highdicom Documentation

Release 0.22.0

Markus D. Herrmann

Apr 11, 2024

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/imagingdatacommons/highdicom ~/highdicom
pip install ~/highdicom
```

CHAPTER

THREE

USER GUIDE

3.1 Quick Start

This section gives simple examples of how to create various types of DICOM object with *highdicom*. See *Information Object Definitions (IODs)* for more detail on the options available within each.

3.1.1 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',
```

(continues on next page)

(continued from previous page)

```
    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 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
```

(continues on next page)

(continued from previous page)

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

For more information see *Segmentation (SEG) Images*.

3.1.2 Parsing Segmentation (SEG) images

Finding relevant segments in a segmentation image instance and retrieving masks for them:

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

# Read SEG Image data set from PS3.10 files on disk into a Segmentation
# object
# This example is a test file in the highdicom git repository
```

(continues on next page)

(continued from previous page)

```
seg = hd.seg.segread('data/test_files/seg_image_ct_binary_overlap.dcm')

# Check the number of segments
assert seg.number_of_segments == 2

# Find segments (identified by their segment number) that have segmented
# property type "Bone"
bone_segment_numbers = seg.get_segment_numbers(
    segmented_property_type=codes.SCT.Bone
)
assert bone_segment_numbers ==  [1]

# List SOP Instance UIDs of the images from which the segmentation was
# derived
for study_uid, series_uid, sop_uid in seg.get_source_image_uids():
    print(study_uid, series_uid, sop_uid)
    # '1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.1, 1.3.6.1.4.1.5962.1.1.0.0.0.
↪1196530851.28319.0.2, 1.3.6.1.4.1.5962.1.1.0.0.0.1196530851.28319.0.93'
    # ...

# Here is a list of known SOP Instance UIDs that are a subset of those
# from which the segmentation was derived
source_image_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',
]

# Retrieve a binary segmentation mask for these images for the bone segment
mask = seg.get_pixels_by_source_instance(
    source_sop_instance_uids=source_image_uids,
    segment_numbers=bone_segment_numbers,
)
# Output is a numpy array of shape (instances x rows x columns x segments)
assert mask.shape == (2, 16, 16, 1)
assert np.unique(mask).tolist() == [0, 1]

# Alternatively, retrieve the segmentation mask for the full list of segments
# (2 in this case), and combine the resulting array into a "label mask", where
# pixel value represents segment number
mask = seg.get_pixels_by_source_instance(
    source_sop_instance_uids=source_image_uids,
    combine_segments=True,
    skip_overlap_checks=True,  # the segments in this image overlap
)
# Output is a numpy array of shape (instances x rows x columns)
assert mask.shape == (2, 16, 16)
assert np.unique(mask).tolist() == [0, 1, 2]
```

For more information see *Segmentation (SEG) Images*.

3.1.3 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,
)

# 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
)
```

(continues on next page)

(continued from previous page)

```
# 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
    )
]

# 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,
    series_number=1,
    series_instance_uid=hd.UID(),
```

(continues on next page)

(continued from previous page)

```
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Manufacturer'
)

print(sr_dataset)
```

For more information see *Structured Report (SR) Overview* and *The TID1500 Measurement Report Template*.

3.1.4 Parsing Structured Report (SR) documents

Highdicom has special support for parsing structured reports conforming to the TID1500 "Measurement Report" template using specialized Python classes for templates.

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

# This example is in the highdicom test data files in the repository
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# First we explore finding measurement groups. There are three types of
# measurement groups (image measurement, planar roi measurement groups, and
# volumetric roi measurement groups)

# Get a list of all image measurement groups referencing an image with a
# particular SOP Instance UID
groups = sr.content.get_image_measurement_groups(
    referenced_sop_instance_uid="1.3.6.1.4.1.5962.1.1.1.1.1.20040119072730.12322",
)
assert len(groups) == 1

# Get a list of all image measurement groups with a particular tracking UID
groups = sr.content.get_image_measurement_groups(
    tracking_uid="1.2.826.0.1.3680043.10.511.3.77718622501224431322963356892468048",
)
assert len(groups) == 1

# Get a list of all planar ROI measurement groups with finding type "Nodule"
# AND finding site "Lung"
groups = sr.content.get_planar_roi_measurement_groups(
    finding_type=codes.SCT.Nodule,
    finding_site=codes.SCT.Lung,
)
assert len(groups) == 1

# Get a list of all volumetric ROI measurement groups (with no filters)
groups = sr.content.get_volumetric_roi_measurement_groups()
assert len(groups) == 1

# Get a list of all planar ROI measurement groups with graphic type CIRCLE
```

(continues on next page)

(continued from previous page)

```
groups = sr.content.get_planar_roi_measurement_groups(
    graphic_type=hd.sr.GraphicTypeValues.CIRCLE,
)
assert len(groups) == 1

# Get a list of all planar ROI measurement groups stored as regions
groups = sr.content.get_planar_roi_measurement_groups(
    reference_type=codes.DCM.ImageRegion,
)
assert len(groups) == 2

# Get a list of all volumetric ROI measurement groups stored as volume
# surfaces
groups = sr.content.get_volumetric_roi_measurement_groups(
    reference_type=codes.DCM.VolumeSurface,
)
assert len(groups) == 1

# Next, we explore the properties of measurement groups that can
# be conveniently accessed with Python properties

# Use the first (only) image measurement group as an example
group = sr.content.get_image_measurement_groups()[0]

# tracking_identifier returns a Python str
assert group.tracking_identifier == "Image0001"

# tracking_uid returns a hd.UID, a subclass of str
assert group.tracking_uid == "1.2.826.0.1.3680043.10.511.3.
↪77718622501224431322963356892468048"

# source_images returns a list of hd.sr.SourceImageForMeasurementGroup,
# which in turn have some properties to access data
assert isinstance(group.source_images[0], hd.sr.SourceImageForMeasurementGroup)
ref_sop_uid = group.source_images[0].referenced_sop_instance_uid
assert ref_sop_uid == "1.3.6.1.4.1.5962.1.1.1.1.1.20040119072730.12322"

# for the various optional pieces of information in a measurement, accessing
# the relevant property returns None if the information is not present
assert group.finding_type is None

# Now use the first planar ROI group as a second example
group = sr.content.get_planar_roi_measurement_groups()[0]

# finding_type returns a CodedConcept
assert group.finding_type == codes.SCT.Nodule

# finding_sites returns a list of hd.sr.FindingSite objects
assert isinstance(group.finding_sites[0], hd.sr.FindingSite)
# the value of a finding site is a CodedConcept
assert group.finding_sites[0].value == codes.SCT.Lung
```

(continues on next page)

(continued from previous page)

```
# reference_type returns a CodedConcept (the same values used above for
# filtering)
assert group.reference_type == codes.DCM.ImageRegion

# since this has reference type ImageRegion, we can access the referenced
# using 'roi', which will return an hd.sr.ImageRegion object
assert isinstance(group.roi, hd.sr.ImageRegion)

# the graphic type and actual ROI coordinates (as a numpy array) can be
# accessed with the graphic_type and value properties of the roi
assert group.roi.graphic_type == hd.sr.GraphicTypeValues.CIRCLE
assert isinstance(group.roi.value, np.ndarray)
assert group.roi.value.shape == (2, 2)

# Next, we explore getting individual measurements out of measurement
# groups

# Use the first planar measurement group as an example
group = sr.content.get_planar_roi_measurement_groups()[0]

# Get a list of all measurements
measurements = group.get_measurements()

# Get the first measurements for diameter
measurement = group.get_measurements(name=codes.SCT.Diameter)[0]

# Access the measurement's name
assert measurement.name == codes.SCT.Diameter

# Access the measurement's value
assert measurement.value == 10.0

# Access the measurement's unit
assert measurement.unit == codes.UCUM.mm

# Get the diameter measurement in this group
evaluation = group.get_qualitative_evaluations(
    name=codes.DCM.LevelOfSignificance
)[0]

# Access the measurement's name
assert evaluation.name == codes.DCM.LevelOfSignificance

# Access the measurement's value
assert evaluation.value == codes.SCT.NotSignificant
```

For more information see *Parsing Measurement Reports*.

Additionally, there are low-level utilities that you can use to find content items in the content tree of any structured report documents:

```
from pathlib import Path
```

(continues on next page)

(continued from previous page)

```
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 "Measurement Report"
if sr_dataset.ContentTemplateSequence[0].TemplateIdentifier == 'TID1500':
    # Determine who made the observations reported in the document
    observers = hd.sr.utils.find_content_items(
        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)
```

3.1.5 Creating Microscopy Bulk Simple Annotation (ANN) objects

Microscopy Bulk Simple Annotations store large numbers of annotations of objects in microscopy images in a space-efficient way.

```
from pydicom import dcmread
from pydicom.sr.codedict import codes
from pydicom.sr.coding import Code
import highdicom as hd
import numpy as np

# Load a slide microscopy image from the highdicom test data (if you have
# cloned the highdicom git repo)
sm_image = dcmread('data/test_files/sm_image.dcm')

# Graphic data containing two nuclei, each represented by a single point
# expressed in 2D image coordinates
graphic_data = [
    np.array([[34.6, 18.4]]),
    np.array([[28.7, 34.9]]),
]

# You may optionally include measurements corresponding to each annotation
# This is a measurement object representing the areas of each of the two
# nuclei
area_measurement = hd.ann.Measurements(
    name=codes.SCT.Area,
    unit=codes.UCUM.SquareMicrometer,
    values=np.array([20.4, 43.8]),
)

# An annotation group represents a single set of annotations of the same
# type. Multiple such groups may be included in a bulk annotations object
# This group represents nuclei annotations produced by a manual "algorithm"
nuclei_group = hd.ann.AnnotationGroup(
    number=1,
    uid=hd.UID(),
    label='nuclei',
    annotated_property_category=codes.SCT.AnatomicalStructure,
    annotated_property_type=Code('84640000', 'SCT', 'Nucleus'),
    algorithm_type=hd.ann.AnnotationGroupGenerationTypeValues.MANUAL,
    graphic_type=hd.ann.GraphicTypeValues.POINT,
    graphic_data=graphic_data,
    measurements=[area_measurement],
)

# Include the annotation group in a bulk annotation object
bulk_annotations = hd.ann.MicroscopyBulkSimpleAnnotations(
    source_images=[sm_image],
    annotation_coordinate_type=hd.ann.AnnotationCoordinateTypeValues.SCOORD,
    annotation_groups=[nuclei_group],
    series_instance_uid=hd.UID(),
    series_number=10,
```

(continues on next page)

(continued from previous page)

```
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='MGH Pathology',
    manufacturer_model_name='MGH Pathology Manual Annotations',
    software_versions='0.0.1',
    device_serial_number='1234',
    content_description='Nuclei Annotations',
)

bulk_annotations.save_as('nuclei_annotations.dcm')
```

For more information see *Microscopy Bulk Simple Annotation (ANN) Objects*.

3.1.6 Parsing Microscopy Bulk Simple Annotation (ANN) objects

The following example demonstrates loading in a small bulk microscopy annotations file, finding an annotation group representing annotation of nuclei, and extracting the graphic data for the annotation as well as the area measurements corresponding to those annotations.

```
from pydicom import dcmread
from pydicom.sr.codedict import codes
from pydicom.sr.coding import Code
import highdicom as hd

# Load a bulk annotation file and convert to highdicom object
ann_dataset = dcmread('data/test_files/sm_annotations.dcm')
ann = hd.ann.MicroscopyBulkSimpleAnnotations.from_dataset(ann_dataset)

# Search for annotation groups by filtering for annotated property type of
# 'nucleus', and take the first such group
group = ann.get_annotation_groups(
    annotated_property_type=Code('84640000', 'SCT', 'Nucleus'),
)[0]

# Determine the graphic type and the number of annotations
assert group.number_of_annotations == 2
assert group.graphic_type == hd.ann.GraphicTypeValues.POINT

# Get the graphic data as a list of numpy arrays, we have to pass the
# coordinate type from the parent object here
graphic_data = group.get_graphic_data(
    coordinate_type=ann.AnnotationCoordinateType
)

# For annotations of graphic type "POINT" and coordinate type "SCOORD" (2D
# image coordinates), each annotation is a (1 x 2) NumPy array
assert graphic_data[0].shape == (1, group.number_of_annotations)

# Annotations may also optionally contain measurements
names, values, units = group.get_measurements(name=codes.SCT.Area)
```

(continues on next page)

(continued from previous page)

```
# The name and the unit are returned as a list of CodedConcepts
# and the values are returned in a numpy array of shape (number of
# annotations x number of measurements)
assert names[0] == codes.SCT.Area
assert units[0] == codes.UCUM.SquareMicrometer
assert values.shape == (group.number_of_annotations, 1)
```

For more information see *Microscopy Bulk Simple Annotation (ANN) Objects*.

3.1.7 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')

# 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
```

(continues on next page)

(continued from previous page)

```
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,
    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
```

(continues on next page)

(continued from previous page)

```
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']

    # 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'))
```

3.1.8 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
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
```

(continues on next page)

(continued from previous page)

```
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')
```

3.2 General Concepts

This section covers topics that are generally applicable across various parts of the library.

3.2.1 Coding

"Coding" is a key concept used throughout *highdicom*. By "coding", we are referring to the use of standardized nomenclatures or terminologies to describe medical (or related) concepts. For example, instead of using the English word "liver" (or a word in another human language) to describe the liver, we instead use a code such as '10200004' from the SNOMED-CT nomenclature to describe the liver in standardized way. Use of coding is vital to ensure that these concepts are expressed unambiguously within DICOM files. Coding is especially fundamental within structured reporting, but is also found in other places around the DICOM standard and, in turn, *highdicom*.

To communicate a concept in DICOM using a coding scheme, three elements are necessary:

- A **coding scheme**: an identifier of the pre-defined terminology used to define the concept.
- A code **value**: the code value conveys a unique identifier for the specific concept. It is often a number or alphanumeric string that may not have any inherent meaning outside of the terminology.

- A code **meaning**. The code meaning conveys the concept in a way that is understandable to humans.

Any coding scheme that operates in this way may be used within DICOM objects, including ones that you create yourself. However, it is highly recommended to use a well-known and widely accepted standard terminology to ensure that your DICOM objects will be as widely understood and as interoperable as possible. Examples of widely used medical terminologies include:

- The DCM terminology. This terminology is defined within the DICOM standard itself and is used to refer to DICOM concepts, as well as other concepts within the radiology workflow.
- SNOMED-CT. This terminology contains codes to describe medical concepts including anatomy, diseases, and procedures.
- RadLex. A standardized terminology for concepts in radiology.
- UCUM. A terminology specifically to describe units of measurement.

See this page for a list of terminologies used within DICOM.

Highdicom defines the *highdicom.sr.CodedConcept* class to encapsulate a coded concept. To create a *highdicom.sr.CodedConcept*, you pass values for the coding scheme, code value, and code meaning. For example, to describe a tumor using the SNOMED-CT terminology, you could do this:

```
import highdicom as hd

tumor_code = hd.sr.CodedConcept(
    value="108369006",
    scheme_designator="SCT",
    meaning="Tumor"
)
```

Codes within Pydicom

The *pydicom* library, upon which *highdicom* is built, has its own class `pydicom.sr.coding.Code` that captures coded concepts in the same way that *highdicom.sr.CodedConcept* does. The reason for the difference is that the *highdicom* class is a sub-class of `pydicom.Dataset` with the relevant attributes such that it can be included directly into a DICOM object. *pydicom* also includes within it values for a large number of coded concepts within the DCM, SNOMED-CT, and UCUM terminologies. For example, instead of manually creating the "tumor" concept above, we could have just used the pre-defined value in *pydicom*:

```
from pydicom.sr.codedict import codes

tumor_code = codes.SCT.Tumor
print(tumor_code.value)
# '1083690006'
print(tumor_code.scheme_designator)
# 'SCT'
print(tumor_code.meaning)
# 'tumor'
```

Here are some other examples of codes within *pydicom*:

```
from pydicom.sr.codedict import codes

# A patient, as described by the DCM terminology
patient_code = codes.DCM.Patient
```

(continues on next page)

(continued from previous page)

```
print(patient_code)
# Code(value='121025', scheme_designator='DCM', meaning='Patient', scheme_version=None)

# A centimeter, a described by the UCUM coding scheme
cm_code = codes.UCUM.cm
print(cm_code)
# Code(value='cm', scheme_designator='UCUM', meaning='cm', scheme_version=None)
```

The two classes can be used interoperably throughout highdicom: anywhere in the *highdicom* API that you can pass a *highdicom.sr.CodedConcept*, you can pass an `pydicom.sr.coding.Code` instead and it will be converted behind the scenes for you. Furthermore, equality is defined between the two classes such that it evaluates to true if they represent the same concept, and they hash to the same value if you use them within sets or as keys in dictionaries.

```
import highdicom as hd
from pydicom.sr.codedict import codes

tumor_code_hd = hd.sr.CodedConcept(
    value="108369006",
    scheme_designator="SCT",
    meaning="Tumor"
)
tumor_code = codes.SCT.Tumor

assert tumor_code_hd == tumor_code
assert len({tumor_code_hd, tumor_code}) == 1
```

For equality and hashing, two codes are considered equivalent if they have the same coding scheme, and value, regardless of how their meaning is represented.

Finding Suitable Codes

The *pydicom* code dictionary allows searching for concepts via simple string matching. However, for more advanced searching it is generally advisable to search the documentation for the coding scheme itself.

```
from pydicom.sr.codedict import codes

print(codes.SCT.dir('liver'))
# ['DeliveredRadiationDose',
# 'HistoryOfPrematureDelivery',
# 'Liver',
# 'LiverStructure']

print(codes.SCT.Liver)
# Code(value='10200004', scheme_designator='SCT', meaning='Liver', scheme_version=None)
```

3.3 Information Object Definitions (IODs)

An Information Object Definition defines a single "type" of DICOM file, such as a Segmentation, Presentation State or Structured Report. The following sections give in-depth explanations of the various IODs implemented within *highdicom*.

3.3.1 Segmentation (SEG) Images

DICOM Segmentation Images (often abbreviated DICOM SEG) are one of the primary IODs (information objects definitions) implemented in the *highdicom* library. SEG images store segmentations of other DICOM images (which we will refer to as *source images*) of other modalities, such as magnetic resonance (MR), computed tomography (CT), slide microscopy (SM) and many others. A segmentation is a partitioning of the source image into different regions. In medical imaging these regions may commonly represent different organs or tissue types, or regions of abnormality (e.g. tumor or infarct) identified within an image.

The crucial difference between SEGs and other IODs that allow for storing image regions is that SEGs store the segmented regions in *raster* format as pixel arrays as opposed to the *vector* descriptions of the region's boundary used by structured reports (SRs), presentation states, and RT structures. This makes them a more natural choice for many automatic image processing algorithms such as convolutional neural networks.

The DICOM standard provides a highly flexible object definition for Segmentation images that is able to cover a large variety of possible use cases. Unfortunately, this flexibility comes with complexity that may make Segmentation images difficult to understand and work with at first.

Segments

A SEG image encodes one or more distinct regions of an image, which are known as *segments*. A single segment could represent, for example, a particular organ or structure (liver, lung, kidney, cell nucleus), tissue (fat, muscle, bone), or abnormality (tumor, infarct). Elsewhere the same concept is known by other names such as *class* or *label*.

Each segment in a DICOM SEG image is represented by a separate 2D *frame* (or set of *frames*) within the Segmentation image. One important ramification of this is that segments need not be *mutually exclusive*, i.e. a given pixel or spatial location within the source image can belong to multiple segments. In other words, the segments within a SEG image may *overlap*. There is an optional attribute called "Segments Overlap" (0062, 0013) that, if present, will indicate whether the segments overlap in a given SEG image.

Segment Descriptions

Within a DICOM SEG image, segments are identified by a Segment Number. Segments are numbered with consecutive segment numbers starting at 1 (i.e., 1, 2, 3, ...). Additionally, each segment present is accompanied by information describing what the segment represents. This information is placed in the "SegmentsSequence" (0062, 0002) attribute of the segmentation file. In *highdcom*, we use the *highdicom.seg.SegmentDescription* class to hold this information. When you construct a DICOM SEG image using *highdicom*, you must construct a single *highdicom.seg.SegmentDescription* object for each segment. The segment description includes the following information:

- **Segment Label**: A human-readable name for the segment (e.g. `"Left Kidney"`). This can be any string.
- **Segmented Property Category**: A coded value describing the category of the segmented region. For example this could specify that the segment represents an anatomical structure, a tissue type, or an abnormality. This is passed as either a *highdicom.sr.CodedConcept*, or a `pydicom.sr.coding.Code` object.
- **Segmented Property Type**: Another coded value that more specifically describes the segmented region, as for example a kidney or tumor. This is passed as either a *highdicom.sr.CodedConcept*, or a `pydicom.sr.coding.Code` object.

- **Algorithm Type**: Whether the segment was produced by an automatic, semi-automatic, or manual algorithm. The valid values are contained within the enum *highdicom.seg.SegmentAlgorithmTypeValues*.
- **Anatomic Regions**: (Optional) A coded value describing the anatomic region in which the segment is found. For example, if the segmented property type is "tumor", this can be used to convey that the tumor is found in the kidney. This is passed as a sequence of coded values as either *highdicom.sr.CodedConcept*, or `pydicom.sr.coding.Code` objects.
- **Tracking ID and UID**: (Optional) These allow you to provide, respectively, a human readable ID and unique ID to a specific segment. This can be used, for example, to uniquely identify particular lesions over multiple imaging studies. These are passed as strings.

Notice that the segment description makes use of coded concepts to ensure that the way a particular anatomical structure is described is standardized and unambiguous (if standard nomenclatures are used). See *Coding* for more information.

Here is an example of constructing a simple segment description for a segment representing a liver that has been manually segmented.

```
from pydicom.sr.codedict import codes

import highdicom as hd


# Liver segment produced by a manual algorithm
liver_description = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='liver',
    segmented_property_category=codes.SCT.Organ,
    segmented_property_type=codes.SCT.Liver,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)
```

In this second example, we describe a segment representing a tumor that has been automatically segmented by an artificial intelligence algorithm. For this, we must first provide more information about the algorithm used in an *highdicom.AlgorithmIdentificationSequence*.

```
# For the next segment, we will describe the specific algorithm used to
# create it
algorithm_identification = hd.AlgorithmIdentificationSequence(
    name='Auto-Tumor',
    version='v1.0',
    family=codes.cid7162.ArtificialIntelligence
)

# Kidney tumor segment produced by the above algorithm
tumor_description = hd.seg.SegmentDescription(
    segment_number=2,
    segment_label='kidney tumor',
    segmented_property_category=codes.SCT.MorphologicallyAbnormalStructure,
    segmented_property_type=codes.SCT.Tumor,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.AUTOMATIC,
    algorithm_identification=algorithm_identification,
    anatomic_regions=[codes.SCT.Kidney]
)
```

Binary and Fractional SEGs

One particularly important characteristic of a segmentation image is its "Segmentation Type" (0062,0001), which may take the value of either `"BINARY"` or `"FRACTIONAL"` and describes the values that pixels within the segmentation may take. Pixels in a `"BINARY"` segmentation image may only take values 0 or 1, i.e. each pixel either belongs to the segment or does not.

By contrast, pixels in a `"FRACTIONAL"` segmentation image lie in the range 0 to 1. A second attribute, "Segmentation Fractional Type" (0062,0010) specifies how these values should be interpreted. There are two options, represented by the enumerated type *highdicom.seg.SegmentationFractionalTypeValues*:

- `"PROBABILITY"`, i.e. the number between 0 and 1 represents a probability that a pixel belongs to the segment
- `"OCCUPANCY"` i.e. the number represents the fraction of the volume of the pixel's (or voxel's) area (or volume) that belongs to the segment

A potential source of confusion is that having a Segmentation Type of `"BINARY"` only limits the range of values *within a given segment*. It is perfectly valid for a `"BINARY"` segmentation to have multiple segments. It is therefore not the same sense of the word *binary* that distinguishes *binary* from *multiclass* segmentations.

Highdicom provides the Python enumerations *highdicom.seg.SegmentationTypeValues* and *highdicom.seg.SegmentationFractionalTypeValues* for the valid values of the "Segmentation Type" and "Segmentation Fractional Type" attributes, respectively.

Constructing Basic Binary SEG Images

We have now covered enough to construct a basic binary segmentation image. We use the *highdicom.seg.Segmentation* class and provide a description of each segment, a pixel array of the segmentation mask, the source images as a list of `pydicom.Dataset` objects, and some other basic information. The segmentation pixel array is provided as a numpy array with a boolean or unsigned integer data type containing only the values 0 and 1.

```
import numpy as np

from pydicom import dcmread
from pydicom.sr.codedict import codes
from pydicom.data import get_testdata_file

import highdicom as hd

# Load a CT image
source_image = dcmread(get_testdata_file('CT_small.dcm'))

# Description of liver segment produced by a manual algorithm
liver_description = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='liver',
    segmented_property_category=codes.SCT.Organ,
    segmented_property_type=codes.SCT.Liver,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)

# Pixel array is an unsigned integer array with 0 and 1 values
mask = np.zeros((128, 128), dtype=np.uint8)
mask[10:20, 10:20] = 1
```

(continues on next page)

(continued from previous page)

```
# Construct the Segmentation Image
seg = hd.seg.Segmentation(
    source_images=[source_image],
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=[liver_description],
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Liver Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
)
```

Constructing Binary SEG Images with Multiple Frames

DICOM SEGs are multiframe objects, which means that they may contain more than one frame within the same object. For example, a single SEG image may contain the segmentations for an entire series of CT images. In this case you can pass a 3D numpy array as the `pixel_array` parameter of the constructor. The segmentation masks of each of the input images are stacked down axis 0 of the numpy array. The order of segmentation masks is assumed to match the order of the frames within the `source_images` parameter, i.e. `pixel_array[i, ...]` is the segmentation of `source_images[i]`. Note that highdicom makes no attempt to sort the input source images in any way. It is the responsibility of the user to ensure that they pass the source images in a meaningful order, and that the source images and segmentation frames at the same index correspond.

```
import numpy as np

from pydicom import dcmread
from pydicom.sr.codedict import codes
from pydicom.data import get_testdata_files

import highdicom as hd

# Load a series of CT images as a list of pydicom.Datasets
source_images = [
    dcmread(f) for f in get_testdata_files('dicomdirtests/77654033/CT2/*')
]

# Sort source frames by instance number (note that this is illustrative
# only, sorting by instance number is not generally recommended as this
# attribute is not guaranteed to be present in all types of source image)
source_images = sorted(source_images, key=lambda x: x.InstanceNumber)

# Create a segmentation by thresholding the CT image at 1000 HU
thresholded = [
    im.pixel_array * im.RescaleSlope + im.RescaleIntercept > 1000
    for im in source_images
]
```

(continues on next page)

(continued from previous page)

```
# Stack segmentations of each frame down axis zero. Now we have an array
# with shape (frames x height x width)
mask = np.stack(thresholded, axis=0)

# Description of liver segment produced by a manual algorithm
# Note that now there are multiple frames but still only a single segment
liver_description = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='liver',
    segmented_property_category=codes.SCT.Organ,
    segmented_property_type=codes.SCT.Liver,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)

# Construct the Segmentation Image
seg = hd.seg.Segmentation(
    source_images=source_images,
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=[liver_description],
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Liver Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
)
```

Note that the example of the previous section with a 2D pixel array is simply a convenient shorthand for the special case where there is only a single source frame and a single segment. It is equivalent in every way to passing a 3D array with a single frame down axis 0.

Constructing Binary SEG Images of Multiframe Source Images

Alternatively, we could create a segmentation of a source image that is itself a multiframe image (such as an Enhanced CT, Enhanced MR image, or a Whole Slide Microscopy image). In this case, we just pass the single source image object, and the `pixel_array` input with one segmentation frame in axis 0 for each frame of the source file, listed in ascending order by frame number. I.e. `pixel_array[i, ...]` is the segmentation of frame `i + 1` of the single source image (the offset of +1 is because numpy indexing starts at 0 whereas DICOM frame indices start at 1).

```
import numpy as np

from pydicom import dcmread
from pydicom.sr.codedict import codes
from pydicom.data import get_testdata_file

import highdicom as hd

# Load an enhanced (multiframe) CT image
```

(continues on next page)

(continued from previous page)

```
source_dcm = dcmread(get_testdata_file('eCT_Supplemental.dcm'))

# Apply some basic processing to correctly scale the source images
pixel_xform_seq = source_dcm.SharedFunctionalGroupsSequence[0]\
    .PixelValueTransformationSequence[0]
slope = pixel_xform_seq.RescaleSlope
intercept = pixel_xform_seq.RescaleIntercept
image_array = source_dcm.pixel_array * slope + intercept

# Create a segmentation by thresholding the CT image at 0 HU
mask = image_array > 0

# Description of liver segment produced by a manual algorithm
# Note that now there are multiple frames but still only a single segment
liver_description = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='liver',
    segmented_property_category=codes.SCT.Organ,
    segmented_property_type=codes.SCT.Liver,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)

# Construct the Segmentation Image
seg = hd.seg.Segmentation(
    source_images=[source_dcm],
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=[liver_description],
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Liver Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
)
```

Constructing Binary SEG Images with Multiple Segments

To further generalize our initial example, we can include multiple segments representing, for example, multiple organs. The first change is to include the descriptions of all segments in the `segment_descriptions` parameter. Note that the `segment_descriptions` list must contain segment descriptions ordered consecutively by their `segment_number`, starting with `segment_number=1`.

The second change is to include the segmentation mask of each segment within the `pixel_array` passed to the constructor. There are two methods of doing this. The first is to stack the masks for the multiple segments down axis 3 (the fourth axis) of the `pixel_array`. The shape of the resulting `pixel_array` with F source frames of height H and width W, with S segments, is then (F x H x W x S). The segmentation mask for the segment with `segment_number=i` should be found at `pixel_array[:, :, :, i - 1]` (the offset of -1 is because segments are numbered starting at 1 but numpy array indexing starts at 0).

Note that when multiple segments are used, the first dimension (F) must always be present even if there is a single source frame.

```
# Load a series of CT images as a list of pydicom.Datasets
source_images = [
    dcmread(f) for f in get_testdata_files('dicomdirtests/77654033/CT2/*')
]

# Sort source frames by instance number
source_images = sorted(source_images, key=lambda x: x.InstanceNumber)
image_array = np.stack([
    im.pixel_array * im.RescaleSlope + im.RescaleIntercept
    for im in source_images
], axis=0)

# Create a segmentation by thresholding the CT image at 1000 HU
thresholded_0 = image_array > 1000

# ...and a second below 500 HU
thresholded_1 = image_array < 500

# Stack the two segments down axis 3
mask = np.stack([thresholded_0, thresholded_1], axis=3)

# Description of bone segment produced by a manual algorithm
bone_description = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='bone',
    segmented_property_category=codes.SCT.Tissue,
    segmented_property_type=codes.SCT.Bone,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)
# Description of liver segment produced by a manual algorithm
liver_description = hd.seg.SegmentDescription(
    segment_number=2,
    segment_label='liver',
    segmented_property_category=codes.SCT.Organ,
    segmented_property_type=codes.SCT.Liver,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)
segment_descriptions = [bone_description, liver_description]

# Construct the Segmentation Image
seg = hd.seg.Segmentation(
    source_images=source_images,
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=segment_descriptions,
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
```

(continues on next page)

(continued from previous page)

```
    manufacturer_model_name='Multi-Organ Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
)
```

The second way to pass segmentation masks for multiple labels is as a "label map". A label map is a 3D array (or 2D in the case of a single frame) in which each pixel's value determines which segment it belongs to, i.e. a pixel with value 1 belongs to segment 1 (which is the first item in the `segment_descriptions`). A pixel with value 0 belongs to no segments. The label map form is more convenient to work with in many applications, however it is limited to representing segmentations that do not overlap (i.e. those in which a single pixel can belong to at most one segment). The more general form does not have this limitation: a given pixel may belong to any number of segments. Note that passing a "label map" is purely a convenience provided by *highdicom*, it makes no difference to how the segmentation is actually stored (*highdicom* splits the label map into multiple single-segment frames and stores these, as required by the standard).

Therefore, The following snippet produces an equivalent SEG image to the previous snippet, but passes the mask as a label map rather than as a stack of segments.

```
# Load a CT image
source_images = [
    dcmread(f) for f in get_testdata_files('dicomdirtests/77654033/CT2/*')
]

# Sort source frames by instance number
source_images = sorted(source_images, key=lambda x: x.InstanceNumber)
image_array = np.stack([
    im.pixel_array * im.RescaleSlope + im.RescaleIntercept
    for im in source_images
], axis=0)

# Create the same two segments as above as a label map
mask = np.zeros_like(image_array, np.uint8)
mask[image_array > 1000] = 1
mask[image_array < 500] = 2

# Construct the Segmentation Image
seg = hd.seg.Segmentation(
    source_images=source_images,
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=segment_descriptions,
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Multi-Organ Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
)
```

Constructing SEG Images from a Total Pixel Matrix

Some digital pathology images are represented as "tiled" images, in which the full image (known as the "total pixel matrix") is divided up into smaller rectangular regions in the row and column dimensions and each region ("tile") is stored as a frame in a multiframe DICOM image.

Segmentations of such images are stored as a tiled image in the same manner. There are a two options in *highdicom* for doing this. You can either pass each tile/frame individually stacked as a 1D list down the first dimension of the `pixel_array` as we have already seen (with the location of each frame either matching that of the corresponding frame in the source image or explicitly specified in the `plane_positions` argument), or you can pass the 2D total pixel matrix of the segmentation and have *highdicom* automatically create the tiles for you.

To enable this latter option, pass the `pixel_array` as a single frame (i.e. a 2D labelmap array, a 3D labelmap array with a single frame stacked down the first axis, or a 4D array with a single frame stacked down the first dimension and any number of segments stacked down the last dimension) and set the `tile_pixel_array` argument to `True`. You can optionally choose the size (in pixels) of each tile using the `tile_size` argument, or, by default, the tile size of the source image will be used (regardless of whether the segmentation is represented at the same resolution as the source image).

If you need to specify the plane positions of the image explicitly, you should pass a single item to the `plane_positions` argument giving the location of the top left corner of the full total pixel matrix. Otherwise, all the usual options are available to you.

```
# Use an example slide microscopy image from the highdicom test data
# directory
sm_image = dcmread('data/test_files/sm_image.dcm')

# The source image has multiple frames/tiles, but here we create a mask
# corresponding to the entire total pixel matrix
mask = np.zeros(
    (
        sm_image.TotalPixelMatrixRows,
        sm_image.TotalPixelMatrixColumns
    ),
    dtype=np.uint8,
)
mask[38:43, 5:41] = 1

property_category = hd.sr.CodedConcept("91723000", "SCT", "Anatomical Structure")
property_type = hd.sr.CodedConcept("84640000", "SCT", "Nucleus")
segment_descriptions = [
    hd.seg.SegmentDescription(
        segment_number=1,
        segment_label='Segment #1',
        segmented_property_category=property_category,
        segmented_property_type=property_type,
        algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
    ),
]

seg = hd.seg.Segmentation(
    source_images=[sm_image],
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=segment_descriptions,
```

(continues on next page)

(continued from previous page)

```
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Slide Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
    tile_pixel_array=True,
)

# The result stores the mask as a set of 10 tiles of the non-empty region of
# the total pixel matrix, each of size (10, 10), matching # the tile size of
# the source image
assert seg.NumberOfFrames == 10
assert seg.pixel_array.shape == (10, 10, 10)
```

"TILED_FULL" and "TILED_SPARSE"

When the segmentation is stored as a tiled image, there are two ways in which the locations of each frame/tile may be specified in the resulting object. These are defined by the value of the "DimensionOrganizationType" attribute:

- "TILED_SPARSE": The position of each tile is explicitly defined in the "PerFrameFunctionalGroupsSequence" of the object. This requires a potentially very long sequence to store all the per-frame metadata, but does allow for the omission of empty frames from the segmentation and other irregular tiling strategies.
- "TILED_FULL": The position of each tile is implicitly defined using a predetermined order of the frames. This saves the need to store the pre-frame metadata but does not allow for the omission of empty frames of the segmentation and is generally less flexible. It may also be simpler for a receiving application to process, since the tiles are guaranteed to be regularly and consistently ordered.

You can control this behavior by specifying the `dimension_organization_type` parameter and passing a value of the *highdicom.DimensionOrganizationTypeValues* enum. The default value is "TILED_SPARSE". Generally, the "TILED_FULL" option will be used in combination with `tile_pixel_array` argument.

```
# Using the same example as above, this time as TILED_FULL
seg = hd.seg.Segmentation(
    source_images=[sm_image],
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=segment_descriptions,
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Slide Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
    tile_pixel_array=True,
    omit_empty_frames=False,
    dimension_organization_type=hd.DimensionOrganizationTypeValues.TILED_FULL,
```

(continues on next page)

(continued from previous page)

```
)

# The result stores the mask as a set of 25 tiles of the entire region of
# the total pixel matrix, each of size (10, 10), matching the tile size of
# the source image
assert seg.NumberOfFrames == 25
assert seg.pixel_array.shape == (25, 10, 10)
```

Multi-resolution Pyramids

Whole slide digital pathology images can often be very large and as such it is common to represent them as *multi-resolution pyramids* of images, i.e. to store multiple versions of the same image at different resolutions. This helps viewers render the image at different zoom levels.

Within DICOM, this can also extend to segmentations derived from whole slide images. Multiple different SEG images may be stored, each representing the same segmentation at a different resolution, as different instances within a DICOM series.

highdicom provides the *highdicom.seg.create_segmentation_pyramid()* function to assist with this process. This function handles multiple related scenarios:

- Constructing a segmentation of a source image pyramid given a segmentation pixel array of the highest resolution source image. Highdicom performs the downsampling automatically to match the resolution of the other source images. For this case, pass multiple `source_images` and a single item in `pixel_arrays`.
- Constructing a segmentation of a source image pyramid given user-provided segmentation pixel arrays for each level in the source pyramid. For this case, pass multiple `source_images` and a matching number of `pixel_arrays`.
- Constructing a segmentation of a single source image given multiple user-provided downsampled segmentation pixel arrays. For this case, pass a single item in `source_images`, and multiple items in `pixel_arrays`).
- Constructing a segmentation of a single source image and a single segmentation pixel array by downsampling by a given list of `downsample_factors`. For this case, pass a single item in `source_images`, a single item in `pixel_arrays`, and a list of one or more desired `downsample_factors`.

Here is a simple of example of specifying a single source image and segmentation array, and having *highdicom* create a multi-resolution pyramid segmentation series at user-specified downsample factors.

```
import highdicom as hd
from pydicom import dcmread
import numpy as np


# Use an example slide microscopy image from the highdicom test data
# directory
sm_image = dcmread('data/test_files/sm_image.dcm')

# The source image has multiple frames/tiles, but here we create a mask
# corresponding to the entire total pixel matrix
mask = np.zeros(
    (
        sm_image.TotalPixelMatrixRows,
        sm_image.TotalPixelMatrixColumns
```

(continues on next page)

(continued from previous page)

```
    ),
    dtype=np.uint8,
)
mask[38:43, 5:41] = 1

property_category = hd.sr.CodedConcept("91723000", "SCT", "Anatomical Structure")
property_type = hd.sr.CodedConcept("84640000", "SCT", "Nucleus")
segment_descriptions = [
    hd.seg.SegmentDescription(
        segment_number=1,
        segment_label='Segment #1',
        segmented_property_category=property_category,
        segmented_property_type=property_type,
        algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
    ),
]

# This will create a segmentation series of three images: one at the
# original source image resolution (implicit), one at half the size, and
# another at a quarter of the original size.
seg_pyramid = hd.seg.create_segmentation_pyramid(
    source_images=[sm_image],
    pixel_arrays=[mask],
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=segment_descriptions,
    series_instance_uid=hd.UID(),
    series_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Slide Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
    downsample_factors=[2.0, 4.0]
)
```

Note that the *highdicom.seg.create_segmentation_pyramid()* function always behaves as if the `tile_pixel_array` input is `True` within the segmentation constructor, i.e. it assumes that the input segmentation masks represent total pixel matrices.

Representation of Fractional SEGs

Although the pixel values of `"FRACTIONAL"` segmentation images can be considered to lie within a continuous range between 0 and 1, they are in fact not stored this way. Instead they are quantized and scaled so that they may be stored as unsigned 8-bit integers between 0 and the value of the "Maximum Fractional Value" (0062,000E) attribute. Thus, assuming a "Maximum Fractional Value" of 255, a pixel value of x should be interpreted as a probability or occupancy value of $x/255$. You can control the "Maximum Fractional Value" by passing the `max_fractional_value` parameter. 255 is used as the default.

When constructing `"FRACTIONAL"` segmentation images, you pass a floating-point valued pixel array and *highdicom* handles this quantization for you. If you wish, you may change the "Maximum Fractional Value" from the default of 255 (which gives the maximum possible level of precision). Note that this does entail a loss of precision.

Similarly, *highdicom* will rescale stored values back down to the range 0-1 by default in its methods for retrieving pixel arrays (more on this below).

Otherwise, constructing "FRACTIONAL" segs is identical to constructing binary ones "BINARY", with the limitation that fractional SEGs may not use the "label map" method to pass multiple segments but must instead stack them along axis 3.

The example below shows a simple example of constructing a fractional seg representing a probabilistic segmentation of the liver.

```
import numpy as np

from pydicom import dcmread
from pydicom.sr.codedict import codes
from pydicom.data import get_testdata_file

import highdicom as hd

# Load a CT image
source_image = dcmread(get_testdata_file('CT_small.dcm'))

# Description of liver segment produced by a manual algorithm
liver_description = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='liver',
    segmented_property_category=codes.SCT.Organ,
    segmented_property_type=codes.SCT.Liver,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)

# Pixel array is an float array with values between 0 and 1
mask = np.zeros((128, 128), dtype=float)
mask[10:20, 10:20] = 0.5
mask[30:40, 30:40] = 0.75

# Construct the Segmentation Image
seg = hd.seg.Segmentation(
    source_images=[source_image],
    pixel_array=mask,
    segmentation_type=hd.seg.SegmentationTypeValues.FRACTIONAL,
    fractional_type=hd.seg.SegmentationFractionalTypeValues.PROBABILITY,
    segment_descriptions=[liver_description],
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Liver Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
)
```

Implicit Conversion to Fractional

Note that any segmentation pixel array that *highdicom* allows you to store as a `"BINARY"` SEG (i.e. a binary segmentation with segments stacked down axis 3, or a label-map style segmentation) may also be stored as a `"FRACTIONAL"` SEG. You just pass the integer array, specify the `segmentaton_type` as `"FRACTIONAL"` and *highdicom* does the conversion for you. Input pixels with value 1 will be automatically stored with value `max_fractional_value`. We recommend that if you do this, you specify `max_fractional_value=1` to clearly communicate that the segmentation is inherently binary in nature.

Why would you want to make this seemingly rather strange choice? Well, `"FRACTIONAL"` SEGs tend to compress *much* better than `"BINARY"` ones (see next section). Note however, that this is arguably an misuse of the intent of the standard, so *caveat emptor*.

Compression

The types of pixel compression available in segmentation images depends on the segmentation type. Pixels in a `"BINARY"` segmentation image are "bit-packed" such that 8 pixels are grouped into 1 byte in the stored array. If a given frame contains a number of pixels that is not divisible by 8 exactly, a single byte will straddle a frame boundary into the next frame if there is one, or the byte will be padded with zeroes of there are no further frames. This means that retrieving individual frames from segmentation images in which each frame size is not divisible by 8 becomes problematic. No further compression may be applied to frames of `"BINARY"` segmentation images.

Pixels in `"FRACTIONAL"` segmentation images may be compressed using one of the lossless compression methods available within DICOM. Currently *highdicom* supports the following compressed transfer syntaxes when creating `"FRACTIONAL"` segmentation images: `"RLELossless"`, `"JPEG2000Lossless"`, and `"JPEGLSLossless"`.

Note that there may be advantages to using `"FRACTIONAL"` segmentations to store segmentation images that are binary in nature (i.e. only taking values 0 and 1):

- If the segmentation is very simple or sparse, the lossless compression methods available in `"FRACTIONAL"` images may be more effective than the "bit-packing" method required by `"BINARY"` segmentations.
- The clear frame boundaries make retrieving individual frames from `"FRACTIONAL"` image files possible.

Multiprocessing

When creating large, multiframe `"FRACTIONAL"` segmentations using a compressed transfer syntax, the time taken to compress the frames can become large and dominate the time taken to create the segmentation. By default, frames are compressed in series using the main process, however the `workers` parameter allows you to specify a number of additional worker processes that will be used to compress frames in parallel. Setting `workers` to a negative number uses all available processes on your machine. Note that while this is likely to result in significantly lower creations times for segmentations with a very large number of frames, for segmentations with only a few frames the additional overhead of spawning processes may in fact slow the entire segmentation creation process down.

Geometry of SEG Images

In the simple cases we have seen so far, the geometry of the segmentation `pixel_array` has matched that of the source images, i.e. there is a spatial correspondence between a given pixel in the `pixel_array` and the corresponding pixel in the relevant source frame. While this covers most use cases, DICOM SEGs actually allow for more general segmentations in which there is a more complicated geometrical relationship between the source frames and the segmentation masks. This could arise when a source image is resampled or transformed before the segmentation method is applied, such that there is no longer a simple correspondence between pixels in the segmentation mask and pixels in the original source DICOM image.

Highdicom supports this case by allowing you to manually specify the plane positions of the each frame in the segmentation mask, and further the orientations and pixel spacings of these planes if they do not match that in the source images. In this case, the correspondence between the items of the `source_images` list and axis 0 of the segmentation `pixel_array` is broken and the number of frames in each may differ.

```
import numpy as np

from pydicom import dcmread
from pydicom.sr.codedict import codes
from pydicom.data import get_testdata_files

import highdicom as hd

# Load a CT image
source_images = [
    dcmread(f) for f in get_testdata_files('dicomdirtests/77654033/CT2/*')
]

# Sort source frames by instance number
source_images = sorted(source_images, key=lambda x: x.InstanceNumber)

# Now the shape and size of the mask does not have to match the source
# images
mask = np.zeros((2, 100, 100), np.uint8)
mask[0, 50:60, 50:60] = 1

# Define custom positions for each frame
positions = [
    hd.PlanePositionSequence(
        hd.CoordinateSystemNames.PATIENT,
        [100.0, 50.0, -50.0]
    ),
    hd.PlanePositionSequence(
        hd.CoordinateSystemNames.PATIENT,
        [100.0, 50.0, -48.0]
    ),
]

# Define a custom orientation and spacing for the segmentation mask
orientation = hd.PlaneOrientationSequence(
    hd.CoordinateSystemNames.PATIENT,
    [0.0, 1.0, 0.0, -1.0, 0.0, 0.0]
)
spacings = hd.PixelMeasuresSequence(
    slice_thickness=2.0,
    pixel_spacing=[2.0, 2.0]
)

# Description of liver segment produced by a manual algorithm
# Note that now there are multiple frames but still only a single segment
liver_description = hd.seg.SegmentDescription(
    segment_number=1,
    segment_label='liver',
```

(continues on next page)

(continued from previous page)

```
    segmented_property_category=codes.SCT.Organ,
    segmented_property_type=codes.SCT.Liver,
    algorithm_type=hd.seg.SegmentAlgorithmTypeValues.MANUAL,
)

# Construct the Segmentation Image
seg = hd.seg.Segmentation(
    source_images=source_images,
    pixel_array=mask,
    plane_positions=positions,
    plane_orientation=orientation,
    pixel_measures=spacings,
    segmentation_type=hd.seg.SegmentationTypeValues.BINARY,
    segment_descriptions=[liver_description],
    series_instance_uid=hd.UID(),
    series_number=1,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Foo Corp.',
    manufacturer_model_name='Liver Segmentation Algorithm',
    software_versions='0.0.1',
    device_serial_number='1234567890',
)
```

Organization of Frames in SEGs

After construction, there may be many 2D frames within an SEG image, each referring to the segmentation of a certain 2D source image or frame (or a resampled plane defined by its plane position and orientation) for a certain segment. Note that this may mean that there are multiple frames of the SEG image that are derived from each frame of the input image or series. These frames are stored within the SEG as an array indexed by a frame number (consecutive integers starting at 1). The DICOM standard gives the creator of a SEG a lot of freedom about how to organize the resulting frames within the 1D list within the SEG. To complicate matters further, frames in the segmentation image that would otherwise be "empty" (contain only 0s) may be omitted from the SEG image entirely (this is *highdicom*'s default behavior but can be turned off if you prefer by specifying `omit_empty_frames=False` in the constructor).

Every `pydicom.Dataset` has the `.pixel_array` property, which, in the case of a multiframe image, returns the full list of frames in the image as an array of shape (frames x rows x columns), with frames organized in whatever manner they were organized in by the creator of the object. A *highdicom.seg.Segmentation* is a sub-class of `pydicom.Dataset`, and therefore also has the `.pixel_array` property. However, given the complexities outlined above, *it is not recommended* to use to the `.pixel_array` property with SEG images since the meaning of the resulting array is unclear without referring to other metadata within the object in all but the most trivial cases (single segment and/or single source frame with no empty frames). This may be particularly confusing and perhaps offputting to those working with SEG images for the first time.

The order in which the creator of a SEG image has chosen to organize the frames of the SEG image is described by the "DimensionIndexSequence" attribute (0020, 9222) of the SEG object. Referring to this, and the information held about a given frame within the item of the "PerFrameFunctionalGroupsSequence" attribute (5200, 9230) with the matching frame number, it is possible to determine the meaning of a certain segmentation frame. We will not describe the full details of this mechanism here.

Instead, *highdicom* provides a family of methods to help users reconstruct segmentation masks from SEG objects in a predictable and more intuitive way. We recommend using these methods over the basic `.pixel_array` in nearly all circumstances.

Reading Existing Segmentation Images

Since a segmentation is a DICOM object just like any other image, you can read it in from a file using `pydicom` to give you a `pydicom.Dataset`. However, if you read the file in using the *highdicom.seg.segread()* function, the segmentation will have type *highdicom.seg.Segmentation*. This adds several extra methods that make it easier to work with the segmentation.

```
import highdicom as hd

seg = hd.seg.segread('data/test_files/seg_image_ct_binary.dcm')
assert isinstance(seg, hd.seg.Segmentation)
```

Alternatively, you can convert an existing `pydicom.Dataset` into a *highdicom.seg.Segmentation* using the *highdicom.seg.Segmentation.from_dataset()* method. This is useful if you receive the object over network rather than reading from file.

```
import highdicom as hd
import pydicom

dcm = pydicom.dcmread('data/test_files/seg_image_ct_binary.dcm')

# Convert to highdicom Segmentation object
seg = hd.Segmentation.from_dataset(dcm)

assert isinstance(seg, hd.seg.Segmentation)
```

By default this operation copies the underlying dataset, which may be slow for large objects. You can use `copy=False` to change the type of the object without copying the data.

Since *highdicom.seg.Segmentation* is a subclass of `pydicom.Dataset`, you can still perform *pydicom* operations on it, such as access DICOM attributes by their keyword, in the usual way.

```
import highdicom as hd
import pydicom

seg = hd.seg.segread('data/test_files/seg_image_ct_binary.dcm')
assert isinstance(seg, pydicom.Dataset)

# Accessing DICOM attributes as usual in pydicom
seg.PatientName
# 'Doe^Archibald'
```

Searching For Segments

When working with existing SEG images you can use the method *highdicom.seg.Segmentation.get_segment_numbers()* to search for segments whose descriptions meet certain criteria. For example:

```
from pydicom.sr.codedict import codes

import highdicom as hd


# This is a test file in the highdicom git repository
```

(continues on next page)

(continued from previous page)

```
seg = hd.seg.segread('data/test_files/seg_image_ct_binary_overlap.dcm')

# Check the number of segments
assert seg.number_of_segments == 2

# Check the range of segment numbers
assert seg.segment_numbers == range(1, 3)

# Search for segments by label (returns segment numbers of all matching
# segments)
assert seg.get_segment_numbers(segment_label='first segment')) == [1]
assert seg.get_segment_numbers(segment_label='second segment')) == [2]

# Search for segments by segmented property type (returns segment numbers
# of all matching segments)
assert seg.get_segment_numbers(segmented_property_type=codes.SCT.Bone)) == [1]
assert seg.get_segment_numbers(segmented_property_type=codes.SCT.Spine)) == [2]

# Search for segments by tracking UID (returns segment numbers of all
# matching segments)
assert seg.get_segment_numbers(tracking_uid='1.2.826.0.1.3680043.10.511.3.
↪832710468158945490940433306322750 67')) == [1]
assert seg.get_segment_numbers(tracking_uid='1.2.826.0.1.3680043.10.511.3.
↪100424149696294296938803390163947 72')) == [2]

# You can also get the full description for a given segment, and access
# the information in it via properties
segment_1_description = seg.get_segment_description(1)
assert segment_1_description.segment_label) == 'first segment'
assert segment_1_description.tracking_uid) == '1.2.826.0.1.3680043.10.511.3.
↪832710468158945490940433306322750 67'
```

Reconstructing Segmentation Masks From DICOM SEGs

Highdicom provides the *highdicom.seg.Segmentation.get_pixels_by_source_instance()* and *highdicom.seg.Segmentation.get_pixels_by_source_frame()* methods to handle reconstruction of segmentation masks from SEG objects in which each frame in the SEG object is derived from a single source frame. The only difference between the two methods is that the *highdicom.seg.Segmentation.get_pixels_by_source_instance()* is used when the segmentation is derived from a source series consisting of multiple single-frame instances, while *highdicom.seg.Segmentation.get_pixels_by_source_frame()* is used when the segmentation is derived from a single multiframe source instance.

When reconstructing a segmentation mask using *highdicom.seg.Segmentation.get_pixels_by_source_instance()*, the user must provide a list of SOP Instance UIDs of the source images for which the segmentation mask should be constructed. Whatever order is chosen here will be used to order the frames of the output segmentation mask, so it is up to the user to sort them according to their needs. The default behavior is that the output pixel array is of shape (F x H x W x S), where F is the number of source instance UIDs, H and W are the height and width of the frames, and S is the number of segments included in the segmentation. In this way, the output of this method matches the input *pixel_array* to the constructor that would create the SEG object if it were created with *highdicom*.

The following example (and those in later sections) use DICOM files from the *highdicom* test data, which may be found

in the highdicom repository on GitHub.

```
import numpy as np
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'
    ]
)
assert pixels.shape == (2, 16, 16, 1)
assert np.unique(pixels).tolist() == [0, 1]
```

This second example demonstrates reconstructing segmentation masks from a segmentation derived from a multiframe image, in this case a whole slide microscopy image, and also demonstrates an example with multiple, in this case 20, segments:

```
import highdicom as hd

# Read in the segmentation using highdicom
seg = hd.seg.segread('data/test_files/seg_image_sm_numbers.dcm')

assert seg.number_of_segments == 20

# SOP Instance UID of the single multiframe image from which the
# segmentation was derived
_, _, source_sop_instance_uid = seg.get_source_image_uids()[0]

# Get the segmentation array for a subset of these images:
pixels = seg.get_pixels_by_source_frame(
    source_sop_instance_uid=source_sop_instance_uid,
    source_frame_numbers=range(1, 26),
)

# Source frames are stacked down the first dimension, segments are stacked
# down the fourth dimension
assert pixels.shape == (25, 10, 10, 20)

# Each segment is still binary
assert np.unique(pixels).tolist() == [0, 1]
```

Note that these two methods may only be used when the segmentation's metadata indicates that each segmentation frame

is derived from exactly one source instance or frame of a source instance. If this is not the case, a `RuntimeError` is raised.

In the general case, the *highdicom.seg.Segmentation.get_pixels_by_dimension_index_values()* method is available to query directly by the underlying dimension index values. We will not cover this advanced topic.

Reconstructing Specific Segments

A further optional parameter, `segment_numbers`, allows the user to request only a subset of the segments available within the SEG object by providing a list of segment numbers. In this case, the output array will have a dimension equal to the number of segments requested, with the segments stacked in the order they were requested (which may not be ascending by segment number).

```
import highdicom as hd

# Read in the segmentation using highdicom
seg = hd.seg.segread('data/test_files/seg_image_sm_numbers.dcm')

assert seg.number_of_segments == 20

# SOP Instance UID of the single multiframe image from which the
# segmentation was derived
_, _, source_sop_instance_uid = seg.get_source_image_uids()[0]

# Get the segmentation array for a subset of these images:
pixels = seg.get_pixels_by_source_frame(
    source_sop_instance_uid=source_sop_instance_uid,
    source_frame_numbers=range(1, 26),
    assert_missing_frames_are_empty=True,
    segment_numbers=[10, 9, 8]
)

# Source frames are stacked down the first dimension, segments are stacked
# down the fourth dimension
assert pixels.shape == (25, 10, 10, 3)
```

After this, the array `pixels[:, :, :, 0]` contains the pixels for segment number 10, `pixels[:, :, :, 1]` contains the pixels for segment number 9, and `pixels[:, :, :, 2]` contains the pixels for segment number 8.

Reconstructing Segmentation Masks as "Label Maps"

If the segments do not overlap, it is possible to combine the multiple segments into a simple "label map" style mask, as described above. This can be achieved by specifying the `combine_segments` parameter as `True`. In this case, the output will have shape (F x H x W), and a pixel value of $i > 0$ indicates that the pixel belongs to segment i or a pixel value of 0 represents that the pixel belongs to none of the requested segments. Again, this mirrors the way you would have passed this segmentation mask to the constructor to create the object if you had used a label mask. If the segments overlap, *highdicom* will raise a `RuntimeError`. Alternatively, if you specify the `skip_overlap_checks` parameter as `True`, no error will be raised and each pixel will be given the value of the highest segment number of those present in the pixel (or the highest segment value after relabelling has been applied if you pass `relabel=True`, see below). Note that combining segments is only possible when the segmentation type is `"BINARY"`, or the segmentation type is `"FRACTIONAL"` but the only two values are actually present in the image.

Here, we repeat the above example but request the output as a label map:

```
import highdicom as hd

# Read in the segmentation using highdicom
seg = hd.seg.segread('data/test_files/seg_image_sm_numbers.dcm')

# SOP Instance UID of the single multiframe image from which the
# segmentation was derived
_, _, source_sop_instance_uid = seg.get_source_image_uids()[0]

# Get the segmentation array for a subset of these images:
pixels = seg.get_pixels_by_source_frame(
    source_sop_instance_uid=source_sop_instance_uid,
    source_frame_numbers=range(1, 26),
    assert_missing_frames_are_empty=True,
    segment_numbers=[10, 9, 8],
    combine_segments=True,
)

# Source frames are stacked down the first dimension, now there is no
# fourth dimension
assert pixels.shape == (25, 10, 10)

assert np.unique(pixels).tolist() == [0, 8, 9, 10]
```

In the default behavior, the pixel values of the output label map correspond to the original segment numbers to which those pixels belong. Therefore we see that the output array contains values 8, 9, and 10, corresponding to the three segments that we requested (in addition to 0, meaning no segment). However, when you are specifying a subset of segments, you may wish to "relabel" these segments such that in the output array the first segment you specify (10 in the above example) is indicated by pixel value 1, the second segment (9 in the example) is indicated by pixel value 2, and so on. This is achieved using the `relabel` parameter.

```
import highdicom as hd

# Read in the segmentation using highdicom
seg = hd.seg.segread('data/test_files/seg_image_sm_numbers.dcm')

# SOP Instance UID of the single multiframe image from which the
# segmentation was derived
_, _, source_sop_instance_uid = seg.get_source_image_uids()[0]

# Get the segmentation array for a subset of these images:
pixels = seg.get_pixels_by_source_frame(
    source_sop_instance_uid=source_sop_instance_uid,
    source_frame_numbers=range(1, 26),
    assert_missing_frames_are_empty=True,
    segment_numbers=[10, 9, 8],
    combine_segments=True,
    relabel=True,
)

# Source frames are stacked down the first dimension, now there is no
# fourth dimension
assert pixels.shape == (25, 10, 10)
```

(continues on next page)

(continued from previous page)

```
# Now the output segments have been relabelled to 1, 2, 3
assert np.unique(pixels).tolist() == [0, 1, 2, 3]
```

Reconstructing Fractional Segmentations

For "FRACTIONAL" SEG objects, *highdicom* will rescale the pixel values in the segmentation masks from the integer values as which they are stored back down to the range *0.0* to *1.0* as floating point values by scaling by the "MaximumFractionalValue" attribute. If desired, this behavior can be disabled by specifying `rescale_fractional=False`, in which case the raw integer array as stored in the SEG will be returned.

```
import numpy as np
import highdicom as hd

# Read in the segmentation using highdicom
seg = hd.seg.segread('data/test_files/seg_image_ct_true_fractional.dcm')

assert seg.segmentation_type == hd.seg.SegmentationTypeValues.FRACTIONAL

# List the source images for this segmentation:
sop_uids = [uids[2] for uids in seg.get_source_image_uids()]

# Get the segmentation array for a subset of these images:
pixels = seg.get_pixels_by_source_instance(
    source_sop_instance_uids=sop_uids,
)

# Each segment values are now floating point
assert pixels.dtype == np.float32

print(np.unique(pixels))
# [0.        0.2509804 0.5019608]
```

Reconstructing Total Pixel Matrices from Tiled Segmentations

For segmentations of digital pathology images that are stored as tiled images, the *highdicom.seg.Segmentation.get_pixels_by_source_frame()* method will return the segmentation mask as a set of frames stacked down the first dimension of the array. However, for such images, you typically want to work with the large 2D total pixel matrix that is formed by correctly arranging the tiles into a 2D array. *highdicom* provides the *highdicom.seg.Segmentation.get_total_pixel_matrix()* method for this purpose.

Called without any parameters, it returns a 3D array containing the full total pixel matrix. The first two dimensions are the spatial dimensions, and the third is the segments dimension. Behind the scenes highdicom has stitched together the required frames stored in the original file for you. Like with the other methods described above, setting `combine_segments` to `True` combines all the segments into, in this case, a 2D array.

```
import highdicom as hd

# Read in the segmentation using highdicom
seg = hd.seg.segread('data/test_files/seg_image_sm_control.dcm')
```

(continues on next page)

(continued from previous page)

```
# Get the full total pixel matrix
mask = seg.get_total_pixel_matrix()

expected_shape = (
    seg.TotalPixelMatrixRows,
    seg.TotalPixelMatrixColumns,
    seg.number_of_segments,
)
assert mask.shape == expected_shape

# Combine the segments into a single array
mask = seg.get_total_pixel_matrix(combine_segments=True)

assert mask.shape == (seg.TotalPixelMatrixRows, seg.TotalPixelMatrixColumns)
```

Furthermore, you can request a sub-region of the full total pixel matrix by specifying the start and/or stop indices for the rows and/or columns within the total pixel matrix. Note that this method follows DICOM 1-based convention for indexing rows and columns, i.e. the first row and column of the total pixel matrix are indexed by the number 1 (not 0 as is common within Python). Negative indices are also supported to index relative to the last row or column, with -1 being the index of the last row or column. Like for standard Python indexing, the stop indices are specified as one beyond the final row/column in the returned array. Note that the requested region does not have to start or stop at the edges of the underlying frames: *highdicom* stitches together only the relevant parts of the frames to create the requested image for you.

```
import highdicom as hd

# Read in the segmentation using highdicom
seg = hd.seg.segread('data/test_files/seg_image_sm_control.dcm')

# Get a region of the total pixel matrix
mask = seg.get_total_pixel_matrix(
    combine_segments=True,
    row_start=20,
    row_end=40,
    column_start=10,
    column_end=20,
)

assert mask.shape == (20, 10)

# A further example using negative indices. Since row_end is not provided,
# the default behavior is to include the last row in the total pixel matrix.
mask = seg.get_total_pixel_matrix(
    combine_segments=True,
    row_start=21,
    column_start=-30,
    column_end=-25,
)

assert mask.shape == (30, 5)
```

Viewing DICOM SEG Images

Unfortunately, DICOM SEG images are not widely supported by DICOM viewers. Viewers that do support SEG include:

- The OHIF Viewer, an open-source web-based viewer.
- 3D Slicer, an open-source desktop application for 3D medical image computing. It supports both display and creation of DICOM SEG files via the "Quantitative Reporting" plugin.

Note that these viewers may not support all features of segmentation images that *highdicom* is able to encode.

3.3.2 Structured Report Documents (SRs)

Structured report documents are DICOM files that contain information derived from a medical image in a structured and computer-readable way. *Highdicom* supports structured reports through the `highdicom.sr` sub-package.

Since SRs are a complex topic, this section is sub-divided as follows:

Structured Report (SR) Overview

Structured report documents are DICOM files that contain information derived from a medical image in a structured and computer-readable way. *Highdicom* supports structured reports through the `highdicom.sr` sub-package.

SRs are highly complex, and this page attempts to give a basic introduction while also describing the implementation within *highdicom*. A more thorough explanation my be found in:

- *DICOM Structured Reporting*. David Clunie. PixelMed Publishing, 2000. Digital copy available here.

Content Items

At their core, structured reports are collections of "content items". Each content item is a collection of DICOM attributes (a DICOM dataset) that are intended to convey a single piece of information. Each content item consists of a "name", which is always a coded concept describing what information is being conveyed, and a "value", which actually contains the information of interest. In a loose analogy, you can think of this as similar to other sorts of key-value mappings such as Python dictionaries and JSON documents. There are multiple different types of values (known as "value types"), and accordingly, there are a number of different types of content item. The classes representing these content items in *highdicom* are:

- `highdicom.sr.CodeContentItem`: The value is a coded concept.
- `highdicom.sr.CompositeContentItem`: The value is a reference to another (composite) DICOM object (for example an image or segmentation image).
- `highdicom.sr.ContainerContentItem`: The value is a template container containing other content items (more on this later).
- `highdicom.sr.DateContentItem`: The value is a date.
- `highdicom.sr.DateTimeContentItem`: The value is a date and a time.
- `highdicom.sr.NumContentItem`: The value is a decimal number.
- `highdicom.sr.PnameContentItem`: The value is a person's name.
- `highdicom.sr.ScoordContentItem`: The value is a (2D) spatial coordinate in the image coordinate system.
- `highdicom.sr.Scoord3DContentItem`: The value is a 3D spatial coordinate in the frame of reference coordinate system.

- *highdicom.sr.TcoordContentItem*: The value is a temporal coordinate defined relative to some start point.
- *highdicom.sr.TextContentItem*: The value is a general string.
- *highdicom.sr.TimeContentItem*: The value is a time.
- *highdicom.sr.WaveformContentItem*: The value is a reference to a waveform stored within another DICOM object.
- *highdicom.sr.UIDRefContentItem*: The value is a UID (unique identifier).

These classes are all subclasses pf `pydicom.Dataset` and you can view and interact with their attributes as you can with any pydicom dataset.

You can look at the API for each class to see how to construct content items of each type. Here are some simple examples for the more common types:

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

# A code content item expressing that the severity is mild
mild_item = hd.sr.CodeContentItem(
   name=codes.SCT.Severity,
   value=codes.SCT.Mild,
)

# A num content item expressing that the depth is 3.4cm
depth_item = hd.sr.NumContentItem(
   name=codes.DCM.Depth,
   value=3.4,
   unit=codes.UCUM.cm,
)

# A scoord content item expressing a point in 3D space of a particular
# frame of reference
region_item = hd.sr.Scoord3DContentItem(
   name=codes.DCM.ImageRegion,
   graphic_type=hd.sr.GraphicTypeValues3D.POINT,
   graphic_data=np.array([[10.6, 2.3, -9.6]]),
   frame_of_reference_uid="1.2.826.0.1.3680043.10.511.3.
↪881318293336312419137721414753385 66",
)

# A composite content item referencing another image as the source for a
# segmentation
source_item = hd.sr.CompositeContentItem(
   name=codes.DCM.SourceImageForSegmentation,
   referenced_sop_class_uid="1.2.840.10008.5.1.4.1.1.2",
   referenced_sop_instance_uid="1.2.826.0.1.3680043.10.511.3.
↪214292651010449660756870848035495 17",
)
```

Graphic Data Content Items (SCOORD and SCOORD3D)

Two types of Content Item that are worth discussing in greater detail are the *highdicom.sr.ScoordContentItem* and *highdicom.sr.Scoord3DContentItem*. These two types both encode "graphic data" in the form of points/lines/polygons to allow describing locations of an image in the report.

Scoord (spatial coordinate) Content Items describe locations in 2D image coordinates. Image coordinates are decimal numbers with sub-pixel accuracy that are defined in a coordinate system from (0.0, 0.0) at the top left corner of the top left pixel of the image and (rows, columns) at the bottom right corner of the bottom right pixel of the image. I.e. the center of the top left pixel is at location (0.5, 0.5).

Scoord3D (3D spatial coordinate) Content Items describe locations in the 3D frame of reference that the corresponding image (or images) are defined within. The points are expressed in millimeters relative to the origin of the coordinate system (which is not generally the same as the origin of any particular image, which is given by the "ImagePositionPatient" or "ImagePositionSlide" attribute of the image). Points expressed in this way do not change if the underlying image is resampled.

See the *highdicom.spatial* module for useful utilities for moving between these two coordinate systems.

Each of these has a distinct but similar list of graphical objects that can be represented, defined by the enumerations *highdicom.sr.GraphicTypeValues* (for Scoord Content Items) and *highdicom.sr.GraphicTypeValues3D*. These types are:

Graphic Type Values (Scoord):

- `CIRCLE`
- `ELLIPSE`
- `MULTIPOINT`
- `POINT`
- `POLYLINE`

Graphic Type 3D Values (Scoord3D):

- `ELLIPSE`
- `ELLIPSOID`
- `MULTIPOINT`
- `POINT`
- `POLYLINE`
- `POLYGON`

highdicom uses NumPy NdArrays to pass data into the constructors of the content items. These arrays should have dimensions (N, 2) for Scoord Content Items and (N, 3) for Scoord3D Content Items, where N is the number of points. The permissible number of points depends upon the graphic type. For example, a `POINT` is described by exactly one point, a `CIRCLE` is described by exactly 2 points (the center and a point on the circumference), and a `POLYLINE` may contain 2 or more points. See the documentation of the relevant enumeration class (*highdicom.sr.GraphicTypeValues* or *highdicom.sr.GraphicTypeValues3D*) for specific details on all graphic types.

Furthermore, *highdicom* will reconstruct the graphic data stored into a content item into a NumPy array of the correct shape if you use the *value* property of the content item.

Here are some examples of creating Scoord and Scoord3D Content Items and accessing their graphic data:

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

circle_data = np.array(
    [
        [10.0, 10.0],
        [11.0, 11.0],
    ]
)
circle_item = hd.sr.ScoordContentItem(
    name=codes.DCM.ImageRegion,
    graphic_type=hd.sr.GraphicTypeValues.CIRCLE,
    graphic_data=circle_data,
)
assert np.array_equal(circle_data, circle_item.value)

multipoint_data = np.array(
    [
        [100.0, 110.0, -90.0],
        [130.0, 70.0, -80.0],
        [-10.0, 400.0, 80.0],
    ]
)
multipoint_item = hd.sr.Scoord3DContentItem(
    name=codes.DCM.ImageRegion,
    graphic_type=hd.sr.GraphicTypeValues3D.MULTIPOINT,
    graphic_data=multipoint_data,
    frame_of_reference_uid="1.2.826.0.1.3680043.10.511.3.
↪88131829333631241913772141475338566",
)
assert np.array_equal(multipoint_data, multipoint_item.value)
```

Nesting of Content Items and Sequences

Each content item in an SR document may additionally have an attribute named "ContentSequence", which is a sequence of other Content Items that are the children of that Content Item. *Highdicom* has the class *highdicom.sr.ContentSequence* to encapsulate this behavior.

Using Content Sequences containing further Content Items, whose sequences may in turn contain further items, and so on, it is possible to build highly nested structures of content items in a "tree" structure.

When this is done, it is necessary to include a "relationship type" attribute in each child content item (i.e. all Content Items except the one at the root of the tree) that encodes the relationship that the child item has with the parent (the Content Item whose Content Sequence the parent belongs to).

The possible relationship types are defined with the enumeration *highdicom.sr.RelationshipTypeValues* (see the documentation of that class for more detail):

- CONTAINS
- HAS_ACQ_CONTEXT
- HAS_CONCEPT_MOD

- HAS_OBS_CONTEXT
- HAS_PROPERTIES
- INFERRED_FROM
- SELECTED_FROM

If you construct Content Items with the relationship type, you can nest Content Items like this:

```
import highdicom as hd
from pydicom.sr.codedict import codes

# A measurement derived from an image
depth_item = hd.sr.NumContentItem(
    name=codes.DCM.Depth,
    value=3.4,
    unit=codes.UCUM.cm,
)

# The source image from which the measurement was inferred
source_item = hd.sr.CompositeContentItem(
    name=codes.DCM.SourceImage,
    referenced_sop_class_uid="1.2.840.10008.5.1.4.1.1.2",
    referenced_sop_instance_uid="1.3.6.1.4.1.5962.1.1.1.1.1.20040119072730.12322",
    relationship_type=hd.sr.RelationshipTypeValues.INFERRED_FROM,
)

# A tracking identifier identifying the measurement
tracking_item = hd.sr.UIDRefContentItem(
    name=codes.DCM.TrackingIdentifier,
    value=hd.UID(),  # a newly generated UID
    relationship_type=hd.sr.RelationshipTypeValues.HAS_OBS_CONTEXT,
)

# Nest the source item below the depth item
depth_item.ContentSequence = [source_item, tracking_item]
```

Structured Reporting IODs

By nesting Content Items and Content Sequences in this way, you can create a Structured Report DICOM object. There are many IODs (Information Object Definitions) for Structured Reports, and *highdicom* currently implements three of them:

- *highdicom.sr.EnhancedSR* – Does not support Scoord 3D Content Items.
- *highdicom.sr.ComprehensiveSR* – Does not support Scoord 3D Content Items. In terms of functionality currently supported by *highdicom*, this is equivalent to the EnhancedSR.
- *highdicom.sr.Comprehensive3DSR* – This is the most general form of SR, but is relatively new and may not be supported by all systems. It does support Scoord 3D Content Items.

The constructors for these classes take a number of parameters specifying the content of the structured report, the evidence from which it was derived in the form of a list of `pydicom.Dataset` objects, as well as various metadata associated with the report.

The content is provided as the `content` parameter, which should be a single content item representing the "root" of the (potentially) nested structure containing all Content Items in the report.

Using the depth item constructed above as the root Content Item, we can create a Structured Report like this (here we use an example dataset from the highdicom test data):

```
# Path to single-frame CT image instance stored as PS3.10 file
image_dataset = pydicom.dcmread("data/test_files/ct_image.dcm")

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

Note that this is just a toy example and we do **not** recommend producing SRs like this in practice. Instead of this arbitrary structure of Content Items, it is far better to follow an existing **template** that encapsulates a standardized structure of Content Items.

Structured Reporting Templates

The DICOM standard defines a large number of Structured Reporting templates, which are essentially sets of constraints on the pattern of Content Items within a report. Each template is intended for a particular purpose.

Highdicom currently implements only the TID1500 "Measurement Report" template and its many sub-templates. This template is highly flexible and provides a standardized way to store general measurements and evaluations from one or more images or image regions (expressed in image or frame of reference coordinates).

The following page gives a detailed overview of how to use the Measurement Report template within *highdicom*.

The TID1500 Measurement Report Template

The TID1500 "Measurement Report" template is a general-purpose template for communicating measurements and qualitative qualitative evaluations derived from one or more images or regions of images. It is recommended to read the previous page on *Structured Report (SR) Overview* before this page.

Highdicom represents the various sub-templates of the TID1500 template as Python classes. Using these classes will guide you through the process of creating TID 1500 SRs in a modular and structured way, and will perform various checks on the inputs you provide.

Overview of TID1500 Content

A diagram of the structure of TID1500 content is shown here:

Fig. 1: Simplified diagram of the structure of the TID1500 template and major subtemplates. Note that this is intended to give a quick overview, please refer to the standard itself for full details.

At the top level, the Measurement Report template (*highdicom.sr.MeasurementReport*) represents a report containing various measurements and various metadata about the process through which they were created.

A measurement report contains one or more "Measurement Groups", where each group contains measurements and/or qualitative evaluations about a particular image or image region. There are three types of Measurement Group, each of which refer to different types of region:

- *highdicom.sr.MeasurementsAndQualitativeEvaluations* (TID1501): Refers to one or more entire images or image frames.
- *highdicom.sr.PlanarROIMeasurementsAndQualitativeEvaluations* (TID1410): Refers to a 2D region within a single image.
- *highdicom.sr.VolumetricROIMeasurementsAndQualitativeEvaluations* (TID1411): Refers to a 3D region within an image or image series.

A single Measurement Report may contain a mixture of Measurement Groups of these different types in any combination (as long as there is at least one group).

Each Measurement Group contains a number of Measurements (TID300) - numerical values derived from an image, such as a length or volume - and/or Qualitative Evaluations - categorical values derived from an image, such as classification of a tumor morphology.

When constructing the content, it is necessary to start at the bottom of the content tree with the Measurements and Evaluations and work up, by adding them into Measurement Groups, adding these groups to a Measurement Report, and then creating the document that contains the report. However, here we will describe the structure from the top down as it makes the big picture clearer.

Measurement Report (TID1500)

Every TID1500 Structured Report contains exactly one Measurement Report at the root of its content tree. This is represented by the class *highdicom.sr.MeasurementReport*.

The first ingredient in the Measurement Report is the "Observation Context", which contains metadata describing the way the observations that led to the report were made. This includes information such as the person or device that made the observations, and the subject about which the observations were made:

```
from pydicom.sr.codedict import codes
import highdicom as hd

observer_person_context = hd.sr.ObserverContext(
    observer_type=codes.DCM.Person,
    observer_identifying_attributes=hd.sr.PersonObserverIdentifyingAttributes(
        name='Doe^John'
    )
)
observer_device_context = hd.sr.ObserverContext(
    observer_type=codes.DCM.Device,
```

(continues on next page)

(continued from previous page)

```
    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,
)
```

The second required ingredient is a procedure code describing the procedure that was performed to result in the observations. Finally, we have the image measurement groups that the report contains (described below). There are some further optional parameters, such as a title for the report. Combining these we can construct the Measurement Report, and use it to construct the SR document:

```
from pydicom.sr.codedict import codes
import highdicom as hd

measurement_report = hd.sr.MeasurementReport(
    observation_context=observation_context,  # from above
    procedure_reported=codes.LN.CTUnspecifiedBodyRegion,
    imaging_measurements=[...],  # list of measurement groups, see below
    title=codes.DCM.ImagingMeasurementReport,
)

# Create the Structured Report instance
sr_dataset = hd.sr.Comprehensive3DSR(
    evidence=[...],  # all datasets referenced in the report
    content=measurement_report,
    series_number=1,
    series_instance_uid=hd.UID(),
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Manufacturer'
)
```

Measurement Groups

A Measurement Report contains one or more Measurement Groups. There are three types of Measurement Groups, corresponding to entire images, 2D regions of interest, and 3D regions of interest. The three types may be mixed and matched within a single Measurement Report in any combination.

Measurements And Qualitative Evaluations Group (TID1501)

The first, and simplest, type of Measurement Group applies to one or more entire images (or alternatively one or more entire frames in the case of multiframe source images). This is implemented using *highdicom.sr.MeasurementsAndQualitativeEvaluations*.

This class also accepts a parameter `source_images`, which is a sequence of *highdicom.sr.SourceImageForMeasurementGroup* items specifying the images (or frames) to which the measurement group applies. If this is omitted, the measurement group is assumed to include all images referenced in the SR document (as passed in the `evidence` parameter of the relevant Structured Report object's `__init__` method).

The following is a simple example:

```
import highdicom as hd
from pydicom import dcmread

im = dcmread('/path/to/file.dcm')

# A tracking identifier for this measurement group
tracking_id = hd.sr.TrackingIdentifier(
   identifier='Image0001',
   uid=hd.UID(),
)

# An object describing the source image for the measurements
source_image = hd.sr.SourceImageForMeasurementGroup.from_source_image(im)

# Construct the measurement group
group = hd.sr.MeasurementsAndQualitativeEvaluations(
   source_images=[source_image],
   tracking_identifier=tracking_id,
   measurements=[...],
   qualitative_evaluations=[...],
)
```

Planar ROI Image Measurements Group (TID1410)

This type of Measurement Group applies to a specific planar sub-region of the source image or images. This is implemented in the class *highdicom.sr.PlanarROIMeasurementsAndQualitativeEvaluations*.

This class takes a parameter specifying the region. There are two distinct options here:

- `referenced_region`: The image region is specified directly in the SR using a *highdicom.sr.ImageRegion* or *highdicom.sr.ImageRegion3D* passed as the `referenced_region` parameter. In this case, the coordinates defining the region are stored within the measurement group itself. The choice between *highdicom.sr.ImageRegion* and *highdicom.sr.ImageRegion3D* determines whether the image region is defined in 2D image coordinates or 3D frame-of-reference coordinates. Either way, the region must be planar.
- `referenced_segment`: The region is specified indirectly as a reference to a single slice of a single segment stored in a separate DICOM Segmentation Image object, specified by passing a *highdicom.sr.ReferencedSegmentationFrame* to the `referenced_segment` parameter, which contains UIDs to identify the Segmentation Image along with the segment number of the specific segment and the frames within which it is stored.

Note that **either** `referenced_region` or `referenced_segment` should be passed, and not both (or neither).

The following example uses an *highdicom.sr.ImageRegion* as the `referenced_region`:

```
import highdicom as hd
import numpy as np
from pydicom import dcmread

im = dcmread('/path/to/file.dcm')

# A tracking identifier for this measurement group
tracking_id = hd.sr.TrackingIdentifier(
   identifier='Region0001',
   uid=hd.UID(),
)

# Define the image region (a circle) using image coordinates
region = hd.sr.ImageRegion(
   graphic_type=hd.sr.GraphicTypeValues.CIRCLE,
   graphic_data=np.array([[45.0, 55.0], [45.0, 65.0]]),
   source_image=hd.sr.SourceImageForRegion.from_source_image(im),
)

# Construct the measurement group
group = hd.sr.PlanarROIMeasurementsAndQualitativeEvaluations(
   referenced_region=region,
   tracking_identifier=tracking_id,
   measurements=[...],
   qualitative_evaluations=[...],
)
```

This example uses an *highdicom.sr.ImageRegion3D* as the `referenced_region`:

```
import highdicom as hd
import numpy as np
from pydicom import dcmread

im = dcmread('/path/to/file.dcm')

# A tracking identifier for this measurement group
tracking_id = hd.sr.TrackingIdentifier(
   identifier='Region3D0001',
   uid=hd.UID(),
)

# Define the image region (a point) using frame-of-reference coordinates
region = hd.sr.ImageRegion3D(
   graphic_type=hd.sr.GraphicTypeValues3D.POINT,
   graphic_data=np.array([[123.5, 234.1, -23.7]]),
   frame_of_reference_uid=im.FrameOfReferenceUID,
)

# Construct the measurement group
group = hd.sr.PlanarROIMeasurementsAndQualitativeEvaluations(
   referenced_region=region,
   tracking_identifier=tracking_id,
```

(continues on next page)

(continued from previous page)

```
    measurements=[...],
    qualitative_evaluations=[...],
)
```

The final example uses an *highdicom.sr.ReferencedSegmentationFrame* as the `referenced_segment`:

```
import highdicom as hd
import numpy as np
from pydicom import dcmread

# The image dataset referenced
im = dcmread('/path/to/file.dcm')

# A segmentation dataset, assumed to contain a segmentation of the source
# image above
seg = dcmread('/path/to/seg.dcm')

# A tracking identifier for this measurement group
tracking_id = hd.sr.TrackingIdentifier(
    identifier='Region3D0001',
    uid=hd.UID(),
)

# Define the image region using a specific segment from the segmentation
ref_segment = hd.sr.ReferencedSegmentationFrame.from_segmentation(
    segmentation=seg,
    segment_number=1,
)

# Construct the measurement group
group = hd.sr.PlanarROIMeasurementsAndQualitativeEvaluations(
    referenced_segment=ref_segment,
    tracking_identifier=tracking_id,
    measurements=[...],
    qualitative_evaluations=[...],
)
```

Volumetric ROI Image Measurements Group (TID1411)

This type of Measurement Group applies to a specific volumetric sub-region of the source image or images. This is implemented in the class *highdicom.sr.VolumetricROIMeasurementsAndQualitativeEvaluations*.

Like the similar Planar ROI class, this class takes a parameter specifying the region. In this case there are three options:

- `referenced_regions`: The image region is specified directly in the SR in image coordinates using one or more objects of type *highdicom.sr.ImageRegion* passed as the `referenced_regions` parameter, representing the volumetric region as a set of 2D regions across multiple images or frames.
- `referenced_volume_surface`: The region is specified directly in the SR as a single volumetric region defined in frame of reference coordinates using a single *highdicom.sr.VolumeSurface* object passed to the `referenced_volume_surface` parameter.
- `referenced_segment`: The region is specified indirectly as a reference to an entire segment (which may spread

across multiple images or frames) of a Segmentation Image object, specified by passing a *highdicom.sr.ReferencedSegment* to the `referenced_segment` parameter, which contains UIDs to identify the Segmentation Image along with the segment number of the specific segment within it.

Note that exactly one of `referenced_regions`, `referenced_volume_surface`, or `referenced_segment` should be passed.

The following example uses a list of *highdicom.sr.ImageRegion* objects as the `referenced_regions`:

```
import highdicom as hd
import numpy as np
from pydicom import dcmread

im1 = dcmread('/path/to/file1.dcm')
im2 = dcmread('/path/to/file2.dcm')

# A tracking identifier for this measurement group
tracking_id = hd.sr.TrackingIdentifier(
   identifier='Region0001',
   uid=hd.UID(),
)

# Define the image regions (a circle in two images) using image coordinates
region1 = hd.sr.ImageRegion(
   graphic_type=hd.sr.GraphicTypeValues.CIRCLE,
   graphic_data=np.array([[45.0, 55.0], [45.0, 65.0]]),
   source_image=hd.sr.SourceImageForRegion.from_source_image(im1),
)
region2 = hd.sr.ImageRegion(
   graphic_type=hd.sr.GraphicTypeValues.CIRCLE,
   graphic_data=np.array([[40.0, 50.0], [40.0, 60.0]]),
   source_image=hd.sr.SourceImageForRegion.from_source_image(im2),
)

# Construct the measurement group
group = hd.sr.VolumetricROIMeasurementsAndQualitativeEvaluations(
   referenced_regions=[region1, region2],
   tracking_identifier=tracking_id,
   measurements=[...],
   qualitative_evaluations=[...],
)
```

This example uses a *highdicom.sr.VolumeSurface* object as the `referenced_volume_surface`:

```
import highdicom as hd
import numpy as np
from pydicom import dcmread

im = dcmread('/path/to/file.dcm')

# A tracking identifier for this measurement group
tracking_id = hd.sr.TrackingIdentifier(
   identifier='Region0001',
   uid=hd.UID(),
)
```

(continues on next page)

(continued from previous page)

```
# Define the image region (a point) using frame-of-reference coordinates
volume_surface = hd.sr.VolumeSurface(
    graphic_type=hd.sr.GraphicTypeValues.POINT,
    graphic_data=np.array([[123.5, 234.1, -23.7]]),
    source_images=[hd.sr.SourceImageForSegmentation.from_source_image(im)],
    frame_of_reference_uid=im.FrameOfReferenceUID,
)

# Construct the measurement group
group = hd.sr.VolumetricROIMeasurementsAndQualitativeEvaluations(
   referenced_volume_surface=volume_surface,
   tracking_identifier=tracking_id,
   measurements=[...],
   qualitative_evaluations=[...],
)
```

The final example uses an *highdicom.sr.ReferencedSegment* as the `referenced_segment`:

```
import highdicom as hd
import numpy as np
from pydicom import dcmread

# The image dataset referenced
im = dcmread('/path/to/file.dcm')

# A segmentation dataset, assumed to contain a segmentation of the source
# image above
seg = dcmread('/path/to/seg.dcm')

# A tracking identifier for this measurement group
tracking_id = hd.sr.TrackingIdentifier(
   identifier='Region3D0001',
   uid=hd.UID(),
)

# Define the image region using a specific segment from the segmentation
ref_segment = hd.sr.ReferencedSegment.from_segmentation(
   segmentation=seg,
   segment_number=1,
)

# Construct the measurement group
group = hd.sr.VolumetricROIMeasurementsAndQualitativeEvaluations(
   referenced_segment=ref_segment,
   tracking_identifier=tracking_id,
   measurements=[...],
   qualitative_evaluations=[...],
)
```

Further Parameters for Measurement Groups

The three types of measurement group are more alike than different. The following parameters may be used for all Measurement Groups, regardless of type (some have been omitted for brevity):

- **tracking_identifier** (*highdicom.sr.TrackingIdentifier*)**:**
  Identifier for tracking measurement groups. This allows this region to be referred to unambiguously in future objects.
- **finding_type** (*highdicom.sr.CodedConcept*, **optional**)
  Type of observed finding
- **algorithm_id:** (*highdicom.sr.AlgorithmIdentification*, **optional**)
  Identification of algorithm used for making measurements.
- **finding_sites:** (**Sequence of** *highdicom.sr.FindingSite*, **optional**)
  Coded description of one or more anatomic locations at which finding was observed
- **measurements:** (**Sequence of** *highdicom.sr.Measurement*, **optional**)
  Numeric measurements
- **qualitative_evaluations:** (**Sequence of** *highdicom.sr.CodedConcept*, **optional**)
  Coded name-value pairs that describe qualitative qualitative_evaluations
- **finding_category:** (*highdicom.sr.CodedConcept*, **optional**)
  Category of observed finding, e.g., anatomic structure or morphologically abnormal structure

Measurements and Qualitative Evaluations

Finally, we get down to the bottom of the content tree, and the measurements and qualitative evaluations themselves. Information derived from the images or image regions represented by the measurement group may be stored as either measurements, qualitative evaluations, or a mixture or the two. These two concepts play a similar role in the SR, but measurements have numerical values and qualitative evaluations have categorical values.

Qualitative Evaluations

A Qualitative Evaluation is essentially a categorical value inferred from an image. For example, this could represent a diagnosis derived from the referenced region or a severity grading. These are represented in *highdicom* using the class `highdicom.sr.QualitativeEvalution`, which is essentially a single *highdicom.sr.CodeContentItem* within a special template.

To create a Qualitative Evaluation, just pass the `name` and `value` parameters as coded values:

```
import highdicom as hd
from pydicom.sr.codedict import codes

# An evaluation of disease severity as "mild"
severity_item = hd.sr.QualitativeEvalution(
   name=codes.SCT.Severity,
   value=codes.SCT.Mild,
)

# An evaluation of tumor morphology as adenocarcinoma
morphology_item = hd.sr.QualitativeEvalution(
   name=codes.SCT.AssociatedMorphology,
```

(continues on next page)

(continued from previous page)

```
    value=codes.SCT.Anenocarcinoma,
)
```

Measurements (TID300)

A Measurement is essentially a numerical (decimal) value derived from the image or image region. In *highdicom*, a measurement is represented by the class *highdicom.sr.Measurement*. It is a small template that contains at its core a *highdicom.sr.NumContentItem* containing the value, a *highdicom.sr.CodeContentItem* specifying the unit of the measurement, and optionally several more content items describing further context or qualifications for the measurement.

Here is a basic example:

```
import highdicom as hd
from pydicom.sr.codedict import codes

# A volume measurement
measurement = hd.sr.Measurement(
    name=codes.SCT.Volume,
    value=1983.123,
    unit=codes.UCUM.CubicMillimeter,
)
```

In addition, the following optional parameters are available (see the API reference for more information):

- **Qualifier:** Qualification of the measurement.
- **Tracking Identifier:** Identifier for uniquely identifying and tracking measurements.
- **Algorithm:** Identification of algorithm used for making measurements.
- **Derivation:** How the value was computed.
- **Finding Sites:** Coded description of one or more anatomic locations corresponding to the image region from which measurement was taken.
- **Method:** Measurement method.
- **Properties:** Measurement properties, including qualitative 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:** Referenced images which were used as sources for the measurement.
- **Referenced Real World Value Map:** Referenced real world value map for referenced source images used to generate the measurement.

Putting It All Together

The snippet below is a full example of creating an SR document using the TID1500 template. You can find the file created by this snippet in the highdicom test data within the highdicom repository at `data/test_files/sr_document_with_multiple_groups.dcm`.

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

im = pydicom.dcmread("data/test_files/ct_image.dcm")

# Information about the observer
observer_person_context = hd.sr.ObserverContext(
    observer_type=codes.DCM.Person,
    observer_identifying_attributes=hd.sr.PersonObserverIdentifyingAttributes(
        name='Doe^John'
    )
)
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,
)

# An object describing the source image for the measurements
source_image = hd.sr.SourceImageForMeasurementGroup.from_source_image(im)

# First, we define an image measurement group for the CT image describing
# the intensity histogram at a certain vertebral level

# A tracking identifier for this measurement group
im_tracking_id = hd.sr.TrackingIdentifier(
   identifier='Image0001',
   uid=hd.UID(),
)

# A measurement using an IBSI code (not in pydicom)
histogram_intensity_code = hd.sr.CodedConcept(
    value="X6K6",
    meaning="Intensity Histogram Mean",
    scheme_designator="IBSI",
)
hist_measurement = hd.sr.Measurement(
    name=histogram_intensity_code,
    value=-119.0738525390625,
    unit=codes.UCUM.HounsfieldUnit,
```

(continues on next page)

(continued from previous page)

```
)
im_evaluation = hd.sr.QualitativeEvaluation(
    name=codes.SCT.AnatomicalPosition,
    value=codes.SCT.LevelOfT4T5IntervertebralDisc,
)

# Construct the measurement group
im_group = hd.sr.MeasurementsAndQualitativeEvaluations(
   source_images=[source_image],
   tracking_identifier=im_tracking_id,
   measurements=[hist_measurement],
   qualitative_evaluations=[im_evaluation],
)

# Next, we define a planar ROI measurement group describing a lung nodule

# A tracking identifier for this measurement group
lung_nodule_roi_tracking_id = hd.sr.TrackingIdentifier(
   identifier='LungNodule0001',
   uid=hd.UID(),
)

# Define the image region (a circle) using image coordinates
region = hd.sr.ImageRegion(
   graphic_type=hd.sr.GraphicTypeValues.CIRCLE,
   graphic_data=np.array([[45.0, 55.0], [45.0, 65.0]]),
   source_image=hd.sr.SourceImageForRegion.from_source_image(im),
)
nodule_measurement = hd.sr.Measurement(
    name=codes.SCT.Diameter,
    value=10.0,
    unit=codes.UCUM.mm,
)
nodule_evaluation = hd.sr.QualitativeEvaluation(
    name=codes.DCM.LevelOfSignificance,
    value=codes.SCT.NotSignificant,
)

# Construct the measurement group
planar_group_1 = hd.sr.PlanarROIMeasurementsAndQualitativeEvaluations(
   referenced_region=region,
   tracking_identifier=lung_nodule_roi_tracking_id,
   finding_type=codes.SCT.Nodule,
   finding_category=codes.SCT.MorphologicallyAbnormalStructure,
   finding_sites=[hd.sr.FindingSite(codes.SCT.Lung)],
   measurements=[nodule_measurement],
   qualitative_evaluations=[nodule_evaluation],
)

# Next, we define a second planar ROI measurement group describing the
# aorta
```

(continues on next page)

(continued from previous page)

```
# A tracking identifier for this measurement group
aorta_roi_tracking_id = hd.sr.TrackingIdentifier(
   identifier='Aorta0001',
   uid=hd.UID(),
)

# Define the image region (a circle) using image coordinates
region = hd.sr.ImageRegion(
   graphic_type=hd.sr.GraphicTypeValues.POLYLINE,
   graphic_data=np.array([[25.0, 45.0], [45.0, 45.0], [45.0, 65.0], [25.0, 65.0]]),
   source_image=hd.sr.SourceImageForRegion.from_source_image(im),
)
aorta_measurement = hd.sr.Measurement(
    name=codes.SCT.Diameter,
    value=20.0,
    unit=codes.UCUM.mm,
)

# Construct the measurement group
planar_group_2 = hd.sr.PlanarROIMeasurementsAndQualitativeEvaluations(
   referenced_region=region,
   tracking_identifier=aorta_roi_tracking_id,
   finding_type=codes.SCT.Aorta,
   finding_category=structure_code,
   measurements=[aorta_measurement],
)

# Finally, we define a volumetric ROI measurement group describing a
# vertebral body

# A tracking identifier for this measurement group
volumetric_roi_tracking_id = hd.sr.TrackingIdentifier(
   identifier='Vertebra0001',
   uid=hd.UID(),
)

# Define the region (a point) using frame of reference coordinates
volume_surface = hd.sr.VolumeSurface(
    graphic_type=hd.sr.GraphicTypeValues3D.POINT,
    graphic_data=np.array([[123.5, 234.1, -23.7]]),
    source_images=[hd.sr.SourceImageForSegmentation.from_source_image(im)],
    frame_of_reference_uid=im.FrameOfReferenceUID,
)
vol_measurement = hd.sr.Measurement(
    name=codes.SCT.Volume,
    value=200.0,
    unit=codes.UCUM.CubicMillimeter,
)

# Construct the measurement group
vol_group = hd.sr.VolumetricROIMeasurementsAndQualitativeEvaluations(
   referenced_volume_surface=volume_surface,
```

(continues on next page)

(continued from previous page)

```
    tracking_identifier=volumetric_roi_tracking_id,
    finding_category=structure_code,
    finding_type=codes.SCT.Vertebra,
    measurements=[vol_measurement],
)

measurement_report = hd.sr.MeasurementReport(
    observation_context=observation_context,  # from above
    procedure_reported=codes.LN.CTUnspecifiedBodyRegion,
    imaging_measurements=[im_group, planar_group_1, planar_group_2, vol_group],
    title=codes.DCM.ImagingMeasurementReport,
)

# Create the Structured Report instance
sr_dataset = hd.sr.Comprehensive3DSR(
    evidence=[im],  # all datasets referenced in the report
    content=measurement_report,
    series_number=1,
    series_instance_uid=hd.UID(),
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='Manufacturer'
)
sr_dataset.save_as("sr_document_with_multiple_groups.dcm")
```

Parsing Measurement Reports

In addition to the ability to create TID 1500 Structured Reports, *highdicom* also includes functionality to help you find and extract information from existing SR documents in this format.

First you must get the SR dataset into the format of a *highdicom* class. You can do this using the *highdicom.sr.srread()* function:

```
import highdicom as hd

# This example is in the highdicom test data files in the repository
sr = hd.sr.srread("data/test_files/sr_document.dcm")
```

Alternatively, if you already have a `pydicom.Dataset` in memory, you can use the relevant `from_dataset` method like this:

```
import pydicom
import highdicom as hd

sr_dataset = pydicom.dcmread("data/test_files/sr_document.dcm")

# Use the appropriate class depending on the specific IOD, here it is a
# Comprehensive3DSR
sr = hd.sr.Comprehensive3DSR.from_dataset(sr_dataset)
```

If the Structured Report conforms to the TID 1500 measurement report template, when you access the `content` property, a *highdicom.sr.MeasurementReport* object will be returned. Otherwise, a general *highdicom.sr.ContentSequence* object is returned.

The resulting *highdicom.sr.MeasurementReport* object has methods that allow you to find and access the content of the report conveniently.

Searching For Measurement Groups

To search for measurement groups, the *highdicom.sr.MeasurementReport* class has *highdicom.sr.MeasurementReport.get_image_measurement_groups()*, *highdicom.sr.MeasurementReport.get_planar_roi_measurement_groups()*, and *highdicom.sr.MeasurementReport.get_volumetric_roi_measurement_groups()* methods, each of which returns a list of the measurement groups of the three different types from the structured SR. You can additionally provide filters to return only those measurement groups that meet certain criteria.

The available search criteria include: tracking UID, finding type, finding site, referenced SOP instance UID, and referenced SOP class UID. If you provide multiple criteria, the methods return those groups that meet *all* the specified criteria.

The returned objects are of type *highdicom.sr.MeasurementsAndQualitativeEvaluations*, *highdicom.sr.PlanarROIMeasurementsAndQualitativeEvaluations*, or *highdicom.sr.VolumetricROIMeasurementsAndQualitativeEvaluations*, respectively, representing the entire sub-template in the SR content tree.

Here are just some examples of using these methods to find measurement groups of interest within a measurement report. As an example SR document, we use the SR document created on the previous page (see *Putting It All Together* for the relevant snippet).

```
import highdicom as hd
from pydicom.sr.codedict import codes

# This example is in the highdicom test data files in the repository
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# Get a list of all image measurement groups referencing an image with a
# particular SOP Instance UID
groups = sr.content.get_image_measurement_groups(
    referenced_sop_instance_uid="1.3.6.1.4.1.5962.1.1.1.1.1.20040119072730.12322",
)
assert len(groups) == 1

# Get a list of all image measurement groups with a particular tracking UID
groups = sr.content.get_image_measurement_groups(
    tracking_uid="1.2.826.0.1.3680043.10.511.3.77718622501224431322963356892468048",
)
assert len(groups) == 1

# Get a list of all planar ROI measurement groups with finding type "Nodule"
# AND finding site "Lung"
groups = sr.content.get_planar_roi_measurement_groups(
    finding_type=codes.SCT.Nodule,
    finding_site=codes.SCT.Lung,
)
assert len(groups) == 1

# Get a list of all volumetric ROI measurement groups (with no filters)
```

(continues on next page)

(continued from previous page)

```
groups = sr.content.get_volumetric_roi_measurement_groups()
assert len(groups) == 1
```

Additionally for *highdicom.sr.MeasurementReport.get_planar_roi_measurement_groups()*, and *highdicom.sr.MeasurementReport.get_volumetric_roi_measurement_groups()* it is possible to filter by graphic type and reference type (how the ROI is specified in the measurement group).

To search by graphic type, pass an instance of either the *highdicom.sr.GraphicTypeValues* or *highdicom.sr.GraphicTypeValues3D* enums:

```
import highdicom as hd
from pydicom.sr.codedict import codes

# This example is in the highdicom test data files in the repository
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# Get a list of all planar ROI measurement groups with graphic type CIRCLE
groups = sr.content.get_planar_roi_measurement_groups(
    graphic_type=hd.sr.GraphicTypeValues.CIRCLE,
)
assert len(groups) == 1
```

For reference type, you should provide one of the following values (which reflect how the SR document stores the information internally):

- `CodedConcept(value="111030", meaning="Image Region", scheme_designator="DCM")` aka `pydicom.sr.codedict.codes.DCM.ImageRegion` for ROIs defined in the SR as image regions (vector coordinates for planar regions defined within the SR document).
- `CodedConcept(value="121231", meaning="Volume Surface", scheme_designator="DCM")` aka `pydicom.sr.codedict.codes.DCM.VolumeSurface` for ROIs defined in the SR as a volume surface (vector coordinates for a volumetric region defined within the SR document).
- `CodedConcept(value="121191", meaning="Referenced Segment", scheme_designator="DCM")` aka `pydicom.sr.codedict.codes.DCM.ReferencedSegment` for ROIs defined in the SR indirectly by referencing a segment stored in a DICOM Segmentation Image.
- `CodedConcept(value="121191", meaning="Region In Space", scheme_designator="DCM")` For ROIs defined in the SR indirectly by referencing a region stored in a DICOM RT Struct object (this is not currently supported by the *highdicom* constructor, but is an option in the standard). Unfortunately this code is not including in `pydicom.sr.codedict.codes` at this time.

```
import highdicom as hd
from pydicom.sr.codedict import codes

# This example is in the highdicom test data files in the repository
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# Get a list of all planar ROI measurement groups stored as regions
groups = sr.content.get_planar_roi_measurement_groups(
    reference_type=codes.DCM.ImageRegion,
)
assert len(groups) == 2

# Get a list of all volumetric ROI measurement groups stored as volume
```

(continues on next page)

(continued from previous page)

```
# surfaces
groups = sr.content.get_volumetric_roi_measurement_groups(
    reference_type=codes.DCM.VolumeSurface,
)
assert len(groups) == 1
```

Accessing Data in Measurement Groups

Once you have found measurement groups, there are various properties on the returned object that allow you to access the information that you may need. These may be in the form of basic Python data types extracted from the measurement group's content items, or *highdicom* classes representing full sub-templates that in turn have methods and properties defined on them. These classes are the same classes that you use to construct the objects.

The following example demonstrates some examples, see the API documentation of the relevant class for a full list.

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

# This example is in the highdicom test data files in the repository
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# Use the first (only) image measurement group as an example
group = sr.content.get_image_measurement_groups()[0]

# tracking_identifier returns a Python str
assert group.tracking_identifier == "Image0001"

# tracking_uid returns a hd.UID, a subclass of str
assert group.tracking_uid == "1.2.826.0.1.3680043.10.511.3.
↪77718622501224431322963356892468048"

# source_images returns a list of hd.sr.SourceImageForMeasurementGroup, which
# in turn have some properties to access data
assert isinstance(group.source_images[0], hd.sr.SourceImageForMeasurementGroup)
assert group.source_images[0].referenced_sop_instance_uid == "1.3.6.1.4.1.5962.1.1.1.1.1.
↪20040119072730.12322"

# for the various optional pieces of information in a measurement, accessing
# the relevant property returns None if the information is not present
assert group.finding_type is None

# Now use the first planar ROI group as a second example
group = sr.content.get_planar_roi_measurement_groups()[0]

# finding_type returns a CodedConcept
assert group.finding_type == codes.SCT.Nodule

# finding_sites returns a list of hd.sr.FindingSite objects (a sub-template)
assert isinstance(group.finding_sites[0], hd.sr.FindingSite)
# the value of a finding site is a CodedConcept
```

(continues on next page)

(continued from previous page)

```
assert group.finding_sites[0].value == codes.SCT.Lung

# reference_type returns a CodedConcept (the same values used above for
# filtering)
assert group.reference_type == codes.DCM.ImageRegion

# since this has reference type ImageRegion, we can access the referenced roi
# using 'roi', which will return an hd.sr.ImageRegion object
assert isinstance(group.roi, hd.sr.ImageRegion)

# the graphic type and actual ROI coordinates (as a numpy array) can be
# accessed with the graphic_type and value properties of the roi
assert group.roi.graphic_type == hd.sr.GraphicTypeValues.CIRCLE
assert isinstance(group.roi.value, np.ndarray)
assert group.roi.value.shape == (2, 2)
```

A volumetric group returns a *highdicom.sr.VolumeSurface* or list of *highdicom.sr.ImageRegion* objects, depending on the reference type. If instead, a planar/volumetric measurement group uses the `ReferencedSegment` reference type, the referenced segment can be accessed by the `group.referenced_segmention_frame` property (for planar groups) or `group.referenced_segment` property (for volumetric groups), which return objects of type *highdicom.sr.ReferencedSegmentationFrame* and *highdicom.sr.ReferencedSegment* respectively.

Searching for Measurements

Each measurement group may optionally contain any number of "measurements", represented by the TID300 "Measurement" template and the *highdicom.sr.Measurement* class that implements it in *highdicom*. A measurement contains a numerical measurement derived from the image, along with the physical unit of the measurement and various other optional descriptive metadata

You can search for measurements within a measurements group using the `get_measurements()` method on the relevant measurement group class. You can optionally provide a `name` parameter, which should be a coded value that allows you to find measurements with a particular name.

```
import highdicom as hd
from pydicom.sr.codedict import codes

# Use the same example file in the highdicom test data
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# Use the first planar measurement group as an example
group = sr.content.get_planar_roi_measurement_groups()[0]

# Get a list of all measurements
measurements = group.get_measurements()

# Get a list of measurements for diameter
measurements = group.get_measurements(name=codes.SCT.Diameter)
```

Note that although there will usually be only a single measurement with a given name within a measurement group, multiple measurements with the same name are not disallowed by the standard. Consequently, the `get_measurements()` method returns a list containing 0 or more measurements.

Accessing Data in Measurements

You can access the name of a measurement with the `name` property (returns a *highdicom.sr.CodedConcept*), its numerical value with the `value` property (returns a `float`), and the unit with the `unit` property.

```
import highdicom as hd
from pydicom.sr.codedict import codes

# Use the same example file in the highdicom test data
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# Use the first planar measurement group as an example
group = sr.content.get_planar_roi_measurement_groups()[0]

# Get the diameter measurement in this group
measurement = group.get_measurements(name=codes.SCT.Diameter)[0]

# Access the measurement's name
assert measurement.name == codes.SCT.Diameter

# Access the measurement's value
assert measurement.value == 10.0

# Access the measurement's unit
assert measurement.unit == codes.UCUM.mm
```

Additionally, the properties `method`, `finding_sites`, `qualifier`, `referenced_images`, and `derivation` allow you to access further optional metadata that may be present in the stored measurement.

Searching for Evaluations

In addition to numerical measurements, measurement groups may also contain "Qualitative Evaluations". These contain an evaluation of the image represented using a coded concept.

Similar to measurements, you can search for evaluations with the `get_qualitative_evaluations()` method. You can optionally filter by name with the `name` parameter. You can access the name and value of the returned evaluations with the `name` and `value` properties.

```
import highdicom as hd
from pydicom.sr.codedict import codes

# Use the same example file in the highdicom test data
sr = hd.sr.srread("data/test_files/sr_document_with_multiple_groups.dcm")

# Use the first planar measurement group as an example
group = sr.content.get_planar_roi_measurement_groups()[0]

# Get the level of significance evaluation in this group
evaluation = group.get_qualitative_evaluations(
    name=codes.DCM.LevelOfSignificance
)[0]

# Access the evaluation's name
```

(continues on next page)

(continued from previous page)

```
assert evaluation.name == codes.DCM.LevelOfSignificance

# Access the evaluation's value
assert evaluation.value == codes.SCT.NotSignificant
```

3.3.3 Key Object Selection (KOS) Documents

This page is under construction, and more detail will be added soon.

3.3.4 Microscopy Bulk Simple Annotation (ANN) Objects

The Microscopy Bulk Simple Annotation IOD is an IOD designed specifically to store large numbers of similar annotations and measurements from microscopy images. Annotations of microscopy images typically refer to very large numbers of cells or cellular structures. Storing these in a Structured Report Document, with its highly nested structure, would be very inefficient in storage space and unnecessarily complex and slow to parse. Microscopy Bulk Simple Annotation objects ("bulk annotations") solve this problem by allowing you to store large number of similar annotations or measurements in efficient arrays without duplication of the descriptive metadata.

Each bulk annotation object contains one or more Annotation Groups, each of which contains a set of graphical annotations, and optionally one or more numerical measurements relating to those graphical annotations.

Constructing Annotation Groups

An Annotation Group is a set of multiple similar annotations from a microscopy image. For example, a single annotation group may contain all annotations of cell nuclei, lymphocytes, or regions of necrosis in the image. In *highdicom*, an annotation group is represented by a *highdicom.ann.AnnotationGroup*.

Each annotation group contains some required metadata that describes the contents of the group, as well as some further optional metadata that may contain further details about the group or the derivation of the annotations it contains. The required metadata elements include:

- A `number` (`int`), an integer number for the group.
- A `label` (`str`) giving a human-readable label for the group.
- A `uid` (`str` or *highdicom.UID*) uniquely identifying the group. Usually, you will want to generate UID for this.
- An `annotated_property_category` and `annotated_property_type` (*highdicom.sr.CodedConcept*) coded values (see *Coding*) describing the category and specific structure that has been annotated.
- A `graphic_type` (*highdicom.ann.GraphicTypeValues*) indicating the "form" of the annotations. Permissible values are `"ELLIPSE"`, `"POINT"`, `"POLYGON"`, `"RECTANGLE"`, and `"POLYLINE"`.
- The `algorithm_type` (*highdicom.ann.AnnotationGroupGenerationTypeValues*), the type of the algorithm used to generate the annotations (`"MANUAL"`, `"SEMIAUTOMATIC"`, or `"AUTOMATIC"`).

Further optional metadata may optionally be provided, see the API documentation for more information.

The actual annotation data is passed to the group as a list of `numpy.ndarray` objects, each of shape (N x D). N is the number of coordinates required for each individual annotation and is determined by the graphic type (see `highdicom.ann.GraphicType`). D is either 2 – meaning that the coordinates are expressed as a (Column,Row) pair in image coordinates – or 3 – meaning that the coordinates are expressed as a (X,Y,Z) triple in 3D frame of reference coordinates.

When considering which type of coordinate to use, bear in mind that the 2D image coordinates refer only to one image in a image pyramid, whereas 3D frame of reference coordinates are more easily used with any image in the pyramid. Also note that although you can include multiple annotation groups in a single bulk annotation object, they must all use the same coordinate type.

Here is a simple example of constructing an annotation group:

```
from pydicom.sr.codedict import codes
from pydicom.sr.coding import Code
import highdicom as hd
import numpy as np

# Graphic data containing two nuclei, each represented by a single point
# expressed in 2D image coordinates
graphic_data = [
    np.array([[34.6, 18.4]]),
    np.array([[28.7, 34.9]]),
]

# Nuclei annotations produced by a manual algorithm
nuclei_group = hd.ann.AnnotationGroup(
    number=1,
    uid=hd.UID(),
    label='nuclei',
    annotated_property_category=codes.SCT.AnatomicalStructure,
    annotated_property_type=Code('84640000', 'SCT', 'Nucleus'),
    algorithm_type=hd.ann.AnnotationGroupGenerationTypeValues.MANUAL,
    graphic_type=hd.ann.GraphicTypeValues.POINT,
    graphic_data=graphic_data,
)
```

Note that including two nuclei would be very unusual in practice: annotations often number in the thousands or even millions within a large whole slide image.

Including Measurements

In addition to the coordinates of the annotations themselves, it is also possible to attach one or more continuous-valued numeric *measurements* corresponding to those annotations. The measurements are passed as a *highdicom.ann.Measurements* object, which contains the *name* of the measurement (as a coded value), the *unit* of the measurement (also as a coded value) and an array of the measurements themselves (as a `numpy.ndarray`).

The length of the measurement array for any measurements attached to an annotation group must match exactly the number of annotations in the group. Value *i* in the array therefore represents the measurement of annotation *i*.

Here is the above example with an area measurement included:

```
from pydicom.sr.codedict import codes
from pydicom.sr.coding import Code
import highdicom as hd
import numpy as np

# Graphic data containing two nuclei, each represented by a single point
# expressed in 2D image coordinates
graphic_data = [
```

(continues on next page)

(continued from previous page)

```
    np.array([[34.6, 18.4]]),
    np.array([[28.7, 34.9]]),
]

# Measurement object representing the areas of each of the two nuclei
area_measurement = hd.ann.Measurements(
    name=codes.SCT.Area,
    unit=codes.UCUM.SquareMicrometer,
    values=np.array([20.4, 43.8]),
)

# Nuclei annotations produced by a manual algorithm
nuclei_group = hd.ann.AnnotationGroup(
    number=1,
    uid=hd.UID(),
    label='nuclei',
    annotated_property_category=codes.SCT.AnatomicalStructure,
    annotated_property_type=Code('84640000', 'SCT', 'Nucleus'),
    algorithm_type=hd.ann.AnnotationGroupGenerationTypeValues.MANUAL,
    graphic_type=hd.ann.GraphicTypeValues.POINT,
    graphic_data=graphic_data,
    measurements=[area_measurement],
)
```

Constructing MicroscopyBulkSimpleAnnotation Objects

When you have constructed the annotation groups, you can include them into a bulk annotation object along with a bit more metadata using the *highdicom.ann.MicroscopyBulkSimpleAnnotations* constructor. You also need to pass the image from which the annotations were derived so that *highdicom* can copy all the patient, study and slide-level metadata:

```
from pydicom import dcmread
import highdicom as hd

# Load a slide microscopy image from the highdicom test data (if you have
# cloned the highdicom git repo)
sm_image = dcmread('data/test_files/sm_image.dcm')

bulk_annotations = hd.ann.MicroscopyBulkSimpleAnnotations(
    source_images=[sm_image],
    annotation_coordinate_type=hd.ann.AnnotationCoordinateTypeValues.SCOORD,
    annotation_groups=[nuclei_group],
    series_instance_uid=hd.UID(),
    series_number=10,
    sop_instance_uid=hd.UID(),
    instance_number=1,
    manufacturer='MGH Pathology',
    manufacturer_model_name='MGH Pathology Manual Annotations',
    software_versions='0.0.1',
    device_serial_number='1234',
    content_description='Nuclei Annotations',
```

(continues on next page)

(continued from previous page)

```
)

bulk_annotations.save_as('nuclei_annotations.dcm')
```

The result is a complete DICOM object that can be written out as a DICOM file, transmitted over network, etc.

Reading Existing Bulk Annotation Objects

You can read an existing bulk annotation object from file using the *highdicom.ann.annread()* function:

```
from pydicom import dcmread
import highdicom as hd

ann = hd.ann.annread('data/test_files/sm_annotations.dcm')

assert isinstance(ann, hd.ann.MicroscopyBulkSimpleAnnotations)
```

Alternatively you can converting an existing `pydicom.Dataset` representing a bulk annotation object to the *highdicom* object like this:

```
from pydicom import dcmread
import highdicom as hd

ann_dcm = dcmread('data/test_files/sm_annotations.dcm')

ann = hd.ann.MicroscopyBulkSimpleAnnotations.from_dataset(ann_dcm)

assert isinstance(ann, hd.ann.MicroscopyBulkSimpleAnnotations)
```

Note that these examples (and the following examples) uses an example file that you can access from the test data in the *highdicom* repository. It was created using exactly the code in the construction example above.

Accessing Annotation Groups

Usually the next step when working with bulk annotation objects is to find the relevant annotation groups. You have two ways to do this.

If you know either the number or the UID of the group, you can access the group directly (since either of these should uniquely identify a group). The *highdicom.ann.MicroscopyBulkSimpleAnnotations.get_annotation_group()* method is used for this purpose:

```
# Access a group by its number
group = ann.get_annotation_group(number=1)
assert isinstance(group, hd.ann.AnnotationGroup)

# Access a group by its UID
group = ann.get_annotation_group(
    uid='1.2.826.0.1.3680043.10.511.3.406708363279713023756236135339936 86'
)
assert isinstance(group, hd.ann.AnnotationGroup)
```

Alternatively, you can search for groups that match certain filters such as the annotation property type or category, label, or graphic type. The *highdicom.ann.MicroscopyBulkSimpleAnnotations.get_annotation_groups()*

method (note groups instead of group) is used for this. It returns a list of matching groups, since the filters may match multiple groups.

```
from pydicom.sr.coding import Code

# Search for groups by annotated property type
groups = ann.get_annotation_groups(
    annotated_property_type=Code('84640000', 'SCT', 'Nucleus'),
)
assert len(groups) == 1 and isinstance(groups[0], hd.ann.AnnotationGroup)

# If there are no matches, an empty list is returned
groups = ann.get_annotation_groups(
    annotated_property_type=Code('53982002', "SCT", "Cell membrane"),
)
assert len(groups) == 0

# Search for groups by label
groups = ann.get_annotation_groups(label='nuclei')
assert len(groups) == 1 and isinstance(groups[0], hd.ann.AnnotationGroup)

# Search for groups by label and graphic type together (results must match
# *all* provided filters)
groups = ann.get_annotation_groups(
    label='nuclei',
    graphic_type=hd.ann.GraphicTypeValues.POINT,
)
assert len(groups) == 1 and isinstance(groups[0], hd.ann.AnnotationGroup)
```

Extracting Information From Annotation Groups

When you have found a relevant group, you can use the Python properties on the object to conveniently access metadata and the graphic data of the annotations. For example (see *highdicom.ann.AnnotationGroup* for a full list):

```
# Access the label
assert group.label == 'nuclei'

# Access the number
assert group.number == 1

# Access the UID
assert group.uid == '1.2.826.0.1.3680043.10.511.3.406708363279713023756236135339936 86'

# Access the annotated property type (returns a CodedConcept)
assert group.annotated_property_type == Code('84640000', 'SCT', 'Nucleus')

# Access the graphic type, describing the "form" of each annotation
assert group.graphic_type == hd.ann.GraphicTypeValues.POINT
```

You can access the entire array of annotations at once using *highdicom.ann.AnnotationGroup.get_graphic_data()*. You need to pass the annotation coordinate type from the parent bulk annotation object to the group so that it knows how to interpret the coordinate data. This method returns a list of 2D numpy arrays of shape (*N* x *D*), mirroring how you would have passed the data in to create the annotation with *highdicom*.

```
import numpy as np

graphic_data = group.get_graphic_data(
    coordinate_type=ann.annotation_coordinate_type,
)
assert len(graphic_data) == 2 and isinstance(graphic_data[0], np.ndarray)
```

Alternatively, you can access the coordinate array for a specific annotation using its (one-based) index in the annotation list:

```
# Get the number of annotations
assert group.number_of_annotations == 2

# Access an annotation using 1-based index
first_annotation = group.get_coordinates(
    annotation_number=1,
    coordinate_type=ann.AnnotationCoordinateType,
)
assert np.array_equal(first_annotation, np.array([[34.6, 18.4]]))
```

Extracting Measurements From Annotation Groups

You can use the *highdicom.ann.AnnotationGroup.get_measurements()* method to access any measurements included in the group. By default, this will return all measurements in the group, but you can also filter for measurements matching a certain name.

Measurements are returned as a tuple of `(names, values, units)`, where `names` is a list of nnames as `highdicom.sr.CondedConcept` objects, `units` is a list of units also as `highdicom.sr.CondedConcept` objects, and the values is a `numpy.ndarray` of values of shape (N by M) where N is the number of annotations and M is the number of measurements. This return format is intended to facilitate the loading of measurements into tables or dataframes for further analysis.

```
from pydicom.sr.codedict import codes

names, values, units = group.get_measurements()
assert names[0] == codes.SCT.Area
assert units[0] == codes.UCUM.SquareMicrometer
assert values.shape == (2, 1)
```

3.3.5 Parametric Maps

This page is under construction, and more detail will be added soon.

3.3.6 Presentation States

This page is under construction, and more detail will be added soon.

3.3.7 Secondary Capture (SC) Images

This page is under construction, and more detail will be added soon.

3.3.8 Legacy Converted Enhanced Images

This page is under construction, and more detail will be added soon.

CHAPTER

FOUR

DEVELOPER GUIDE

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

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

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

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

4.1 Pull requests

We encourage contributions from the users of the library (provided that they fit within the scope of the project).

If you are planning to make a contribution to the library, we encourage you to leave an issue first on the issue tracker detailing your proposed contribution. This way, the maintainers can vet your proposal, make sure it is within the scope of the project, and guide you through the process of creating a successful pull request. Before creating a pull request on Github, read the coding style guideline, run the tests and check PEP8 compliance.

We follow a gitflow-like process for development. Therefore, please do not commit code changes to the `master` branch. New features should be implemented in a separate branch called `feature/*`, and a pull request should be created with the target set as the development branch with the name of the *next* release (e.g. `v0.22.0dev`). Bug fixes that do not affect the public API of the project should be applied in separate branch called `bugfix/*` and a pull request should be created with targeted at `master` branch.

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

CODE OF CONDUCT

Highdicom has adopted the Contributor Covenant to govern the community around the project.

Find the full Code of Conduct here.

CHAPTER

SIX

CONFORMANCE STATEMENT

CHAPTER

SEVEN

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

EIGHT

LICENSE

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

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

API DOCUMENTATION

10.1 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*, *copy=True*)

Construct instance from an existing data element sequence.

Parameters

- **sequence** (*pydicom.sequence.Sequence*) – Data element sequence representing the Algorithm Identification Sequence
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for Content Qualification attribute.

PRODUCT = 'PRODUCT'

RESEARCH = 'RESEARCH'

SERVICE = 'SERVICE'

class highdicom.**CoordinateSystemNames**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for coordinate system names.

PATIENT = 'PATIENT'

SLIDE = 'SLIDE'

class highdicom.**DimensionOrganizationTypeValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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)

classmethod **from_dataset**(*dataset*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns

Issuer of identifier

Return type

highdicom.IssuerOfIdentifier

property **issuer_of_identifier: str**

Identifier of the issuer.

Type

str

Return type

str

property **issuer_of_identifier_type: Optional[*UniversalEntityIDTypeValues*]**

Type of the issuer.

Type

highdicom.UniversalEntityIDTypeValues

Return type

typing.Optional[*highdicom.enum.UniversalEntityIDTypeValues*]

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

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.**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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for Patient's Sex attribute.

F = 'F'

Female

M = 'M'

Male

O = 'O'

Other

class highdicom.**PhotometricInterpretationValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *copy=True*)

Create a PixelMeasuresSequence from an existing Sequence.

Parameters

- **sequence** (*pydicom.sequence.Sequence*) – Sequence to be converted.
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for Planar Representation attribute.

COMPLEMENT = 1

UNSIGNED_INTEGER = 0

class highdicom.**PlanarConfigurationValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *copy=True*)

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.
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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*, *copy=True*)

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.
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.

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.**PresentationLUTShapeValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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

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*, *specimen_type=None*, *specimen_short_description=None*, *specimen_detailed_description=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
- **specimen_type** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – The anatomic pathology specimen type of the specimen (see CID 8103 "Anatomic Pathology Specimen Type" for options).
- **specimen_short_description** (*str, optional*) – Short description of the examined specimen.
- **specimen_detailed_description** (*str, optional*) – Detailed description of the examined specimen.
- **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 issuer_of_specimen_id: Optional[*IssuerOfIdentifier*]

Issuer of identifier for the specimen.

Type
IssuerOfIdentifier

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

property primary_anatomic_structures: Optional[List[*CodedConcept*]]

List of anatomic structures of the specimen.

Type
List[*highdicom.sr.CodedConcept*]

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

property specimen_detailed_description: Optional[str]

Detailed description of specimen.

Type
str

Return type
typing.Optional[str]

property specimen_id: str

Specimen identifier.

Type
str

Return type
str

property specimen_location: Optional[Union[str, Tuple[float, float, float]]]

Specimen location in container.

Type
Tuple[float, float, float]

Return type
typing.Union[str, typing.Tuple[float, float, float], None]

property specimen_preparation_steps: List[*SpecimenPreparationStep*]

Specimen preparation steps.

Type
highdicom.SpecimenPreparationStep

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

property specimen_short_description: Optional[str]

Short description of specimen.

Type
str

Return type
typing.Optional[str]

property specimen_type: Optional[*CodedConcept*]

Specimen type.

Type
highdicom.sr.CodedConcept

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

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*, *specimen_container=None*, *specimen_type=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*) – The issuer of the identifier of the processed specimen.

- **fixative** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – Fixative used during processing (see CID 8114 "Specimen Fixative" for options).
- **embedding_medium** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – Embedding medium used during processing see CID 8115 "Specimen Embedding Media" for options).
- **specimen_container** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – Container the specimen resides in (see CID 8101 "Container Type" for options).
- **specimen_type** (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept], optional*) – The anatomic pathology specimen type of the specimen (see CID 8103 "Anatomic Pathology Specimen Type" for options).

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 issuer_of_specimen_id: Optional[str]

Issuer of specimen id

Type
str

Return type
typing.Optional[str]

property processing_datetime: Optional[datetime]

Processing datetime

Type
datetime.datetime

Return type
typing.Optional[datetime.datetime]

property processing_description: Optional[Union[str, *CodedConcept*]]

Processing description

Type
Union[str, *highdicom.sr.CodedConcept*]

Return type
typing.Union[str, *highdicom.sr.coding.CodedConcept*, None]

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_container: Optional[*CodedConcept*]

Specimen container

Type
highdicom.sr.CodedConcept

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

property specimen_id: str

Specimen identifier

Type
str

Return type
str

property specimen_type: Optional[*CodedConcept*]

Specimen type

Type
highdicom.sr.CodedConcept

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

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

append(*val*)

Append a content item to the sequence.

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

Return type
None

property **description:** *CodedConcept*

Processing step description

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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

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)

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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

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)
```

property info: str

Return the UID info from the UID dictionary.

Return type
str

property is_compressed: bool

Return True if a compressed transfer syntax UID.

Return type
bool

property is_deflated: bool

Return True if a deflated transfer syntax UID.

Return type
bool

property is_encapsulated: bool

Return True if an encasulated transfer syntax UID.

Return type
bool

property is_implicit_VR: bool

Return True if an implicit VR transfer syntax UID.

Return type
bool

property is_little_endian: bool

Return True if a little endian transfer syntax UID.

Return type
bool

property is_private: bool

Return True if the UID isn't an officially registered DICOM UID.

Return type
bool

property is_retired: bool

Return True if the UID is retired, False otherwise or if private.

Return type
bool

property is_transfer_syntax: bool

Return True if a transfer syntax UID.

Return type
bool

property is_valid: bool

Return True if *self* is a valid UID, False otherwise.

Return type
bool

property keyword: str

Return the UID keyword from the UID dictionary.

Return type
str

property name: str

Return the UID name from the UID dictionary.

Return type
str

property type: str

Return the UID type from the UID dictionary.

Return type
str

class highdicom.**UniversalEntityIDTypeValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

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.**VOILUTFunctionValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

10.1.1 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.

10.1.2 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** (*Union[PhotometricInterpretationValues, str]*) – 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 syntax)

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.

10.1.3 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
- **OSError** – 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
OSError – 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
OSError – When frame could not be read

10.1.4 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, 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, 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, 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, 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.**are_points_coplanar**(*points*, *tol=1e-05*)

Check whether a set of 3D points are coplanar (to within a tolerance).

Parameters

- **points** (*np.ndarray*) – Numpy array of shape (n x 3) containing 3D points.
- **tol** (*float*) – Tolerance on the distance of the furthest point from the plane of best fit.

Returns

True if the points are coplanar within a tolerance tol, False otherwise. Note that if n < 4, points are always coplanar.

Return type

bool

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.

10.1.5 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

10.1.6 highdicom.utils module

highdicom.utils.**are_plane_positions_tiled_full**(*plane_positions*, *rows*, *columns*)

Determine whether a list of plane positions matches "TILED_FULL".

This takes a list of plane positions for each frame and determines whether the plane positions satisfy the requirements of "TILED_FULL". Plane positions match the TILED_FULL dimension organization type if they are non-overlapping, and cover the entire image plane in the order specified in the standard.

The test implemented in this function is necessary and sufficient for the use of TILED_FULL in a newly created tiled image (thus allowing the plane positions to be omitted from the image and defined implicitly).

Parameters

- **plane_positions** (*Sequence[PlanePositionSequence]*) – Plane positions of each frame.
- **rows** (*int*) – Number of rows in each frame.
- **columns** (*int*) – Number of columns in each frame.

Returns
True if the supplied plane positions satisfy the requirements for TILED_FULL. False otherwise.

Return type
bool

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 for an image using the TILED_FULL DimensionOrganizationType.

Parameters
dataset (*pydicom.dataset.Dataset*) – VL Whole Slide Microscopy Image or Segmentation Image using the "TILED_FULL" DimensionOrganizationType.

Returns
Plane Position Sequence per frame

Return type
List[*highdicom.PlanePositionSequence*]

Raises
ValueError – When *dataset* does not represent a VL Whole Slide Microscopy Image or Segmentation Image or the image does not use the "TILED_FULL" dimension organization type.

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
Position, 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.**get_tile_array**(*pixel_array*, *row_offset*, *column_offset*, *tile_rows*, *tile_columns*, *pad=True*)

Extract a tile from a total pixel matrix array.

Parameters

- **pixel_array** (*np.ndarray*) – Array representing a total pixel matrix. The first two dimensions are treated as the rows and columns, respectively, of the total pixel matrix. Any subsequent dimensions are not used but are retained in the output array.
- **row_offset** (*int*) – Offset of the first row of the requested tile from the top of the total pixel matrix (1-based index).
- **column_offset** (*int*) – Offset of the first column of the requested tile from the left of the total pixel matrix (1-based index).
- **tile_rows** (*int*) – Number of rows per tile.
- **tile_columns** (int) – Number of columns per tile.
- **pad** (*bool*) – Whether to pad the returned array with zeros at the right and/or bottom to ensure that it matches the correct tile size. Otherwise, the returned array is not padded and may be smaller than the full tile size.

Returns
Returned pixel array for the requested tile.

Return type
np.ndarray

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.**iter_tiled_full_frame_data**(*dataset*)

Get data on the position of each tile in a TILED_FULL image.

This works only with images with Dimension Organization Type of "TILED_FULL".

Unlike *highdicom.utils.compute_plane_position_slide_per_frame()*, this functions returns the data in their basic Python types rather than wrapping as *highdicom.PlanePositionSequence*

Parameters
dataset (*pydicom.dataset.Dataset*) – VL Whole Slide Microscopy Image or Segmentation Image using the "TILED_FULL" DimensionOrganizationType.

Return type
typing.Generator[typing.Tuple[int, int, int, int, float, float, float], None, None]

Returns

- **channel** (*int*) – 1-based integer index of the "channel". The meaning of "channel" depends on the image type. For segmentation images, the channel is the segment number. For other images, it is the optical path number.

- **focal_plane_index** (*int*) – 1-based integer index of the focal plane.
- **column_position** (*int*) – 1-based column position of the tile (measured left from the left side of the total pixel matrix).
- **row_position** (*int*) – 1-based row position of the tile (measured down from the top of the total pixel matrix).
- **x** (*float*) – X coordinate in the frame-of-reference coordinate system in millimeter units.
- **y** (*float*) – Y coordinate in the frame-of-reference coordinate system in millimeter units.
- **z** (*float*) – Z coordinate in the frame-of-reference coordinate system in millimeter units.

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

Tiles an image into smaller frames (rectangular regions).

Follows the convention used in image with Dimension Organization Type "TILED_FULL" images.

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

10.2 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*

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.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*

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.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*

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

10.3 highdicom.ann package

Package for creation of Annotation (ANN) instances.

class highdicom.ann.**AnnotationCoordinateTypeValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *copy=True*)

Construct instance from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an item of the Annotation Group Sequence.
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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 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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for attribute Annotation Group Generation Type.

AUTOMATIC = 'AUTOMATIC'

MANUAL = 'MANUAL'

SEMIAUTOMATIC = 'SEMIAUTOMATIC'

class highdicom.ann.**GraphicTypeValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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 point 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 four ordered coordinates.

The coordinates shall be coplanar and represent 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 fourth 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*, *copy=True*)

Construct instance from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an item of the Measurements Sequence.
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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]*) – 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*

property annotation_coordinate_type: *AnnotationCoordinateTypeValues*

Annotation coordinate type.

Type
highdicom.ann.AnnotationCoordinateTypeValues

Return type
highdicom.ann.enum.AnnotationCoordinateTypeValues

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

classmethod from_dataset(*dataset*, *copy=True*)

Construct instance from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing a Microscopy Bulk Simple Annotations instance.
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

highdicom.ann.**annread**(*fp*)

Read a bulk annotations object stored in DICOM File Format.

Parameters

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

Returns

Bulk annotations object read from the file.

Return type

highdicom.ann.MicroscopyBulkSimpleAnnotations

10.4 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

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=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_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*]

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*]]

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.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 fulfilled 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

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

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]

10.5 highdicom.pm package

Package for creation of Parametric Map instances.

class highdicom.pm.**DerivedPixelContrastValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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 transforming 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*).

---

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.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.

10.6 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*

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.pr.**AnnotationUnitsValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*

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.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 (maximum 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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*

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.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 transforming 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 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
- **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*

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.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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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

10.7 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:
...     print(name)
RowPositionInTotalImagePixelMatrix
ColumnPositionInTotalImagePixelMatrix
XOffsetInSlideCoordinateSystem
YOffsetInSlideCoordinateSystem
```

(continues on next page)

(continued from previous page)

```
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*) – Array of dimension index values. The first dimension corresponds to the items in the input plane_positions sequence. The second dimension corresponds to the dimensions of the dimension index. The third dimension (if any) corresponds to the multiplicity of the values, and is omitted if this is 1 for all dimensions.
- **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*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *copy=True*)

Construct instance from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an item of the Segment Sequence.
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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*, *workers=0*, *dimension_organization_type=None*, *tile_pixel_array=False*, *tile_size=None*, *pyramid_uid=None*, *pyramid_label=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 output segmentation series.
- **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"), JPEG 2000 Lossless ("1.2.840.10008.1.2.4.90"), and JPEG LS Lossless ("1.2.840.10008.1.2.4.00").

- **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.
- **workers** (*Union[int, concurrent.futures.Executor], optional*) – Number of worker processes to use for frame compression. If 0, no workers are used and compression is performed in the main process (this is the default behavior). If negative, as many processes are created as the machine has processors.

  Alternatively, you may directly pass an instance of a class derived from concurrent.futures.Executor (most likely an instance of concurrent.futures.ProcessPoolExecutor) for highdicom to use. You may wish to do this either to have greater control over the setup of the executor, or to avoid the setup cost of spawning new processes each time this __init__ method is called if your application creates a large number of Segmentations.

  Note that if you use worker processes, you must ensure that your main process uses the if __name__ == "__main__" idiom to guard against spawned child processes creating further workers.
- **dimension_organization_type** (*Union[highdicom.enum.DimensionOrganizationTypeValues, str, None], optional*) – Dimension organization type to use for the output image.
- **tile_pixel_array** (*bool, optional*) – If True, *highdicom* will automatically convert an input total pixel matrix into a sequence of frames representing tiles of the segmentation. This is valid only when the source image supports tiling (e.g. VL Whole Slide Microscopy images).

  If True, the input pixel array must consist of a single "frame", i.e. must be either a 2D numpy array, a 3D numpy array with a size of 1 down the first dimension (axis zero), or a 4D numpy array also with a size of 1 down the first dimension. The input pixel array is treated as the total pixel matrix of the segmentation, and this is tiled along the row and column dimension to create an output image with multiple, smaller frames.

If no `pixel_measures`, `plane_positions`, `plane_orientation` are supplied, the total pixel matrix of the segmentation is assumed to correspond to the total pixel matrix of the (single) source image. If `plane_positions` is supplied, the sequence should contain a single item representing the plane position of the entire total pixel matrix. Plane positions of the newly created tiles will derived automatically from this.

If False, the pixel array is already considered to consist of one or more existing frames, as described above.

- **tile_size** (*Union[Sequence[int], None], optional*) – Tile size to use when tiling the input pixel array. If `None` (the default), the tile size is copied from the source image. Otherwise the tile size is specified explicitly as (number of rows, number of columns). This value is ignored if `tile_pixel_array` is False.
- **pyramid_uid** (*Optional[str], optional*) – Unique identifier for the pyramid containing this segmentation. Should only be used if this segmentation is part of a multi-resolution pyramid.
- **pyramid_label** (*Optional[str], optional*) – Human readable label for the pyramid containing this segmentation. Should only be used if this segmentation is part of a multi-resolution pyramid.
- ****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.

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

classmethod from_dataset(*dataset*, *copy=True*)

Create instance from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing a Segmentation image.
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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*, *skip_overlap_checks=False*, *dtype=None*)

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) according 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.
- **skip_overlap_checks** (*bool*) – If True, skip checks for overlap between different segments. By default, checks are performed to ensure that the segments do not overlap. However, this reduces performance. If checks are skipped and multiple segments do overlap, the segment with the highest segment number (after relabelling, if applicable) will be placed into the output array.
- **dtype** (*Union[type, str, numpy.dtype, None]*) – Data type of the returned array. If None, an appropriate type will be chosen automatically. If the returned values are rescaled fractional values, this will be numpy.float32. Otherwise, the smallest unsigned integer type that accommodates all of the output values will be chosen.

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*, *skip_overlap_checks=False*, *dtype=None*)

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** (*Optional[Sequence[int]], 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) according 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 and is larger than the frame number of the highest referenced frame, highdicom is unable to check that the frame number is valid in the source image. By default, highdicom will raise an error in this situation. 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.
- **skip_overlap_checks** (*bool*) – If True, skip checks for overlap between different segments. By default, checks are performed to ensure that the segments do not overlap. However, this reduces performance. If checks are skipped and multiple segments do overlap, the segment with the highest segment number (after relabelling, if applicable) will be placed into the output array.
- **dtype** (*Union[type, str, numpy.dtype, None]*) – Data type of the returned array. If None, an appropriate type will be chosen automatically. If the returned values are rescaled fractional values, this will be numpy.float32. Otherwise, the smallest unsigned integer type that accommodates all of the output values will be chosen.

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], dtype=uint8))
```

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
```

(continues on next page)

(continued from previous page)

```
... )
>>> pixels.shape, np.unique(pixels)
((3, 10, 10), array([0, 1, 2, 3], dtype=uint8))
```

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*, *skip_overlap_checks=False*, *dtype=None*)

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) according 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.
- **skip_overlap_checks** (*bool*) – If True, skip checks for overlap between different segments. By default, checks are performed to ensure that the segments do not overlap. However, this reduces performance. If checks are skipped and multiple segments do overlap, the segment with the highest segment number (after relabelling, if applicable) will be placed into the output array.
- **dtype** (*Union[type, str, numpy.dtype, None]*) – Data type of the returned array. If None, an appropriate type will be chosen automatically. If the returned values are rescaled fractional values, this will be numpy.float32. Otherwise, the smallest unsigned integer type that accommodates all of the output values will be chosen.

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
```

(continues on next page)

(continued from previous page)

```
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_total_pixel_matrix(*row_start=1*, *row_end=None*, *column_start=1*, *column_end=None*, *segment_numbers=None*, *combine_segments=False*, *relabel=False*, *rescale_fractional=True*, *skip_overlap_checks=False*, *dtype=None*)

Get the pixel array as a (region of) the total pixel matrix.

This is intended for retrieving segmentation masks derived from multi-frame (enhanced) source images that are tiled. The method returns (a region of) the 2D total pixel matrix implied by the frames within the segmentation.

The output array will have 3 dimensions under the default behavior, and 2 dimensions if `combine_segments` is set to `True`. The first two dimensions are the rows and columns of the total pixel matrix, respectively. By default, the full total pixel matrix is returned, however a smaller region may be requested using the `row_start`, `row_end`, `column_start` and `column_end` parameters as 1-based indices into the total pixel matrix.

When `combine_segments` is `False` (the default behavior), the segments are stacked down the final (3rd) 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

- **row_start** (*int, optional*) – 1-based row index in the total pixel matrix of the first row to include in the output array. May be negative, in which case the last row is considered index -1.
- **row_end** (*Union[int, None], optional*) – 1-based row index in the total pixel matrix of the first row beyond the last row to include in the output array. A `row_end` value of `n` will include rows `n - 1` and below, similar to standard Python indexing. If `None`, rows up until the final row of the total pixel matrix are included. May be negative, in which case the last row is considered index -1.
- **column_start** (*int, optional*) – 1-based column index in the total pixel matrix of the first column to include in the output array. May be negative, in which case the last column is considered index -1.
- **column_end** (*Union[int, None], optional*) – 1-based column index in the total pixel matrix of the first column beyond the last column to include in the output array. A `column_end` value of `n` will include columns `n - 1` and below, similar to standard Python indexing. If `None`, columns up until the final column of the total pixel matrix are included. May be negative, in which case the last column is considered index -1.
- **segment_numbers** (*Optional[Sequence[int]], 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) according to the position of the original segment numbers in `segment_numbers` parameter. If `combine_segments` is `False`, this has no effect.
- **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.

- **skip_overlap_checks** (*bool*) – If True, skip checks for overlap between different segments. By default, checks are performed to ensure that the segments do not overlap. However, this reduces performance. If checks are skipped and multiple segments do overlap, the segment with the highest segment number (after relabelling, if applicable) will be placed into the output array.
- **dtype** (*Union[type, str, numpy.dtype, None]*) – Data type of the returned array. If None, an appropriate type will be chosen automatically. If the returned values are rescaled fractional values, this will be numpy.float32. Otherwise, the smallest unsigned integer type that accommodates all of the output values will be chosen.

Returns
pixel_array – Pixel array representing the segmentation's total pixel matrix.

Return type
np.ndarray

Note: This method uses 1-based indexing of rows and columns in order to match the conventions used in the DICOM standard. The first row of the total pixel matrix is row 1, and the last is `self.TotalPixelMatrixRows`. This is is unlike standard Python and NumPy indexing which is 0-based. For negative indices, the two are equivalent with the final row/column having index -1.

---

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for attribute Segmentation Fractional Type.

OCCUPANCY = 'OCCUPANCY'

PROBABILITY = 'PROBABILITY'

class highdicom.seg.**SegmentationTypeValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for attribute Segmentation Type.

BINARY = 'BINARY'

FRACTIONAL = 'FRACTIONAL'

class highdicom.seg.**SegmentsOverlapValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enumerated values for attribute Segments Overlap.

NO = 'NO'

UNDEFINED = 'UNDEFINED'

YES = 'YES'

class highdicom.seg.**SpatialLocationsPreservedValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.**create_segmentation_pyramid**(*source_images*, *pixel_arrays*, *segmentation_type*, *segment_descriptions*, *series_instance_uid*, *series_number*, *manufacturer*, *manufacturer_model_name*, *software_versions*, *device_serial_number*, *downsample_factors=None*, *sop_instance_uids=None*, *pyramid_uid=None*, *pyramid_label=None*, ***kwargs*)

Construct a multi-resolution segmentation pyramid series.

A multi-resolution pyramid represents the same segmentation array at multiple resolutions.

This function handles multiple related scenarios:

- Constructing a segmentation of a source image pyramid given a segmentation pixel array of the highest resolution source image. Highdicom performs the downsampling automatically to match the resolution of the other source images. For this case, pass multiple `source_images` and a single item in `pixel_arrays`.
- Constructing a segmentation of a source image pyramid given user-provided segmentation pixel arrays for each level in the source pyramid. For this case, pass multiple `source_images` and a matching number of `pixel_arrays`.
- Constructing a segmentation of a single source image given multiple user-provided downsampled segmentation pixel arrays. For this case, pass a single item in `source_images`, and multiple items in `pixel_arrays`).
- Constructing a segmentation of a single source image and a single segmentation pixel array by downsampling by a given list of `downsample_factors`. For this case, pass a single item in `source_images`, a single item in `pixel_arrays`, and a list of one or more desired `downsample_factors`.

In all cases, the items in both `source_images` and `pixel_arrays` should be sorted in pyramid order from highest resolution (smallest spacing) to lowest resolution (largest spacing), and the pixel array in `pixel_arrays[0]` must be the segmentation of the source image in `source_images[0]` with spatial locations preserved (a one-to-one correspondence between pixels in the source image's total pixel matrix and the provided segmentation pixel array).

In all cases, the provided pixel arrays should be total pixel matrices. Tiling is performed automatically.

Parameters

- **source_images** (*Sequence[pydicom.Dataset]*) – List of source images. If there are multiple source images, they should be from the same series and pyramid.

- **pixel_arrays** (*Sequence[numpy.ndarray]*) – List of segmentation pixel arrays. Each should be a total pixel matrix.
- **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_number** (*int*) – Number of the output segmentation series.
- **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
- **downsample_factors** (*Optional[Sequence[float]], optional*) – Factors by which to downsample the pixel array to create each of the output segmentation objects. This should be provided if and only if a single source image and single pixel array are provided. Note that the original array is always used to create the first segmentation output, so the number of created segmententation instances is one greater than the number of items in this list. Items must be numbers greater than 1 and sorted in ascending order. A downsampling factor of *n* implies that the output array is *1/n* time the size of input pixel array. For example a list `[2, 4, 8]` would be produce 4 output segmentation instances. The first is the same size as the original pixel array, the next is half the size, the next is a quarter of the size of the original, and the last is one eighth the size of the original. Output sizes are rounded to the nearest integer.
- **series_instance_uid** (*Optional[str], optional*) – UID of the output segmentation series. If not specified, UIDs are generated automatically using highdicom's prefix.
- **sop_instance_uids** (*Optional[List[str]], optional*) – SOP instance UIDS of the output instances. If not specified, UIDs are generated automatically using highdicom's prefix.
- **pyramid_uid** (*Optional[str], optional*) – UID for the output imaging pyramid. If not specified, a UID is generated using highdicom's prefix.
- **pyramid_label** (*Optional[str], optional*) – A human readable label for the output pyramid.
- ****kwargs** (*Any*) – Any further parameters are passed directly to the constructor of the :class:highdicom.seg.Segmentation object. However the following parameters are disallowed: `instance_number`, `sop_instance_uid`, `plane_orientation`, `plane_positions`, `pixel_measures`, `pixel_array`, `tile_pixel_array`.

Note: Downsampling is performed via simple nearest neighbor interpolation. If more control is needed over the downsampling process (for example anti-aliasing), explicitly pass the downsampled arrays.

Return type
typing.List[*highdicom.seg.sop.Segmentation*]

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

10.7.1 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.

10.8 highdicom.sr package

Package for creation 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

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type CODE
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.CodeContentItem

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: *CodedConcept*

coded concept

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct a CodedConcept from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing a coded concept.
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type COMPOSITE
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.CompositeContentItem

property name: *CodedConcept*

coded name of the content item

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

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 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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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** (*Union[pydicom.dataset.Dataset, pydicom.sequence.Sequence]*) – Root container content items that should be included in the SR document. This should either be a single dataset, or a sequence of datasets containing a single item.
- **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 fulfilled 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.

property content: *ContentSequence*

SR document content

Type
highdicom.sr.value_types.ContentSequence

Return type
highdicom.sr.value_types.ContentSequence

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

classmethod from_dataset(*dataset*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing a Comprehensive 3D SR document
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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** (*Union[pydicom.dataset.Dataset, pydicom.sequence.Sequence]*) – Root container content items that should be included in the SR document. This should either be a single dataset, or a sequence of datasets containing a single item.
- **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 fulfilled 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.

property content: *ContentSequence*

SR document content

Type
highdicom.sr.value_types.ContentSequence

Return type
highdicom.sr.value_types.ContentSequence

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

classmethod from_dataset(*dataset*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing a Comprehensive SR document
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type CONTAINER
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.ContainerContentItem

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 template_id: Optional[str]

template identifier

Type
Union[str, None]

Return type
typing.Optional[str]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type DATE
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.DateContentItem

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: date

date

Type
datetime.date

Return type
datetime.date

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type DATETIME
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.DateTimeContentItem

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: datetime

datetime

Type
datetime.datetime

Return type
datetime.datetime

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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

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*)

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

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

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 (restricted 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** (*Union[pydicom.dataset.Dataset, pydicom.sequence.Sequence]*) – Root container content items that should be included in the SR document. This should either be a single dataset, or a sequence of datasets containing a single item.
- **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 fulfilled 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.

property content: *ContentSequence*

SR document content

Type
highdicom.sr.value_types.ContentSequence

Return type
highdicom.sr.value_types.ContentSequence

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

classmethod from_dataset(*dataset*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing a Comprehensive SR document
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
SR document

Return type
highdicom.sr.sop._SR

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Constructed object

Return type
highdicom.sr.FindingSite

property laterality: Optional[CodedConcept]

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

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 topographical_modifier: Optional[CodedConcept]

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

property value: CodedConcept

coded concept

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

property value_type: ValueTypeValues

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

class highdicom.sr.GraphicTypeValues(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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 reference 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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type IMAGE
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.ImageContentItem

property name: *CodedConcept*

coded name of the content item

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

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 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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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.

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.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

`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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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*, *copy=True*)

Construct object from an existing dataset.

Parameters

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

- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Constructed object

Return type
highdicom.sr.ImageRegion

property graphic_type: *GraphicTypeValues*

graphic type

Type
GraphicTypeValues

Return type
highdicom.sr.enum.GraphicTypeValues

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: ndarray

n x 2 array of 2D spatial coordinates

Type
numpy.ndarray

Return type
numpy.ndarray

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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

property frame_of_reference_uid: *UID*

frame of reference UID

Type
highdicom.UID

Return type
highdicom.uid.UID

classmethod from_dataset(*dataset*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Constructed object

Return type
highdicom.sr.ImageRegion3D

property graphic_type: *GraphicTypeValues3D*

graphic type

Type
GraphicTypeValues3D

Return type
highdicom.sr.enum.GraphicTypeValues3D

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: ndarray

n x 3 array of 3D spatial coordinates

Type
numpy.ndarray

Return type
numpy.ndarray

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Constructed object

Return type
highdicom.sr.LongitudinalTemporalOffsetFromEvent

property name: *CodedConcept*

coded name of the content item

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 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 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]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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 corresponding 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

append(*val*)

Append a content item to the sequence.

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

Return type
None

property derivation: Optional[*CodedConcept*]

derivation

Type
Union[*highdicom.sr.CodedConcept*, None]

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

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

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

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

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

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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

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=True*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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_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*]

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*]

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.**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

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*, *copy=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

- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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.

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

property finding_category: Optional[*CodedConcept*]

finding category

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*]

property finding_type: Optional[*CodedConcept*]

finding type

Type
Union[*highdicom.sr.CodedConcept*, None]

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

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._MeasurementsAndQualitativeEvaluations

get_measurements(*name=None*)

Get measurements.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Name of measurement

Returns
Measurements

Return type
List[*highdicom.sr.Measurement*]

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*]

get_qualitative_evaluations(*name=None*)

Get qualitative evaluations.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Name of evaluation

Returns
Qualitative evaluations

Return type
List[*highdicom.sr.QualitativeEvaluation*]

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

property method: Optional[*CodedConcept*]

measurement method

Type
Union[*highdicom.sr.CodedConcept*, None]

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

property source_images: List[*SourceImageForMeasurementGroup*]

source images

Type
List[*highdicom.sr.SourceImageForMeasurementGroup*]

Return type
typing.List[*highdicom.sr.content.SourceImageForMeasurementGroup*]

property tracking_identifier: Optional[str]

tracking identifier

Type
Union[str, None]

Return type
typing.Optional[str]

property tracking_uid: Optional[*UID*]

tracking unique identifier

Type
Union[*highdicom.UID*, None]

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

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

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*, *copy=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

- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type NUM
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.NumContentItem

property **name**: *CodedConcept*

coded name of the content item

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 **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 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]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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)

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.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

`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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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

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

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*)

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

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

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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 corresponding 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.

---

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

property finding_category: Optional[*CodedConcept*]

finding category

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*]

property finding_type: Optional[*CodedConcept*]

finding type

Type
Union[*highdicom.sr.CodedConcept*, None]

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

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

get_measurements(*name=None*)

Get measurements.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Name of measurement

Returns
Measurements

Return type
List[*highdicom.sr.Measurement*]

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*]

get_qualitative_evaluations(*name=None*)

Get qualitative evaluations.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Name of evaluation

Returns
Qualitative evaluations

Return type
List[*highdicom.sr.QualitativeEvaluation*]

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

property method: Optional[*CodedConcept*]

measurement method

Type
Union[*highdicom.sr.CodedConcept*, None]

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

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]

property tracking_identifier: Optional[str]

tracking identifier

Type
Union[str, None]

Return type
typing.Optional[str]

property tracking_uid: Optional[*UID*]

tracking unique identifier

Type
Union[*highdicom.UID*, None]

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

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*, *copy=True*)

Construct object from existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type PNAME
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.PnameContentItem

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: PersonName

person name

Type
pydicom.valuerep.PersonName

Return type
pydicom.valuerep.PersonName

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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

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*)

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

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

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

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

- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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

property name: *CodedConcept*

coded name of the content item

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

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 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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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.

---

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_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

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*]

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

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

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

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_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

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

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*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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 millimeter 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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD3D

- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.Scoord3DContentItem

property graphic_type: *GraphicTypeValues3D*

graphic type

Type
GraphicTypeValues3D

Return type
highdicom.sr.enum.GraphicTypeValues3D

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: ndarray

n x 3 array of 3D spatial coordinates

Type
numpy.ndarray

Return type
numpy.ndarray

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns

Content Item

Return type

highdicom.sr.ScoordContentItem

property graphic_type: *GraphicTypeValues*

graphic type

Type

GraphicTypeValues

Return type

highdicom.sr.enum.GraphicTypeValues

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: ndarray

n x 2 array of 2D spatial coordinates

Type
numpy.ndarray

Return type
numpy.ndarray

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type IMAGE
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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

property name: *CodedConcept*

coded name of the content item

Type

highdicom.sr.CodedConcept

Return type

highdicom.sr.coding.CodedConcept

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 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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type IMAGE
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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

property name: *CodedConcept*

coded name of the content item

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

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 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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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

property name: *CodedConcept*

coded name of the content item

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

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 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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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

property name: *CodedConcept*

coded name of the content item

Type
highdicom.sr.CodedConcept

Return type
highdicom.sr.coding.CodedConcept

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 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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type SCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

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

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: *UID*

UID

Type
highdicom.UID

Return type
highdicom.uid.UID

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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

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

append(*val*)

Append a content item to the sequence.

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

Return type
None

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]

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*)

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

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.SubjectContextFetus(*subject_id*)

Bases: Template

TID 1008 Subject Context Fetus

Parameters
subject_id (*str*) – identifier of the fetus for longitudinal tracking

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*)

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

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

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)

append(*val*)

Append a content item to the sequence.

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

Return type
None

property container_identifier: Optional[str]

specimen container identifier

Type
Union[str, None]

Return type
typing.Optional[str]

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*)

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

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

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type TCOORD
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.TcoordContentItem

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 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]]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

class highdicom.sr.TemporalRangeTypeValues(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)**

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type TEXT
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.TextContentItem

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: str

text value

Type
str

Return type
str

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type TIME
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.TimeContentItem

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: time

time

Type
datetime.time

Return type
datetime.time

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

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

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*, *copy=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
- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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

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*, *copy=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

- **copy** (*bool*) – If True, the underlying sequence is deep-copied such that the original sequence remains intact. If False, this operation will alter the original sequence in place.

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.**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*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type UIDREF
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns
Content Item

Return type
highdicom.sr.UIDRefContentItem

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**: *UID*

UID

Type
highdicom.UID

Return type
highdicom.uid.UID

property **value_type**: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

class highdicom.sr.**ValueTypeValues**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.

---

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

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

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*]

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

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

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.

---

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

property finding_category: Optional[*CodedConcept*]

finding category

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*]

property finding_type: Optional[*CodedConcept*]

finding type

Type
Union[*highdicom.sr.CodedConcept*, None]

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

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

get_measurements(*name=None*)

Get measurements.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Name of measurement

Returns
Measurements

Return type
List[*highdicom.sr.Measurement*]

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*]

`get_qualitative_evaluations`(*name=None*)

Get qualitative evaluations.

Parameters
name (*Union[pydicom.sr.coding.Code, highdicom.sr.CodedConcept, None], optional*) – Name of evaluation

Returns
Qualitative evaluations

Return type
List[*highdicom.sr.QualitativeEvaluation*]

`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

`property method: Optional[CodedConcept]`

measurement method

Type
Union[*highdicom.sr.CodedConcept*, None]

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

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]

property tracking_identifier: Optional[str]

tracking identifier

Type
Union[str, None]

Return type
typing.Optional[str]

property tracking_uid: Optional[*UID*]

tracking unique identifier

Type
Union[*highdicom.UID*, None]

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

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

Bases: *ContentItem*

DICOM SR document content item for value type WAVEFORM.

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_waveform_channels** (*Union[Sequence[Tuple[int, int]], None], optional*) – Pairs of waveform number (number of item in the Waveform Sequence) and channel definition number (number of item in the Channel Definition Sequence) to which the reference applies in case of a multi-channel waveform
- **relationship_type** (*Union[highdicom.sr.RelationshipTypeValues, str, None], optional*) – Type of relationship with parent content item

classmethod from_dataset(*dataset*, *copy=True*)

Construct object from an existing dataset.

Parameters

- **dataset** (*pydicom.dataset.Dataset*) – Dataset representing an SR Content Item with value type WAVEFORM
- **copy** (*bool*) – If True, the underlying dataset is deep-copied such that the original dataset remains intact. If False, this operation will alter the original dataset in place.

Returns

Content Item

Return type

highdicom.sr.WaveformContentItem

property name: *CodedConcept*

coded name of the content item

Type

highdicom.sr.CodedConcept

Return type

highdicom.sr.coding.CodedConcept

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 referenced_waveform_channels: Optional[List[Tuple[int, int]]]

referenced waveform channels

Type
Union[List[Tuple[int, int]], None]

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

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: 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*]

property value_type: *ValueTypeValues*

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

Type
ValueTypeValues

Return type
highdicom.sr.enum.ValueTypeValues

highdicom.sr.**srread**(*fp*)

Read a Structured Report (SR) document in DICOM File format.

The object is returned as an instance of the highdicom class corresponding to the dataset's IOD. Currently supported IODs are:

- Enhanced SR via class *EnhancedSR*
- Comprehensive SR via class *ComprehensiveSR*
- Comprehensive 3D SR via class *Comprehensive3DSR*

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

Raises
RuntimeError: – If the DICOM file has an IOD not supported by highdicom.

Returns
Structured Report document read from the file.

Return type
Union[*highdicom.sr.EnhancedSR*, *highdicom.sr.ComprehensiveSR*, *highdicom.sr.Comprehensive3DSR*]

10.8.1 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

10.9 highdicom.sc package

Package for creation of Secondary Capture (SC) Image instances.

class highdicom.sc.**ConversionTypeValues**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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.SpecimenDescription], 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*

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

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.SpecimenDescription], 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

ELEVEN

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

S

T

U

V

W

X

Y

Z