction (second triplet: vertical, top to bottom, increasing row index) direction expressed in the three-dimensional patient or slide coordinate system defined by the frame of reference
- **pixel_spacing** (*Sequence[float]*) – Spacing between pixels in millimeter unit along the column direction (first value: spacing between rows, vertical, top to bottom, increasing row index) and the rows direction (second value: spacing between columns: horizontal, left to right, increasing column index)
- **spacing_between_slices** (*float, optional*) – Distance (in the coordinate defined by the frame of reference) between neighboring slices. Default: 1

Raises

- **TypeError** – When *image_position*, *image_orientation* or *pixel_spacing* is not a sequence.
- **ValueError** – When *image_position*, *image_orientation* or *pixel_spacing* has an incorrect length.

__call__(*coordinates*)

Apply the inverse of an affine transformation matrix to a batch of coordinates in the frame of reference to obtain the corresponding pixel matrix indices.

Parameters

coordinates (*numpy.ndarray*) – Array of (x, y, z) coordinates in the coordinate system defined by the frame of reference. Array should have shape `(n, 3)`, where *n* is the number of coordinates, the first column represents the *X* offsets, the second column represents the *Y* offsets and the third column represents the *Z* offsets

Returns

Array of (column, row, slice) indices, where *column* and *row* are zero-based indices to the total pixel matrix and the *slice* index represents the signed distance of the input coordinate in the direction normal to the plane of the total pixel matrix. The *row* and *column* indices are constrained by the dimension of the total pixel matrix. Note, however, that in general, the resulting coordinate may not lie within the imaging plane, and consequently the *slice* offset may be non-zero.

Return type

numpy.ndarray

Raises

ValueError – When *coordinates* has incorrect shape.

property affine: ndarray

4 x 4 affine transformation matrix

Type
numpy.ndarray

Return type
numpy.ndarray

class highdicom.spatial.**ReferenceToPixelTransformer**(*image_position*, *image_orientation*, *pixel_spacing*, *spacing_between_slices=1.0*)

Bases: object

Class for transforming reference coordinates to pixel indices.

This class facilitates the mapping of coordinates in the patient or slide coordinate system defined by the frame of reference into the total pixel matrix.

Reference coordinates are (x, y, z) triplets of floating-point values, where the (0.0, 0.0) point is located at the origin of the frame of reference.

Pixel indices are (column, row) pairs of zero-based integer values, where the (0, 0) index is located at the **center** of the top left hand corner pixel of the pixel matrix.

Examples

```
>>> transformer = ReferenceToPixelTransformer(
...     image_position=[56.0, 34.2, 1.0],
...     image_orientation=[1.0, 0.0, 0.0, 0.0, 1.0, 0.0],
...     pixel_spacing=[0.5, 0.5]
... )
>>>
>>> ref_coords = np.array([[56., 39.2,  1. ], [58.5, 36.7, 1.]])
>>> pixel_indices = transformer(ref_coords)
>>> print(pixel_indices)
[[ 0 10  0]
 [ 5  5  0]]
```

Warning: This class shall not be used to map 3D spatial coordinates (SCOORD3D) to spatial coordinates (SCOORD). Use the *highdicom.spatial.ReferenceToImageTransformer* class instead.

Construct transformation object.

Builds an inverse of an affine transformation matrix for mapping coordinates from the frame of reference into the two dimensional pixel matrix.

Parameters

- **image_position** (*Sequence[float]*) – Position of the slice (image or frame) in the frame of reference, i.e., the offset of the top left hand corner pixel in the pixel matrix from the origin of the reference coordinate system along the X, Y, and Z axis
- **image_orientation** (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing column index) and the column direction (second triplet: vertical, top to bottom, increasing row index) direction expressed in the three-dimensional patient or slide coordinate system defined by the frame of reference
- **pixel_spacing** (*Sequence[float]*) – Spacing between pixels in millimeter unit along the column direction (first value: spacing between rows, vertical, top to bottom, increasing

row index) and the rows direction (second value: spacing between columns: horizontal, left to right, increasing column index)

- **spacing_between_slices** (*float, optional*) – Distance (in the coordinate defined by the frame of reference) between neighboring slices. Default: 1

Raises

- **TypeError** – When *image_position*, *image_orientation* or *pixel_spacing* is not a sequence.
- **ValueError** – When *image_position*, *image_orientation* or *pixel_spacing* has an incorrect length.

__call__(*coordinates*)

Transform frame of reference coordinates into image pixel indices.

Parameters

coordinates (*numpy.ndarray*) – Array of (x, y, z) coordinates in the coordinate system defined by the frame of reference. Array has shape `(n, 3)`, where *n* is the number of coordinates, the first column represents the *X* offsets, the second column represents the *Y* offsets and the third column represents the *Z* offsets

Returns

Array of (column, row) zero-based indices at pixel resolution. Array of integer values with shape `(n, 2)`, where *n* is the number of indices, the first column represents the *column* index and the second column represents the *row* index. The `(0, 0)` coordinate is located at the **center** of the top left pixel in the total pixel matrix.

Return type

numpy.ndarray

Note: The returned pixel indices may be negative if *coordinates* fall outside of the total pixel matrix.

Raises

ValueError – When *indices* has incorrect shape.

property affine: ndarray

4 x 4 affine transformation matrix

Type

numpy.ndarray

Return type

`numpy.ndarray`

highdicom.spatial.**create_rotation_matrix**(*image_orientation*)

Builds a rotation matrix.

Parameters

image_orientation (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing column index) and the column direction (second triplet: vertical, top to bottom, increasing row index) direction expressed in the three-dimensional patient or slide coordinate system defined by the frame of reference

Returns

3 x 3 rotation matrix

Return type

numpy.ndarray

highdicom.spatial.**map_coordinate_into_pixel_matrix**(*coordinate*, *image_position*, *image_orientation*, *pixel_spacing*, *spacing_between_slices=1.0*)

Map a reference coordinate into an index to the total pixel matrix.

Parameters

- **coordinate** (*Sequence[float]*) – (x, y, z) coordinate in the coordinate system in millimeter unit.
- **image_position** (*Sequence[float]*) – Position of the slice (image or frame) in the frame of reference, i.e., the offset of the center of top left hand corner pixel in the total pixel matrix from the origin of the reference coordinate system along the X, Y, and Z axis
- **image_orientation** (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing column index) and the column direction (second triplet: vertical, top to bottom, increasing row index) direction expressed in the three-dimensional patient or slide coordinate system defined by the frame of reference
- **pixel_spacing** (*Sequence[float]*) – Spacing between pixels in millimeter unit along the column direction (first value: spacing between rows, vertical, top to bottom, increasing row index) and the rows direction (second value: spacing between columns: horizontal, left to right, increasing column index)
- **spacing_between_slices** (*float, optional*) – Distance (in the coordinate defined by the frame of reference) between neighboring slices. Default: `1.0`

Returns

(column, row, slice) index, where *column* and *row* are pixel indices in the total pixel matrix, *slice* represents the signed distance of the input coordinate in the direction normal to the plane of the total pixel matrix. If the *slice* offset is `0`, then the input coordinate lies in the imaging plane, otherwise it lies off the plane of the total pixel matrix and *column* and *row* indices may be interpreted as the projections of the input coordinate onto the imaging plane.

Return type

Tuple[int, int, int]

Note: This function is a convenient wrapper around *highdicom.spatial.ReferenceToPixelTransformer*. When mapping a large number of coordinates, consider using these underlying functions directly for speedup.

Raises

- **TypeError** – When *image_position*, *image_orientation*, or *pixel_spacing* is not a sequence.
- **ValueError** – When *image_position*, *image_orientation*, or *pixel_spacing* has an incorrect length.

highdicom.spatial.**map_pixel_into_coordinate_system**(*index*, *image_position*, *image_orientation*, *pixel_spacing*)

Map an index to the pixel matrix into the reference coordinate system.

Parameters

- **index** (*Sequence[float]*) – (column, row) zero-based index at pixel resolution in the range [0, Columns - 1] and [0, Rows - 1], respectively.

- **image_position** (*Sequence[float]*) – Position of the slice (image or frame) in the frame of reference, i.e., the offset of the center of top left hand corner pixel in the total pixel matrix from the origin of the reference coordinate system along the X, Y, and Z axis
- **image_orientation** (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing column index) and the column direction (second triplet: vertical, top to bottom, increasing row index) direction expressed in the three-dimensional patient or slide coordinate system defined by the frame of reference
- **pixel_spacing** (*Sequence[float]*) – Spacing between pixels in millimeter unit along the column direction (first value: spacing between rows, vertical, top to bottom, increasing row index) and the row direction (second value: spacing between columns: horizontal, left to right, increasing column index)

Returns
(x, y, z) coordinate in the coordinate system defined by the frame of reference

Return type
Tuple[float, float, float]

Note: This function is a convenient wrapper around *highdicom.spatial.PixelToReferenceTransformer* for mapping an individual coordinate. When mapping a large number of coordinates, consider using this class directly for speedup.

Raises

- **TypeError** – When *image_position*, *image_orientation*, or *pixel_spacing* is not a sequence.
- **ValueError** – When *image_position*, *image_orientation*, or *pixel_spacing* has an incorrect length.

8.1.5 highdicom.valuerep module

Functions for working with DICOM value representations.

highdicom.valuerep.**check_person_name**(*person_name*)

Check value is valid for the value representation "person name".

The DICOM Person Name (PN) value representation has a specific format with multiple components (family name, given name, middle name, prefix, suffix) separated by caret characters ('^'), where any number of components may be missing and trailing caret separators may be omitted. Unfortunately it is both easy to make a mistake when constructing names with this format, and impossible to check for certain whether it has been done correctly.

This function checks for strings representing person names that have a high likelihood of having been encoded incorrectly and raises an exception if such a case is found.

A string is considered to be an invalid person name if it contains no caret characters.

Note: A name consisting of only a family name component (e.g. `'Bono'`) is valid according to the standard but will be disallowed by this function. However if necessary, such a name can be still be encoded by adding a trailing caret character to disambiguate the meaning (e.g. `'Bono^'`).

Parameters
person_name (*Union[str, pydicom.valuerep.PersonName]*) – Name to check.

Raises

- **ValueError** – If the provided value is highly likely to be an invalid person name.
- **TypeError** – If the provided person name has an invalid type.

Return type
None

8.1.6 highdicom.utils module

highdicom.utils.**compute_plane_position_slide_per_frame**(*dataset*)

Computes the plane position for each frame in given dataset with respect to the slide coordinate system.

Parameters
dataset (*pydicom.dataset.Dataset*) – VL Whole Slide Microscopy Image

Returns
Plane Position Sequence per frame

Return type
List[*highdicom.PlanePositionSequence*]

Raises
ValueError – When *dataset* does not represent a VL Whole Slide Microscopy Image

highdicom.utils.**compute_plane_position_tiled_full**(*row_index*, *column_index*, *x_offset*, *y_offset*, *rows*, *columns*, *image_orientation*, *pixel_spacing*, *slice_thickness=None*, *spacing_between_slices=None*, *slice_index=None*)

Compute the position of a frame (image plane) in the frame of reference defined by the three-dimensional slide coordinate system.

This information is not provided in image instances with Dimension Orientation Type TILED_FULL and therefore needs to be computed.

Parameters

- **row_index** (*int*) – One-based Row index value for a given frame (tile) along the column direction of the tiled Total Pixel Matrix, which is defined by the second triplet in *image_orientation* (values should be in the range [1, *n*], where *n* is the number of tiles per column)
- **column_index** (*int*) – One-based Column index value for a given frame (tile) along the row direction of the tiled Total Pixel Matrix, which is defined by the first triplet in *image_orientation* (values should be in the range [1, *n*], where *n* is the number of tiles per row)
- **x_offset** (*float*) – X offset of the Total Pixel Matrix in the slide coordinate system in millimeters
- **y_offset** (*float*) – Y offset of the Total Pixel Matrix in the slide coordinate system in millimeters
- **rows** (*int*) – Number of rows per Frame (tile)
- **columns** (*int*) – Number of columns per Frame (tile)
- **image_orientation** (*Sequence[float]*) – Cosines of the row direction (first triplet: horizontal, left to right, increasing Column index) and the column direction (second triplet:

vertical, top to bottom, increasing Row index) direction for X, Y, and Z axis of the slide coordinate system defined by the Frame of Reference

- **pixel_spacing** (*Sequence[float]*) – Spacing between pixels in millimeter unit along the column direction (first value: spacing between rows, vertical, top to bottom, increasing Row index) and the row direction (second value: spacing between columns, horizontal, left to right, increasing Column index)
- **slice_thickness** (*Union[float, None], optional*) – Thickness of a focal plane in micrometers
- **spacing_between_slices** (*Union[float, None], optional*) – Distance between neighboring focal planes in micrometers
- **slice_index** (*Union[int, None], optional*) – Relative one-based index of the focal plane in the array of focal planes within the imaged volume from the slide to the coverslip

Returns
Positon of the plane in the slide coordinate system

Return type
highdicom.PlanePositionSequence

Raises
TypeError – When only one of *slice_index* and *spacing_between_slices* is provided

highdicom.utils.**is_tiled_image**(*dataset*)

Determine whether a dataset represents a tiled image.

Returns
True if the dataset is a tiled image. False otherwise.

Return type
bool

highdicom.utils.**tile_pixel_matrix**(*total_pixel_matrix_rows*, *total_pixel_matrix_columns*, *rows*, *columns*)

Tiles an image into smaller frames (rectangular regions).

Parameters

- **total_pixel_matrix_rows** (*int*) – Number of rows in the Total Pixel Matrix
- **total_pixel_matrix_columns** (*int*) – Number of columns in the Total Pixel Matrix
- **rows** (*int*) – Number of rows per Frame (tile)
- **columns** (*int*) – Number of columns per Frame (tile)

Returns
One-based (Column, Row) index of each Frame (tile)

Return type
Iterator

8.2 highdicom.legacy package

Package for creation of Legacy Converted Enhanced CT, MR or PET Image instances.

class highdicom.legacy.**LegacyConvertedEnhancedCTImage**(*legacy_datasets*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for Legacy Converted Enhanced CT Image instances.

Parameters

- **legacy_datasets** (*Sequence[pydicom.dataset.Dataset]*) – DICOM data sets of legacy single-frame image instances that should be converted
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements. The following compressed transfer syntaxes are supported: JPEG 2000 Lossless ("1.2.840.10008.1.2.4.90") and JPEG-LS Lossless ("1.2.840.10008.1.2.4.80").
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

class highdicom.legacy.**LegacyConvertedEnhancedMRImage**(*legacy_datasets*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for Legacy Converted Enhanced MR Image instances.

Parameters

- **legacy_datasets** (*Sequence[pydicom.dataset.Dataset]*) – DICOM data sets of legacy single-frame image instances that should be converted
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements. The following compressed transfer syntaxes are supported: JPEG 2000 Lossless ("1.2.840.10008.1.2.4.90") and JPEG-LS Lossless ("1.2.840.10008.1.2.4.80").
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

class highdicom.legacy.**LegacyConvertedEnhancedPETImage**(*legacy_datasets*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for Legacy Converted Enhanced PET Image instances.

Parameters

- **legacy_datasets** (*Sequence[pydicom.dataset.Dataset]*) – DICOM data sets of legacy single-frame image instances that should be converted
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements. The following compressed transfer syntaxes are supported: JPEG 2000 Lossless ("1.2.840.10008.1.2.4.90") and JPEG-LS Lossless ("1.2.840.10008.1.2.4.80").
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

8.3 highdicom.ann package

Package for creation of Annotation (ANN) instances.

class highdicom.ann.**AnnotationCoordinateTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Annotation Coordinate Type.

SCOORD = '2D'

Two-dimensional spatial coordinates denoted by (Column,Row) pairs.

The coordinate system is the pixel matrix of an image and individual coordinates are defined relative to center of the (1,1) pixel of either the total pixel matrix of the entire image or of the pixel matrix of an individual frame, depending on the value of Pixel Origin Interpretation.

Coordinates have pixel unit.

SCOORD3D = '3D'

Three-dimensional spatial coordinates denoted by (X,Y,Z) triplets.

The coordinate system is the Frame of Reference (slide or patient) and the coordinates are defined relative to origin of the Frame of Reference.

Coordinates have millimeter unit.

class highdicom.ann.**AnnotationGroup**(*number*, *uid*, *label*, *annotated_property_category*, *annotated_property_type*, *graphic_type*, *graphic_data*, *algorithm_type*, *algorithm_identification=None*, *measurements=None*, *description=None*, *anatomic_regions=None*, *primary_anatomic_structures=None*)

Bases: Dataset

Dataset describing a group of annotations.

Parameters

- **number** (*int*) – One-based number for identification of the annotation group
- **uid** (*str*) – Unique identifier of the annotation group
- **label** (*str*) – User-defined label for identification of the annotation group
- **annotated_property_category** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Category of the property the annotated regions of interest represents, e.g., `Code("49755003", "SCT", "Morphologically Abnormal Structure")` (see CID 7150 "Segmentation Property Categories")
- **annotated_property_type** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Property the annotated regions of interest represents, e.g., `Code("108369006", "SCT", "Neoplasm")` (see CID 8135 "Microscopy Annotation Property Types")
- **graphic_type** (*Union[str, highdicom.ann.GraphicTypeValues]*) – Graphic type of annotated regions of interest
- **graphic_data** (*Sequence[numpy.ndarray]*) – Array of ordered spatial coordinates, where each row of an array represents a (Column,Row) coordinate pair or (X,Y,Z) coordinate triplet.
- **algorithm_type** (*Union[str, highdicom.ann.AnnotationGroupGenerationTypeValues]*) – Type of algorithm that was used to generate the annotation
- **algorithm_identification** (*Union[highdicom.AlgorithmIdentificationSequence, None], optional*) – Information useful for identification of the algorithm, such as its name or version. Required unless the *algorithm_type* is `"MANUAL"`
- **measurements** (*Union[Sequence[highdicom.ann.Measurements], None], optional*) – One or more sets of measurements for annotated regions of interest
- **description** (*Union[str, None], optional*) – Description of the annotation group
- **anatomic_regions** (*Union[Sequence[Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]], None], optional*) – Anatomic region(s) into which annotations fall
- **primary_anatomic_structures** (*Union[Sequence[Union[highdicom.sr.Code, highdicom.sr.CodedConcept]], None], optional*) – Anatomic structure(s) the annotations represent (see CIDs for domain-specific primary anatomic structures)

property algorithm_identification: Optional[*AlgorithmIdentificationSequence*]

Union[highdicom.AlgorithmIdentificationSequence, None]: Information useful for identification of the algorithm, if any.

Return type

typing.Optional[*highdicom.content.AlgorithmIdentificationSequence*]

property algorithm_type: *AnnotationGroupGenerationTypeValues*

algorithm type

Type

highdicom.ann.AnnotationGroupGenerationTypeValues

Return type
highdicom.ann.enum.AnnotationGroupGenerationTypeValues

property anatomic_regions: List[*CodedConcept*]

List[highdicom.sr.CodedConcept]: List of anatomic regions into which the annotations fall. May be empty.

Return type
typing.List[*highdicom.sr.coding.CodedConcept*]

property annotated_property_category: *CodedConcept*

coded annotated property category

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property annotated_property_type: *CodedConcept*

coded annotated property type

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

classmethod from_dataset(*dataset*)

Construct instance from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an item of the Annotation Group Sequence.

Returns
Item of the Annotation Group Sequence

Return type
highdicom.ann.AnnotationGroup

get_coordinates(*annotation_number*, *coordinate_type*)

Get spatial coordinates of a graphical annotation.

Parameters

- **annotation_number** (*int*) – One-based identification number of the annotation
- **coordinate_type** (*Union[str, highdicom.ann.AnnotationCoordinateTypeValues]*) – Coordinate type of annotation

Returns
Two-dimensional array of floating-point values representing either 2D or 3D spatial coordinates of a graphical annotation

Return type
numpy.ndarray

get_graphic_data(*coordinate_type*)

Get spatial coordinates of all graphical annotations.

Parameters
coordinate_type (*Union[str, highdicom.ann.AnnotationCoordinateTypeValues]*) – Coordinate type of annotations

Returns
Two-dimensional array of floating-point values representing either 2D or 3D spatial coordinates for each graphical annotation

Return type
List[numpy.ndarray]

`get_measurements`(*name=None*)

Get measurements.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Name by which measurements should be filtered

Return type
typing.Tuple[typing.List[*highdicom.sr.coding.CodedConcept*], numpy.ndarray, typing.List[*highdicom.sr.coding.CodedConcept*]]

Returns

- **names** (*List[highdicom.sr.CodedConcept]*) – Names of measurements
- **values** (*numpy.ndarray*) – Two-dimensional array of measurement floating point values. The array has shape n x m, where where *n* is the number of annotations and *m* is the number of measurements. The array may contain `numpy.nan` values in case a measurement is not available for a given annotation.
- **units** (*List[highdicom.sr.CodedConcept]*) – Units of measurements

`property graphic_type: GraphicTypeValues`

graphic type

Type
highdicom.ann.GraphicTypeValues

Return type
highdicom.ann.enum.GraphicTypeValues

`property label: str`

label

Type
str

Return type
str

`property number: int`

one-based identification number

Type
int

Return type
int

`property number_of_annotations: int`

Number of annotations in group

Type
int

Return type
int

property primary_anatomic_structures: List[*CodedConcept*]

List[highdicom.sr.CodedConcept]: List of anatomic anatomic structures the annotations represent. May be empty.

Return type
typing.List[*highdicom.sr.coding.CodedConcept*]

property uid: *UID*

unique identifier

Type
highdicom.UID

Return type
highdicom.uid.UID

class highdicom.ann.**AnnotationGroupGenerationTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Annotation Group Generation Type.

AUTOMATIC = 'AUTOMATIC'

MANUAL = 'MANUAL'

SEMIAUTOMATIC = 'SEMIAUTOMATIC'

class highdicom.ann.**GraphicTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Graphic Type.

Note: Coordinates may be either (Column,Row) pairs defined in the 2-dimensional Total Pixel Matrix or (X,Y,Z) triplets defined in the 3-dimensional Frame of Reference (patient or slide coordinate system).

Warning: Despite having the same names, the definition of values for the Graphic Type attribute of the ANN modality may differ from those of the SR modality (SCOORD or SCOORD3D value types).

ELLIPSE = 'ELLIPSE'

An ellipse defined by four coordinates.

The first two coordinates specify the endpoints of the major axis and the second two coordinates specify the endpoints of the minor axis.

POINT = 'POINT'

An individual piont defined by a single coordinate.

POLYGON = 'POLYGON'

Connected line segments defined by three or more ordered coordinates.

The coordinates shall be coplanar and form a closed polygon.

> **Warning:** In contrast to the corresponding SR Graphic Type for content items of SCOORD3D value type, the first and last points shall NOT be the same.

POLYLINE = 'POLYLINE'

Connected line segments defined by two or more ordered coordinates.

The coordinates shall be coplanar.

RECTANGLE = 'RECTANGLE'

Connected line segments defined by three or more ordered coordinates.

The coordinates shall be coplanar and form a closed, rectangular polygon. The first coordinate is the top left hand corner, the second coordinate is the top right hand corner, the third coordinate is the bottom right hand corner, and the forth coordinate is the bottom left hand corner.

The edges of the rectangle need not be aligned with the axes of the coordinate system.

class highdicom.ann.**Measurements**(*name*, *values*, *unit*)

Bases: Dataset

Dataset describing measurements of annotations.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **values** (*numpy.ndarray*) – One-dimensional array of floating-point values. Some values may be NaN (numpy.nan) if no measurement is available for a given annotation. Values must be sorted such that the *n*-th value represents the measurement for the *n*-th annotation.
- **unit** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code], optional*) – Coded units of measurement (see CID 7181 "Abstract Multi-dimensional Image Model Component Units")

classmethod from_dataset(*dataset*)

Construct instance from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an item of the Measurements Sequence.

Returns
Item of the Measurements Sequence

Return type
highdicom.ann.Measurements

get_values(*number_of_annotations*)

Get measured values for annotations.

Parameters
number_of_annotations (*int*) – Number of annotations in the annotation group

Returns
One-dimensional array of floating-point numbers of length *number_of_annotations*. The array may be sparse and annotations for which no measurements are available have value numpy.nan.

Return type
numpy.ndarray

Raises

IndexError – In case the measured values cannot be indexed given the indices stored in the Annotation Index List.

property name: *CodedConcept*

coded name

Type

highdicom.sr.CodedConcept

Return type

highdicom.sr.coding.CodedConcept

property unit: *CodedConcept*

coded unit

Type

highdicom.sr.CodedConcept

Return type

highdicom.sr.coding.CodedConcept

class highdicom.ann.**MicroscopyBulkSimpleAnnotations**(*source_images*, *annotation_coordinate_type*, *annotation_groups*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *content_description=None*, *content_creator_name=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, *pixel_origin_interpretation=PixelOriginInterpretationValues.VOLU*, *content_label=None*, ***kwargs*)

Bases: *SOPClass*

SOP class for the Microscopy Bulk Simple Annotations IOD.

Parameters

- **source_images** (*Sequence[pydicom.dataset.Dataset]*) – Image instances from which annotations were derived. In case of "2D" Annotation Coordinate Type, only one source image shall be provided. In case of "3D" Annotation Coordinate Type, one or more source images may be provided. All images shall have the same Frame of Reference UID.
- **annotation_coordinate_type** (*Union[str, highdicom.ann.AnnotationCoordinateTypeValues]*) – Type of coordinates (two-dimensional coordinates relative to origin of Total Pixel Matrix in pixel unit or three-dimensional coordinates relative to origin of Frame of Reference (Slide) in millimeter/micrometer unit)
- **annotation_groups** (*Sequence[highdicom.ann.AnnotationGroup]*) – Groups of annotations (vector graphics and corresponding measurements)
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **manufacturer** (*Union[str, None], optional*) – Name of the manufacturer (developer) of the device (software) that creates the instance

- **manufacturer_model_name** (*str*) – Name of the device model (name of the software library or application) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Version(s) of the software that creates the instance
- **device_serial_number** (*str*) – Manufacturer's serial number of the device
- **content_description** (*Union[str, None], optional*) – Description of the annotation
- **content_creator_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the creator of the annotation (if created manually)
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements.
- **content_label** (*Union[str, None], optional*) – Content label
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

classmethod from_dataset(*dataset*)

Construct instance from an existing dataset.

Parameters

dataset (*pydicom.dataset.Dataset*) – Dataset representing a Microscopy Bulk Simple Annotations instance.

Returns

Microscopy Bulk Simple Annotations instance

Return type

highdicom.ann.MicroscopyBulkSimpleAnnotations

get_annotation_group(*number=None*, *uid=None*)

Get an individual annotation group.

Parameters

- **number** (*Union[int, None], optional*) – Identification number of the annotation group
- **uid** (*Union[str, None], optional*) – Unique identifier of the annotation group

Returns

Annotation group

Return type

highdicom.ann.AnnotationGroup

Raises

- **TypeError** – When neither *number* nor *uid* is provided.
- **ValueError** – When no group item or more than one item is found matching either *number* or *uid*.

get_annotation_groups(*annotated_property_category=None*, *annotated_property_type=None*, *label=None*, *graphic_type=None*, *algorithm_type=None*, *algorithm_name=None*, *algorithm_family=None*, *algorithm_version=None*)

Get annotation groups matching search criteria.

Parameters

- **annotated_property_category** (*Union[Code, CodedConcept, None], optional*) – Category of annotated property (e.g., codes.SCT.MorphologicAbnormality)
- **annotated_property_type** (*Union[Code, CodedConcept, None], optional*) – Type of annotated property (e.g., codes.SCT.Neoplasm)
- **label** (*Union[str, None], optional*) – Annotation group label
- **graphic_type** (*Union[str, GraphicTypeValues, None], optional*) – Graphic type (e.g., highdicom.ann.GraphicTypeValues.POLYGON)
- **algorithm_type** (*Union[str, AnnotationGroupGenerationTypeValues, None], optional*) – Algorithm type (e.g., highdicom.ann.AnnotationGroupGenerationTypeValues.AUTOMATIC)
- **algorithm_name** (*Union[str, None], optional*) – Algorithm name
- **algorithm_family** (*Union[Code, CodedConcept, None], optional*) – Algorithm family (e.g., codes.DCM.ArtificialIntelligence)
- **algorithm_version** (*Union[str, None], optional*) – Algorithm version

Returns
Annotation groups

Return type
List[*highdicom.ann.AnnotationGroup*]

class highdicom.ann.**PixelOriginInterpretationValues**(*value*)

Bases: Enum

Enumerated values for attribute Pixel Origin Interpretation.

FRAME = 'FRAME'

Relative to an individual image frame.

Coordinates have been defined and need to be interpreted relative to the (1,1) pixel of an individual image frame.

VOLUME = 'VOLUME'

Relative to the Total Pixel Matrix of a VOLUME image.

Coordinates have been defined and need to be interpreted relative to the (1,1) pixel of the Total Pixel Matrix of the entire image.

8.4 highdicom.ko package

Package for creation of Key Object Selection instances.

class highdicom.ko.**KeyObjectSelection**(*document_title*, *referenced_objects*, *observer_person_context=None*, *observer_device_context=None*, *description=None*)

Bases: *ContentSequence*

Sequence of structured reporting content item describing a selection of DICOM objects according to structured reporting template TID 2010 Key Object Selection.

Parameters

- **document_title** (*Union[pydicom.sr.coding.Code, highdicom.srCodedConcept]*) – Coded title of the document (see CID 7010)
- **referenced_objects** (*Sequence[pydicom.dataset.Dataset]*) – Metadata of selected objects that should be referenced
- **observer_person_context** (*Union[highdicom.sr.ObserverContext, None], optional*) – Observer context describing the person that selected the objects
- **observer_device_context** (*Union[highdicom.sr.ObserverContext, None], optional*) – Observer context describing the device that selected the objects
- **description** (*Union[str, None], optional*) – Description of the selected objects

classmethod from_sequence(*sequence*, *is_root=True*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing "Key Object Selection" SR Content Items of Value Type CONTAINER (sequence shall only contain a single item)
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns

Content Sequence containing root CONTAINER SR Content Item

Return type

highdicom.ko.KeyObjectSelection

get_observer_contexts(*observer_type=None*)

Get observer contexts.

Parameters

observer_type (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Type of observer ("Device" or "Person") for which should be filtered

Returns

Observer contexts

Return type

List[*highdicom.sr.ObserverContext*]

get_references(*value_type=None*, *sop_class_uid=None*)

Get referenced objects.

Parameters

- **value_type** (*Union[highdicom.sr.ValueTypeValues, None], optional*) – Value type of content items that reference objects
- **sop_class_uid** (*Union[str, None], optional*) – SOP Class UID of referenced object

Returns

Content items that reference objects

Return type

List[Union[*highdicom.sr.ImageContentItem*, *highdicom.sr.CompositeContentItem*, highdicom.sr.WaveformContentItem]]

class highdicom.ko.**KeyObjectSelectionDocument**(*evidence*, *content*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer=None*, *institution_name=None*, *institutional_department_name=None*, *requested_procedures=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

Key Object Selection Document SOP class.

Parameters

- **evidence** (*Sequence[pydicom.dataset.Dataset]*) – Instances that are referenced in the content tree and from which the created KO document instance should inherit patient and study information
- **content** (highdicom.ko.KeyObjectSelection) – Content items that should be included in the document
- **series_instance_uid** (*str*) – Series Instance UID of the document series
- **series_number** (*int*) – Series Number of the document series
- **sop_instance_uid** (*str*) – SOP Instance UID that should be assigned to the document instance
- **instance_number** (*int*) – Number that should be assigned to this document instance
- **manufacturer** (*str, optional*) – Name of the manufacturer of the device that creates the document instance (in a research setting this is typically the same as *institution_name*)
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the document instance
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the document instance
- **requested_procedures** (*Union[Sequence[pydicom.dataset.Dataset], None], optional*) – Requested procedures that are being fullfilled by creation of the document
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

Raises

ValueError – When no *evidence* is provided

property **content**: *KeyObjectSelection*

document content

Type

highdicom.ko.KeyObjectSelection

Return type

highdicom.ko.content.KeyObjectSelection

classmethod **from_dataset**(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing a Key Object Selection Document

Returns
Key Object Selection Document

Return type
highdicom.ko.KeyObjectSelectionDocument

resolve_reference(*sop_instance_uid*)

Resolve reference for an object included in the document content.

Parameters
sop_instance_uid (*str*) – SOP Instance UID of a referenced object

Returns
Study, Series, and SOP Instance UID

Return type
Tuple[str, str, str]

8.5 highdicom.pm package

Package for creation of Parametric Map instances.

class highdicom.pm.**DerivedPixelContrastValues**(*value*)

Bases: Enum

Enumerated values for value 4 of attribute Image Type or Frame Type.

ADDITION = 'ADDITION'

DIVISION = 'DIVISION'

ENERGY_PROP_WT = 'ENERGY_PROP_WT'

FILTERED = 'FILTERED'

MASKED = 'MASKED'

MAXIMUM = 'MAXIMUM'

MEAN = 'MEAN'

MEDIAN = 'MEDIAN'

MINIMUM = 'MINIMUM'

MULTIPLICATION = 'MULTIPLICATION'

NONE = 'NONE'

QUANTITY = 'QUANTITY'

RESAMPLED = 'RESAMPLED'

STD_DEVIATION = 'STD_DEVIATION'

SUBTRACTION = 'SUBTRACTION'

class highdicom.pm.**DimensionIndexSequence**(*coordinate_system*)

Bases: Sequence

Sequence of data elements describing dimension indices for the patient or slide coordinate system based on the Dimension Index functional group macro. .. note:: The order of indices is fixed.

Parameters
coordinate_system (*Union[str, highdicom.CoordinateSystemNames]*) – Subject ("PATIENT" or "SLIDE") that was the target of imaging

get_index_keywords()

Get keywords of attributes that specify the position of planes.

Returns
Keywords of indexed attributes

Return type
List[str]

get_index_position(*pointer*)

Get relative position of a given dimension in the dimension index.

Parameters
pointer (*str*) – Name of the dimension (keyword of the attribute), e.g., "XOffsetInSlideCoordinateSystem"

Returns
Zero-based relative position

Return type
int

Examples

```
>>> dimension_index = DimensionIndexSequence("SLIDE")
>>> i = dimension_index.get_index_position("XOffsetInSlideCoordinateSystem")
>>> x_offsets = dimension_index[i]
```

get_index_values(*plane_positions*)

Get the values of indexed attributes.

Parameters
plane_positions (*Sequence[highdicom.PlanePositionSequence]*) – Plane position of frames in a multi-frame image or in a series of single-frame images

Return type
typing.Tuple[numpy.ndarray, numpy.ndarray]

Returns

- **dimension_index_values** (*numpy.ndarray*) – 2D array of spatial dimension index values
- **plane_indices** (*numpy.ndarray*) – 1D array of planes indices for sorting frames according to their spatial position specified by the dimension index.

get_plane_positions_of_image(*image*)

Get plane positions of frames in multi-frame image.

Parameters
image (*Dataset*) – Multi-frame image

Returns
Plane position of each frame in the image

Return type
List[*highdicom.PlanePositionSequence*]

get_plane_positions_of_series(*images*)

Gets plane positions for series of single-frame images.

Parameters
images (*Sequence[Dataset]*) – Series of single-frame images

Returns
Plane position of each frame in the image

Return type
List[*highdicom.PlanePositionSequence*]

class highdicom.pm.**ImageFlavorValues**(*value*)

Bases: Enum

Enumerated values for value 3 of attribute Image Type or Frame Type.

ANGIO = 'ANGIO'

ANGIO_TIME = 'ANGIO_TIME'

ASL = 'ASL'

ATTENUATION = 'ATTENUATION'

CARDIAC = 'CARDIAC'

CARDIAC_CASCORE = 'CARDIAC_CASCORE'

CARDIAC_CTA = 'CARDIAC_CTA'

CARDIAC_GATED = 'CARDIAC_GATED'

CARDRESP_GATED = 'CARDRESP_GATED'

CINE = 'CINE'

DIFFUSION = 'DIFFUSION'

DIXON = 'DIXON'

DYNAMIC = 'DYNAMIC'

FLOW_ENCODED = 'FLOW_ENCODED'

FLUID_ATTENUATED = 'FLUID_ATTENUATED'

FLUOROSCOPY = 'FLUOROSCOPY'

```
FMRI = 'FMRI'

LOCALIZER = 'LOCALIZER'

MAX_IP = 'MAX_IP'

METABOLITE_MAP = 'METABOLITE_MAP'

MIN_IP = 'MIN_IP'

MOTION = 'MOTION'

MULTIECHO = 'MULTIECHO'

M_MODE = 'M_MODE'

NON_PARALLEL = 'NON_PARALLEL'

PARALLEL = 'PARALLEL'

PERFUSION = 'PERFUSION'

POST_CONTRAST = 'POST_CONTRAST'

PRE_CONTRAST = 'PRE_CONTRAST'

PROTON_DENSITY = 'PROTON_DENSITY'

REALTIME = 'REALTIME'

REFERENCE = 'REFERENCE'

RESP_GATED = 'RESP_GATED'

REST = 'REST'

STATIC = 'STATIC'

STIR = 'STIR'

STRESS = 'STRESS'

T1 = 'T1'

T2 = 'T2'

T2_STAR = 'T2_STAR'

TAGGING = 'TAGGING'

TEMPERATURE = 'TEMPERATURE'

TOF = 'TOF'

VELOCITY = 'VELOCITY'

VOLUME = 'VOLUME'

WHOLE_BODY = 'WHOLE_BODY'
```

class highdicom.pm.**ParametricMap**(*source_images*, *pixel_array*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *contains_recognizable_visual_features*, *real_world_value_mappings*, *window_center*, *window_width*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, *content_description=None*, *content_creator_name=None*, *pixel_measures=None*, *plane_orientation=None*, *plane_positions=None*, *content_label=None*, *content_qualification=ContentQualificationValues.RESEARCH*, *image_flavor=ImageFlavorValues.VOLUME*, *derived_pixel_contrast=DerivedPixelContrastValues.QUANTITY*, *content_creator_identification=None*, *palette_color_lut_transformation=None*, ***kwargs*)

Bases: *SOPClass*

SOP class for a Parametric Map.

Note: This class only supports creation of Parametric Map instances with a value of interest (VOI) lookup table that describes a linear transformation that equally applies to all frames in the image.

Parameters

- **source_images** (*Sequence[pydicom.dataset.Dataset]*) – One or more single- or multi-frame images (or metadata of images) from which the parametric map was derived
- **pixel_array** (*numpy.ndarray*) – 2D, 3D, or 4D array of unsigned integer or floating-point data type representing one or more channels (images derived from source images via an image transformation) for one or more spatial image positions:
  - In case of a 2D array, the values represent a single channel for a single 2D frame and the array shall have shape `(r, c)`, where `r` is the number of rows and `c` is the number of columns.
  - In case of a 3D array, the values represent a single channel for multiple 2D frames at different spatial image positions and the array shall have shape `(n, r, c)`, where `n` is the number of frames, `r` is the number of rows per frame, and `c` is the number of columns per frame.
  - In case of a 4D array, the values represent multiple channels for multiple 2D frames at different spatial image positions and the array shall have shape `(n, r, c, m)`, where `n` is the number of frames, `r` is the number of rows per frame, `c` is the number of columns per frame, and `m` is the number of channels.
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **manufacturer** (*str*) – Name of the manufacturer (developer) of the device (software) that creates the instance
- **manufacturer_model_name** (*str,*) – Name of the model of the device (software) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Versions of relevant software used to create the data

- **device_serial_number** (*str*) – Serial number (or other identifier) of the device (software) that creates the instance
- **contains_recognizable_visual_features** (*bool*) – Whether the image contains recognizable visible features of the patient
- **real_world_value_mappings** (*Union[Sequence[highdicom.map. RealWorldValueMapping], Sequence[Sequence[highdicom.map. RealWorldValueMapping]]*) – Descriptions of how stored values map to real-world values. Each channel encoded in *pixel_array* shall be described with one or more real-world value mappings. Multiple mappings might be used for different representations such as log versus linear scales or for different representations in different units. If *pixel_array* is a 2D or 3D array and only one channel exists at each spatial image position, then one or more real-world value mappings shall be provided in a flat sequence. If *pixel_array* is a 4D array and multiple channels exist at each spatial image position, then one or more mappings shall be provided for each channel in a nested sequence of length `m`, where `m` shall match the channel dimension of the *pixel_array* `.

  In some situations the mapping may be difficult to describe (e.g., in case of a transformation performed by a deep convolutional neural network). The real-world value mapping may then simply describe an identity function that maps stored values to unit-less real-world values.
- **window_center** (*Union[int, float, None], optional*) – Window center (intensity) for rescaling stored values for display purposes by applying a linear transformation function. For example, in case of floating-point values in the range `[0.0, 1.0]`, the window center may be `0.5`, in case of floating-point values in the range `[-1.0, 1.0]` the window center may be `0.0`, in case of unsigned integer values in the range `[0, 255]` the window center may be `128`.
- **window_width** (*Union[int, float, None], optional*) – Window width (contrast) for rescaling stored values for display purposes by applying a linear transformation function. For example, in case of floating-point values in the range `[0.0, 1.0]`, the window width may be `1.0`, in case of floating-point values in the range `[-1.0, 1.0]` the window width may be `2.0`, and in case of unsigned integer values in the range `[0, 255]` the window width may be `256`. In case of unbounded floating-point values, a sensible window width should be chosen to allow for stored values to be displayed on 8-bit monitors.
- **transfer_syntax_uid** (*Union[str, None], optional*) – UID of transfer syntax that should be used for encoding of data elements. Defaults to Explicit VR Little Endian (UID `"1.2.840.10008.1.2.1"`)
- **content_description** (*Union[str, None], optional*) – Brief description of the parametric map image
- **content_creator_name** (*Union[str, None], optional*) – Name of the person that created the parametric map image
- **pixel_measures** (*Union[*highdicom.PixelMeasuresSequence*, None], optional*) – Physical spacing of image pixels in *pixel_array*. If `None`, it will be assumed that the parametric map image has the same pixel measures as the source image(s).
- **plane_orientation** (*Union[*highdicom.PlaneOrientationSequence*, None], optional*) – Orientation of planes in *pixel_array* relative to axes of three-dimensional patient or slide coordinate space. If `None`, it will be assumed that the parametric map image as the same plane orientation as the source image(s).
- **plane_positions** (*Union[Sequence[*PlanePositionSequence*], None], optional*) – Position of each plane in *pixel_array* in the three-dimensional patient or slide coordinate space. If `None`, it will be assumed that the parametric map image has

the same plane position as the source image(s). However, this will only work when the first dimension of *pixel_array* matches the number of frames in *source_images* (in case of multi-frame source images) or the number of *source_images* (in case of single-frame source images).

- **content_label** (*Union[str, None], optional*) – Content label
- **content_qualification** (*Union[str, highdicom.ContentQualificationValues], optional*) – Indicator of whether content was produced with approved hardware and software
- **image_flavor** (*Union[str, highdicom.pm.ImageFlavorValues], optional*) – Overall representation of the image type
- **derived_pixel_contrast** (*Union[str, highdicom.pm.DerivedPixelContrast], optional*) – Contrast created by combining or processing source images with the same geometry
- **content_creator_identification** (*Union[highdicom.ContentCreatorIdentificationCodeSequence, None], optional*) – Identifying information for the person who created the content of this parametric map.
- **palette_color_lut_transformation** (*Union[highdicom.PaletteColorLUTTransformation, None], optional*) – Description of the Palette Color LUT Transformation for tranforming grayscale into RGB color pixel values
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

Raises

ValueError – When * Length of *source_images* is zero. * Items of *source_images* are not all part of the same study and series. * Items of *source_images* have different number of rows and columns. * Length of *plane_positions* does not match number of 2D planes in *pixel_array* (size of first array dimension). * Transfer Syntax specified by *transfer_syntax_uid* is not supported for data type of *pixel_array*.

Note: The assumption is made that planes in *pixel_array* are defined in the same frame of reference as *source_images*. It is further assumed that all image frame have the same type (i.e., the same *image_flavor* and *derived_pixel_contrast*).

---

class highdicom.pm.**RealWorldValueMapping**(*lut_label*, *lut_explanation*, *unit*, *value_range*, *slope=None*, *intercept=None*, *lut_data=None*, *quantity_definition=None*)

Bases: `Dataset`

Class representing the Real World Value Mapping Item Macro.

Parameters

- **lut_label** (*str*) – Label (identifier) used to identify transformation. Must be less than or equal to 16 characters.
- **lut_explanation** (*str*) – Explanation (short description) of the meaning of the transformation
- **unit** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Unit of the real world values. This may be not applicable, because the values may not have a (known) unit. In this case, use `pydicom.sr.codedict.codes.UCUM.NoUnits`.

- **value_range** (*Union[Tuple[int, int], Tuple[float, float]]*) – Upper and lower value of range of stored values to which the mapping should be restricted. For example, values may be stored as floating-point values with double precision, but limited to the range `(-1.0, 1.0)` or `(0.0, 1.0)` or stored as 16-bit unsigned integer values but limited to range `(0, 4094)`. Note that the type of the values in `value_range` is significant and is used to determine whether values are stored as integers or floating-point values. Therefore, use ``(0.0, 1.0)` instead of `(0, 1)` to specify a range of floating-point values.
- **slope** (*Union[int, float, None], optional*) – Slope of the linear mapping function applied to values in *value_range*.
- **intercept** (*Union[int, float, None], optional*) – Intercept of the linear mapping function applied to values in *value_range*.
- **lut_data** (*Union[Sequence[int], Sequence[float], None], optional*) – Sequence of values to serve as a lookup table for mapping stored values into real-world values in case of a non-linear relationship. The sequence should contain an entry for each value in the specified *value_range* such that `len(sequence) == value_range[1] - value_range[0] + 1`. For example, in case of a value range of `(0, 255)`, the sequence shall have `256` entries - one for each value in the given range.
- **quantity_definition** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Description of the quantity represented by real world values (see CID 7180 "Abstract Multi-dimensional Image Model Component Semantics")

Note: Either *slope* and *intercept* or *lut_data* must be specified. Specify *slope* and *intercept* if the mapping can be described by a linear function. Specify *lut_data* if the relationship between stored and real-world values is non-linear. Note, however, that a non-linear relationship can only be described for values that are stored as integers. Values stored as floating-point numbers must map linearly to real-world values.

8.6 highdicom.pr package

Package for creation of Presentation State instances.

class highdicom.pr.**AdvancedBlending**(*referenced_images*, *blending_input_number*, *modality_lut_transformation=None*, *voi_lut_transformations=None*, *palette_color_lut_transformation=None*)

Bases: `Dataset`

Class for an item of the Advanced Blending Sequence.

Parameters

- **referenced_images** (*Sequence[pydicom.Dataset]*) – Images that should be referenced
- **blending_input_number** (*int*) – Relative one-based index of the item for input into the blending operation
- **modality_lut_transformation** (*Union[highdicom.ModalityLUTTransformation, None], optional*) – Description of the Modality LUT Transformation for transforming modality dependent into modality independent pixel values

- **voi_lut_transformations** (*Union[Sequence[highdicom.pr.SoftcopyVOILUTTransformation], None], optional*) – Description of the VOI LUT Transformation for transforming modality pixel values into pixel values that are of interest to a user or an application
- **palette_color_lut_transformation** (*Union[highdicom.PaletteColorLUTTransformation, None], optional*) – Description of the Palette Color LUT Transformation for transforming grayscale into RGB color pixel values

class highdicom.pr.**AdvancedBlendingPresentationState**(*referenced_images*, *blending*, *blending_display*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *content_label*, *content_description=None*, *graphic_annotations=None*, *graphic_layers=None*, *graphic_groups=None*, *concept_name=None*, *institution_name=None*, *institutional_department_name=None*, *content_creator_name=None*, *content_creator_identification=None*, *icc_profile=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for an Advanced Blending Presentation State object.

An Advanced Blending Presentation State object includes instructions for the blending of one or more pseudo-color or color images by software. If the referenced images are grayscale images, they first need to be pseudo-colored.

Parameters

- **referenced_images** (*Sequence[pydicom.Dataset]*) – Images that should be referenced. This list should contain all images that are referenced across all *blending* items.
- **blending** (*Sequence[highdicom.pr.AdvancedBlending]*) – Description of groups of images that should be blended to form a pseudo-color image.
- **blending_display** (*Sequence[highdicom.pr.BlendingDisplay]*) – Description of the blending operations and the images to be used. Each item results in an individual pseudo-color RGB image, which may reused in a following step.
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **manufacturer** (*str*) – Name of the manufacturer of the device (developer of the software) that creates the instance
- **manufacturer_model_name** (*str*) – Name of the device model (name of the software library or application) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Version(s) of the software that creates the instance

- **device_serial_number** (*Union[str, None]*) – Manufacturer's serial number of the device
- **content_label** (*str*) – A label used to describe the content of this presentation state. Must be a valid DICOM code string consisting only of capital letters, underscores and spaces.
- **content_description** (*Union[str, None], optional*) – Description of the content of this presentation state.
- **graphic_annotations** (*Union[Sequence[highdicom.pr.GraphicAnnotation], None], optional*) – Graphic annotations to include in this presentation state.
- **graphic_layers** (*Union[Sequence[highdicom.pr.GraphicLayer], None], optional*) – Graphic layers to include in this presentation state. All graphic layers referenced in "graphic_annotations" must be included.
- **graphic_groups** (*Optional[Sequence[highdicom.pr.GraphicGroup]], optional*) – Description of graphic groups used in this presentation state.
- **concept_name** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – A coded description of the content of this presentation state.
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance.
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance.
- **content_creator_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the person who created the content of this presentation state.
- **content_creator_identification** (*Union[highdicom.ContentCreatorIdentificationCodeSequence, None], optional*) – Identifying information for the person who created the content of this presentation state.
- **icc_profile** (*Union[bytes, None], optional*) – ICC color profile to include in the presentation state. If none is provided, a default profile will be included for the sRGB color space. The profile must follow the constraints listed in C.11.15.
- **transfer_syntax_uid** (*Union[str, highdicom.UID], optional*) – Transfer syntax UID of the presentation state.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

class highdicom.pr.**AnnotationUnitsValues**(*value*)

Bases: Enum

Enumerated values for annotation units, describing how the stored values relate to the image position.

DISPLAY = 'DISPLAY'

Display coordinates.

Display coordinates in pixel unit specified with sub-pixel resolution, where (0.0, 0.0) is the top left hand corner of the displayed area and (1.0, 1.0) is the bottom right hand corner of the displayed area. Values are between 0.0 and 1.0.

MATRIX = 'MATRIX'

Image coordinates relative to the total pixel matrix of a tiled image.

Image coordinates in pixel unit specified with sub-pixel resolution such that the origin, which is at the Top Left Hand Corner (TLHC) of the TLHC pixel of the Total Pixel Matrix, is (0.0, 0.0), the Bottom Right Hand

Corner (BRHC) of the TLHC pixel is (1.0, 1.0), and the BRHC of the BRHC pixel of the Total Pixel Matrix is (Total Pixel Matrix Columns,Total Pixel Matrix Rows). The values must be within the range (0.0, 0.0) to (Total Pixel Matrix Columns, Total Pixel Matrix Rows). MATRIX may be used only if the referenced image is tiled (i.e. has attributes Total Pixel Matrix Rows and Total Pixel Matrix Columns).

PIXEL = 'PIXEL'

Image coordinates within an individual image image frame.

Image coordinates in pixel unit specified with sub-pixel resolution such that the origin, which is at the Top Left Hand Corner (TLHC) of the TLHC pixel is (0.0, 0.0), the Bottom Right Hand Corner (BRHC) of the TLHC pixel is (1.0, 1.0), and the BRHC of the BRHC pixel is (Columns, Rows). The values must be within the range (0, 0) to (Columns, Rows).

class highdicom.pr.**BlendingDisplay**(*blending_mode*, *blending_display_inputs*, *blending_input_number=None*, *relative_opacity=None*)

Bases: Dataset

Class for an item of the Blending Display Sequence attribute.

Parameters

- **blending_mode** (*Union[str, highdicom.pr.BlendingModeValues]*) – Method for weighting the different input images during the blending operation using alpha composition with premultiplication
- **blending_display_inputs** (*Sequence[highdicom.pr.BlendingDisplayInput]*) – Inputs for the blending operation. The order of items determines the order in which images will be blended.
- **blending_input_number** (*Union[int, None], optional*) – One-based identification index number of the result. Required if the output of the blending operation should not be directly displayed but used as input for a subsequent blending operation.
- **relative_opacity** (*Union[float, None], optional*) – Relative opacity (alpha value) that should be premultiplied with pixel values of the foreground image. Pixel values of the background image will be premultilied with 1 - *relative_opacity*. Required if *blending_mode* is "FOREGROUND". Will be ignored otherwise.

class highdicom.pr.**BlendingDisplayInput**(*blending_input_number*)

Bases: Dataset

Class for an item of the Blending Display Input Sequence attribute.

Parameters

blending_input_number (*int*) – One-based identification index number of the input series to which the blending information should be applied

class highdicom.pr.**BlendingModeValues**(*value*)

Bases: Enum

Enumerated values for the Blending Mode attribute.

Pixel values are additively blended using alpha compositioning with premultiplied alpha. The Blending Mode attribute describes how the premultiplier alpha value is computed for each image.

EQUAL = 'EQUAL'

Additive blending of two or more images with equal alpha premultipliers.

Pixel values of *n* images are additively blended in an iterative fashion after premultiplying pixel values with a constant alpha value, which is either 0 or 1/n of the value of the Relative Opacity attribute: 1/n * Relative Opacity * first value + 1/n * Relative Opacity * second value

FOREGROUND = 'FOREGROUND'

Additive blending of two images with different alpha premultipliers.

The first image serves as background and the second image serves as foreground. Pixel values of the two images are additively blended after premultiplying the pixel values of each image with a different alpha value, which is computed from the value of the Relative Opacity attribute: Relative Opacity * first value + (1 - Relative Opacity) * second value

class highdicom.pr.**ColorSoftcopyPresentationState**(*referenced_images*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *content_label*, *content_description=None*, *graphic_annotations=None*, *graphic_layers=None*, *graphic_groups=None*, *concept_name=None*, *institution_name=None*, *institutional_department_name=None*, *content_creator_name=None*, *content_creator_identification=None*, *icc_profile=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for a Color Softcopy Presentation State object.

A Color Softcopy Presentation State object includes instructions for the presentation of a color image by software.

Parameters

- **referenced_images** (*Sequence[pydicom.Dataset]*) – Images that should be referenced
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **manufacturer** (*str*) – Name of the manufacturer of the device (developer of the software) that creates the instance
- **manufacturer_model_name** (*str*) – Name of the device model (name of the software library or application) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Version(s) of the software that creates the instance
- **device_serial_number** (*Union[str, None]*) – Manufacturer's serial number of the device
- **content_label** (*str*) – A label used to describe the content of this presentation state. Must be a valid DICOM code string consisting only of capital letters, underscores and spaces.
- **content_description** (*Union[str, None], optional*) – Description of the content of this presentation state.
- **graphic_annotations** (*Union[Sequence[highdicom.pr.GraphicAnnotation], None], optional*) – Graphic annotations to include in this presentation state.

- **graphic_layers** (*Union[Sequence[highdicom.pr.GraphicLayer], None], optional*) – Graphic layers to include in this presentation state. All graphic layers referenced in "graphic_annotations" must be included.
- **graphic_groups** (*Optional[Sequence[highdicom.pr.GraphicGroup]], optional*) – Description of graphic groups used in this presentation state.
- **concept_name** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – A coded description of the content of this presentation state.
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance.
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance.
- **content_creator_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the person who created the content of this presentation state.
- **content_creator_identification** (*Union[highdicom.ContentCreatorIdentificationCodeSequence, None], optional*) – Identifying information for the person who created the content of this presentation state.
- **icc_profile** (*Union[bytes, None], optional*) – ICC color profile to include in the presentation state. If none is provided, the profile will be copied from the referenced images. The profile must follow the constraints listed in C.11.15.
- **transfer_syntax_uid** (*Union[str, highdicom.UID], optional*) – Transfer syntax UID of the presentation state.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

class highdicom.pr.**GraphicAnnotation**(*referenced_images*, *graphic_layer*, *referenced_frame_number=None*, *referenced_segment_number=None*, *graphic_objects=None*, *text_objects=None*)

Bases: `Dataset`

Dataset describing related graphic and text objects.

Parameters

- **referenced_images** (*Sequence[pydicom.dataset.Dataset]*) – Sequence of referenced datasets. Graphic and text objects shall be rendered on all images in this list.
- **graphic_layer** (highdicom.pr.GraphicLayer) – Graphic layer to which this annotation should belong.
- **referenced_frame_number** (*Union[int, Sequence[int], None], optional*) – Frame number(s) in a multiframe image upon which annotations shall be rendered.
- **referenced_segment_number** (*Union[int, Sequence[int], None], optional*) – Frame number(s) in a multi-frame image upon which annotations shall be rendered.
- **graphic_objects** (*Union[Sequence[highdicom.pr.GraphicObject], None], optional*) – Graphic objects to render over the referenced images.
- **text_objects** (*Union[Sequence[highdicom.pr.TextObject], None], optional*) – Text objects to render over the referenced images.

class highdicom.pr.**GraphicGroup**(*graphic_group_id*, *label*, *description=None*)

Bases: Dataset

Dataset describing a grouping of annotations.

Note: GraphicGroup s represent an independent concept from GraphicLayer s. Where a GraphicLayer (*highdicom.pr.GraphicLayer*) specifies which annotations are rendered first, a GraphicGroup specifies which annotations belong together and shall be handled together (e.g., rotate, move) independent of the GraphicLayer to which they are assigned.

Each annotation (*highdicom.pr.GraphicObject* or *highdicom.pr.TextObject*) may optionally be assigned to a single GraphicGroup upon construction, whereas assignment to a *highdicom.pr.GraphicLayer* is required.

For example, suppose a presentation state is to include two GraphicObject s, each accompanied by a corresponding TextObject that indicates the meaning of the graphic and should be rendered above the GraphicObject if they overlap. In this situation, it may be useful to group each TextObject with the corresponding GraphicObject as a distinct GraphicGroup (giving two GraphicGroup s each containing one TextObject and one GraphicObject) and also place both GraphicObject s in one GraphicLayer and both TextObject s in a second GraphicLayer with a higher order to control rendering.

Parameters

- **graphic_group_id** (*int*) – A positive integer that uniquely identifies this graphic group.
- **label** (*str*) – Name used to identify the Graphic Group (maximum 64 characters).
- **description** (*Union[str, None], optional*) – Description of the group (maxiumum 10240 characters).

property **graphic_group_id: int**

The ID of the graphic group.

Type
int

Return type
int

class highdicom.pr.**GraphicLayer**(*layer_name*, *order*, *description=None*, *display_color=None*)

Bases: Dataset

A layer of graphic annotations that should be rendered together.

Parameters

- **layer_name** (*str*) – Name for the layer. Should be a valid DICOM Code String (CS), i.e. 16 characters or fewer containing only uppercase letters, spaces and underscores.
- **order** (*int*) – Integer indicating the order in which this layer should be rendered. Lower values are rendered first.
- **description** (*Union[str, None], optional*) – A description of the contents of this graphic layer.
- **display_color** (*Union[CIELabColor, None], optional*) – A default color value for rendering this layer.

class highdicom.pr.**GraphicObject**(*graphic_type*, *graphic_data*, *units*, *is_filled=False*, *tracking_id=None*, *tracking_uid=None*, *graphic_group=None*)

Bases: `Dataset`

Dataset describing a graphic annotation object.

Parameters

- **graphic_type** (*Union[highdicom.pr.GraphicTypeValues, str]*) – Type of the graphic data.
- **graphic_data** (*numpy.ndarray*) – Graphic data contained in a 2D NumPy array. The shape of the array should be (N, 2), where N is the number of 2D points in this graphic object. Each row of the array therefore describes a (column, row) value for a single 2D point, and the interpretation of the points depends upon the graphic type. See *highdicom.pr.enum.GraphicTypeValues* for details.
- **units** (*Union[highdicom.pr.AnnotationUnitsValues, str]*) – The units in which each point in graphic data is expressed.
- **is_filled** (*bool, optional*) – Whether the graphic object should be rendered as a solid shape (`True`), or just an outline (`False`). Using `True` is only valid when the graphic type is `'CIRCLE'` or `'ELLIPSE'`, or the graphic type is `'INTERPOLATED'` or `'POLYLINE'` and the first and last points are equal giving a closed shape.
- **tracking_id** (*str, optional*) – User defined text identifier for tracking this finding or feature. Shall be unique within the domain in which it is used.
- **tracking_uid** (*str, optional*) – Unique identifier for tracking this finding or feature.
- **graphic_group** (*Union[highdicom.pr.GraphicGroup, None]*) – Graphic group to which this annotation belongs.

property **graphic_data**: `ndarray`

n x 2 array of 2D coordinates

Type
numpy.ndarray

Return type
`numpy.ndarray`

property **graphic_group_id**: `Optional[int]`

The ID of the graphic group, if any.

Type
Union[int, None]

Return type
`typing.Optional[int]`

property **graphic_type**: *GraphicTypeValues*

graphic type

Type
highdicom.pr.GraphicTypeValues

Return type
highdicom.pr.enum.GraphicTypeValues

property tracking_id: Optional[str]

tracking identifier

Type
Union[str, None]

Return type
typing.Optional[str]

property tracking_uid: Optional[*UID*]

tracking UID

Type
Union[*highdicom.UID*, None]

Return type
typing.Optional[*highdicom.uid.UID*]

property units: *AnnotationUnitsValues*

annotation units

Type
highdicom.pr.AnnotationUnitsValues

Return type
highdicom.pr.enum.AnnotationUnitsValues

class highdicom.pr.GraphicTypeValues(*value*)

Bases: Enum

Enumerated values for attribute Graphic Type.

See C.10.5.2.

CIRCLE = 'CIRCLE'

A circle defined by two (column,row) pairs.

The first pair is the central point and the second pair is a point on the perimeter of the circle.

ELLIPSE = 'ELLIPSE'

An ellipse defined by four pixel (column,row) pairs.

The first two pairs specify the endpoints of the major axis and the second two pairs specify the endpoints of the minor axis.

INTERPOLATED = 'INTERPOLATED'

List of end points between which a line is to be interpolated.

The exact nature of the interpolation is an implementation detail of the software rendering the object.

Each point is represented by a (column,row) pair.

POINT = 'POINT'

A single point defined by two values (column,row).

POLYLINE = 'POLYLINE'

List of end points between which straight lines are to be drawn.

Each point is represented by a (column,row) pair.

class highdicom.pr.**GrayscaleSoftcopyPresentationState**(*referenced_images*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *content_label*, *content_description=None*, *graphic_annotations=None*, *graphic_layers=None*, *graphic_groups=None*, *concept_name=None*, *institution_name=None*, *institutional_department_name=None*, *content_creator_name=None*, *content_creator_identification=None*, *modality_lut_transformation=None*, *voi_lut_transformations=None*, *presentation_lut_transformation=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for a Grayscale Softcopy Presentation State (GSPS) object.

A GSPS object includes instructions for the presentation of a grayscale image by software.

Parameters

- **referenced_images** (*Sequence[pydicom.Dataset]*) – Images that should be referenced
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **manufacturer** (*str*) – Name of the manufacturer of the device (developer of the software) that creates the instance
- **manufacturer_model_name** (*str*) – Name of the device model (name of the software library or application) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Version(s) of the software that creates the instance
- **device_serial_number** (*Union[str, None]*) – Manufacturer's serial number of the device
- **content_label** (*str*) – A label used to describe the content of this presentation state. Must be a valid DICOM code string consisting only of capital letters, underscores and spaces.
- **content_description** (*Union[str, None], optional*) – Description of the content of this presentation state.
- **graphic_annotations** (*Union[Sequence[highdicom.pr.GraphicAnnotation], None], optional*) – Graphic annotations to include in this presentation state.
- **graphic_layers** (*Union[Sequence[highdicom.pr.GraphicLayer], None], optional*) – Graphic layers to include in this presentation state. All graphic layers referenced in "graphic_annotations" must be included.

- **graphic_groups** (*Optional[Sequence[highdicom.pr.GraphicGroup]], optional*) – Description of graphic groups used in this presentation state.
- **concept_name** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – A coded description of the content of this presentation state.
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance.
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance.
- **content_creator_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the person who created the content of this presentation state.
- **content_creator_identification** (*Union[highdicom.ContentCreatorIdentificationCodeSequence, None], optional*) – Identifying information for the person who created the content of this presentation state.
- **modality_lut_transformation** (*Union[highdicom.ModalityLUTTransformation, None], optional*) – Description of the Modality LUT Transformation for transforming modality dependent into modality independent pixel values. If no value is provided, the modality transformation in the referenced images, if any, will be used.
- **voi_lut_transformations** (*Union[Sequence[highdicom.pr.SoftcopyVOILUTTransformation], None], optional*) – Description of the VOI LUT Transformation for transforming modality pixel values into pixel values that are of interest to a user or an application. If no value is provided, the VOI LUT transformation in the referenced images, if any, will be used.
- **presentation_lut_transformation** (*Union[highdicom.PresentationLUTTransformation, None], optional*) – Description of the Presentation LUT Transformation for transforming polarity pixel values into device-independent presentation values
- **transfer_syntax_uid** (*Union[str, highdicom.UID], optional*) – Transfer syntax UID of the presentation state.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

class highdicom.pr.**PseudoColorSoftcopyPresentationState**(*referenced_images*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *palette_color_lut_transformation*, *content_label*, *content_description=None*, *graphic_annotations=None*, *graphic_layers=None*, *graphic_groups=None*, *concept_name=None*, *institution_name=None*, *institutional_department_name=None*, *content_creator_name=None*, *content_creator_identification=None*, *modality_lut_transformation=None*, *voi_lut_transformations=None*, *icc_profile=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for a Pseudo-Color Softcopy Presentation State object.

A Pseudo-Color Softcopy Presentation State object includes instructions for the presentation of a grayscale image as a color image by software.

Parameters

- **referenced_images** (*Sequence[pydicom.Dataset]*) – Images that should be referenced.
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **manufacturer** (*str*) – Name of the manufacturer of the device (developer of the software) that creates the instance
- **manufacturer_model_name** (*str*) – Name of the device model (name of the software library or application) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Version(s) of the software that creates the instance
- **device_serial_number** (*Union[str, None]*) – Manufacturer's serial number of the device
- **palette_color_lut_transformation** (highdicom.PaletteColorLUTTransformation) – Description of the Palette Color LUT Transformation for tranforming grayscale into RGB color pixel values
- **content_label** (*str*) – A label used to describe the content of this presentation state. Must be a valid DICOM code string consisting only of capital letters, underscores and spaces.
- **content_description** (*Union[str, None], optional*) – Description of the content of this presentation state.

- **graphic_annotations** (*Union[Sequence[highdicom.pr.GraphicAnnotation], None], optional*) – Graphic annotations to include in this presentation state.
- **graphic_layers** (*Union[Sequence[highdicom.pr.GraphicLayer], None], optional*) – Graphic layers to include in this presentation state. All graphic layers referenced in "graphic_annotations" must be included.
- **graphic_groups** (*Optional[Sequence[highdicom.pr.GraphicGroup]], optional*) – Description of graphic groups used in this presentation state.
- **concept_name** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – A coded description of the content of this presentation state.
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance.
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance.
- **content_creator_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the person who created the content of this presentation state.
- **content_creator_identification** (*Union[highdicom.ContentCreatorIdentificationCodeSequence, None], optional*) – Identifying information for the person who created the content of this presentation state.
- **modality_lut_transformation** (*Union[highdicom.ModalityLUTTransformation, None], optional*) – Description of the Modality LUT Transformation for tranforming modality dependent into modality independent pixel values
- **voi_lut_transformations** (*Union[Sequence[highdicom.pr.SoftcopyVOILUTTransformation], None], optional*) – Description of the VOI LUT Transformation for tranforming modality pixel values into pixel values that are of interest to a user or an application
- **icc_profile** (*Union[bytes, None], optional*) – ICC color profile to include in the presentation state. If none is provided, the profile will be copied from the referenced images. The profile must follow the constraints listed in C.11.15.
- **transfer_syntax_uid** (*Union[str, highdicom.UID], optional*) – Transfer syntax UID of the presentation state.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

class highdicom.pr.**SoftcopyVOILUTTransformation**(*window_center=None*, *window_width=None*, *window_explanation=None*, *voi_lut_function=None*, *voi_luts=None*, *referenced_images=None*)

Bases: *VOILUTTransformation*

Dataset describing the VOI LUT Transformation as part of the Pixel Transformation Sequence to transform the modality pixel values into pixel values that are of interest to a user or an application.

The description is specific to the application of the VOI LUT Transformation in the context of a Softcopy Presentation State, where potentially only a subset of explicitly referenced images should be transformed.

Parameters

- **window_center** (*Union[float, Sequence[float], None], optional*) – Center value of the intensity window used for display.

- **window_width** (*Union[float, Sequence[float], None], optional*) – Width of the intensity window used for display.
- **window_explanation** (*Union[str, Sequence[str], None], optional*) – Free-form explanation of the window center and width.
- **voi_lut_function** (*Union[highdicom.VOILUTFunctionValues, str, None], optional*) – Description of the LUT function parametrized by `window_center`. and `window_width`.
- **voi_luts** (*Union[Sequence[highdicom.VOILUT], None], optional*) – Intensity lookup tables used for display.
- **referenced_images** (*Union[highdicom.ReferencedImageSequence, None], optional*) – Images to which the VOI LUT Transformation described in this dataset applies. Note that if unspecified, the VOI LUT Transformation applies to every frame of every image referenced in the presentation state object that this dataset is included in.

Note: Either `window_center` and `window_width` should be provided or `voi_luts` should be provided, or both. `window_explanation` should only be provided if `window_center` is provided.

---

class highdicom.pr.**TextJustificationValues**(*value*)

Bases: `Enum`

Enumerated values for attribute Bounding Box Text Horizontal Justification.

CENTER = 'CENTER'

LEFT = 'LEFT'

RIGHT = 'RIGHT'

class highdicom.pr.**TextObject**(*text_value*, *units*, *bounding_box=None*, *anchor_point=None*, *text_justification=TextJustificationValues.CENTER*, *anchor_point_visible=True*, *tracking_id=None*, *tracking_uid=None*, *graphic_group=None*)

Bases: `Dataset`

Dataset describing a text annotation object.

Parameters

- **text_value** (*str*) – The unformatted text value.
- **units** (*Union[highdicom.pr.AnnotationUnitsValues, str]*) – The units in which the coordinates of the bounding box and/or anchor point are expressed.
- **bounding_box** (*Union[Tuple[float, float, float, float], None], optional*) – Coordinates of the bounding box in which the text should be displayed, given in the following order [left, top, right, bottom], where 'left' and 'right' are the horizontal offsets of the left and right sides of the box, respectively, and 'top' and 'bottom' are the vertical offsets of the upper and lower sides of the box.
- **anchor_point** (*Union[Tuple[float, float], None], optional*) – Location of a point in the image to which the text value is related, given as a (Column, Row) pair.
- **anchor_point_visible** (*bool, optional*) – Whether the relationship between the anchor point and the text should be displayed in the image, for example via a line or arrow. This parameter is ignored if the anchor_point is not provided.

- **tracking_id** (*str, optional*) – User defined text identifier for tracking this finding or feature. Shall be unique within the domain in which it is used.
- **tracking_uid** (*str, optional*) – Unique identifier for tracking this finding or feature.
- **graphic_group** (*Union[highdicom.pr.GraphicGroup, None], optional*) – Graphic group to which this annotation belongs.

Note: Either the `anchor_point` or the `bounding_box` parameter (or both) must be provided to localize the text in the image.

---

property anchor_point: Optional[Tuple[float, float]]

Union[Tuple[float, float], None]: anchor point as a (Row, Column) pair of image coordinates

Return type
typing.Optional[typing.Tuple[float, float]]

property bounding_box: Optional[Tuple[float, float, float, float]]

Union[Tuple[float, float, float, float], None]: bounding box in the format [left, top, right, bottom]

Return type
typing.Optional[typing.Tuple[float, float, float, float]]

property graphic_group_id: Optional[int]

The ID of the graphic group, if any.

Type
Union[int, None]

Return type
typing.Optional[int]

property text_value: str

unformatted text value

Type
str

Return type
str

property tracking_id: Optional[str]

tracking identifier

Type
Union[str, None]

Return type
typing.Optional[str]

property tracking_uid: Optional[*UID*]

tracking UID

Type
Union[*highdicom.UID*, None]

Return type
typing.Optional[*highdicom.uid.UID*]

property units: *AnnotationUnitsValues*

annotation units

Type
highdicom.pr.AnnotationUnitsValues

Return type
highdicom.pr.enum.AnnotationUnitsValues

8.7 highdicom.seg package

Package for creation of Segmentation (SEG) instances.

class highdicom.seg.**DimensionIndexSequence**(*coordinate_system*)

Bases: Sequence

Sequence of data elements describing dimension indices for the patient or slide coordinate system based on the Dimension Index functional group macro.

Note: The order of indices is fixed.

Parameters
coordinate_system (*Union[str, highdicom.CoordinateSystemNames, None]*) – Subject ("PATIENT" or "SLIDE") that was the target of imaging. If None, the imaging does not belong within a frame of reference.

get_index_keywords()

Get keywords of attributes that specify the position of planes.

Returns
Keywords of indexed attributes

Return type
List[str]

Note: Includes only keywords of indexed attributes that specify the spatial position of planes relative to the total pixel matrix or the frame of reference, and excludes the keyword of the Referenced Segment Number attribute.

Examples

```
>>> dimension_index = DimensionIndexSequence('SLIDE')
>>> plane_positions = [
...     PlanePositionSequence('SLIDE', [10.0, 0.0, 0.0], [1, 1]),
...     PlanePositionSequence('SLIDE', [30.0, 0.0, 0.0], [1, 2]),
...     PlanePositionSequence('SLIDE', [50.0, 0.0, 0.0], [1, 3])
... ]
>>> values, indices = dimension_index.get_index_values(plane_positions)
>>> names = dimension_index.get_index_keywords()
>>> for name in names:
```

(continues on next page)

(continued from previous page)

```
...     print(name)
ColumnPositionInTotalImagePixelMatrix
RowPositionInTotalImagePixelMatrix
XOffsetInSlideCoordinateSystem
YOffsetInSlideCoordinateSystem
ZOffsetInSlideCoordinateSystem
>>> index = names.index("XOffsetInSlideCoordinateSystem")
>>> print(values[:, index])
[10. 30. 50.]
```

get_index_position(*pointer*)

Get relative position of a given dimension in the dimension index.

Parameters

pointer (*str*) – Name of the dimension (keyword of the attribute), e.g., "ReferencedSegmentNumber"

Returns

Zero-based relative position

Return type

int

Examples

```
>>> dimension_index = DimensionIndexSequence("SLIDE")
>>> i = dimension_index.get_index_position("ReferencedSegmentNumber")
>>> dimension_description = dimension_index[i]
>>> dimension_description
(0020, 9164) Dimension Organization UID          ...
(0020, 9165) Dimension Index Pointer             AT: (0062, 000b)
(0020, 9167) Functional Group Pointer            AT: (0062, 000a)
(0020, 9421) Dimension Description Label         LO: 'Segment Number'
```

get_index_values(*plane_positions*)

Get values of indexed attributes that specify position of planes.

Parameters

plane_positions (*Sequence[highdicom.PlanePositionSequence]*) – Plane position of frames in a multi-frame image or in a series of single-frame images

Return type

typing.Tuple[numpy.ndarray, numpy.ndarray]

Returns

- **dimension_index_values** (*numpy.ndarray*) – 2D array of dimension index values
- **plane_indices** (*numpy.ndarray*) – 1D array of planes indices for sorting frames according to their spatial position specified by the dimension index

---

Note: Includes only values of indexed attributes that specify the spatial position of planes relative to the total pixel matrix or the frame of reference, and excludes values of the Referenced Segment Number attribute.

---

get_plane_positions_of_image(*image*)

Gets plane positions of frames in multi-frame image.

Parameters
image (*Dataset*) – Multi-frame image

Returns
Plane position of each frame in the image

Return type
List[*highdicom.PlanePositionSequence*]

get_plane_positions_of_series(*images*)

Gets plane positions for series of single-frame images.

Parameters
images (*Sequence[Dataset]*) – Series of single-frame images

Returns
Plane position of each frame in the image

Return type
List[*highdicom.PlanePositionSequence*]

class highdicom.seg.**SegmentAlgorithmTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Segment Algorithm Type.

AUTOMATIC = 'AUTOMATIC'

MANUAL = 'MANUAL'

SEMIAUTOMATIC = 'SEMIAUTOMATIC'

class highdicom.seg.**SegmentDescription**(*segment_number*, *segment_label*, *segmented_property_category*, *segmented_property_type*, *algorithm_type*, *algorithm_identification=None*, *tracking_uid=None*, *tracking_id=None*, *anatomic_regions=None*, *primary_anatomic_structures=None*)

Bases: Dataset

Dataset describing a segment based on the Segment Description macro.

Parameters

- **segment_number** (*int*) – Number of the segment.
- **segment_label** (*str*) – Label of the segment
- **segmented_property_category** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Category of the property the segment represents, e.g. Code("49755003", "SCT", "Morphologically Abnormal Structure") (see CID 7150 "Segmentation Property Categories")
- **segmented_property_type** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Property the segment represents, e.g. Code("108369006", "SCT", "Neoplasm") (see CID 7151 "Segmentation Property Types")
- **algorithm_type** (*Union[str, highdicom.seg.SegmentAlgorithmTypeValues]*) – Type of algorithm

- **algorithm_identification** (*Union[highdicom.AlgorithmIdentificationSequence, None], optional*) – Information useful for identification of the algorithm, such as its name or version. Required unless the algorithm type is *MANUAL*
- **tracking_uid** (*Union[str, None], optional*) – Unique tracking identifier (universally unique)
- **tracking_id** (*Union[str, None], optional*) – Tracking identifier (unique only with the domain of use)
- **anatomic_regions** (*Union[Sequence[Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]], None], optional*) – Anatomic region(s) into which segment falls, e.g. `Code("41216001", "SCT", "Prostate")` (see CID 4 "Anatomic Region", CID 4031 "Common Anatomic Regions", as as well as other CIDs for domain-specific anatomic regions)
- **primary_anatomic_structures** (*Union[Sequence[Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]], None], optional*) – Anatomic structure(s) the segment represents (see CIDs for domain-specific primary anatomic structures)

Notes

When segment descriptions are passed to a segmentation instance they must have consecutive segment numbers, starting at 1 for the first segment added.

property algorithm_identification: Optional[*AlgorithmIdentificationSequence*]

Union[highdicom.AlgorithmIdentificationSequence, None] Information useful for identification of the algorithm, if any.

Return type
typing.Optional[*highdicom.content.AlgorithmIdentificationSequence*]

property algorithm_type: *SegmentAlgorithmTypeValues*

highdicom.seg.SegmentAlgorithmTypeValues: Type of algorithm used to create the segment.

Return type
highdicom.seg.enum.SegmentAlgorithmTypeValues

property anatomic_regions: List[*CodedConcept*]

List[highdicom.sr.CodedConcept]: List of anatomic regions into which the segment falls. May be empty.

Return type
typing.List[*highdicom.sr.coding.CodedConcept*]

classmethod from_dataset(*dataset*)

Construct instance from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an item of the Segment Sequence.

Returns
Segment description.

Return type
highdicom.seg.SegmentDescription

property primary_anatomic_structures: List[*CodedConcept*]

List[highdicom.sr.CodedConcept]: List of anatomic anatomic structures the segment represents. May be empty.

Return type
typing.List[*highdicom.sr.coding.CodedConcept*]

property segment_label: str

Label of the segment.

Type
str

Return type
str

property segment_number: int

Number of the segment.

Type
int

Return type
int

property segmented_property_category: *CodedConcept*

highdicom.sr.CodedConcept: Category of the property the segment represents.

Return type
highdicom.sr.coding.CodedConcept

property segmented_property_type: *CodedConcept*

highdicom.sr.CodedConcept: Type of the property the segment represents.

Return type
highdicom.sr.coding.CodedConcept

property tracking_id: Optional[str]

Tracking identifier for the segment, if any.

Type
Union[str, None]

Return type
typing.Optional[str]

property tracking_uid: Optional[str]

Union[str, None]: Tracking unique identifier for the segment, if any.

Return type
typing.Optional[str]

class highdicom.seg.**Segmentation**(*source_images*, *pixel_array*, *segmentation_type*, *segment_descriptions*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *fractional_type=SegmentationFractionalTypeValues.PROBABILITY*, *max_fractional_value=255*, *content_description=None*, *content_creator_name=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, *pixel_measures=None*, *plane_orientation=None*, *plane_positions=None*, *omit_empty_frames=True*, *content_label=None*, *content_creator_identification=None*, ***kwargs*)

Bases: *SOPClass*

SOP class for the Segmentation IOD.

Parameters

- **source_images** (*Sequence[Dataset]*) – One or more single- or multi-frame images (or metadata of images) from which the segmentation was derived
- **pixel_array** (*numpy.ndarray*) – Array of segmentation pixel data of boolean, unsigned integer or floating point data type representing a mask image. The array may be a 2D, 3D or 4D numpy array.

  If it is a 2D numpy array, it represents the segmentation of a single frame image, such as a planar x-ray or single instance from a CT or MR series.

  If it is a 3D array, it represents the segmentation of either a series of source images (such as a series of CT or MR images) a single 3D multi-frame image (such as a multi-frame CT/MR image), or a single 2D tiled image (such as a slide microscopy image).

  If `pixel_array` represents the segmentation of a 3D image, the first dimension represents individual 2D planes. Unless the `plane_positions` parameter is provided, the frame in `pixel_array[i, ...]` should correspond to either `source_images[i]` (if `source_images` is a list of single frame instances) or source_images[0].pixel_array[i, ...] if `source_images` is a single multiframe instance.

  Similarly, if `pixel_array` is a 3D array representing the segmentation of a tiled 2D image, the first dimension represents individual 2D tiles (for one channel and z-stack) and these tiles correspond to the frames in the source image dataset.

  If `pixel_array` is an unsigned integer or boolean array with binary data (containing only the values `True` and `False` or `0` and `1`) or a floating-point array, it represents a single segment. In the case of a floating-point array, values must be in the range 0.0 to 1.0.

  Otherwise, if `pixel_array` is a 2D or 3D array containing multiple unsigned integer values, each value is treated as a different segment whose segment number is that integer value. This is referred to as a *label map* style segmentation. In this case, all segments from 1 through `pixel_array.max()` (inclusive) must be described in *segment_descriptions*, regardless of whether they are present in the image. Note that this is valid for segmentations encoded using the `"BINARY"` or `"FRACTIONAL"` methods.

  Note that that a 2D numpy array and a 3D numpy array with a single frame along the first dimension may be used interchangeably as segmentations of a single frame, regardless of their data type.

  If `pixel_array` is a 4D numpy array, the first three dimensions are used in the same way as the 3D case and the fourth dimension represents multiple segments. In this case `pixel_array[:, :, :, i]` represents segment number `i + 1` (since numpy indexing is 0-based but segment numbering is 1-based), and all segments from 1 through `pixel_array.shape[-1] + 1` must be described in `segment_descriptions`.

  Furthermore, a 4D array with unsigned integer data type must contain only binary data (`True` and `False` or `0` and `1`). In other words, a 4D array is incompatible with the *label map* style encoding of the segmentation.

  Where there are multiple segments that are mutually exclusive (do not overlap) and binary, they may be passed using either a *label map* style array or a 4D array. A 4D array is required if either there are multiple segments and they are not mutually exclusive (i.e. they overlap) or there are multiple segments and the segmentation is fractional.

  Note that if the segmentation of a single source image with multiple stacked segments is required, it is necessary to include the singleton first dimension in order to give a 4D array.

  For `"FRACTIONAL"` segmentations, values either encode the probability of a given pixel belonging to a segment (if *fractional_type* is `"PROBABILITY"`) or the extent to which a segment

occupies the pixel (if *fractional_type* is "OCCUPANCY").

- **segmentation_type** (*Union[str, highdicom.seg.SegmentationTypeValues]*) – Type of segmentation, either "BINARY" or "FRACTIONAL"
- **segment_descriptions** (*Sequence[highdicom.seg.SegmentDescription]*) – Description of each segment encoded in *pixel_array*. In the case of pixel arrays with multiple integer values, the segment description with the corresponding segment number is used to describe each segment.
- **series_instance_uid** (*str*) – UID of the series
- **series_number** (*int*) – Number of the series within the study
- **sop_instance_uid** (*str*) – UID that should be assigned to the instance
- **instance_number** (*int*) – Number that should be assigned to the instance
- **manufacturer** (*str*) – Name of the manufacturer of the device (developer of the software) that creates the instance
- **manufacturer_model_name** (*str*) – Name of the device model (name of the software library or application) that creates the instance
- **software_versions** (*Union[str, Tuple[str]]*) – Version(s) of the software that creates the instance
- **device_serial_number** (*str*) – Manufacturer's serial number of the device
- **fractional_type** (*Union[str, highdicom.seg.SegmentationFractionalTypeValues, None], optional*) – Type of fractional segmentation that indicates how pixel data should be interpreted
- **max_fractional_value** (*int, optional*) – Maximum value that indicates probability or occupancy of 1 that a pixel represents a given segment
- **content_description** (*Union[str, None], optional*) – Description of the segmentation
- **content_creator_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the creator of the segmentation (if created manually)
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements. The following lossless compressed transfer syntaxes are supported for encapsulated format encoding in case of FRACTIONAL segmentation type: RLE Lossless ("1.2.840.10008.1.2.5") and JPEG 2000 Lossless ("1.2.840.10008.1.2.4.90").
- **pixel_measures** (*Union[highdicom.PixelMeasures, None], optional*) – Physical spacing of image pixels in *pixel_array*. If None, it will be assumed that the segmentation image has the same pixel measures as the source image(s).
- **plane_orientation** (*Union[highdicom.PlaneOrientationSequence, None], optional*) – Orientation of planes in *pixel_array* relative to axes of three-dimensional patient or slide coordinate space. If None, it will be assumed that the segmentation image as the same plane orientation as the source image(s).
- **plane_positions** (*Union[Sequence[highdicom.PlanePositionSequence], None], optional*) – Position of each plane in *pixel_array* in the three-dimensional patient or slide coordinate space. If None, it will be assumed that the segmentation image has the same plane position as the source image(s). However, this will only work when the first dimension of *pixel_array* matches the number of frames in *source_images* (in case of

multi-frame source images) or the number of *source_images* (in case of single-frame source images).

- **omit_empty_frames** (*bool, optional*) – If True (default), frames with no non-zero pixels are omitted from the segmentation image. If False, all frames are included.
- **content_label** (*Union[str, None], optional*) – Content label
- **content_creator_identification** (*Union[highdicom.ContentCreatorIdentificationCodeSequence, None], optional*) – Identifying information for the person who created the content of this segmentation.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

Raises

ValueError – When

* Length of *source_images* is zero. * Items of *source_images* are not all part of the same study and series. * Items of *source_images* have different number of rows and columns. * Length of *plane_positions* does not match number of segments encoded in *pixel_array*. * Length of *plane_positions* does not match number of 2D planes in *pixel_array* (size of first array dimension).

Note: The assumption is made that segments in *pixel_array* are defined in the same frame of reference as *source_images*.

---

add_segments(*pixel_array*, *segment_descriptions*, *plane_positions=None*, *omit_empty_frames=True*)

To ensure correctness of segmentation images, this method was deprecated in highdicom 0.8.0. For more information and migration instructions see *here*.

Return type

None

are_dimension_indices_unique(*dimension_index_pointers*)

Check if a list of index pointers uniquely identifies frames.

For a given list of dimension index pointers, check whether every combination of index values for these pointers identifies a unique frame per segment in the segmentation image. This is a pre-requisite for indexing using this list of dimension index pointers in the *Segmentation.get_pixels_by_dimension_index_values()* method.

Parameters

dimension_index_pointers (*Sequence[Union[int, pydicom.tag.BaseTag]]*) – Sequence of tags serving as dimension index pointers.

Returns

True if the specified list of dimension index pointers uniquely identifies frames in the segmentation image. False otherwise.

Return type

bool

Raises

KeyError – If any of the elements of the dimension_index_pointers are not valid dimension index pointers in this segmentation image.

classmethod from_dataset(*dataset*)

Create instance from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing a Segmentation image.

Returns
Representation of the supplied dataset as a highdicom Segmentation.

Return type
highdicom.seg.Segmentation

`get_default_dimension_index_pointers()`

Get the default list of tags used to index frames.

The list of tags used to index dimensions depends upon how the segmentation image was constructed, and is stored in the DimensionIndexPointer attribute within the DimensionIndexSequence. The list returned by this method matches the order of items in the DimensionIndexSequence, but omits the ReferencedSegmentNumber attribute, since this is handled differently to other tags when indexing frames in highdicom.

Returns
List of tags used as the default dimension index pointers.

Return type
List[pydicom.tag.BaseTag]

`get_pixels_by_dimension_index_values(`*dimension_index_values*, *dimension_index_pointers=None*, *segment_numbers=None*, *combine_segments=False*, *relabel=False*, *assert_missing_frames_are_empty=False*, *rescale_fractional=True*`)`

Get a pixel array for a list of dimension index values.

This is intended for retrieving segmentation masks using the index values within the segmentation object, without referring to the source images from which the segmentation was derived.

The output array will have 4 dimensions under the default behavior, and 3 dimensions if `combine_segments` is set to `True`. The first dimension represents the source frames. `pixel_array[i, ...]` represents the segmentation frame with index `dimension_index_values[i]`. The next two dimensions are the rows and columns of the frames, respectively.

When `combine_segments` is `False` (the default behavior), the segments are stacked down the final (4th) dimension of the pixel array. If `segment_numbers` was specified, then `pixel_array[:, :, :, i]` represents the data for segment `segment_numbers[i]`. If `segment_numbers` was unspecified, then `pixel_array[:, :, :, i]` represents the data for segment `parser.segment_numbers[i]`. Note that in neither case does `pixel_array[:, :, :, i]` represent the segmentation data for the segment with segment number `i`, since segment numbers begin at 1 in DICOM.

When `combine_segments` is `True`, then the segmentation data from all specified segments is combined into a multi-class array in which pixel value is used to denote the segment to which a pixel belongs. This is only possible if the segments do not overlap and either the type of the segmentation is `BINARY` or the type of the segmentation is `FRACTIONAL` but all values are exactly 0.0 or 1.0. the segments do not overlap. If the segments do overlap, a `RuntimeError` will be raised. After combining, the value of a pixel depends upon the `relabel` parameter. In both cases, pixels that appear in no segments with have a value of `0`. If `relabel` is `False`, a pixel that appears in the segment with segment number `i` (according to the original segment numbering of the segmentation object) will have a value of `i`. If `relabel` is `True`, the value of a pixel in segment `i` is related not to the original segment number, but to the index of that segment number in the `segment_numbers` parameter of this method. Specifically, pixels belonging to the segment with segment number `segment_numbers[i]` is given the value `i + 1` in the output pixel array (since 0 is reserved for pixels that belong to no segments). In this case, the values in the output pixel array will always lie in the range `0` to `len(segment_numbers)` inclusive.

Parameters

- **dimension_index_values** (*Sequence[Sequence[int]]*) – Dimension index values for the requested frames. Each element of the sequence is a sequence of 1-based index values representing the dimension index values for a single frame of the output segmentation. The order of the index values within the inner sequence is determined by the `dimension_index_pointers` parameter, and as such the length of each inner sequence must match the length of `dimension_index_pointers` parameter.
- **dimension_index_pointers** (*Union[Sequence[Union[int, pydicom.tag.BaseTag]], None], optional*) – The data element tags that identify the indices used in the `dimension_index_values` parameter. Each element identifies a data element tag by which frames are ordered in the segmentation image dataset. If this parameter is set to `None` (the default), the value of *Segmentation.get_default_dimension_index_pointers()* is used. Valid values of this parameter are are determined by the construction of the segmentation image and include any permutation of any subset of elements in the *Segmentation.get_default_dimension_index_pointers()* list.
- **segment_numbers** (*Union[Sequence[int], None], optional*) – Sequence containing segment numbers to include. If unspecified, all segments are included.
- **combine_segments** (*bool, optional*) – If True, combine the different segments into a single label map in which the value of a pixel represents its segment. If False (the default), segments are binary and stacked down the last dimension of the output array.
- **relabel** (*bool, optional*) – If True and `combine_segments` is `True`, the pixel values in the output array are relabelled into the range `0` to `len(segment_numbers)` (inclusive) accoring to the position of the original segment numbers in `segment_numbers` parameter. If `combine_segments` is `False`, this has no effect.
- **assert_missing_frames_are_empty** (*bool, optional*) – Assert that requested source frame numbers that are not referenced by the segmentation image contain no segments. If a source frame number is not referenced by the segmentation image, highdicom is unable to check that the frame number is valid in the source image. By default, highdicom will raise an error if any of the requested source frames are not referenced in the source image. To override this behavior and return a segmentation frame of all zeros for such frames, set this parameter to True.
- **rescale_fractional** (*bool, optional*) – If this is a FRACTIONAL segmentation and `rescale_fractional` is True, the raw integer-valued array stored in the segmentation image output will be rescaled by the MaximumFractionalValue such that each pixel lies in the range 0.0 to 1.0. If False, the raw integer values are returned. If the segmentation has BINARY type, this parameter has no effect.

Returns

pixel_array – Pixel array representing the segmentation. See notes for full explanation.

Return type

np.ndarray

Examples

Read a test image of a segmentation of a slide microscopy image

```
>>> import highdicom as hd
>>> from pydicom.datadict import keyword_for_tag, tag_for_keyword
>>> from pydicom import dcmread
>>>
>>> ds = dcmread('data/test_files/seg_image_sm_control.dcm')
>>> seg = hd.seg.Segmentation.from_dataset(ds)
```

Get the default list of dimension index values

```
>>> for tag in seg.get_default_dimension_index_pointers():
...     print(keyword_for_tag(tag))
ColumnPositionInTotalImagePixelMatrix
RowPositionInTotalImagePixelMatrix
XOffsetInSlideCoordinateSystem
YOffsetInSlideCoordinateSystem
ZOffsetInSlideCoordinateSystem
```

Use a subset of these index pointers to index the image

```
>>> tags = [
...     tag_for_keyword('ColumnPositionInTotalImagePixelMatrix'),
...     tag_for_keyword('RowPositionInTotalImagePixelMatrix')
... ]
>>> assert seg.are_dimension_indices_unique(tags)  # True
```

It is therefore possible to index using just this subset of dimension indices

```
>>> pixels = seg.get_pixels_by_dimension_index_values(
...     dimension_index_pointers=tags,
...     dimension_index_values=[[1, 1], [1, 2]]
... )
>>> pixels.shape
(2, 10, 10, 20)
```

get_pixels_by_source_frame(*source_sop_instance_uid*, *source_frame_numbers*, *segment_numbers=None*, *combine_segments=False*, *relabel=False*, *ignore_spatial_locations=False*, *assert_missing_frames_are_empty=False*, *rescale_fractional=True*)

Get a pixel array for a list of frames within a source instance.

This is intended for retrieving segmentation masks derived from multi-frame (enhanced) source images. All source frames for which segmentations are requested must belong within the same SOP Instance UID.

The output array will have 4 dimensions under the default behavior, and 3 dimensions if `combine_segments` is set to `True`. The first dimension represents the source frames. `pixel_array[i, ...]` represents the segmentation of `source_frame_numbers[i]`. The next two dimensions are the rows and columns of the frames, respectively.

When `combine_segments` is `False` (the default behavior), the segments are stacked down the final (4th) dimension of the pixel array. If `segment_numbers` was specified, then `pixel_array[:, :, :, i]` represents the data for segment `segment_numbers[i]`. If `segment_numbers` was unspecified, then `pixel_array[:, :, :, i]` represents the data for segment `parser.segment_numbers[i]`. Note

that in neither case does `pixel_array[:, :, :, i]` represent the segmentation data for the segment with segment number `i`, since segment numbers begin at 1 in DICOM.

When `combine_segments` is `True`, then the segmentation data from all specified segments is combined into a multi-class array in which pixel value is used to denote the segment to which a pixel belongs. This is only possible if the segments do not overlap and either the type of the segmentation is `BINARY` or the type of the segmentation is `FRACTIONAL` but all values are exactly 0.0 or 1.0. the segments do not overlap. If the segments do overlap, a `RuntimeError` will be raised. After combining, the value of a pixel depends upon the `relabel` parameter. In both cases, pixels that appear in no segments with have a value of `0`. If `relabel` is `False`, a pixel that appears in the segment with segment number `i` (according to the original segment numbering of the segmentation object) will have a value of `i`. If `relabel` is `True`, the value of a pixel in segment `i` is related not to the original segment number, but to the index of that segment number in the `segment_numbers` parameter of this method. Specifically, pixels belonging to the segment with segment number `segment_numbers[i]` is given the value `i + 1` in the output pixel array (since 0 is reserved for pixels that belong to no segments). In this case, the values in the output pixel array will always lie in the range `0` to `len(segment_numbers)` inclusive.

Parameters

- **source_sop_instance_uid** (*str*) – SOP Instance UID of the source instance that contains the source frames.
- **source_frame_numbers** (*Sequence[int]*) – A sequence of frame numbers (1-based) within the source instance for which segmentations are requested.
- **segment_numbers** (*Sequence[int, None], optional*) – Sequence containing segment numbers to include. If unspecified, all segments are included.
- **combine_segments** (*bool, optional*) – If True, combine the different segments into a single label map in which the value of a pixel represents its segment. If False (the default), segments are binary and stacked down the last dimension of the output array.
- **relabel** (*bool, optional*) – If True and `combine_segments` is `True`, the pixel values in the output array are relabelled into the range `0` to `len(segment_numbers)` (inclusive) accoring to the position of the original segment numbers in `segment_numbers` parameter. If `combine_segments` is `False`, this has no effect.
- **ignore_spatial_locations** (*bool, optional*) – Ignore whether or not spatial locations were preserved in the derivation of the segmentation frames from the source frames. In some segmentation images, the pixel locations in the segmentation frames may not correspond to pixel locations in the frames of the source image from which they were derived. The segmentation image may or may not specify whether or not spatial locations are preserved in this way through use of the optional (0028,135A) SpatialLocationsPreserved attribute. If this attribute specifies that spatial locations are not preserved, or is absent from the segmentation image, highdicom's default behavior is to disallow indexing by source frames. To override this behavior and retrieve segmentation pixels regardless of the presence or value of the spatial locations preserved attribute, set this parameter to True.
- **assert_missing_frames_are_empty** (*bool, optional*) – Assert that requested source frame numbers that are not referenced by the segmentation image contain no segments. If a source frame number is not referenced by the segmentation image, highdicom is unable to check that the frame number is valid in the source image. By default, highdicom will raise an error if any of the requested source frames are not referenced in the source image. To override this behavior and return a segmentation frame of all zeros for such frames, set this parameter to True.
- **rescale_fractional** (*bool*) – If this is a FRACTIONAL segmentation and `rescale_fractional` is True, the raw integer-valued array stored in the segmentation image output will be rescaled by the MaximumFractionalValue such that each pixel lies in

the range 0.0 to 1.0. If False, the raw integer values are returned. If the segmentation has BINARY type, this parameter has no effect.

Returns

pixel_array – Pixel array representing the segmentation. See notes for full explanation.

Return type

np.ndarray

Examples

Read in an example from the highdicom test data derived from a multiframe slide microscopy image:

```
>>> import highdicom as hd
>>>
>>> seg = hd.seg.segread('data/test_files/seg_image_sm_control.dcm')
```

List the source image SOP instance UID for this segmentation:

```
>>> sop_uid = seg.get_source_image_uids()[0][2]
>>> sop_uid
'1.2.826.0.1.3680043.9.7433.3.128575161848499511430445138772822227'
```

Get the segmentation array for 3 of the frames in the multiframe source image. The resulting segmentation array has 3 10 x 10 frames, one for each source frame. The final dimension contains the 20 different segments present in this segmentation.

```
>>> pixels = seg.get_pixels_by_source_frame(
...     source_sop_instance_uid=sop_uid,
...     source_frame_numbers=[4, 5, 6]
... )
>>> pixels.shape
(3, 10, 10, 20)
```

This time, select only 4 of the 20 segments:

```
>>> pixels = seg.get_pixels_by_source_frame(
...     source_sop_instance_uid=sop_uid,
...     source_frame_numbers=[4, 5, 6],
...     segment_numbers=[10, 11, 12, 13]
... )
>>> pixels.shape
(3, 10, 10, 4)
```

Instead create a multiclass label map for each source frame. Note that segments 6, 8, and 10 are present in the three chosen frames.

```
>>> pixels = seg.get_pixels_by_source_frame(
...     source_sop_instance_uid=sop_uid,
...     source_frame_numbers=[4, 5, 6],
...     combine_segments=True
... )
>>> pixels.shape, np.unique(pixels)
((3, 10, 10), array([ 0,  6,  8, 10]))
```

Now relabel the segments to give a pixel map with values between 0 and 3 (inclusive):

```
>>> pixels = seg.get_pixels_by_source_frame(
...     source_sop_instance_uid=sop_uid,
...     source_frame_numbers=[4, 5, 6],
...     segment_numbers=[6, 8, 10],
...     combine_segments=True,
...     relabel=True
... )
>>> pixels.shape, np.unique(pixels)
((3, 10, 10), array([0, 1, 2, 3]))
```

get_pixels_by_source_instance(*source_sop_instance_uids*, *segment_numbers=None*, *combine_segments=False*, *relabel=False*, *ignore_spatial_locations=False*, *assert_missing_frames_are_empty=False*, *rescale_fractional=True*)

Get a pixel array for a list of source instances.

This is intended for retrieving segmentation masks derived from (series of) single frame source images.

The output array will have 4 dimensions under the default behavior, and 3 dimensions if `combine_segments` is set to `True`. The first dimension represents the source instances. `pixel_array[i, ...]` represents the segmentation of `source_sop_instance_uids[i]`. The next two dimensions are the rows and columns of the frames, respectively.

When `combine_segments` is `False` (the default behavior), the segments are stacked down the final (4th) dimension of the pixel array. If `segment_numbers` was specified, then `pixel_array[:, :, :, i]` represents the data for segment `segment_numbers[i]`. If `segment_numbers` was unspecified, then `pixel_array[:, :, :, i]` represents the data for segment `parser.segment_numbers[i]`. Note that in neither case does `pixel_array[:, :, :, i]` represent the segmentation data for the segment with segment number `i`, since segment numbers begin at 1 in DICOM.

When `combine_segments` is `True`, then the segmentation data from all specified segments is combined into a multi-class array in which pixel value is used to denote the segment to which a pixel belongs. This is only possible if the segments do not overlap and either the type of the segmentation is `BINARY` or the type of the segmentation is `FRACTIONAL` but all values are exactly 0.0 or 1.0. the segments do not overlap. If the segments do overlap, a `RuntimeError` will be raised. After combining, the value of a pixel depends upon the `relabel` parameter. In both cases, pixels that appear in no segments with have a value of `0`. If `relabel` is `False`, a pixel that appears in the segment with segment number `i` (according to the original segment numbering of the segmentation object) will have a value of `i`. If `relabel` is `True`, the value of a pixel in segment `i` is related not to the original segment number, but to the index of that segment number in the `segment_numbers` parameter of this method. Specifically, pixels belonging to the segment with segment number `segment_numbers[i]` is given the value `i + 1` in the output pixel array (since 0 is reserved for pixels that belong to no segments). In this case, the values in the output pixel array will always lie in the range `0` to `len(segment_numbers)` inclusive.

Parameters

- **source_sop_instance_uids** (*str*) – SOP Instance UID of the source instances to for which segmentations are requested.
- **segment_numbers** (*Union[Sequence[int], None], optional*) – Sequence containing segment numbers to include. If unspecified, all segments are included.
- **combine_segments** (*bool, optional*) – If True, combine the different segments into a single label map in which the value of a pixel represents its segment. If False (the default), segments are binary and stacked down the last dimension of the output array.

- **relabel** (*bool, optional*) – If True and `combine_segments` is `True`, the pixel values in the output array are relabelled into the range `0` to `len(segment_numbers)` (inclusive) accoring to the position of the original segment numbers in `segment_numbers` parameter. If `combine_segments` is `False`, this has no effect.
- **ignore_spatial_locations** (*bool, optional*) – Ignore whether or not spatial locations were preserved in the derivation of the segmentation frames from the source frames. In some segmentation images, the pixel locations in the segmentation frames may not correspond to pixel locations in the frames of the source image from which they were derived. The segmentation image may or may not specify whether or not spatial locations are preserved in this way through use of the optional (0028,135A) SpatialLocationsPreserved attribute. If this attribute specifies that spatial locations are not preserved, or is absent from the segmentation image, highdicom's default behavior is to disallow indexing by source frames. To override this behavior and retrieve segmentation pixels regardless of the presence or value of the spatial locations preserved attribute, set this parameter to True.
- **assert_missing_frames_are_empty** (*bool, optional*) – Assert that requested source frame numbers that are not referenced by the segmentation image contain no segments. If a source frame number is not referenced by the segmentation image, highdicom is unable to check that the frame number is valid in the source image. By default, highdicom will raise an error if any of the requested source frames are not referenced in the source image. To override this behavior and return a segmentation frame of all zeros for such frames, set this parameter to True.
- **rescale_fractional** (*bool, optional*) – If this is a FRACTIONAL segmentation and `rescale_fractional` is True, the raw integer-valued array stored in the segmentation image output will be rescaled by the MaximumFractionalValue such that each pixel lies in the range 0.0 to 1.0. If False, the raw integer values are returned. If the segmentation has BINARY type, this parameter has no effect.

Returns

pixel_array – Pixel array representing the segmentation. See notes for full explanation.

Return type

np.ndarray

Examples

Read in an example from the highdicom test data:

```
>>> import highdicom as hd
>>>
>>> seg = hd.seg.segread('data/test_files/seg_image_ct_binary.dcm')
```

List the source images for this segmentation:

```
>>> for study_uid, series_uid, sop_uid in seg.get_source_image_uids():
...     print(sop_uid)
1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.93
1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.94
1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.95
1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.96
```

Get the segmentation array for a subset of these images:

```
>>> pixels = seg.get_pixels_by_source_instance(
...     source_sop_instance_uids=[
...         '1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.93',
...         '1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.94'
...     ]
... )
>>> pixels.shape
(2, 16, 16, 1)
```

get_segment_description(*segment_number*)

Get segment description for a segment.

Parameters

segment_number (*int*) – Segment number for the segment, as a 1-based index.

Returns

Description of the given segment.

Return type

highdicom.seg.SegmentDescription

get_segment_numbers(*segment_label=None*, *segmented_property_category=None*, *segmented_property_type=None*, *algorithm_type=None*, *tracking_uid=None*, *tracking_id=None*)

Get a list of segment numbers matching provided criteria.

Any number of optional filters may be provided. A segment must match all provided filters to be included in the returned list.

Parameters

- **segment_label** (*Union[str, None], optional*) – Segment label filter to apply.
- **segmented_property_category** (*Union[Code, CodedConcept, None], optional*) – Segmented property category filter to apply.
- **segmented_property_type** (*Union[Code, CodedConcept, None], optional*) – Segmented property type filter to apply.
- **algorithm_type** (*Union[SegmentAlgorithmTypeValues, str, None], optional*) – Segmented property type filter to apply.
- **tracking_uid** (*Union[str, None], optional*) – Tracking unique identifier filter to apply.
- **tracking_id** (*Union[str, None], optional*) – Tracking identifier filter to apply.

Returns

List of all segment numbers matching the provided criteria.

Return type

List[int]

Examples

Get segment numbers of all segments that both represent tumors and were generated by an automatic algorithm from a segmentation object `seg`:

```
>>> from pydicom.sr.codedict import codes
>>> from highdicom.seg import SegmentAlgorithmTypeValues, Segmentation
>>> from pydicom import dcmread
>>> ds = dcmread('data/test_files/seg_image_sm_control.dcm')
>>> seg = Segmentation.from_dataset(ds)
>>> segment_numbers = seg.get_segment_numbers(
...     segmented_property_type=codes.SCT.ConnectiveTissue,
...     algorithm_type=SegmentAlgorithmTypeValues.AUTOMATIC
... )
>>> segment_numbers
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Get segment numbers of all segments identified by a given institution-specific tracking ID:

```
>>> segment_numbers = seg.get_segment_numbers(
...     tracking_id='Segment #4'
... )
>>> segment_numbers
[4]
```

Get segment numbers of all segments identified a globally unique tracking UID:

```
>>> uid = '1.2.826.0.1.3680043.8.498.42540123542017542395135803252098380233'
>>> segment_numbers = seg.get_segment_numbers(tracking_uid=uid)
>>> segment_numbers
[13]
```

get_source_image_uids()

Get UIDs for all source SOP instances referenced in the dataset.

Returns

List of tuples containing Study Instance UID, Series Instance UID and SOP Instance UID for every SOP Instance referenced in the dataset.

Return type

List[Tuple[*highdicom.UID*, *highdicom.UID*, *highdicom.UID*]]

get_tracking_ids(*segmented_property_category=None*, *segmented_property_type=None*, *algorithm_type=None*)

Get all unique tracking identifiers in this SEG image.

Any number of optional filters may be provided. A segment must match all provided filters to be included in the returned list.

The tracking IDs and the accompanying tracking UIDs are returned in a list of tuples.

Note that the order of the returned list is not significant and will not in general match the order of segments.

Parameters

- **segmented_property_category** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Segmented property category filter to apply.

- **segmented_property_type** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Segmented property type filter to apply.
- **algorithm_type** (*Union[highdicom.seg.SegmentAlgorithmTypeValues, str, None], optional*) – Segmented property type filter to apply.

Returns
List of all unique (Tracking Identifier, Unique Tracking Identifier) tuples that are referenced in segment descriptions in this Segmentation image that match all provided filters.

Return type
List[Tuple[str, pydicom.uid.UID]]

Examples

Read in an example segmentation image in the highdicom test data:

```
>>> import highdicom as hd
>>> from pydicom.sr.codedict import codes
>>>
>>> seg = hd.seg.segread('data/test_files/seg_image_ct_binary_overlap.dcm')
```

List the tracking IDs and UIDs present in the segmentation image:

```
>>> sorted(seg.get_tracking_ids(), reverse=True)  # otherwise its a random order
[('Spine', '1.2.826.0.1.3680043.10.511.3.10042414969629429693880339016394772'),␣
↪('Bone', '1.2.826.0.1.3680043.10.511.3.83271046815894549094043330632275067')]
```

```
>>> for seg_num in seg.segment_numbers:
...     desc = seg.get_segment_description(seg_num)
...     print(desc.segmented_property_type.meaning)
Bone
Spine
```

List tracking IDs only for those segments with a segmented property category of 'Spine':

```
>>> seg.get_tracking_ids(segmented_property_type=codes.SCT.Spine)
[('Spine', '1.2.826.0.1.3680043.10.511.3.10042414969629429693880339016394772')]
```

iter_segments()

Iterates over segments in this segmentation image.

Returns
For each segment in the Segmentation image instance, provides the Pixel Data frames representing the segment, items of the Per-Frame Functional Groups Sequence describing the individual frames, and the item of the Segment Sequence describing the segment

Return type
Iterator[Tuple[numpy.ndarray, Tuple[pydicom.dataset.Dataset, ...], pydicom.dataset.Dataset]]

property number_of_segments: int

The number of segments in this SEG image.

Type
int

Return type
int

property segment_numbers: range

The segment numbers present in the SEG image as a range.

Type
range

Return type
range

property segmentation_fractional_type: Optional[*SegmentationFractionalTypeValues*]

highdicom.seg.SegmentationFractionalTypeValues: Segmentation fractional type.

Return type
typing.Optional[*highdicom.seg.enum.SegmentationFractionalTypeValues*]

property segmentation_type: *SegmentationTypeValues*

Segmentation type.

Type
highdicom.seg.SegmentationTypeValues

Return type
highdicom.seg.enum.SegmentationTypeValues

property segmented_property_categories: List[*CodedConcept*]

Get all unique segmented property categories in this SEG image.

Returns
All unique segmented property categories referenced in segment descriptions in this SEG image.

Return type
List[*CodedConcept*]

property segmented_property_types: List[*CodedConcept*]

Get all unique segmented property types in this SEG image.

Returns
All unique segmented property types referenced in segment descriptions in this SEG image.

Return type
List[*CodedConcept*]

class highdicom.seg.SegmentationFractionalTypeValues(*value*)

Bases: Enum

Enumerated values for attribute Segmentation Fractional Type.

OCCUPANCY = 'OCCUPANCY'

PROBABILITY = 'PROBABILITY'

class highdicom.seg.SegmentationTypeValues(*value*)

Bases: Enum

Enumerated values for attribute Segmentation Type.

BINARY = 'BINARY'

FRACTIONAL = 'FRACTIONAL'

class highdicom.seg.**SegmentsOverlapValues**(*value*)

Bases: Enum

Enumerated values for attribute Segments Overlap.

NO = 'NO'

UNDEFINED = 'UNDEFINED'

YES = 'YES'

class highdicom.seg.**SpatialLocationsPreservedValues**(*value*)

Bases: Enum

Enumerated values for attribute Spatial Locations Preserved.

NO = 'NO'

REORIENTED_ONLY = 'REORIENTED_ONLY'

A projection radiograph that has been flipped, and/or rotated by a multiple of 90 degrees.

YES = 'YES'

highdicom.seg.**segread**(*fp*)

Read a segmentation image stored in DICOM File Format.

Parameters
fp (*Union[str, bytes, os.PathLike]*) – Any file-like object representing a DICOM file containing a Segmentation image.

Returns
Segmentation image read from the file.

Return type
highdicom.seg.Segmentation

8.7.1 highdicom.seg.utils module

Utilities for working with SEG image instances.

highdicom.seg.utils.**iter_segments**(*dataset*)

Iterates over segments of a Segmentation image instance.

Parameters
dataset (*pydicom.dataset.Dataset*) – Segmentation image instance

Returns
For each segment in the Segmentation image instance, provides the Pixel Data frames representing the segment, items of the Per-Frame Functional Groups Sequence describing the individual frames, and the item of the Segment Sequence describing the segment

Return type
Iterator[Tuple[numpy.ndarray, Tuple[pydicom.dataset.Dataset, ...], pydicom.dataset.Dataset]]

Raises
AttributeError – When data set does not contain Content Sequence attribute.

8.8 highdicom.sr package

Package for creationg of Structured Report (SR) instances.

class highdicom.sr.**AlgorithmIdentification**(*name*, *version*, *parameters=None*)

Bases: Template

TID 4019 Algorithm Identification

Parameters

- **name** (*str*) – name of the algorithm
- **version** (*str*) – version of the algorithm
- **parameters** (*Union[Sequence[str], None], optional*) – parameters of the algorithm

class highdicom.sr.**CodeContentItem**(*name*, *value*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type CODE.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **value** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Coded value or an enumerated item representing a coded value
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type CODE

Returns
Content Item

Return type
highdicom.sr.CodeContentItem

property value: *CodedConcept*

coded concept

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

class highdicom.sr.**CodedConcept**(*value*, *scheme_designator*, *meaning*, *scheme_version=None*)

Bases: Dataset

Coded concept of a DICOM SR document content module attribute.

Parameters

- **value** (*str*) – code

- **scheme_designator** (*str*) – designator of coding scheme
- **meaning** (*str*) – meaning of the code
- **scheme_version** (*Union[str, None], optional*) – version of coding scheme

classmethod from_code(*code*)

Construct a CodedConcept for a pydicom Code.

Parameters
code (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Code.

Returns
CodedConcept dataset for the code.

Return type
highdicom.sr.CodedConcept

classmethod from_dataset(*dataset*)

Construct a CodedConcept from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing a coded concept.

Returns
Coded concept representation of the dataset.

Return type
highdicom.sr.CodedConcept

Raises

- **TypeError:** – If the passed dataset is not a pydicom dataset.
- **AttributeError:** – If the dataset does not contain the required elements for a coded concept.

property meaning: str

meaning of the code

Type
str

Return type
str

property scheme_designator: str

designator of the coding scheme (e.g. "DCM")

Type
str

Return type
str

property scheme_version: Optional[str]

version of the coding scheme (if specified)

Type
Union[str, None]

Return type
typing.Optional[str]

property value: str

value of either *CodeValue*, *LongCodeValue* or *URNCodeValue* attribute

Type
str

Return type
str

class highdicom.sr.CompositeContentItem(*name*, *referenced_sop_class_uid*, *referenced_sop_instance_uid*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type COMPOSITE.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **referenced_sop_class_uid** (*Union[highdicom.UID, str]*) – SOP Class UID of the referenced object
- **referenced_sop_instance_uid** (*Union[highdicom.UID, str]*) – SOP Instance UID of the referenced object
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type COMPOSITE

Returns
Content Item

Return type
highdicom.sr.CompositeContentItem

property referenced_sop_class_uid: *UID*

referenced SOP Class UID

Type
highdicom.UID

Return type
highdicom.uid.UID

property referenced_sop_instance_uid: *UID*

referenced SOP Instance UID

Type
highdicom.UID

Return type
highdicom.uid.UID

property value: Tuple[*UID*, *UID*]

Tuple[highdicom.UID, highdicom.UID]: referenced SOP Class UID and SOP Instance UID

Return type
typing.Tuple[*highdicom.uid.UID*, *highdicom.uid.UID*]

class highdicom.sr.**Comprehensive3DSR**(*evidence*, *content*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer=None*, *is_complete=False*, *is_final=False*, *is_verified=False*, *institution_name=None*, *institutional_department_name=None*, *verifying_observer_name=None*, *verifying_organization=None*, *performed_procedure_codes=None*, *requested_procedures=None*, *previous_versions=None*, *record_evidence=True*, ***kwargs*)

Bases: _SR

SOP class for a Comprehensive 3D Structured Report (SR) document, whose content may include textual and a variety of coded information, numeric measurement values, references to SOP Instances, as well as 2D or 3D spatial or temporal regions of interest within such SOP Instances.

Parameters

- **evidence** (*Sequence[pydicom.dataset.Dataset]*) – Instances that are referenced in the content tree and from which the created SR document instance should inherit patient and study information
- **content** (*pydicom.dataset.Dataset*) – Root container content items that should be included in the SR document
- **series_instance_uid** (*str*) – Series Instance UID of the SR document series
- **series_number** (*int*) – Series Number of the SR document series
- **sop_instance_uid** (*str*) – SOP instance UID that should be assigned to the SR document instance
- **instance_number** (*int*) – Number that should be assigned to this SR document instance
- **manufacturer** (*str, optional*) – Name of the manufacturer of the device that creates the SR document instance (in a research setting this is typically the same as *institution_name*)
- **is_complete** (*bool, optional*) – Whether the content is complete (default: False)
- **is_final** (*bool, optional*) – Whether the report is the definitive means of communicating the findings (default: False)
- **is_verified** (*bool, optional*) – Whether the report has been verified by an observer accountable for its content (default: False)
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance
- **verifying_observer_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the person that verified the SR document (required if *is_verified*)
- **verifying_organization** (*Union[str, None], optional*) – Name of the organization that verified the SR document (required if *is_verified*)
- **performed_procedure_codes** (*Union[List[highdicom.sr.CodedConcept], None], optional*) – Codes of the performed procedures that resulted in the SR document

- **requested_procedures** (*Union[List[pydicom.dataset.Dataset], None], optional*) – Requested procedures that are being fullfilled by creation of the SR document
- **previous_versions** (*Union[List[pydicom.dataset.Dataset], None], optional*) – Instances representing previous versions of the SR document
- **record_evidence** (*bool, optional*) – Whether provided *evidence* should be recorded (i.e. included in Pertinent Other Evidence Sequence) even if not referenced by content items in the document tree (default: `True`)
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

Note: Each dataset in *evidence* must be part of the same study.

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing a Comprehensive 3D SR document

Returns
Comprehensive 3D SR document

Return type
highdicom.sr.Comprehensive3DSR

class highdicom.sr.**ComprehensiveSR**(*evidence*, *content*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer=None*, *is_complete=False*, *is_final=False*, *is_verified=False*, *institution_name=None*, *institutional_department_name=None*, *verifying_observer_name=None*, *verifying_organization=None*, *performed_procedure_codes=None*, *requested_procedures=None*, *previous_versions=None*, *record_evidence=True*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: _SR

SOP class for a Comprehensive Structured Report (SR) document, whose content may include textual and a variety of coded information, numeric measurement values, references to SOP Instances, as well as 2D spatial or temporal regions of interest within such SOP Instances.

Parameters

- **evidence** (*Sequence[pydicom.dataset.Dataset]*) – Instances that are referenced in the content tree and from which the created SR document instance should inherit patient and study information
- **content** (*pydicom.dataset.Dataset*) – Root container content items that should be included in the SR document
- **series_instance_uid** (*str*) – Series Instance UID of the SR document series
- **series_number** (*int*) – Series Number of the SR document series
- **sop_instance_uid** (*str*) – SOP Instance UID that should be assigned to the SR document instance

- **instance_number** (*int*) – Number that should be assigned to this SR document instance
- **manufacturer** (*str, optional*) – Name of the manufacturer of the device that creates the SR document instance (in a research setting this is typically the same as *institution_name*)
- **is_complete** (*bool, optional*) – Whether the content is complete (default: `False`)
- **is_final** (*bool, optional*) – Whether the report is the definitive means of communicating the findings (default: `False`)
- **is_verified** (*bool, optional*) – Whether the report has been verified by an observer accountable for its content (default: `False`)
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance
- **verifying_observer_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the person that verified the SR document (required if *is_verified*)
- **verifying_organization** (*Union[str, None], optional*) – Name of the organization that verified the SR document (required if *is_verified*)
- **performed_procedure_codes** (*Union[List[highdicom.sr.CodedConcept], None], optional*) – Codes of the performed procedures that resulted in the SR document
- **requested_procedures** (*Union[List[pydicom.dataset.Dataset], None], optional*) – Requested procedures that are being fullfilled by creation of the SR document
- **previous_versions** (*Union[List[pydicom.dataset.Dataset], None], optional*) – Instances representing previous versions of the SR document
- **record_evidence** (*bool, optional*) – Whether provided *evidence* should be recorded (i.e. included in Pertinent Other Evidence Sequence) even if not referenced by content items in the document tree (default: `True`)
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

Note: Each dataset in *evidence* must be part of the same study.

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing a Comprehensive SR document

Returns
Comprehensive SR document

Return type
highdicom.sr.ComprehensiveSR

class highdicom.sr.**ContainerContentItem**(*name*, *is_content_continuous=True*, *template_id=None*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type CONTAINER.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – concept name
- **is_content_continous** (*bool, optional*) – whether contained content items are logically linked in a continuous manner or separate items (default: True)
- **template_id** (*Union[str, None], optional*) – SR template identifier
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – type of relationship with parent content item.

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type CONTAINER

Returns
Content Item

Return type
highdicom.sr.ContainerContentItem

property template_id: Optional[str]

template identifier

Type
Union[str, None]

Return type
typing.Optional[str]

class highdicom.sr.**ContentItem**(*value_type*, *name*, *relationship_type*)

Bases: Dataset

Abstract base class for a collection of attributes contained in the DICOM SR Document Content Module.

Parameters

- **value_type** (*Union[str, highdicom.sr.ValueTypeValues]*) – type of value encoded in a content item
- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – coded name or an enumerated item representing a coded name
- **relationship_type** (*Union[str, highdicom.sr.RelationshipTypeValues], optional*) – type of relationship with parent content item

property name: *CodedConcept*

coded name of the content item

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property relationship_type: Optional[*RelationshipTypeValues*]

type of relationship the content item has with its parent (see *highdicom.sr.RelationshipTypeValues*)

Type
RelationshipTypeValues

Return type
typing.Optional[*highdicom.sr.enum.RelationshipTypeValues*]

property value_type: *ValueTypeValues*

type of the content item (see *highdicom.sr.ValueTypeValues*)

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

class highdicom.sr.**ContentSequence**(*items=None*, *is_root=False*, *is_sr=True*)

Bases: Sequence

Sequence of DICOM SR Content Items.

Parameters

- **items** (*Union[Sequence[highdicom.sr.ContentItem], highdicom.sr.ContentSequence, None], optional*) – SR Content items
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree
- **is_sr** (*bool, optional*) – Whether the sequence is use to contain SR Content Items that are intended to be added to an SR document as opposed to other types of IODs based on an acquisition, protocol or workflow context template

append(*val*)

Append a content item to the sequence.

Parameters
item (highdicom.sr.ContentItem) – SR Content Item

Return type
None

extend(*val*)

Extend multiple content items to the sequence.

Parameters
val (*Iterable[highdicom.sr.ContentItem, highdicom.sr.ContentSequence]*) – SR Content Items

Return type
None

find(*name*)

Find contained content items given their name.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept]*) – Name of SR Content Items

Returns
Matched content items

Return type
highdicom.sr.ContentSequence

classmethod from_sequence(*sequence*, *is_root=False*, *is_sr=True*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing SR Content Items
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree
- **is_sr** (*bool, optional*) – Whether the sequence is use to contain SR Content Items that are intended to be added to an SR document as opposed to other types of IODs based on an acquisition, protocol or workflow context template

Returns
Content Sequence containing SR Content Items

Return type
highdicom.sr.ContentSequence

get_nodes()

Get content items that represent nodes in the content tree.

A node is hereby defined as a content item that has a *ContentSequence* attribute.

Returns
Matched content items

Return type
highdicom.sr.ContentSequence[*highdicom.sr.ContentItem*]

index(*val*)

Get the index of a given item.

Parameters
val (highdicom.sr.ContentItem) – SR Content Item

Returns
int

Return type
Index of the item in the sequence

insert(*position*, *val*)

Insert a content item into the sequence at a given position.

Parameters

- **position** (*int*) – Index position
- **val** (highdicom.sr.ContentItem) – SR Content Item

Return type
None

property is_root: bool

whether the sequence is intended for use at the root of the SR content tree.

Type
bool

Return type
bool

property is_sr: bool

whether the sequence is intended for use in an SR document

Type
bool

Return type
bool

class highdicom.sr.DateContentItem(*name*, *value*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type DATE.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **value** (*Union[str, datetime.date, pydicom.valuerep.DA]*) – Date
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type DATE

Returns
Content Item

Return type
highdicom.sr.DateContentItem

property value: date

date

Type
datetime.date

Return type
datetime.date

class highdicom.sr.DateTimeContentItem(*name*, *value*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type DATETIME.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name

- **value** (*Union[str, datetime.datetime, pydicom.valuerep.DT]*) – Datetime
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type DATETIME

Returns
Content Item

Return type
highdicom.sr.DateTimeContentItem

property value: datetime

datetime

Type
datetime.datetime

Return type
datetime.datetime

class highdicom.sr.**DeviceObserverIdentifyingAttributes**(*uid*, *name=None*, *manufacturer_name=None*, *model_name=None*, *serial_number=None*, *physical_location=None*, *role_in_procedure=None*)

Bases: Template

TID 1004 Device Observer Identifying Attributes

Parameters

- **uid** (*str*) – device UID
- **name** (*Union[str, None], optional*) – name of device
- **manufacturer_name** (*Union[str, None], optional*) – name of device's manufacturer
- **model_name** (*Union[str, None], optional*) – name of the device's model
- **serial_number** (*Union[str, None], optional*) – serial number of the device
- **physical_location** (*Union[str, None], optional*) – physical location of the device during the procedure
- **role_in_procedure** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – role of the device in the reported procedure

classmethod from_sequence(*sequence*, *is_root=False*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing SR Content Items of template TID 1004 "Device Observer Identifying Attributes"
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns
Content Sequence containing SR Content Items

Return type
highdicom.sr.templates.DeviceObserverIdentifyingAttributes

property manufacturer_name: Optional[str]

name of device manufacturer

Type
Union[str, None]

Return type
typing.Optional[str]

property model_name: Optional[str]

name of device model

Type
Union[str, None]

Return type
typing.Optional[str]

property name: Optional[str]

name of device

Type
Union[str, None]

Return type
typing.Optional[str]

property physical_location: Optional[str]

location of device

Type
Union[str, None]

Return type
typing.Optional[str]

property serial_number: Optional[str]

device serial number

Type
Union[str, None]

Return type
typing.Optional[str]

property uid: *UID*

unique device identifier

Type
highdicom.UID

Return type
highdicom.uid.UID

class highdicom.sr.**EnhancedSR**(*evidence*, *content*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer=None*, *is_complete=False*, *is_final=False*, *is_verified=False*, *institution_name=None*, *institutional_department_name=None*, *verifying_observer_name=None*, *verifying_organization=None*, *performed_procedure_codes=None*, *requested_procedures=None*, *previous_versions=None*, *record_evidence=True*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: _SR

SOP class for an Enhanced Structured Report (SR) document, whose content may include textual and a minimal amount of coded information, numeric measurement values, references to SOP Instances (retricted to the leaves of the tree), as well as 2D spatial or temporal regions of interest within such SOP Instances.

Parameters

- **evidence** (*Sequence[pydicom.dataset.Dataset]*) – Instances that are referenced in the content tree and from which the created SR document instance should inherit patient and study information
- **content** (*pydicom.dataset.Dataset*) – Root container content items that should be included in the SR document
- **series_instance_uid** (*str*) – Series Instance UID of the SR document series
- **series_number** (*int*) – Series Number of the SR document series
- **sop_instance_uid** (*str*) – SOP Instance UID that should be assigned to the SR document instance
- **instance_number** (*int*) – Number that should be assigned to this SR document instance
- **manufacturer** (*str, optional*) – Name of the manufacturer of the device that creates the SR document instance (in a research setting this is typically the same as *institution_name*)
- **is_complete** (*bool, optional*) – Whether the content is complete (default: False)
- **is_final** (*bool, optional*) – Whether the report is the definitive means of communicating the findings (default: False)
- **is_verified** (*bool, optional*) – Whether the report has been verified by an observer accountable for its content (default: False)
- **institution_name** (*Union[str, None], optional*) – Name of the institution of the person or device that creates the SR document instance
- **institutional_department_name** (*Union[str, None], optional*) – Name of the department of the person or device that creates the SR document instance
- **verifying_observer_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the person that verified the SR document (required if *is_verified*)
- **verifying_organization** (*Union[str, None], optional*) – Name of the organization that verified the SR document (required if *is_verified*)
- **performed_procedure_codes** (*Union[List[highdicom.sr.CodedConcept], None], optional*) – Codes of the performed procedures that resulted in the SR document
- **requested_procedures** (*Union[List[pydicom.dataset.Dataset], None], optional*) – Requested procedures that are being fullfilled by creation of the SR document

- **previous_versions** (*Union[List[pydicom.dataset.Dataset], None], optional*) – Instances representing previous versions of the SR document
- **record_evidence** (*bool, optional*) – Whether provided *evidence* should be recorded (i.e. included in Pertinent Other Evidence Sequence) even if not referenced by content items in the document tree (default: `True`)
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

Note: Each dataset in *evidence* must be part of the same study.

class highdicom.sr.**FindingSite**(*anatomic_location*, *laterality=None*, *topographical_modifier=None*)

Bases: *CodeContentItem*

Content item representing a coded finding site.

Parameters

- **anatomic_location** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – coded anatomic location (region or structure)
- **laterality** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – coded laterality (see CID 244 "Laterality" for options)
- **topographical_modifier** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – coded modifier of anatomic location

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters

dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns

Constructed object

Return type

highdicom.sr.FindingSite

property laterality: Optional[*CodedConcept*]

Return type

typing.Optional[*highdicom.sr.coding.CodedConcept*]

property topographical_modifier: Optional[*CodedConcept*]

Return type

typing.Optional[*highdicom.sr.coding.CodedConcept*]

class highdicom.sr.**GraphicTypeValues**(*value*)

Bases: `Enum`

Enumerated values for attribute Graphic Type.

See C.18.6.1.1.

CIRCLE = 'CIRCLE'

A circle defined by two (Column,Row) coordinates.

The first coordinate is the central point and the second coordinate is a point on the perimeter of the circle.

ELLIPSE = 'ELLIPSE'

An ellipse defined by four pixel (Column,Row) coordinates.

The first two coordinates specify the endpoints of the major axis and the second two coordinates specify the endpoints of the minor axis.

MULTIPOINT = 'MULTIPOINT'

Multiple pixels each denoted by an (Column,Row) coordinates.

POINT = 'POINT'

A single pixel denoted by a single (Column,Row) coordinate.

POLYLINE = 'POLYLINE'

Connected line segments with vertices denoted by (Column,Row) coordinate.

If the first and last coordinates are the same it is a closed polygon.

class highdicom.sr.**GraphicTypeValues3D**(*value*)

Bases: Enum

Enumerated values for attribute Graphic Type 3D.

See C.18.9.1.2.

ELLIPSE = 'ELLIPSE'

An ellipse defined by four (X,Y,Z) coordinates.

The first two coordinates specify the endpoints of the major axis and the second two coordinates specify the endpoints of the minor axis.

ELLIPSOID = 'ELLIPSOID'

A three-dimensional geometric surface defined by six (X,Y,Z) coordinates.

The plane sections of the surface are either ellipses or circles and the surface contains three intersecting orthogonal axes: "a", "b", and "c". The first and second coordinates specify the endpoints of axis "a", the third and fourth coordinates specify the endpoints of axis "b", and the fifth and sixth coordinates specify the endpoints of axis "c".

MULTIPOINT = 'MULTIPOINT'

Multiple points each denoted by an (X,Y,Z) coordinate.

The points need not be coplanar.

POINT = 'POINT'

An individual point denoted by a single (X,Y,Z) coordinate.

POLYGON = 'POLYGON'

Connected line segments with vertices denoted by (X,Y,Z) coordinates.

The first and last coordinates shall be the same forming a closed polygon. The points shall be coplanar.

POLYLINE = 'POLYLINE'

Connected line segments with vertices denoted by (X,Y,Z) coordinates.

The coordinates need not be coplanar.

class highdicom.sr.**ImageContentItem**(*name*, *referenced_sop_class_uid*, *referenced_sop_instance_uid*, *referenced_frame_numbers=None*, *referenced_segment_numbers=None*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type IMAGE.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **referenced_sop_class_uid** (*Union[highdicom.UID, str]*) – SOP Class UID of the referenced image object
- **referenced_sop_instance_uid** (*Union[highdicom.UID, str]*) – SOP Instance UID of the referenced image object
- **referenced_frame_numbers** (*Union[int, Sequence[int], None], optional*) – Number of frame(s) to which the reference applies in case of a multi-frame image
- **referenced_segment_numbers** (*Union[int, Sequence[int], None], optional*) – Number of segment(s) to which the refernce applies in case of a segmentation image
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type IMAGE

Returns
Content Item

Return type
highdicom.sr.ImageContentItem

property referenced_frame_numbers: Optional[List[int]]

referenced frame numbers

Type
Union[List[int], None]

Return type
typing.Optional[typing.List[int]]

property referenced_segment_numbers: Optional[List[int]]

Union[List[int], None] referenced segment numbers

Return type
typing.Optional[typing.List[int]]

property referenced_sop_class_uid: *UID*

referenced SOP Class UID

Type
highdicom.UID

Return type
highdicom.uid.UID

property referenced_sop_instance_uid: *UID*

referenced SOP Instance UID

Type
highdicom.UID

Return type
highdicom.uid.UID

property value: Tuple[*UID*, *UID*]

Tuple[highdicom.UID, highdicom.UID]: referenced SOP Class UID and SOP Instance UID

Return type
typing.Tuple[*highdicom.uid.UID*, *highdicom.uid.UID*]

class highdicom.sr.ImageLibrary(*datasets*)

Bases: Template

TID 1600 Image Library

Parameters
datasets (*Sequence[pydicom.dataset.Dataset]*) – Image Datasets to include in image library. Non-image objects will throw an exception.

class highdicom.sr.ImageLibraryEntryDescriptors(*image*, *additional_descriptors=None*)

Bases: Template

TID 1602 Image Library Entry Descriptors

Parameters

- **image** (*pydicom.dataset.Dataset*) – Metadata of a referenced image instance
- **additional_descriptors** (*Union[Sequence[highdicom.sr.ContentItem], None], optional*) – Optional additional SR Content Items that should be included for description of the referenced image

class highdicom.sr.ImageRegion(*graphic_type*, *graphic_data*, *source_image*, *pixel_origin_interpretation=None*)

Bases: *ScoordContentItem*

Content item representing an image region of interest in the two-dimensional image coordinate space in pixel unit.

Parameters

- **graphic_type** (*Union[highdicom.sr.GraphicTypeValues, str]*) – name of the graphic type
- **graphic_data** (*numpy.ndarray*) – array of ordered spatial coordinates, where each row of the array represents a (column, row) coordinate pair
- **source_image** (highdicom.sr.SourceImageForRegion) – source image to which *graphic_data* relates
- **pixel_origin_interpretation** (*Union[highdicom.sr.PixelOriginInterpretationValues, str, None], optional*) – whether pixel coordinates specified by *graphic_data* are defined relative to the total pixel matrix (highdicom.sr.PixelOriginInterpretationValues.VOLUME) or relative to an individual frame (highdicom.sr.PixelOriginInterpretationValues.FRAME) of the source image (default: highdicom.sr.PixelOriginInterpretationValues.VOLUME)

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns
Constructed object

Return type
highdicom.sr.ImageRegion

class highdicom.sr.**ImageRegion3D**(*graphic_type*, *graphic_data*, *frame_of_reference_uid*)

Bases: *Scoord3DContentItem*

Content item representing an image region of interest in the three-dimensional patient/slide coordinate space in millimeter unit.

Parameters

- **graphic_type** (*Union[highdicom.sr.GraphicTypeValues3D, str]*) – name of the graphic type
- **graphic_data** (*numpy.ndarray*) – array of ordered spatial coordinates, where each row of the array represents a (x, y, z) coordinate triplet
- **frame_of_reference_uid** (*str*) – UID of the frame of reference

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns
Constructed object

Return type
highdicom.sr.ImageRegion3D

class highdicom.sr.**LanguageOfContentItemAndDescendants**(*language*)

Bases: Template

TID 1204 Language of Content Item and Descendants

Parameters
language (highdicom.sr.CodedConcept) – language used for content items included in report

class highdicom.sr.**LongitudinalTemporalOffsetFromEvent**(*value*, *unit*, *event_type*)

Bases: *NumContentItem*

Content item representing a longitudinal temporal offset from an event.

Parameters

- **value** (*Union[int, float]*) – Offset in time from a particular event of significance
- **unit** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Unit of time, e.g., “Days” or “Seconds”
- **event_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Type of event to which offset is relative, e.g., “Baseline” or “Enrollment”

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns
Constructed object

Return type
highdicom.sr.LongitudinalTemporalOffsetFromEvent

class highdicom.sr.**Measurement**(*name*, *value*, *unit*, *qualifier=None*, *tracking_identifier=None*, *algorithm_id=None*, *derivation=None*, *finding_sites=None*, *method=None*, *properties=None*, *referenced_images=None*, *referenced_real_world_value_map=None*)

Bases: Template

TID 300 Measurement

Parameters

- **name** (highdicom.sr.CodedConcept) – Name of the measurement (see CID 7469 "Generic Intensity and Size Measurements" and CID 7468 "Texture Measurements" for options)
- **value** (*Union[int, float]*) – Numeric measurement value
- **unit** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Unit of the numeric measurement value (see CID 7181 "Abstract Multi-dimensional Image Model Component Units" for options)
- **qualifier** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Qualification of numeric measurement value or as an alternative qualitative description
- **tracking_identifier** (*Union[highdicom.sr.TrackingIdentifier, None], optional*) – Identifier for tracking measurements
- **algorithm_id** (*Union[highdicom.sr.AlgorithmIdentification, None], optional*) – Identification of algorithm used for making measurements
- **derivation** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – How the value was computed (see CID 7464 "General Region of Interest Measurement Modifiers" for options)
- **finding_sites** (*Union[Sequence[highdicom.sr.FindingSite], None], optional*) – Coded description of one or more anatomic locations corresonding to the image region from which measurement was taken
- **method** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Measurement method (see CID 6147 "Response Criteria" for options)
- **properties** (*Union[highdicom.sr.MeasurementProperties, None], optional*) – Measurement properties, including evaluations of its normality and/or significance, its relationship to a reference population, and an indication of its selection from a set of measurements
- **referenced_images** (*Union[Sequence[highdicom.sr.SourceImageForMeasurement], None], optional*) – Referenced images which were used as sources for the measurement

- **referenced_real_world_value_map** (*Union[highdicom.sr.RealWorldValueMap, None], optional*) – Referenced real world value map for referenced source images

property derivation: Optional[*CodedConcept*]

derivation

Type
Union[*highdicom.sr.CodedConcept*, None]

Return type
typing.Optional[*highdicom.sr.coding.CodedConcept*]

property finding_sites: List[*FindingSite*]

finding sites

Type
List[*highdicom.sr.FindingSite*]

Return type
typing.List[*highdicom.sr.content.FindingSite*]

classmethod from_sequence(*sequence*, *is_root=False*)

Construct object from a sequence of content items.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Content Sequence containing one SR NUM Content Items
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns
Content Sequence containing one SR NUM Content Items

Return type
highdicom.sr.Measurement

property method: Optional[*CodedConcept*]

method

Type
Union[*highdicom.sr.CodedConcept*, None]

Return type
typing.Optional[*highdicom.sr.coding.CodedConcept*]

property name: *CodedConcept*

coded name of the measurement

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property qualifier: Optional[*CodedConcept*]

qualifier

Type
Union[*highdicom.sr.CodedConcept*, None]

Return type
typing.Optional[*highdicom.sr.coding.CodedConcept*]

property referenced_images: List[*SourceImageForMeasurement*]
referenced images

Type
List[*highdicom.sr.SourceImageForMeasurement*]

Return type
typing.List[*highdicom.sr.content.SourceImageForMeasurement*]

property unit: *CodedConcept*
unit

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property value: Union[int, float]
measured value

Type
Union[int, float]

Return type
typing.Union[int, float]

class highdicom.sr.**MeasurementProperties**(*normality=None*, *level_of_significance=None*, *selection_status=None*, *measurement_statistical_properties=None*, *normal_range_properties=None*, *upper_measurement_uncertainty=None*, *lower_measurement_uncertainty=None*)

Bases: Template

TID 310 Measurement Properties

Parameters

- **normality** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – the extend to which the measurement is considered normal or abnormal (see CID 222 "Normality Codes" for options)
- **level_of_significance** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – the extend to which the measurement is considered normal or abnormal (see CID 220 "Level of Significance" for options)
- **selection_status** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – how the measurement value was selected or computed from a set of available values (see CID 224 "Selection Method" for options)
- **measurement_statistical_properties** (*Union[highdicom.sr.MeasurementStatisticalProperties, None], optional*) – statistical properties of a reference population for a measurement and/or the position of a measurement in such a reference population
- **normal_range_properties** (*Union[highdicom.sr.NormalRangeProperties, None], optional*) – statistical properties of a reference population for a measurement and/or the position of a measurement in such a reference population

- **upper_measurement_uncertainty** (*Union[int, float, None], optional*) – upper range of measurement uncertainty
- **lower_measurement_uncertainty** (*Union[int, float, None], optional*) – lower range of measurement uncertainty

class highdicom.sr.**MeasurementReport**(*observation_context*, *procedure_reported*, *imaging_measurements=None*, *title=None*, *language_of_content_item_and_descendants=None*, *referenced_images=None*)

Bases: Template

TID 1500 Measurement Report

Parameters

- **observation_context** (highdicom.sr.ObservationContext) – description of the observation context
- **procedure_reported** (*Union[Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code], Sequence[Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]]]*) – one or more coded description(s) of the procedure (see CID 100 "Quantitative Diagnostic Imaging Procedures" for options)
- **imaging_measurements** (*Union[Sequence[Union[highdicom.sr.PlanarROIMeasurementsAndQualitativeEvaluations, highdicom.sr.VolumetricROIMeasurementsAndQualitativeEvaluations, highdicom.sr.MeasurementsAndQualitativeEvaluations]]], optional*) – measurements and qualitative evaluations of images or regions within images
- **title** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – title of the report (see CID 7021 "Measurement Report Document Titles" for options)
- **language_of_content_item_and_descendants** (*Union[highdicom.sr.LanguageOfContentItemAndDescendants, None], optional*) – specification of the language of report content items (defaults to English)
- **referenced_images** (*Union[Sequence[pydicom.Dataset], None], optional*) – Images that should be included in the library

classmethod from_sequence(*sequence*, *is_root=True*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing "Measurement Report" SR Content Items of Value Type CONTAINER (sequence shall only contain a single item)
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns

Content Sequence containing root CONTAINER SR Content Item

Return type

highdicom.sr.MeasurementReport

get_image_measurement_groups(*tracking_uid=None*, *finding_type=None*, *finding_site=None*, *referenced_sop_instance_uid=None*, *referenced_sop_class_uid=None*)

Get imaging measurements of images.

Finds (and optionally filters) content items contained in the CONTAINER content item "Measurement Group" as specified by TID 1501 "Measurement and Qualitative Evaluation Group".

Parameters

- **tracking_uid** (*Union[str, None], optional*) – Unique tracking identifier
- **finding_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Finding
- **finding_site** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Finding site
- **referenced_sop_instance_uid** (*Union[str, None], optional*) – SOP Instance UID of the referenced instance.
- **referenced_sop_class_uid** (*Union[str, None], optional*) – SOP Class UID of the referenced instance.

Returns

Sequence of content items for each matched measurement group

Return type

List[*highdicom.sr.MeasurementsAndQualitativeEvaluations*]

get_observer_contexts(*observer_type=None*)

Get observer contexts.

Parameters

observer_type (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Type of observer ("Device" or "Person") for which should be filtered

Returns

Observer contexts

Return type

List[*highdicom.sr.ObserverContext*]

get_planar_roi_measurement_groups(*tracking_uid=None*, *finding_type=None*, *finding_site=None*, *reference_type=None*, *graphic_type=None*, *referenced_sop_instance_uid=None*, *referenced_sop_class_uid=None*)

Get imaging measurement groups of planar regions of interest.

Finds (and optionally filters) content items contained in the CONTAINER content item "Measurement group" as specified by TID 1410 "Planar ROI Measurements and Qualitative Evaluations".

Parameters

- **tracking_uid** (*Union[str, None], optional*) – Unique tracking identifier
- **finding_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Finding
- **finding_site** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Finding site

- **reference_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Type of referenced ROI. Valid values are limited to codes *ImageRegion*, *ReferencedSegmentationFrame*, and *RegionInSpace*.
- **graphic_type** (*Union[highdicom.sr.GraphicTypeValues, highdicom.sr.GraphicTypeValues3D, None], optional*) – Graphic type of image region
- **referenced_sop_instance_uid** (*Union[str, None], optional*) – SOP Instance UID of the referenced instance, which may be a segmentation image, source image for the region or segmentation, or RT struct, depending on *reference_type*
- **referenced_sop_class_uid** (*Union[str, None], optional*) – SOP Class UID of the referenced instance, which may be a segmentation image, source image for the region or segmentation, or RT struct, depending on *reference_type*

Returns
Sequence of content items for each matched measurement group

Return type
List[*highdicom.sr.PlanarROIMeasurementsAndQualitativeEvaluations*]

get_subject_contexts(*subject_class=None*)

Get subject contexts.

Parameters
subject_class (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Type of subject ("Specimen", "Fetus", or "Device") for which should be filtered

Returns
Subject contexts

Return type
List[*highdicom.sr.SubjectContext*]

get_volumetric_roi_measurement_groups(*tracking_uid=None*, *finding_type=None*, *finding_site=None*, *reference_type=None*, *graphic_type=None*, *referenced_sop_instance_uid=None*, *referenced_sop_class_uid=None*)

Get imaging measurement groups of volumetric regions of interest.

Finds (and optionally filters) content items contained in the CONTAINER content item "Measurement group" as specified by TID 1411 "Volumetric ROI Measurements and Qualitative Evaluations".

Parameters

- **tracking_uid** (*Union[str, None], optional*) – Unique tracking identifier
- **finding_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Finding
- **finding_site** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Finding site
- **reference_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Type of referenced ROI. Valid values are limited to codes *ImageRegion*, *ReferencedSegment*, *VolumeSurface* and *RegionInSpace*.
- **graphic_type** (*Union[highdicom.sr.GraphicTypeValues, highdicom.sr.GraphicTypeValues3D, None], optional*) – Graphic type of image region

- **referenced_sop_instance_uid** (*Union[str, None], optional*) – SOP Instance UID of the referenced instance, which may be a segmentation image, source image for the region or segmentation, or RT struct, depending on *reference_type*
- **referenced_sop_class_uid** (*Union[str, None], optional*) – SOP Class UID of the referenced instance, which may be a segmentation image, source image for the region or segmentation, or RT struct, depending on *reference_type*

Returns
Sequence of content items for each matched measurement group

Return type
List[*highdicom.sr.VolumetricROIMeasurementsAndQualitativeEvaluations*]

class highdicom.sr.**MeasurementStatisticalProperties**(*values*, *description=None*, *authority=None*)

Bases: Template

TID 311 Measurement Statistical Properties

Parameters

- **values** (*Sequence[highdicom.sr.NumContentItem]*) – reference values of the population of measurements, e.g., its mean or standard deviation (see CID 226 "Population Statistical Descriptors" and CID 227 "Sample Statistical Descriptors" for options)
- **description** (*Union[str, None], optional*) – description of the reference population of measurements
- **authority** (*Union[str, None], optional*) – authority for a description of the reference population of measurements

class highdicom.sr.**MeasurementsAndQualitativeEvaluations**(*tracking_identifier*, *referenced_real_world_value_map=None*, *time_point_context=None*, *finding_type=None*, *method=None*, *algorithm_id=None*, *finding_sites=None*, *session=None*, *measurements=None*, *qualitative_evaluations=None*, *finding_category=None*, *source_images=None*)

Bases: _MeasurementsAndQualitativeEvaluations

TID 1501 Measurement and Qualitative Evaluation Group

Parameters

- **tracking_identifier** (highdicom.sr.TrackingIdentifier) – Identifier for tracking measurements
- **referenced_real_world_value_map** (*Union[highdicom.sr.RealWorldValueMap, None], optional*) – Referenced real world value map for region of interest
- **time_point_context** (*Union[highdicom.sr.TimePointContext, None], optional*) – Description of the time point context
- **finding_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Type of observed finding
- **method** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Coded measurement method (see CID 6147 "Response Criteria" for options)

- **algorithm_id** (*Union[highdicom.sr.AlgorithmIdentification, None], optional*) – Identification of algorithm used for making measurements
- **finding_sites** (*Sequence[highdicom.sr.FindingSite, None], optional*) – Coded description of one or more anatomic locations at which finding was observed
- **session** (*Union[str, None], optional*) – Description of the session
- **measurements** (*Union[Sequence[highdicom.sr.Measurement], None], optional*) – Numeric measurements
- **qualitative_evaluations** (*Union[Sequence[highdicom.sr.QualitativeEvaluation], None], optional*) – Coded name-value pairs that describe qualitative evaluations
- **finding_category** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Category of observed finding, e.g., anatomic structure or morphologically abnormal structure
- **source_images** (*Optional[Sequence[highdicom.sr.SourceImageForMeasurementGroup]], optional*) – Images to that were the source of the measurements. If not provided, all images that listed in the document tree of the containing SR document are assumed to be source images.

property source_images: List[*SourceImageForMeasurementGroup*]

source images

Type
List[*highdicom.sr.SourceImageForMeasurementGroup*]

Return type
typing.List[*highdicom.sr.content.SourceImageForMeasurementGroup*]

class highdicom.sr.**NormalRangeProperties**(*values*, *description=None*, *authority=None*)

Bases: Template

TID 312 Normal Range Properties

Parameters

- **values** (*Sequence[highdicom.sr.NumContentItem]*) – reference values of the normal range, e.g., its upper and lower bound (see CID 223 "Normal Range Values" for options)
- **description** (*Union[str, None], optional*) – description of the normal range
- **authority** (*Union[str, None], optional*) – authority for the description of the normal range

class highdicom.sr.**NumContentItem**(*name*, *value*, *unit*, *qualifier=None*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type NUM.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **value** (*Union[int, float]*) – Numeric value
- **unit** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code], optional*) – Coded units of measurement (see CID 7181 "Abstract Multi-dimensional Image Model Component Units")

- **qualifier** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Qualification of numeric value or as an alternative to numeric value, e.g., reason for absence of numeric value (see CID 42 "Numeric Value Qualifier" for options)
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type NUM

Returns
Content Item

Return type
highdicom.sr.NumContentItem

property qualifier: Optional[*CodedConcept*]

qualifier

Type
Union[*highdicom.sr.CodedConcept*, None]

Return type
typing.Optional[*highdicom.sr.coding.CodedConcept*]

property unit: *CodedConcept*

unit

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property value: Union[int, float]

measured value

Type
Union[int, float]

Return type
typing.Union[int, float]

class highdicom.sr.**ObservationContext**(*observer_person_context=None*, *observer_device_context=None*, *subject_context=None*)

Bases: Template

TID 1001 Observation Context

Parameters

- **observer_person_context** (*Union[highdicom.sr.ObserverContext, None], optional*) – description of the person that reported the observation
- **observer_device_context** (*Union[highdicom.sr.ObserverContext, None], optional*) – description of the device that was involved in reporting the observation

- **subject_context** (*Union[highdicom.sr.SubjectContext, None], optional*) – description of the imaging subject in case it is not the patient for which the report is generated (e.g., a pathology specimen in a whole-slide microscopy image, a fetus in an ultrasound image, or a pacemaker device in a chest X-ray image)

class highdicom.sr.**ObserverContext**(*observer_type*, *observer_identifying_attributes*)

Bases: Template

TID 1002 Observer Context

Parameters

- **observer_type** (highdicom.sr.CodedConcept) – type of observer (see CID 270 "Observer Type" for options)
- **observer_identifying_attributes** (*Union[highdicom.sr.PersonObserverIdentifyingAttributes, highdicom.sr.DeviceObserverIdentifyingAttributes]*) – observer identifying attributes

property **observer_identifying_attributes**: **Union[***PersonObserverIdentifyingAttributes*, *DeviceObserverIdentifyingAttributes***]**

Union[highdicom.sr.PersonObserverIdentifyingAttributes, highdicom.sr.DeviceObserverIdentifyingAttributes]: observer identifying attributes

Return type
typing.Union[*highdicom.sr.templates.PersonObserverIdentifyingAttributes*, *highdicom.sr.templates.DeviceObserverIdentifyingAttributes*]

property **observer_type**: *CodedConcept*

observer type

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

class highdicom.sr.**PersonObserverIdentifyingAttributes**(*name*, *login_name=None*, *organization_name=None*, *role_in_organization=None*, *role_in_procedure=None*)

Bases: Template

TID 1003 Person Observer Identifying Attributes

Parameters

- **name** (*str*) – name of the person
- **login_name** (*Union[str, None], optional*) – login name of the person
- **organization_name** (*Union[str, None], optional*) – name of the person's organization
- **role_in_organization** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – role of the person within the organization
- **role_in_procedure** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – role of the person in the reported procedure

classmethod from_sequence(*sequence*, *is_root=False*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing SR Content Items of template TID 1003 "Person Observer Identifying Attributes"
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns

Content Sequence containing SR Content Items

Return type

highdicom.sr.PersonObserverIdentifyingAttributes

property login_name: Optional[str]

login name of the person

Type

Union[str, None]

Return type

typing.Optional[str]

property name: str

name of the person

Type

str

Return type

str

property organization_name: Optional[str]

name of the person's organization

Type

Union[str, None]

Return type

typing.Optional[str]

property role_in_organization: Optional[str]

role of the person in the organization

Type

Union[str, None]

Return type

typing.Optional[str]

property role_in_procedure: Optional[str]

role of the person in the procedure

Type

Union[str, None]

Return type

typing.Optional[str]

class highdicom.sr.**PixelOriginInterpretationValues**(*value*)

Bases: Enum

Enumerated values for attribute Pixel Origin Interpretation.

FRAME = 'FRAME'

Relative to the individual frame.

VOLUME = 'VOLUME'

Relative to the Total Pixel Matrix of the VOLUME image.

class highdicom.sr.**PlanarROIMeasurementsAndQualitativeEvaluations**(*tracking_identifier*, *referenced_region=None*, *referenced_segment=None*, *referenced_real_world_value_map=None*, *time_point_context=None*, *finding_type=None*, *method=None*, *algorithm_id=None*, *finding_sites=None*, *session=None*, *measurements=None*, *qualitative_evaluations=None*, *geometric_purpose=None*, *finding_category=None*)

Bases: _ROIMeasurementsAndQualitativeEvaluations

TID 1410 Planar ROI Measurements and Qualitative Evaluations

Parameters

- **tracking_identifier** (highdicom.sr.TrackingIdentifier) – Identifier for tracking measurements
- **referenced_region** (*Union[*highdicom.sr.ImageRegion, highdicom.sr.ImageRegion3D, *None], optional*) – Region of interest in source image
- **referenced_segment** (*Union[*highdicom.sr.ReferencedSegmentationFrame, *None], optional*) – Segmentation for region of interest in source image
- **referenced_real_world_value_map** (*Union[*highdicom.sr.RealWorldValueMap, *None], optional*) – Referenced real world value map for region of interest
- **time_point_context** (*Union[*highdicom.sr.TimePointContext, *None], optional*) – Description of the time point context
- **finding_type** (*Union[*highdicom.sr.CodedConcept, *pydicom.sr.coding.Code, None], optional*) – Type of object that was measured, e.g., organ or tumor
- **method** (*Union[*highdicom.sr.CodedConcept, *pydicom.sr.coding.Code, None], optional*) – Coded measurement method (see CID 6147 “Response Criteria” for options)
- **algorithm_id** (*Union[*highdicom.sr.AlgorithmIdentification, *None], optional*) – Identification of algorithm used for making measurements
- **finding_sites** (*Union[Sequence[*highdicom.sr.FindingSite*], None], optional*) – Coded description of one or more anatomic locations corresonding to the image region from which measurement was taken

- **session** (*Union[str, None], optional*) – Description of the session
- **measurements** (*Union[Sequence[highdicom.sr.Measurement], None], optional*) – Measurements for a region of interest
- **qualitative_evaluations** (*Union[Sequence[highdicom.sr.QualitativeEvaluation], None], optional*) – Coded name-value (question-answer) pairs that describe qualitative evaluations of a region of interest
- **geometric_purpose** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Geometric interpretation of region of interest (see CID 219 "Geometry Graphical Representation" for options)
- **finding_category** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Category of observed finding, e.g., anatomic structure or morphologically abnormal structure

---

Note: Either a segmentation or a region needs to referenced together with the corresponding source image from which the segmentation or region was obtained.

---

classmethod from_sequence(*sequence*, *is_root=False*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing "Measurement Group" SR Content Items of Value Type CONTAINER (sequence shall only contain a single item)
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns

Content Sequence containing root CONTAINER SR Content Item

Return type

highdicom.sr.PlanarROIMeasurementsAndQualitativeEvaluations

property reference_type: Code

pydicom.sr.coding.Code:

The "type" of the ROI reference as a coded concept. This will be one of the following coded concepts from the DCM coding scheme:

- Image Region
- Referenced Segmentation Frame
- Region In Space

Return type

pydicom.sr.coding.Code

property referenced_segmentation_frame: Optional[*ReferencedSegmentationFrame*]

Union[highdicom.sr.ImageContentItem, None]: segmentation frame referenced by the measurements group

Return type

typing.Optional[*highdicom.sr.content.ReferencedSegmentationFrame*]

property roi: Optional[Union[*ImageRegion*, *ImageRegion3D*]]

Union[highdicom.sr.ImageRegion, highdicom.sr.ImageRegion3D, None]: image region defined by spatial coordinates

Return type
typing.Union[*highdicom.sr.content.ImageRegion*, *highdicom.sr.content.ImageRegion3D*, None]

class highdicom.sr.**PnameContentItem**(*name*, *value*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type PNAME.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **value** (*Union[str, pydicom.valuerep.PersonName]*) – Name of the person
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod **from_dataset**(*dataset*)

Construct object from existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type PNAME

Returns
Content Item

Return type
highdicom.sr.PnameContentItem

property value: PersonName

person name

Type
pydicom.valuerep.PersonName

Return type
pydicom.valuerep.PersonName

class highdicom.sr.**QualitativeEvaluation**(*name*, *value*)

Bases: Template

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – concept name
- **value** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – coded value or an enumerated item representing a coded value

classmethod **from_sequence**(*sequence*, *is_root=False*)

Construct object from a sequence of content items.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Content Sequence containing one SR CODE Content Item

- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns
Content Sequence containing one SR CODE Content Item

Return type
highdicom.sr.QualitativeEvaluation

property name: *CodedConcept*

name of the qualitative evaluation

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property value: Union[int, float]

coded value of the qualitative evaluation

Type
Union[int, float]

Return type
typing.Union[int, float]

class highdicom.sr.**RealWorldValueMap**(*referenced_sop_instance_uid*)

Bases: *CompositeContentItem*

Content item representing a reference to a real world value map.

Parameters
referenced_sop_instance_uid (*str*) – SOP Instance UID of the referenced object

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns
Constructed object

Return type
highdicom.sr.RealWorldValueMap

classmethod from_source_value_map(*value_map_dataset*)

Construct the content item directly from an image dataset

Parameters
value_map_dataset (*pydicom.dataset.Dataset*) – dataset representing the real world value map to be referenced

Returns
Content item representing a reference to the image dataset

Return type
highdicom.sr.RealWorldValueMap

class highdicom.sr.**ReferencedSegment**(*sop_class_uid*, *sop_instance_uid*, *segment_number*, *frame_numbers=None*, *source_images=None*, *source_series=None*)

Bases: *ContentSequence*

Content items representing a reference to an individual segment of a segmentation or surface segmentation instance as well as the images that were used as a source for the segmentation.

Parameters

- **sop_class_uid** (*str*) – SOP Class UID of the referenced segmentation object
- **sop_instance_uid** (*str*) – SOP Instance UID of the referenced segmentation object
- **segment_number** (*int*) – number of the segment to which the reference applies
- **frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies (in case a segmentation instance is referenced)
- **source_images** (*Union[Sequence[highdicom.sr.SourceImageForSegmentation], None], optional*) – source images for segmentation
- **source_series** (*Union[highdicom.sr.SourceSeriesForSegmentation, None], optional*) – source series for segmentation

Note: Either *source_images* or *source_series* must be provided.

classmethod **from_segmentation**(*segmentation*, *segment_number*, *frame_numbers=None*)

Construct the content item directly from a segmentation dataset

Parameters

- **segmentation** (*pydicom.dataset.Dataset*) – dataset representing a segmentation containing the referenced segment
- **segment_number** (*int*) – number of the segment to reference within the provided dataset
- **frame_numbers** (*Union[Sequence[int], None], optional*) – list of frames in the segmentation dataset to reference. If not provided, the reference is assumed to apply to all frames of the given segment number. Note that frame numbers are indexed with 1-based indexing.

Returns

Content item representing a reference to the segment

Return type

highdicom.sr.ReferencedSegment

Notes

This method will attempt to deduce source image information from information provided in the segmentation instance. If available, it will used information specific to the segment and frame numbers (if any) provided using the Derivation Image Sequence information in the frames. If this information is not present in the segmentation dataset, it will instead use the information in the Referenced Series Sequence, which applies to all segments and frames present in the segmentation instance.

classmethod **from_sequence**(*sequence*)

Construct an object from items within an existing content sequence.

Parameters
sequence (*Sequence[Dataset]*) – Sequence of datasets to be converted. This is expected to contain a content item with concept name "Referenced Segmentation Frame", and either at least one content item with concept name "Source Image For Segmentation" or a single content item with concept name "Source Series For Segmentation". Any other other items will be ignored.

Returns
Constructed ReferencedSegment object, containing copies of the original content items.

Return type
highdicom.sr.ReferencedSegment

has_source_images()

Returns whether the object contains information about source images.

ReferencedSegment objects must either contain information about source images or source series (and not both).

Returns
True if the object contains information about source images. False if the image contains information about the source series.

Return type
bool

property referenced_frame_numbers: Optional[List[int]]

Union[List[int], None] referenced frame numbers

Return type
typing.Optional[typing.List[int]]

property referenced_segment_numbers: Optional[List[int]]

Union[List[int], None] referenced segment numbers

Return type
typing.Optional[typing.List[int]]

property referenced_sop_class_uid: *UID*

highdicom.UID referenced SOP Class UID

Return type
highdicom.uid.UID

property referenced_sop_instance_uid: *UID*

highdicom.UID referenced SOP Class UID

Return type
highdicom.uid.UID

property source_images_for_segmentation: List[*SourceImageForSegmentation*]

List[highdicom.sr.SourceImageForSegmentation] Source images for the referenced segmentation

Return type
typing.List[*highdicom.sr.content.SourceImageForSegmentation*]

property source_series_for_segmentation: Optional[*SourceSeriesForSegmentation*]

Union[highdicom.sr.SourceSeriesForSegmentation, None] Source series for the referenced segmentation

Return type
typing.Optional[*highdicom.sr.content.SourceSeriesForSegmentation*]

class highdicom.sr.**ReferencedSegmentationFrame**(*sop_class_uid*, *sop_instance_uid*, *frame_number*, *segment_number*, *source_image*)

Bases: *ContentSequence*

Content items representing a reference to an individual frame of a segmentation instance as well as the image that was used as a source for the segmentation.

Parameters

- **sop_class_uid** (*str*) – SOP Class UID of the referenced image object
- **sop_instance_uid** (*str*) – SOP Instance UID of the referenced image object
- **segment_number** (*int*) – Number of the segment to which the reference applies
- **frame_number** (*Union[int, Sequence[int]]*) – Number of the frame to which the reference applies. If the referenced segmentation image is tiled, more than one frame may be specified.
- **source_image** (highdicom.sr.SourceImageForSegmentation) – Source image for segmentation

classmethod **from_segmentation**(*segmentation*, *frame_number=None*, *segment_number=None*)

Construct the content item directly from a segmentation dataset

Parameters

- **segmentation** (*pydicom.dataset.Dataset*) – Dataset representing a segmentation containing the referenced segment.
- **frame_number** (*Union[int, Sequence[int], None], optional*) – Number of the frame(s) that should be referenced
- **segment_number** (*Union[int, None], optional*) – Number of the segment to which the reference applies

Returns

Content item representing a reference to the segment

Return type

highdicom.sr.ReferencedSegment

Notes

This method will attempt to deduce source image information from information provided in the segmentation instance. If available, it will use information specific to the segment and frame numbers (if any) provided using the Derivation Image Sequence item for the given frame. If this information is not present in the segmentation dataset, it will instead use the information in the Referenced Series Sequence, which applies to all segments and frames present in the segmentation instance.

Raises

- **ValueError** – If the dataset provided is not a segmentation dataset. If any of the frames numbers are invalid for the dataset. If multiple elements are found in the Derivation Image Sequence or Source Image Sequence for any of the referenced frames, or if these attributes are absent, if these attributes are absent, if there are multiple elements in the Referenced Instance Sequence.
- **AttributeError** – If the Referenced Series Sequence or Referenced Instance Sequence attributes are absent from the dataset.

classmethod from_sequence(*sequence*)

Construct an object from items within an existing content sequence.

Parameters

sequence (*Sequence[Dataset]*) – Sequence of datasets to be converted. This is expected to contain content items with the following names: "Referenced Segmentation Frame", "Source Image For Segmentation". Any other other items will be ignored.

Returns

Constructed ReferencedSegmentationFrame object, containing copies of the relevant original content items.

Return type

highdicom.sr.ReferencedSegmentationFrame

property referenced_frame_numbers: Optional[List[int]]

Union[List[int], None] referenced frame numbers

Return type

typing.Optional[typing.List[int]]

property referenced_segment_numbers: Optional[List[int]]

Union[List[int], None] referenced segment numbers

Return type

typing.Optional[typing.List[int]]

property referenced_sop_class_uid: *UID*

highdicom.UID referenced SOP Class UID

Return type

highdicom.uid.UID

property referenced_sop_instance_uid: *UID*

highdicom.UID referenced SOP Class UID

Return type

highdicom.uid.UID

property source_image_for_segmentation: *SourceImageForSegmentation*

highdicom.sr.SourceImageForSegmentation Source image for the referenced segmentation

Return type

highdicom.sr.content.SourceImageForSegmentation

class highdicom.sr.**RelationshipTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Relationship Type.

See C.17.3.2.4.

CONTAINS = 'CONTAINS'

Parent item contains child content item.

HAS_ACQ_CONTEXT = 'HAS ACQ CONTEXT'

Has acquisition context.

The child content item describes the conditions present during data acquisition of the source content item.

HAS_CONCEPT_MOD = 'HAS CONCEPT MOD'

Has concept modifier.

The child content item qualifies or describes the concept name of the parent content item.

HAS_OBS_CONTEXT = 'HAS OBS CONTEXT'

Has observation context.

Child content items shall convey any specialization of observation context needed for unambiguous documentation of the parent content item.

HAS_PROPERTIES = 'HAS PROPERTIES'

Child content items describe properties of the parent content item.

INFERRED_FROM = 'INFERRED FROM'

Parent content item is inferred from the child content item.

The Parent content item conveys a measurement or other inference made from the child content item(s). Denotes the supporting evidence for a measurement or judgment.

SELECTED_FROM = 'SELECTED FROM'

Parent content item is selected from the child content items.

The parent content item conveys spatial or temporal coordinates selected from the child content item(s).

class highdicom.sr.**Scoord3DContentItem**(*name*, *graphic_type*, *graphic_data*, *frame_of_reference_uid*, *fiducial_uid=None*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type SCOORD3D.

Note: Spatial coordinates are defined in the patient or specimen-based coordinate system and have milimeter unit.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **graphic_type** (*Union[highdicom.sr.GraphicTypeValues3D, str]*) – Name of the graphic type
- **graphic_data** (*numpy.ndarray[numpy.float]*) – Array of spatial coordinates, where each row of the array represents a (x, y, z) coordinate triplet
- **frame_of_reference_uid** (*Union[highdicom.UID, str]*) – Unique identifier of the frame of reference within which the coordinates are defined
- **fiducial_uid** (*Union[str, None], optional*) – Unique identifier for the content item
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

property frame_of_reference_uid: *UID*

frame of reference UID

Type
highdicom.UID

Return type
highdicom.uid.UID

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD3D

Returns
Content Item

Return type
highdicom.sr.Scoord3DContentItem

property graphic_type: *GraphicTypeValues3D*

graphic type

Type
GraphicTypeValues3D

Return type
highdicom.sr.enum.GraphicTypeValues3D

property value: ndarray

n x 3 array of 3D spatial coordinates

Type
numpy.ndarray

Return type
numpy.ndarray

class highdicom.sr.**ScoordContentItem**(*name*, *graphic_type*, *graphic_data*, *pixel_origin_interpretation=None*, *fiducial_uid=None*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type SCOORD.

Note: Spatial coordinates are defined in image space and have pixel units.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **graphic_type** (*Union[highdicom.sr.GraphicTypeValues, str]*) – Name of the graphic type
- **graphic_data** (*numpy.ndarray*) – Array of ordered spatial coordinates, where each row of the array represents a (Column,Row) pair
- **pixel_origin_interpretation** (*Union[highdicom.sr.PixelOriginInterpretationValues, str, None], optional*) – Whether pixel coordinates specified by *graphic_data* are defined relative to the total pixel matrix (highdicom.sr.PixelOriginInterpretationValues.VOLUME) or relative to an individual frame (highdicom.sr.PixelOriginInterpretationValues.FRAME)

- **fiducial_uid** (*Union[highdicom.UID, str, None], optional*) – Unique identifier for the content item
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns
Content Item

Return type
highdicom.sr.ScoordContentItem

property graphic_type: *GraphicTypeValues*

graphic type

Type
GraphicTypeValues

Return type
highdicom.sr.enum.GraphicTypeValues

property value: ndarray

n x 2 array of 2D spatial coordinates

Type
numpy.ndarray

Return type
numpy.ndarray

class highdicom.sr.**SourceImageForMeasurement**(*referenced_sop_class_uid*, *referenced_sop_instance_uid*, *referenced_frame_numbers=None*)

Bases: *ImageContentItem*

Content item representing a reference to an image that was used as a source for a measurement.

Parameters

- **referenced_sop_class_uid** (*str*) – SOP Class UID of the referenced image object
- **referenced_sop_instance_uid** (*str*) – SOP Instance UID of the referenced image object
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Raises
ValueError – If any referenced frame number is not a positive integer

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type IMAGE

Returns
Constructed object

Return type
highdicom.sr.SourceImageForMeasurement

classmethod from_source_image(*image*, *referenced_frame_numbers=None*)

Construct the content item directly from an image dataset

Parameters

- **image** (*pydicom.dataset.Dataset*) – Dataset representing the image to be referenced
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Returns
Content item representing a reference to the image dataset

Return type
highdicom.sr.SourceImageForMeasurement

class highdicom.sr.**SourceImageForMeasurementGroup**(*referenced_sop_class_uid*, *referenced_sop_instance_uid*, *referenced_frame_numbers=None*)

Bases: *ImageContentItem*

Content item representing a reference to an image that was used as a source.

Parameters

- **referenced_sop_class_uid** (*str*) – SOP Class UID of the referenced image object
- **referenced_sop_instance_uid** (*str*) – SOP Instance UID of the referenced image object
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Raises
ValueError – If any referenced frame number is not a positive integer

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type IMAGE

Returns
Constructed object

Return type
highdicom.sr.SourceImageForMeasurementGroup

classmethod from_source_image(*image*, *referenced_frame_numbers=None*)

Construct the content item directly from an image dataset

Parameters

- **image** (*pydicom.dataset.Dataset*) – Dataset representing the image to be referenced
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Returns

Content item representing a reference to the image dataset

Return type

highdicom.sr.SourceImageForMeasurementGroup

class highdicom.sr.**SourceImageForRegion**(*referenced_sop_class_uid*, *referenced_sop_instance_uid*, *referenced_frame_numbers=None*)

Bases: *ImageContentItem*

Content item representing a reference to an image that was used as a source for a region.

Parameters

- **referenced_sop_class_uid** (*str*) – SOP Class UID of the referenced image object
- **referenced_sop_instance_uid** (*str*) – SOP Instance UID of the referenced image object
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Raises

ValueError – If any referenced frame number is not a positive integer

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters

dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns

Constructed object

Return type

highdicom.sr.SourceImageForRegion

classmethod from_source_image(*image*, *referenced_frame_numbers=None*)

Construct the content item directly from an image dataset

Parameters

- **image** (*pydicom.dataset.Dataset*) – Dataset representing the image to be referenced
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Returns

Content item representing a reference to the image dataset

Return type
highdicom.sr.SourceImageForRegion

class highdicom.sr.**SourceImageForSegmentation**(*referenced_sop_class_uid*, *referenced_sop_instance_uid*, *referenced_frame_numbers=None*)

Bases: *ImageContentItem*

Content item representing a reference to an image that was used as the source for a segmentation.

Parameters

- **referenced_sop_class_uid** (*str*) – SOP Class UID of the referenced image object
- **referenced_sop_instance_uid** (*str*) – SOP Instance UID of the referenced image object
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Raises
ValueError – If any referenced frame number is not a positive integer

classmethod **from_dataset**(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns
Constructed object

Return type
highdicom.sr.SourceImageForSegmentation

classmethod **from_source_image**(*image*, *referenced_frame_numbers=None*)

Construct the content item directly from an image dataset

Parameters

- **image** (*pydicom.dataset.Dataset*) – Dataset representing the image to be referenced
- **referenced_frame_numbers** (*Union[Sequence[int], None], optional*) – numbers of the frames to which the reference applies in case the referenced image is a multi-frame image

Returns
Content item representing a reference to the image dataset

Return type
highdicom.sr.SourceImageForSegmentation

class highdicom.sr.**SourceSeriesForSegmentation**(*referenced_series_instance_uid*)

Bases: *UIDRefContentItem*

Content item representing a reference to a series of images that was used as the source for a segmentation.

Parameters
referenced_series_instance_uid (*str*) – Series Instance UID

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD

Returns
Constructed object

Return type
highdicom.sr.SourceSeriesForSegmentation

classmethod from_source_image(*image*)

Construct the content item directly from an image dataset

Parameters
image (*pydicom.dataset.Dataset*) – dataset representing a single image from the series to be referenced

Returns
Content item representing a reference to the image dataset

Return type
highdicom.sr.SourceSeriesForSegmentation

class highdicom.sr.**SubjectContext**(*subject_class*, *subject_class_specific_context*)

Bases: Template

TID 1006 Subject Context

Parameters

- **subject_class** (highdicom.sr.CodedConcept) – type of subject if the subject of the report is not the patient (see CID 271 "Observation Subject Class" for options)
- **subject_class_specific_context** (*Union[*highdicom.sr.SubjectContextFetus, highdicom.sr.SubjectContextSpecimen, highdicom.sr.SubjectContextDevice*], optional*) – additional context information specific to *subject_class*

property subject_class: *CodedConcept*

type of subject

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property subject_class_specific_context: Union[*SubjectContextFetus***,** *SubjectContextSpecimen***,** *SubjectContextDevice***]**

Union[highdicom.sr.SubjectContextFetus, highdicom.sr.SubjectContextSpecimen, highdicom.sr.SubjectContextDevice]: subject class specific context

Return type
typing.Union[*highdicom.sr.templates.SubjectContextFetus*, *highdicom.sr.templates.SubjectContextSpecimen*, *highdicom.sr.templates.SubjectContextDevice*]

class highdicom.sr.**SubjectContextDevice**(*name*, *uid=None*, *manufacturer_name=None*, *model_name=None*, *serial_number=None*, *physical_location=None*)

Bases: Template

TID 1010 Subject Context Device

Parameters

- **name** (*str*) – name of the observed device
- **uid** (*Union[str, None], optional*) – unique identifier of the observed device
- **manufacturer_name** (*Union[str, None], optional*) – name of the observed device's manufacturer
- **model_name** (*Union[str, None], optional*) – name of the observed device's model
- **serial_number** (*Union[str, None], optional*) – serial number of the observed device
- **physical_location** (*str, optional*) – physical location of the observed device during the procedure

property **device_manufacturer_name: Optional[str]**

name of device manufacturer

Type
Union[str, None]

Return type
typing.Optional[str]

property **device_model_name: Optional[str]**

name of device model

Type
Union[str, None]

Return type
typing.Optional[str]

property **device_name: str**

name of device

Type
str

Return type
str

property **device_physical_location: Optional[str]**

location of device

Type
Union[str, None]

Return type
typing.Optional[str]

property **device_serial_number: Optional[str]**

device serial number

Type
Union[str, None]

Return type
typing.Optional[str]

property device_uid: Optional[str]

unique device identifier

Type
Union[str, None]

Return type
typing.Optional[str]

classmethod from_sequence(*sequence*, *is_root=False*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing SR Content Items of template TID 1010 "Subject Context, Device"
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns
Content Sequence containing SR Content Items

Return type
highdicom.sr.SubjectContextDevice

class highdicom.sr.**SubjectContextFetus**(*subject_id*)

Bases: Template

TID 1008 Subject Context Fetus

Parameters
subject_id (*str*) – identifier of the fetus for longitudinal tracking

classmethod from_sequence(*sequence*, *is_root=False*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing SR Content Items of template TID 1008 "Subject Context, Fetus"
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns
Content Sequence containing SR Content Items

Return type
highdicom.sr.SubjectContextFetus

property subject_id: str

subject identifier

Type
str

Return type
str

class highdicom.sr.**SubjectContextSpecimen**(*uid*, *identifier=None*, *container_identifier=None*, *specimen_type=None*)

Bases: Template

TID 1009 Subject Context Specimen

Parameters

- **uid** (*str*) – Unique identifier of the observed specimen
- **identifier** (*Union[str, None], optional*) – Identifier of the observed specimen (may have limited scope, e.g., only relevant with respect to the corresponding container)
- **container_identifier** (*Union[str, None], optional*) – Identifier of the container holding the speciment (e.g., a glass slide)
- **specimen_type** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Type of the specimen (see CID 8103 "Anatomic Pathology Specimen Types" for options)

property container_identifier: Optional[str]

specimen container identifier

Type
Union[str, None]

Return type
typing.Optional[str]

classmethod **from_sequence**(*sequence*, *is_root=False*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing SR Content Items of template TID 1009 "Subject Context, Specimen"
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns
Content Sequence containing SR Content Items

Return type
highdicom.sr.SubjectContextSpecimen

property specimen_identifier: Optional[str]

specimen identifier

Type
Union[str, None]

Return type
typing.Optional[str]

property specimen_type: Optional[*CodedConcept*]

type of specimen

Type
Union[*highdicom.sr.CodedConcept*, None]

Return type
typing.Optional[*highdicom.sr.coding.CodedConcept*]

property specimen_uid: str

unique specimen identifier

Type
str

Return type
str

class highdicom.sr.**TcoordContentItem**(*name*, *temporal_range_type*, *referenced_sample_positions=None*, *referenced_time_offsets=None*, *referenced_date_time=None*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type TCOORD.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **temporal_range_type** (*Union[highdicom.sr.TemporalRangeTypeValues, str]*) – Name of the temporal range type
- **referenced_sample_positions** (*Union[Sequence[int], None], optional*) – One-based relative sample position of acquired time points within the time series
- **referenced_time_offsets** (*Union[Sequence[float], None], optional*) – Seconds after start of the acquisition of the time series
- **referenced_date_time** (*Union[Sequence[datetime.datetime], None], optional*) – Absolute time points
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type TCOORD

Returns
Content Item

Return type
highdicom.sr.TcoordContentItem

property temporal_range_type: *TemporalRangeTypeValues*

temporal range type

Type
highdicom.sr.TemporalRangeTypeValues

Return type
highdicom.sr.enum.TemporalRangeTypeValues

property value: Union[List[int], List[float], List[datetime]]

time points

Type
Union[List[int], List[float], List[datetime.datetime]]

Return type
typing.Union[typing.List[int], typing.List[float], typing.List[datetime.datetime]]

class highdicom.sr.**TemporalRangeTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Temporal Range Type.

See C.18.7.1.1.

BEGIN = 'BEGIN'

A range that begins at the identified temporal point.

It extends beyond the end of the acquired data.

END = 'END'

A range that ends at the identified temporal point.

It begins before the start of the acquired data and extends to (and includes) the identified temporal point.

MULTIPOINT = 'MULTIPOINT'

Multiple temporal points.

MULTISEGMENT = 'MULTISEGMENT'

Multiple segments, each denoted by two temporal points.

POINT = 'POINT'

A single temporal point.

SEGMENT = 'SEGMENT'

A range between two temporal points.

class highdicom.sr.**TextContentItem**(*name*, *value*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type TEXT.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **value** (*str*) – Description of the concept in free text
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type TEXT

Returns
Content Item

Return type
highdicom.sr.TextContentItem

property value: str

text value

Type
str

Return type
str

class highdicom.sr.TimeContentItem(*name*, *value*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type TIME.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **value** (*Union[str, datetime.time, pydicom.valuerep.TM]*) – Time
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type TIME

Returns
Content Item

Return type
highdicom.sr.TimeContentItem

property value: time

time

Type
datetime.time

Return type
datetime.time

class highdicom.sr.TimePointContext(*time_point*, *time_point_type=None*, *time_point_order=None*, *subject_time_point_identifier=None*, *protocol_time_point_identifier=None*, *temporal_offset_from_event=None*)

Bases: Template

TID 1502 Time Point Context

Parameters

- **time_point** (*str*) – actual value representation of the time point

- **time_point_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – coded type of time point, e.g., "Baseline" or "Posttreatment" (see CID 6146 "Time Point Types" for options)
- **time_point_order** (*Union[int, None], optional*) – number indicating the order of a time point relative to other time points in a time series
- **subject_time_point_identifier** (*Union[str, None], optional*) – identifier of a specific time point in a time series, which is unique within an appropriate local context and specific to a particular subject (patient)
- **protocol_time_point_identifier** (*Union[str, None], optional*) – identifier of a specific time point in a time series, which is unique within an appropriate local context and specific to a particular protocol using the same value for different subjects
- **temporal_offset_from_event** (*Union[highdicom.sr.LongitudinalTemporalOffsetFromEvent, None], optional*) – offset in time from a particular event of significance, e.g., the baseline of an imaging study or enrollment into a clinical trial

class highdicom.sr.**TrackingIdentifier**(*uid=None*, *identifier=None*)

Bases: Template

TID 4108 Tracking Identifier

Parameters

- **uid** (*Union[highdicom.UID, str, None], optional*) – globally unique identifier
- **identifier** (*Union[str, None], optional*) – human readable identifier

class highdicom.sr.**UIDRefContentItem**(*name*, *value*, *relationship_type=None*)

Bases: *ContentItem*

DICOM SR document content item for value type UIDREF.

Parameters

- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code]*) – Concept name
- **value** (*Union[highdicom.UID, str]*) – Unique identifier
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*)

Construct object from an existing dataset.

Parameters
dataset (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type UIDREF

Returns
Content Item

Return type
highdicom.sr.UIDRefContentItem

property value: *UID*

UID

Type
highdicom.UID

Return type
highdicom.uid.UID

class highdicom.sr.**ValueTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Value Type.

See Table C.17.3.2.1.

CODE = 'CODE'

Coded expression of the concept.

COMPOSITE = 'COMPOSITE'

Reference to UIDs of Composite SOP Instances.

CONTAINER = 'CONTAINER'

The content of the CONTAINER.

The value of a CONTAINER Content Item is the collection of Content Items that it contains.

DATE = 'DATE'

Calendar date.

DATETIME = 'DATETIME'

Concatenated date and time.

IMAGE = 'IMAGE'

Reference to UIDs of Image Composite SOP Instances.

NUM = 'NUM'

Numeric value and associated Unit of Measurement.

PNAME = 'PNAME'

Name of person.

SCOORD = 'SCOORD'

Listing of spatial coordinates defined in 2D pixel matrix.

SCOORD3D = 'SCOORD3D'

Listing of spatial coordinates defined in 3D frame of reference.

TCOORD = 'TCOORD'

Listing of temporal coordinates.

TEXT = 'TEXT'

Textual expression of the concept.

TIME = 'TIME'

Time of day.

UIDREF = 'UIDREF'

Unique Identifier.

WAVEFORM = 'WAVEFORM'

Reference to UIDs of Waveform Composite SOP Instances.

class highdicom.sr.**VolumeSurface**(*graphic_type*, *graphic_data*, *frame_of_reference_uid*, *source_images=None*, *source_series=None*)

Bases: *ContentSequence*

Content sequence representing a volume surface in the the three-dimensional patient/slide coordinate system in millimeter unit.

Parameters

- **graphic_type** (*Union[highdicom.sr.GraphicTypeValues3D, str]*) – name of the graphic type. Permissible values are "ELLIPSOID", "POINT", "ELLIPSE" or "POLYGON".
- **graphic_data** (*Union[np.ndarray, Sequence[np.ndarray]]*) – List of graphic data for elements of the volume surface. Each item of the list should be a 2D numpy array representing the graphic data for a single element of type `graphic_type`.

  If *graphic_type* is `"ELLIPSOID"` or `"POINT"`, the volume surface will consist of a single element that defines the entire surface. Therefore, a single 2D NumPy array should be passed as a list of length 1 or as a NumPy array directly.

  If *graphic_type* is `"ELLIPSE"` or `"POLYGON"`, the volume surface will consist of two or more planar regions that together define the surface. Therefore a list of two or more 2D NumPy arrays should be passed.

  Each 2D NumPy array should have dimension N x 3 where each row of the array represents a coordinate in the 3D Frame of Reference. The number, N, and meaning of the coordinates depends upon the value of *graphic_type*. See *highdicom.sr.GraphicTypeValues3D* for details.
- **frame_of_reference_uid** (*str*) – unique identifier of the frame of reference within which the coordinates are defined
- **source_images** (*Union[Sequence[highdicom.sr.SourceImageForSegmentation], None], optional*) – source images for segmentation
- **source_series** (*Union[highdicom.sr.SourceSeriesForSegmentation, None], optional*) – source series for segmentation

Note: Either one or more source images or one source series must be provided.

property **frame_of_reference_uid**: *UID*

Frame of reference UID.

Type
highdicom.UID

Return type
highdicom.uid.UID

classmethod **from_sequence**(*sequence*)

Construct an object from an existing content sequence.

Parameters
sequence (*Sequence[Dataset]*) – Sequence of datasets to be converted. This is expected to contain one or more content items with concept name 'Volume Surface', and either a single content item with concept name 'Source Series For Segmentation', or 1 or more content items with concept name 'Source Image For Segmentation'.

Returns
Constructed VolumeSurface object, containing copies of the original content items.

Return type
highdicom.sr.VolumeSurface

property graphic_data: List[ndarray]

Union[np.ndarray, List[np.ndarray]]: Graphic data arrays that comprise the volume surface. For volume surfaces with graphic type "ELLIPSOID" or "POINT", this will be a single 2D Numpy array representing the graphic data. Otherwise, it will be a list of 2D Numpy arrays representing graphic data for each element of the volume surface.

Return type
typing.List[numpy.ndarray]

property graphic_type: *GraphicTypeValues3D*

Graphic type.

Type
highdicom.sr.GraphicTypeValues3D

Return type
highdicom.sr.enum.GraphicTypeValues3D

has_source_images()

Returns whether the object contains information about source images.

ReferencedSegment objects must either contain information about source images or source series (and not both).

Returns
True if the object contains information about source images. False if the image contains information about the source series.

Return type
bool

property source_images_for_segmentation: List[*SourceImageForSegmentation*]

List[highdicom.sr.SourceImageForSegmentation]: Source images for the volume surface.

Return type
typing.List[*highdicom.sr.content.SourceImageForSegmentation*]

property source_series_for_segmentation: Optional[*SourceSeriesForSegmentation*]

Optional[highdicom.sr.SourceSeriesForSegmentation]: Source series for the volume surface.

Return type
typing.Optional[*highdicom.sr.content.SourceSeriesForSegmentation*]

class highdicom.sr.**VolumetricROIMeasurementsAndQualitativeEvaluations**(*tracking_identifier*, *referenced_regions=None*, *referenced_volume_surface=None*, *referenced_segment=None*, *referenced_real_world_value_map=None*, *time_point_context=None*, *finding_type=None*, *method=None*, *algorithm_id=None*, *finding_sites=None*, *session=None*, *measurements=None*, *qualitative_evaluations=None*, *geometric_purpose=None*, *finding_category=None*)

Bases: _ROIMeasurementsAndQualitativeEvaluations

TID 1411 Volumetric ROI Measurements and Qualitative Evaluations

Parameters

- **tracking_identifier** (highdicom.sr.TrackingIdentifier) – Identifier for tracking measurements
- **referenced_regions** (*Union[Sequence[highdicom.sr.ImageRegion], None], optional*) – Regions of interest in source image(s)
- **referenced_volume_surface** (*Union[highdicom.sr.VolumeSurface, None], optional*) – Volume of interest in source image(s)
- **referenced_segment** (*Union[highdicom.sr.ReferencedSegment, None], optional*) – Segmentation for region of interest in source image
- **referenced_real_world_value_map** (*Union[highdicom.sr.RealWorldValueMap, None], optional*) – Referenced real world value map for region of interest
- **time_point_context** (*Union[highdicom.sr.TimePointContext, None], optional*) – Description of the time point context
- **finding_type** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Type of observed finding
- **method** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Coded measurement method (see CID 6147 "Response Criteria" for options)
- **algorithm_id** (*Union[highdicom.sr.AlgorithmIdentification, None], optional*) – Identification of algorithm used for making measurements
- **finding_sites** (*Union[Sequence[highdicom.sr.FindingSite], None], optional*) – Coded description of one or more anatomic locations at which the finding was observed
- **session** (*Union[str, None], optional*) – Description of the session

- **measurements** (*Union[Sequence[highdicom.sr.Measurement], None], optional*) – Measurements for a volume of interest
- **qualitative_evaluations** (*Union[Sequence[highdicom.sr.QualitativeEvaluation], None], optional*) – Coded name-value (question-answer) pairs that describe qualitative evaluations of a volume of interest
- **geometric_purpose** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Geometric interpretation of region of interest (see CID 219 "Geometry Graphical Representation" for options)
- **finding_category** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Category of observed finding, e.g., anatomic structure or morphologically abnormal structure

---

Note: Either a segmentation, a list of regions or volume needs to referenced together with the corresponding source image(s) or series.

---

classmethod from_sequence(*sequence*, *is_root=False*)

Construct object from a sequence of datasets.

Parameters

- **sequence** (*Sequence[pydicom.dataset.Dataset]*) – Datasets representing "Measurement Group" SR Content Items of Value Type CONTAINER (sequence shall only contain a single item)
- **is_root** (*bool, optional*) – Whether the sequence is used to contain SR Content Items that are intended to be added to an SR document at the root of the document content tree

Returns

Content Sequence containing root CONTAINER SR Content Item

Return type

highdicom.sr.VolumetricROIMeasurementsAndQualitativeEvaluations

property reference_type: Code

pydicom.sr.coding.Code

The "type" of the ROI reference as a coded concept. This will be one of the following coded concepts from the DCM coding scheme:

- Image Region
- Referenced Segment
- Volume Surface
- Region In Space

Return type

pydicom.sr.coding.Code

property referenced_segment: Optional[*ReferencedSegment*]

Union[highdicom.sr.ImageContentItem, None]: segmentation frame referenced by the measurements group

Return type

typing.Optional[*highdicom.sr.content.ReferencedSegment*]

property roi: Optional[Union[*VolumeSurface*, List[*ImageRegion*]]]

Union[highdicom.sr.VolumeSurface, List[highdicom.sr.ImageRegion], None]: volume surface or image regions defined by spatial coordinates

Return type

typing.Union[*highdicom.sr.content.VolumeSurface*, typing.List[*highdicom.sr.content.ImageRegion*], None]

8.8.1 highdicom.sr.utils module

Utilities for working with SR document instances.

highdicom.sr.utils.**collect_evidence**(*evidence*, *content*)

Collect evidence for a SR document.

Any *evidence* that is referenced in *content* via IMAGE or COMPOSITE content items will be grouped together for inclusion in the Current Requested Procedure Evidence Sequence and all remaining evidence will be grouped for potential inclusion in the Pertinent Other Evidence Sequence.

Parameters

- **evidence** (*List[pydicom.dataset.Dataset]*) – Metadata of instances that serve as evidence for the SR document content
- **content** (*pydicom.dataset.Dataset*) – SR document content

Return type

typing.Tuple[typing.List[pydicom.dataset.Dataset], typing.List[pydicom.dataset.Dataset]]

Returns

- **current_requested_procedure_evidence** (*List[pydicom.dataset.Dataset]*) – Items of the Current Requested Procedure Evidence Sequence
- **other_pertinent_evidence** (*List[pydicom.dataset.Dataset]*) – Items of the Pertinent Other Evidence Sequence

Raises

ValueError – When a SOP instance is referenced in *content* but not provided as *evidence*

highdicom.sr.utils.**find_content_items**(*dataset*, *name=None*, *value_type=None*, *relationship_type=None*, *recursive=False*)

Finds content items in a Structured Report document that match a given query.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – SR document instance
- **name** (*Union[highdicom.sr.CodedConcept, pydicom.sr.coding.Code, None], optional*) – Coded name that items should have
- **value_type** (*Union[highdicom.sr.ValueTypeValues, str, None], optional*) – Type of value that items should have (e.g. highdicom.sr.ValueTypeValues.CONTAINER)
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship that items should have with its parent (e.g. highdicom.sr.RelationshipTypeValues.CONTAINS)

- **recursive** (*bool, optional*) – Whether search should be performed recursively, i.e. whether contained child content items should also be queried

Returns
flat list of all content items that matched the query

Return type
List[pydicom.dataset.Dataset]

Raises
AttributeError – When data set does not contain Content Sequence attribute.

highdicom.sr.utils.**get_coded_name**(*item*)

Gets the concept name of a SR Content Item.

Parameters
item (*pydicom.dataset.Dataset*) – Content Item

Returns
Concept name

Return type
highdicom.sr.CodedConcept

highdicom.sr.utils.**get_coded_value**(*item*)

Gets the value of a SR Content Item with Value Type CODE.

Parameters
item (*pydicom.dataset.Dataset*) – Content Item

Returns
Value

Return type
highdicom.sr.CodedConcept

8.9 highdicom.sc package

Package for creation of Secondary Capture (SC) Image instances.

class highdicom.sc.**ConversionTypeValues**(*value*)

Bases: Enum

Enumerated values for attribute Conversion Type.

DF = 'DF'

Digitized Film

DI = 'DI'

Digital Interface

DRW = 'DRW'

Drawing

DV = 'DV'

Digitized Video

SD = 'SD'

Scanned Document

SI = 'SI'

Scanned Image

SYN = 'SYN'

Synthetic Image

WSD = 'WSD'

Workstation

class highdicom.sc.**SCImage**(*pixel_array*, *photometric_interpretation*, *bits_allocated*, *coordinate_system*, *study_instance_uid*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *patient_id=None*, *patient_name=None*, *patient_birth_date=None*, *patient_sex=None*, *accession_number=None*, *study_id=None*, *study_date=None*, *study_time=None*, *referring_physician_name=None*, *pixel_spacing=None*, *laterality=None*, *patient_orientation=None*, *anatomical_orientation_type=None*, *container_identifier=None*, *issuer_of_container_identifier=None*, *specimen_descriptions=None*, *transfer_syntax_uid='1.2.840.10008.1.2.1'*, ***kwargs*)

Bases: *SOPClass*

SOP class for a Secondary Capture (SC) Image, which represents a single-frame image that was converted from a non-DICOM format.

Parameters

- **pixel_array** (*numpy.ndarray*) – Array of unsigned integer pixel values representing a single-frame image; either a 2D grayscale image or a 3D color image (RGB color space)
- **photometric_interpretation** (*Union[str, highdicom.PhotometricInterpretationValues]*) – Interpretation of pixel data; either "MONOCHROME1" or "MONOCHROME2" for 2D grayscale images or "RGB" or "YBR_FULL" for 3D color images
- **bits_allocated** (*int*) – Number of bits that should be allocated per pixel value
- **coordinate_system** (*Union[str, highdicom.CoordinateSystemNames]*) – Subject ("PATIENT" or "SLIDE") that was the target of imaging
- **study_instance_uid** (*str*) – Study Instance UID
- **series_instance_uid** (*str*) – Series Instance UID of the SC image series
- **series_number** (*int*) – Series Number of the SC image series
- **sop_instance_uid** (*str*) – SOP instance UID that should be assigned to the SC image instance
- **instance_number** (*int*) – Number that should be assigned to this SC image instance
- **manufacturer** (*str*) – Name of the manufacturer of the device that creates the SC image instance (in a research setting this is typically the same as *institution_name*)
- **patient_id** (*Union[str, None], optional*) – ID of the patient (medical record number)
- **patient_name** (*Union[str, PersonName, None], optional*) – Name of the patient
- **patient_birth_date** (*Union[str, None], optional*) – Patient's birth date
- **patient_sex** (*Union[str, highdicom.PatientSexValues, None], optional*) – Patient's sex

- **study_id** (*Union[str, None], optional*) – ID of the study
- **accession_number** (*Union[str, None], optional*) – Accession number of the study
- **study_date** (*Union[str, datetime.date, None], optional*) – Date of study creation
- **study_time** (*Union[str, datetime.time, None], optional*) – Time of study creation
- **referring_physician_name** (*Union[str, pydicom.valuerep.PersonName, None], optional*) – Name of the referring physician
- **pixel_spacing** (*Union[Tuple[float, float], None], optional*) – Physical spacing in millimeter between pixels along the row and column dimension
- **laterality** (*Union[str, highdicom.LateralityValues, None], optional*) – Laterality of the examined body part
- **patient_orientation** (typing.Union[typing.Tuple[str, str], typing.Tuple[*highdicom.enum.PatientOrientationValuesBiped*, *highdicom.enum.PatientOrientationValuesBiped*], typing.Tuple[*highdicom.enum.PatientOrientationValuesQuadruped*, *highdicom.enum.PatientOrientationValuesQuadruped*], None]) – Union[Tuple[str, str], Tuple[highdicom.PatientOrientationValuesBiped, highdicom.PatientOrientationValuesBiped], Tuple[highdicom.PatientOrientationValuesQuadruped, highdicom.PatientOrientationValuesQuadruped], None], optional Orientation of the patient along the row and column axes of the image (required if *coordinate_system* is "PATIENT")
- **anatomical_orientation_type** (*Union[str, highdicom.AnatomicalOrientationTypeValues, None], optional*) – Type of anatomical orientation of patient relative to image (may be provide if *coordinate_system* is "PATIENT" and patient is an animal)
- **container_identifier** (*Union[str, None], optional*) – Identifier of the container holding the specimen (required if *coordinate_system* is "SLIDE")
- **issuer_of_container_identifier** (*Union[highdicom.IssuerOfIdentifier, None], optional*) – Issuer of *container_identifier*
- **specimen_descriptions** (*Union[Sequence[highdicom.SpecimenDescriptions], None], optional*) – Description of each examined specimen (required if *coordinate_system* is "SLIDE")
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements. The following compressed transfer syntaxes are supported: RLE Lossless ("1.2.840.10008.1.2.5"), JPEG 2000 Lossless ("1.2.840.10008.1.2.4.90"), JPEG-LS Lossless ("1.2.840.10008.1.2.4.80"), and JPEG Baseline ("1.2.840.10008.1.2.4.50"). Note that JPEG Baseline is a lossy compression method that will lead to a loss of detail in the image.
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

classmethod from_ref_dataset(*ref_dataset*, *pixel_array*, *photometric_interpretation*, *bits_allocated*, *coordinate_system*, *series_instance_uid*, *series_number*, *sop_instance_uid*, *instance_number*, *manufacturer*, *pixel_spacing=None*, *laterality=None*, *patient_orientation=None*, *anatomical_orientation_type=None*, *container_identifier=None*, *issuer_of_container_identifier=None*, *specimen_descriptions=None*, *transfer_syntax_uid='1.2.840.10008.1.2'*, ***kwargs*)

Constructor that copies patient and study from an existing dataset.

This provides a more concise way to construct an SCImage when an existing reference dataset from the study is available. All patient- and study- related attributes required by the main constructor are copied from the `ref_dataset`, if present.

The `ref_dataset` may be any dataset from the study to which the resulting SC image should belong, and contain all the relevant patient and study metadata. It does not need to be specifically related to the contents of the SCImage.

Parameters

- **ref_dataset** (*pydicom.dataset.Dataset*) – An existing dataset from the study to which the SCImage should belong. Patient- and study-related metadata will be copied from this dataset.
- **pixel_array** (*numpy.ndarray*) – Array of unsigned integer pixel values representing a single-frame image; either a 2D grayscale image or a 3D color image (RGB color space)
- **photometric_interpretation** (*Union[str, highdicom.enum.PhotometricInterpretationValues]*) – Interpretation of pixel data; either "MONOCHROME1" or "MONOCHROME2" for 2D grayscale images or "RGB" or "YBR_FULL" for 3D color images
- **bits_allocated** (*int*) – Number of bits that should be allocated per pixel value
- **coordinate_system** (*Union[str, highdicom.enum.CoordinateSystemNames]*) – Subject ("PATIENT" or "SLIDE") that was the target of imaging
- **series_instance_uid** (*str*) – Series Instance UID of the SC image series
- **series_number** (*int*) – Series Number of the SC image series
- **sop_instance_uid** (*str*) – SOP instance UID that should be assigned to the SC image instance
- **instance_number** (*int*) – Number that should be assigned to this SC image instance
- **manufacturer** (*str*) – Name of the manufacturer of the device that creates the SC image instance (in a research setting this is typically the same as *institution_name*)
- **pixel_spacing** (*Union[Tuple[int, int]], optional*) – Physical spacing in millimeter between pixels along the row and column dimension
- **laterality** (*Union[str, highdicom.enum.LateralityValues, None], optional*) – Laterality of the examined body part
- **patient_orientation** (typing.Union[typing.Tuple[str, str], typing.Tuple[*highdicom.enum.PatientOrientationValuesBiped*, *highdicom.enum.PatientOrientationValuesBiped*], typing.Tuple[*highdicom.enum.PatientOrientationValuesQuadruped*, *highdicom.enum.PatientOrientationValuesQuadruped*], None]) – Union[Tuple[str, str], Tuple[highdicom.enum.PatientOrientationValuesBiped, highdicom.enum.PatientOrientationValuesBiped], Tuple[highdicom.enum.PatientOrientationValuesQuadruped, highdicom.enum.PatientOrientationValuesQuadruped], None], optional Orientation of the patient along the row and column axes of the image (required if *coordinate_system* is "PATIENT")
- **anatomical_orientation_type** (*Union[str, highdicom.enum.AnatomicalOrientationTypeValues, None], optional*) – Type of anatomical

orientation of patient relative to image (may be provide if *coordinate_system* is "PATIENT" and patient is an animal)

- **container_identifier** (*Union[str, optional*) – Identifier of the container holding the specimen (required if *coordinate_system* is "SLIDE")
- **issuer_of_container_identifier** (*Union[*highdicom.IssuerOfIdentifier, *None], optional*) – Issuer of *container_identifier*
- **specimen_descriptions** (*Union[Sequence[highdicom.SpecimenDescriptions], None], optional*) – Description of each examined specimen (required if *coordinate_system* is "SLIDE")
- **transfer_syntax_uid** (*str, optional*) – UID of transfer syntax that should be used for encoding of data elements. The following lossless compressed transfer syntaxes are supported: RLE Lossless ("1.2.840.10008.1.2.5"), JPEG 2000 Lossless ("1.2.840.10008.1.2.4.90").
- ****kwargs** (*Any, optional*) – Additional keyword arguments that will be passed to the constructor of *highdicom.base.SOPClass*

Returns

Secondary capture image.

Return type

SCImage

CHAPTER

NINE

RELEASE NOTES

Brief release notes may be found on on Github. This page contains migration notes for major breaking changes to the library's API.

9.1 Deprecation of *add_segments* method

Prior to highdicom 0.8.0, it was possible to add further segments to *highdicom.seg.Segmentation* image after its construction using the *add_segments* method. This was found to produce incorrect Dimension Index Values if the empty frames did not match within all segments added.

To create the Dimension Index Values correctly, the constructor needs access to all segments in the image when it is first created. Therefore, the *add_segments* method was removed in highdicom 0.8.0. Instead, in highdicom 0.8.0 and later, multiple segments can be passed to the constructor by stacking their arrays along the fourth dimension.

Given code that adds segments like this, in highdicom 0.7.0 and earlier:

```
import numpy as np
import highdicom as hd

# Create initial segment mask and description
mask_1 = np.array(
    # ...
)
description_1 = hd.seg.SegmentDescription(
    # ...
)
seg = hd.seg.Segmentation(
    # ...
    pixel_array=mask_1,
    segment_descriptions=[description_1],
    # ...
)

# Create a second segment and add to the existing segmentation
mask_2 = np.array(
    # ...
)
description_2 = hd.seg.SegmentDescription(
    # ...
)
```

(continues on next page)

(continued from previous page)

```
seg.add_segments(
    # ...
    pixel_array=mask_2,
    segment_descriptions=[description_2],
    # ...
)
```

This can be migrated to highdicom 0.8.0 and later by concatenating the arrays along the fourth dimension and calling the constructor at the end.

```
import numpy as np
import highdicom as hd

# Create initial segment mask and description
mask_1 = np.array(
    # ...
)
description_1 = hd.seg.SegmentDescription(
    # ...
)

# Create a second segment and description
mask_2 = np.array(
    # ...
)
description_2 = hd.seg.SegmentDescription(
    # ...
)

combined_segments = np.concatenate([mask_1, mask_2], axis=-1)
combined_descriptions = [description_1, description_2]

seg = hd.seg.Segmentation(
    # ...
    pixel_array=combined_segments,
    segment_descriptions=combined_descriptions,
    # ...
)
```

Note that segments must always be stacked down the fourth dimension (with index 3) of the `pixel_array`. In order to create a segmentation with multiple segments for a single source frame, it is required to add a new dimension (with length 1) as the first dimension (index 0) of the array.

9.2 Correct coordinate mapping

Prior to highdicom 0.14.1, mappings between image coordinates and reference coordinates did not take into account that there are two image coordinate systems, which are shifted by 0.5 pixels.

1. **Pixel indices**: (column, row) indices into the pixel matrix. The values are zero-based integers in the range [0, Columns - 1] and [0, Rows - 1]. Pixel indices are defined relative to the centers of pixels and the (0, 0) index is located at the center of the top left corner hand pixel of the total pixel matrix.
2. **Image coordinates**: (column, row) coordinates in the pixel matrix at sub-pixel resolution. The values are floating-point numbers in the range [0, Columns] and [0, Rows]. Image coordinates are defined relative to the top left corner of the pixels and the (0.0, 0.0) point is located at the top left corner of the top left corner hand pixel of the total pixel matrix.

To account for these differences, introduced two additional transformer classes in highdicom 0.14.1. and made changes to the existing ones. The existing transformer class now map between image coordinates and reference coordinates (*highdicom.spatial.ImageToReferenceTransformer* and *highdicom.spatial.ReferenceToImageTransformer*). While the new transformer classes map between pixel indices and reference coordinates (*highdicom.spatial.PixelToReferenceTransformer* and *highdicom.spatial.ReferenceToPixelTransformer*). Note that you want to use the former classes for converting between spatial coordinates (SCOORD) (*highdicom.sr.ScoordContentItem*) and 3D spatial coordinates (SCOORD3D) (*highdicom.sr.Scoord3DContentItem*) and the latter for determining the position of a pixel in the frame of reference or for projecting a coordinate in the frame of reference onto the image plane.

To make the distinction between pixel indices and image coordinates as clear as possible, we renamed the parameter of the *highdicom.spatial.map_pixel_into_coordinate_system()* function from `coordinate` to `index` and enforce that the values that are provided via the argument are integers rather than floats. In addition, the return value of *highdicom.spatial.map_coordinate_into_pixel_matrix()* is now a tuple of integers.

9.3 Deprecation of *processing_type* parameter

In highdicom 0.15.0, the `processing_type` parameter was removed from the constructor of *highdicom.content.SpecimenPreparationStep*. The parameter turned out to be superfluous, because the argument could be derived from the type of the `processing_procedure` argument.

9.4 Refactoring of *SpecimenPreparationStep* class

In highdicom 0.16.0 and later versions, *highdicom.content.SpecimenPreparationStep* represents an item of the Specimen Preparation Sequence rather than the Specimen Preparation Step Content Item Sequence and the class is consequently derived from `pydicom.dataset.Dataset` instead of `pydicom.sequence.Sequence`. As a consequence, alternative construction of an instance of *highdicom.content.SpecimenPreparationStep* needs to be performed using the `from_dataset()` instead of the `from_sequence()` class method.

9.5 Deprecation of Big Endian Transfer Syntaxes

The use of "Big Endian" transfer syntaxes such as *ExplicitVRBigEndian* is disallowed from highdicom 0.18.0 onwards. The use of Big Endian transfer syntaxes has been retired in the standard for some time. To discourage the use of retired transfer syntaxes and to simplify the logic when encoding and decoding objects in which byte order is relevant, in version 0.17.0 and onwards passing a big endian transfer syntax to the constructor of `highdicom.SOPClass` or any of its subclasses will result in a value error.

Similarly, as of highdicom 0.18.0, it is no longer possible to pass datasets with a Big Endian transfer syntax to the *from_dataset* methods of any of the `highdicom.SOPClass` subclasses.

9.6 Change in MeasurementReport constructor for TID 1601 enhancement

A breaking change was made after highdicom 0.18.4 in the creation of Image Library TID 1601 objects. Previously the Imag Library was constructed by explicitly passing a *pydicom.sequence.Sequence* of *ImageLibraryEntryDescriptors* objects to the `highdicom.sr.MeasurementReport` constructor in the *image_library_groups* argument. Now a *pydicom.sequence.Sequence* of *pydicom.dataset.Dataset* objects is passed in the *referenced_images* argument and the ImageLibrary components are created internally by highdicom. This standardizes the content of the Image Library subcomponents.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z