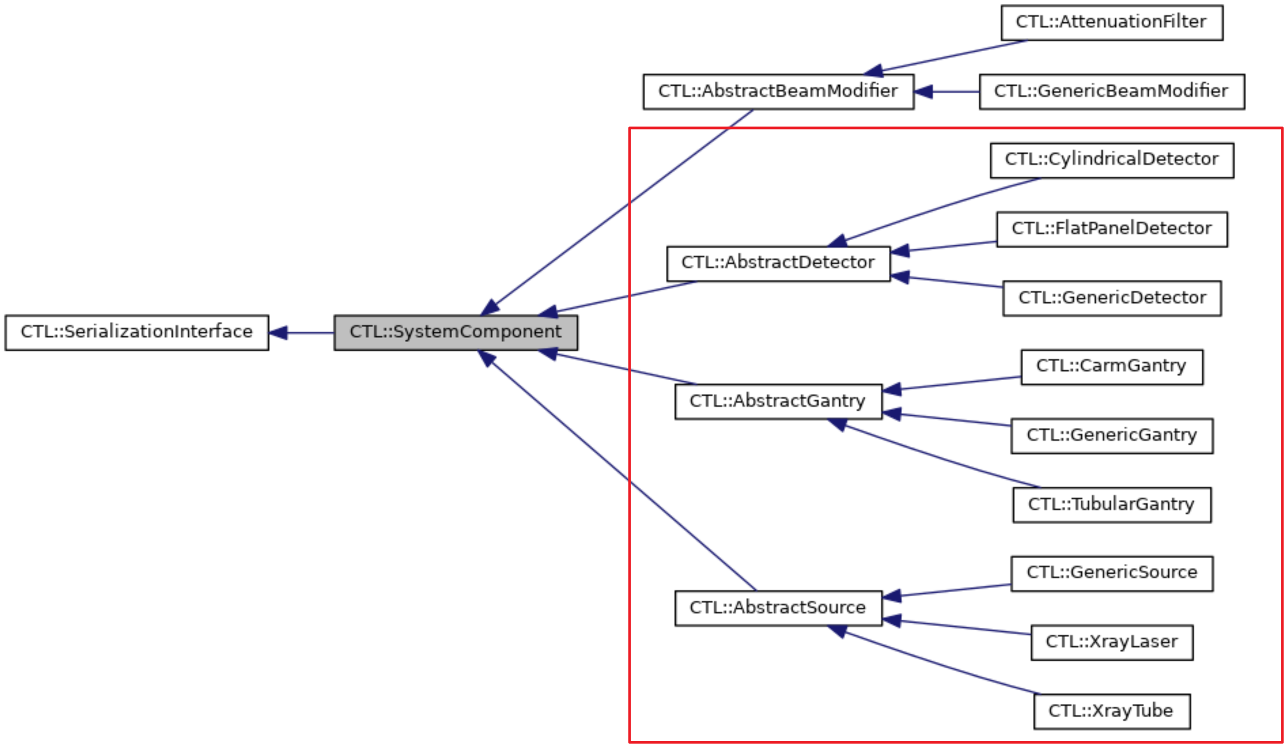
System components in the CTL

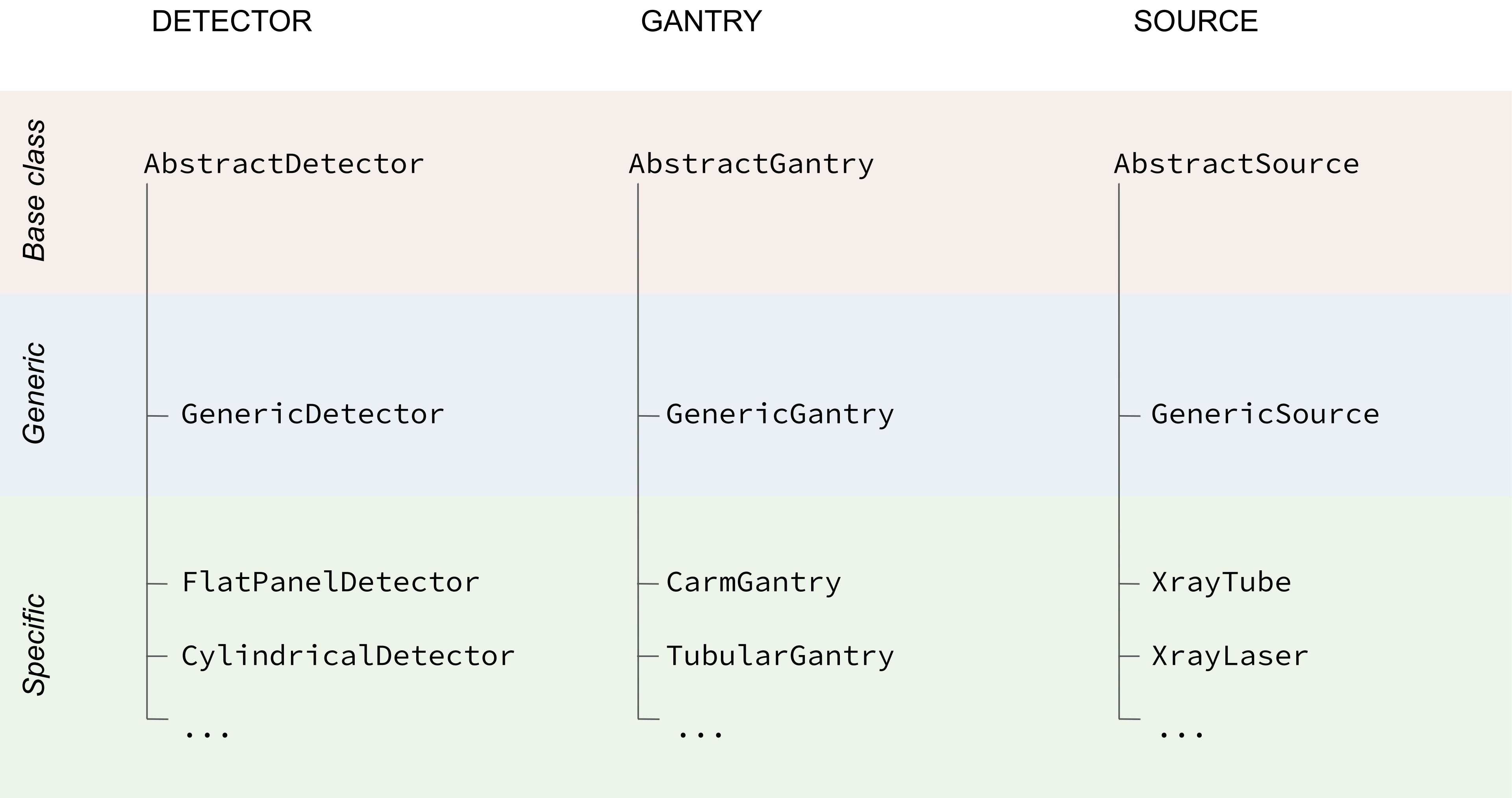


System components in the CTL

Some basic principles:

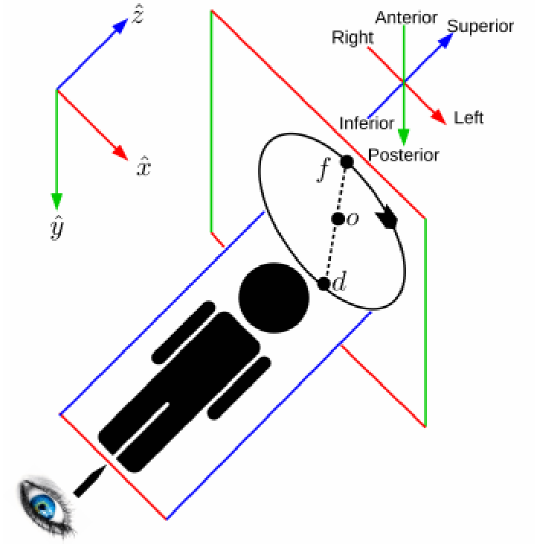
- Components **do not interact** with each other
- Each component manages **only its own** characterizing **properties**
- 4 principle types: **gantries**, **detectors**, **sources**, and beam modifiers
- Each type has a **general interface**
- Sub-classes are used to make **trade-offs** between flexibility and user convenience

Base class (abstract)	• Defines all required interfaces used throughout the CTL. • Includes pure virtual methods that need to be implemented by derived classes. • Can contain other virtual methods that can be overridden if required.	**non-instantiable**
Generic	• a generic, non-restrictive representation of the component type. • Implements (pure) virtual methods from base class. • Provides flexible but complex manipulation (setter).	**full flexibility** **complex interface**
Specific	• specific, “task-related” representations of the component type. • Implement (pure) virtual methods from base class. • Provide simple & convenient manipulation (setter).	**restrictive usage** **convenient interface**

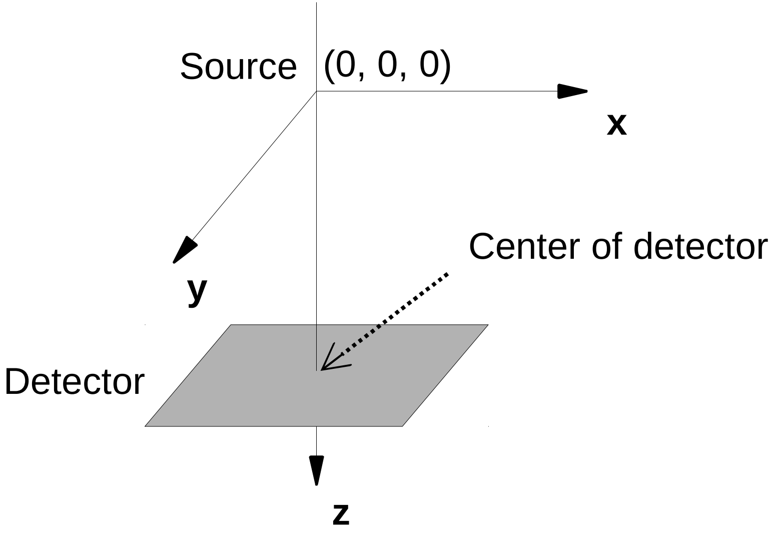

DETECTOR
GANTRY
SOURCE
Base class
Generic
Specific
AbstractDetector
AbstractGantry
AbstractSource
GenericDetector
GenericGantry
GenericSource
FlatPanelDetector
CarmGantry
XrayTube
CylindricalDetector
TubularGantry
XrayLaser
...
...
...

Background: Geometry - coordinate systems

WORLD COORDINATE SYSTEM (WCS)



CT COORDINATE SYSTEM (CTS)



Background: Geometry – `mat::Location`

`mat::Location`

- stores the position of a component in WCS
- also contains a rotation matrix describing the orientation
  → used for different purposes
  - [detector modules] transformation from a single module to whole detector
  - [gantry: source] transformation from CTS to WCS
  - [gantry: detector] transformation from WCS to CTS

```cpp
struct Location
{
    Vector3x1 position = Vector3x1(0.0);
    Matrix3x3 rotation = mat::eye<3>();

    Location() = default;
    Location(const Vector3x1& pos, const Matrix3x3& rot);

    QVariant toVariant() const;
    void fromVariant(const QVariant& variant);
};
```

```cpp
typedef mat::Matrix<3, 3> Matrix3x3;
typedef mat::Matrix<3, 1> Vector3x1;
```

GANTRY

Base class

`AbstractGantry`

Generic

- `GenericGantry`

Specific

- `CarmGantry`
- `TubularGantry`
- ...

GANTRY

Base class

AbstractGantry

- manages the location of a source and detector component in WCS
- does not know what kind of source and detector is “mounted”

```
// abstract interface
protected:virtual mat::Location nominalDetectorLocation() const = 0;
protected:virtual mat::Location nominalSourceLocation() const = 0;
```

- provide the unmodified locations of the detector and source component
  → not intended to be used as typical getter (--> protected); instead:

```
// getter methods
mat::Location sourceLocation() const;
mat::Location detectorLocation() const;
```

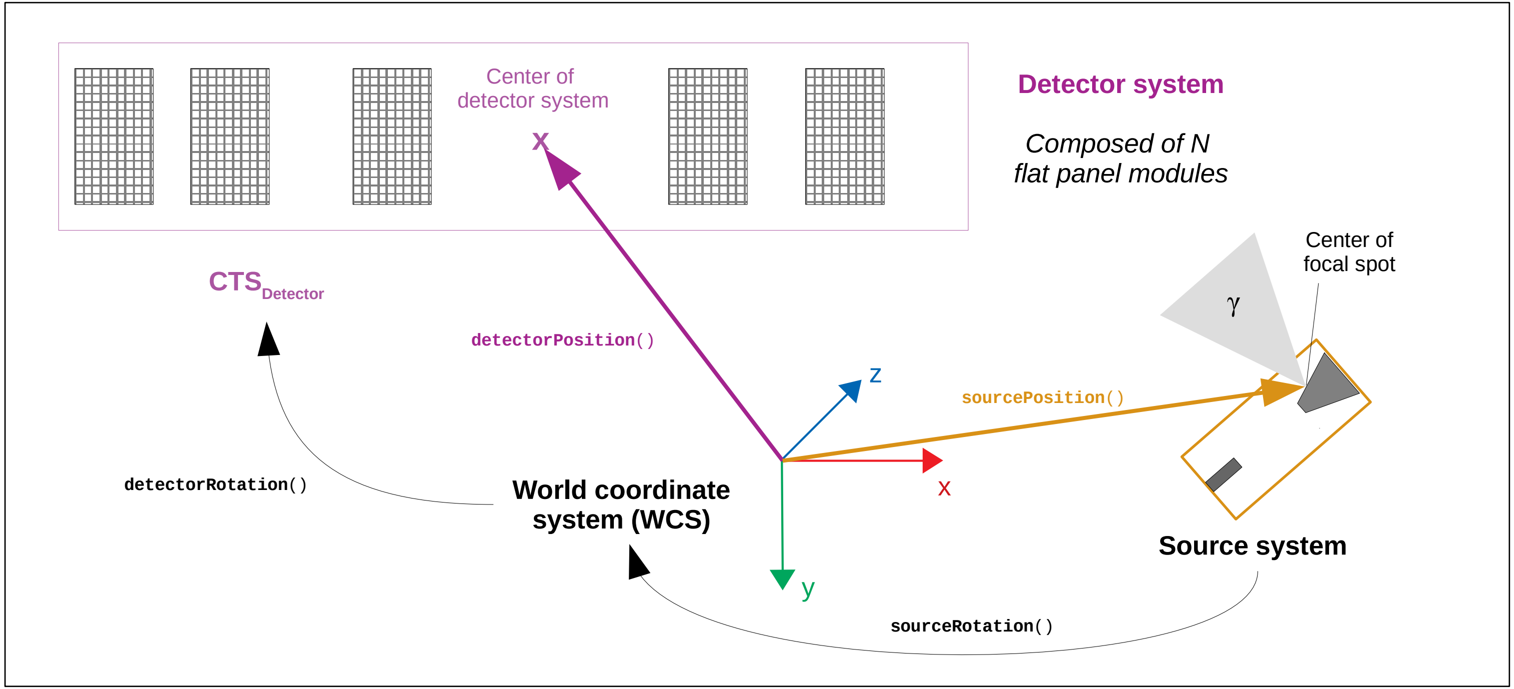
```
// convenience getter
Vector3x1 sourcePosition() const;
Matrix3x3 sourceRotation() const;
Vector3x1 detectorPosition() const;
Matrix3x3 detectorRotation() const;
```

- return the final location of the components, ie. including all potential **displacements** (later)

GANTRY

Base class

AbstractGantry



GANTRY

Base class

AbstractGantry

- **Displacements:**
  additional geometric distortions applied to the components in addition to their nominal geometry

```
// setter methods
void setDetectorDisplacement(const mat::Location& displacement);
void setGantryDisplacement(const mat::Location& displacement);
void setSourceDisplacement(const mat::Location& displacement);
```

→ can be used for several purposes, eg.:

- Miscalibration
- Motion
- Quarter-detector-shift

SOURCE geometry

$$t_{\text{src}}^{\text{total}} = R_{\text{gantry}}^{\text{displ}} \cdot t_{\text{src}}^{\text{nominal}} + R_{\text{src}}^{\text{total}} \cdot t_{\text{src}}^{\text{displ}} + t_{\text{gantry}}^{\text{displ}}$$

$$R_{\text{src}}^{\text{total}} = R_{\text{gantry}}^{\text{displ}} \cdot R_{\text{src}}^{\text{nominal}} \cdot R_{\text{src}}^{\text{displ}}$$

DETECTOR geometry

$$t_{\text{det}}^{\text{total}} = R_{\text{gantry}}^{\text{displ}} \cdot t_{\text{det}}^{\text{nominal}} + \left(R_{\text{det}}^{\text{total}}\right)^T \cdot t_{\text{det}}^{\text{displ}} + t_{\text{gantry}}^{\text{displ}}$$

$$R_{\text{det}}^{\text{total}} = \left(R_{\text{det}}^{\text{displ}}\right)^T \cdot R_{\text{det}}^{\text{nominal}} \cdot \left(R_{\text{gantry}}^{\text{displ}}\right)^T$$

GANTRY

Generic

Specific

GenericGantry

- Allows free specification of the location of detector and source component

```
// setter methods
void setDetectorLocation(const mat::Location& location);
void setSourceLocation(const mat::Location& location);
```

TubularGantry

- Source and detector component move on circular orbit with fixed diameter
- Orbit can be tilted around x-axis
- Configuration is fully specified by:
  - (gantry) rotation angle
  - (gantry) tilt angle
  - (table) pitch position

```
// setter methods
void setPitchPosition(double position);
void setRotationAngle(double angle);
void setTiltAngle(double angle);
```

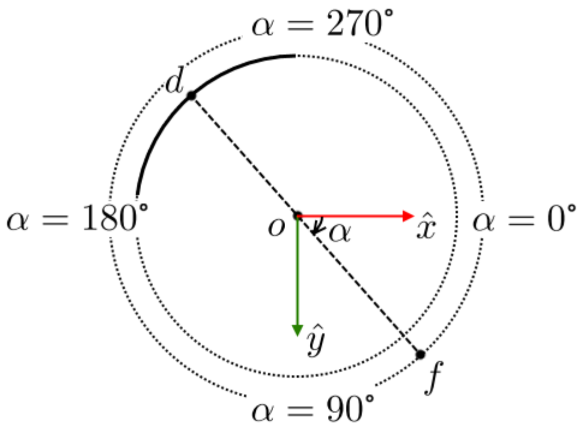
GANTRY

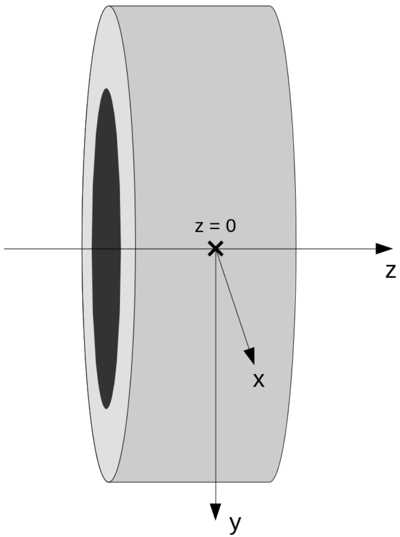
Specific

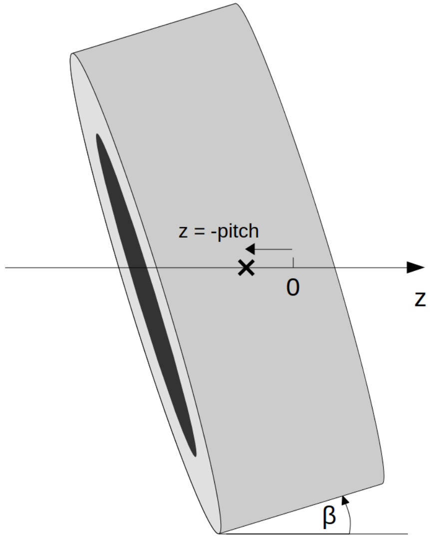
TubularGantry

```
void setRotationAngle(double angle);
```

```
void setPitchPosition(double position);
void setTiltAngle(double angle);
```







GANTRY

Specific

CarmGantry

- Source and detector component move as a union with a mutable distance called “C-arm span”
- Configuration is fully specified by:
  - C-arm span
  - Location of the source (ie. position in WCS and rotation matrix to get there)

```
// setter methods
void setLocation(const mat::Location& location);
void setCarmSpan(double span);
```

DETECTOR

Base class

```
AbstractDetector
```

Generic

```
├─ GenericDetector
```

Specific

```
├─ FlatPanelDetector
├─ CylindricalDetector
└─ ...
```

DETECTOR

Base class

AbstractDetector

- Contains a set of identical flat panel detector modules
  → all modules must have same number of pixels and same pixel size (and same skew)
- Each module has its own location, specifying its geometry with respect to the detector system as a whole (see next slide)

```
// abstract interface
public:virtual QVector<ModuleLocation> moduleLocations() const = 0;
```

- Two (optional) response models
  - Saturation model → response in "intensity" domain (intensity, counts, or extinction)
  - Spectral response model → response in spectral domain

```
// setter methods
void setSaturationModel(AbstractDataModel* model,
                        SaturationModelType type);
void setSpectralResponseModel(AbstractDataModel* model);

bool hasSaturationModel() const;
bool hasSpectralResponseModel() const;
```

considered within:

DetectorSaturationExtension

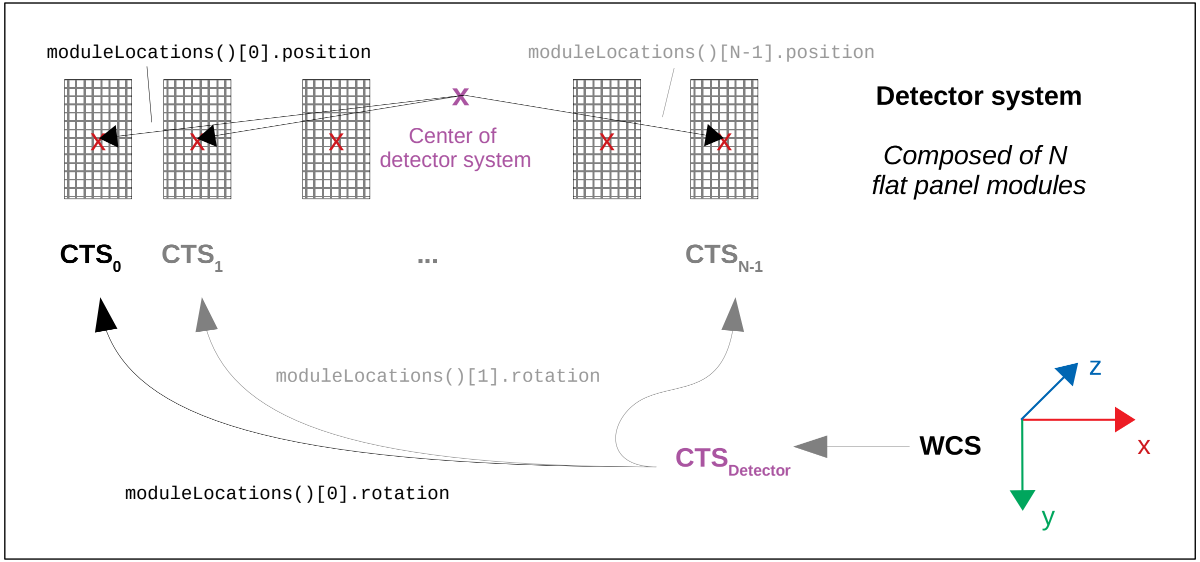
SpectralEffectsExtension

DETECTOR

Base class

AbstractDetector

```
// abstract interface
public:virtual QVector<ModuleLocation> moduleLocations() const = 0;
```



DETECTOR

Generic

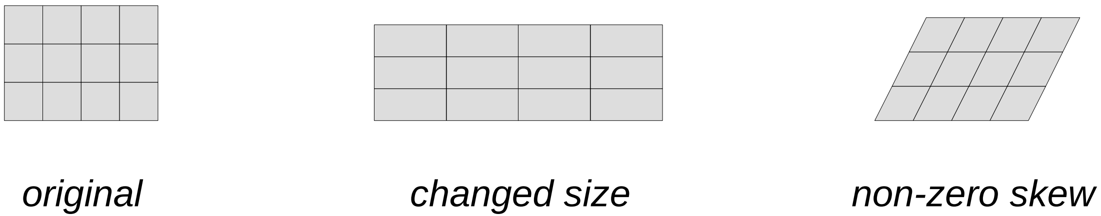
GenericDetector

- Free specification of the location of individual detector modules

```
// setter methods
void setModuleLocations(QVector<ModuleLocation> moduleLocations);
```

- Also allows for dynamic (ie. in between views) change of pixel sizes and allows setting of a non-zero skew factor

```
void setPixelSize(const QSizeF& size);
void setSkewCoefficient(double skewCoefficient);
```



DETECTOR

Specific

`FlatPanelDetector`

- Simplest implementation
- All parameters fixed from moment of construction
  - Fixed pixel size
  - Always skew factor 0
  - Always a **single module** with location 0 (ie. no translation, no rotation)

  → no setters

`CylindricalDetector`

- Places individual detector modules on the surface of a cylinder
  → represents typical curved CT detectors
- All parameters fixed from moment of construction
  - Fixed cylinder radius
  - Fixed number of modules
  - Fixed pixel size
  - Always skew factor 0

DETECTOR

Specific

CylindricalDetector

- Different ways of construction possible
  - Angulation + spacing
  - Radius + fan angle

```
static CylindricalDetector fromAngulationAndSpacing(const QSize& nbPixelPerModule,
                                                    const QSizeF& pixelDimensions,
                                                    uint nbDetectorModules,
                                                    double angulationPerModule,
                                                    double moduleSpacing,
                                                    const QString& name = defaultName());
```

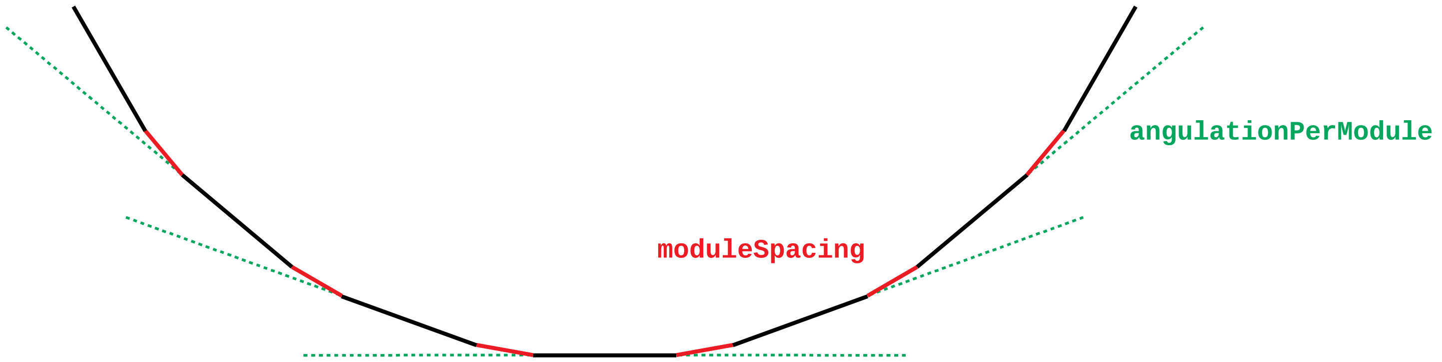
```
static CylindricalDetector fromRadiusAndFanAngle(const QSize& nbPixelPerModule,
                                                 const QSizeF& pixelDimensions,
                                                 uint nbDetectorModules,
                                                 double radius,
                                                 double fanAngle,
                                                 const QString& name = defaultName());
```

DETECTOR

Specific

`CylindricalDetector`

- Different ways of construction possible
  - **Angulation + spacing**
  - Radius + fan angle

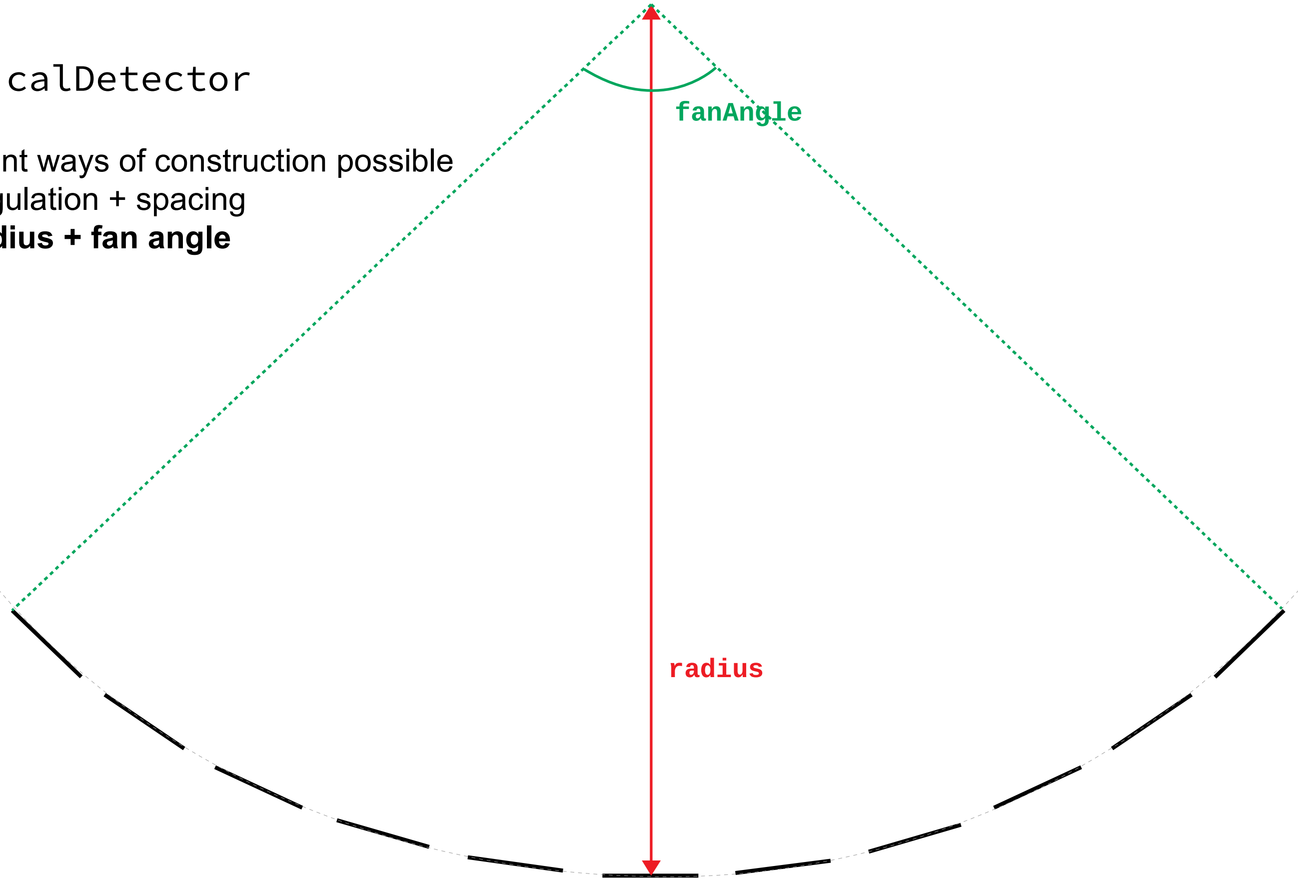


DETECTOR

Specific

`CylindricalDetector`

- Different ways of construction possible
  - Angulation + spacing
  - **Radius + fan angle**



Specific

CylindricalDetector

- Different ways of construction possible
  - Angulation + spacing
  - Radius + fan angle
- several details on the configuration can be retrieved through various getters

```
// getter methods
double angulationOfModule(uint module) const;
double moduleSpacing() const;

// other methods
double coneAngle() const;
double curvatureRadius() const;
double fanAngle() const;
double rowCoverage() const;
```

SOURCE

Base class

```
AbstractSource
│
│
├─ GenericSource
│
├─ XrayTube
│
├─ XrayLaser
│
└─
   ...
```

Generic

Specific

SOURCE

Base class

AbstractSource

- Source properties are characterized by
  - Energy range of emitted radiation (lowest to highest energy that is emitted)
  - Photon flux, ie. the total number of photons emitted to an area of 1cm² in one meter distance

```
// abstract interface
public:virtual EnergyRange nominalEnergyRange() const = 0;
protected:virtual double nominalPhotonFlux() const = 0;
```

```
typedef Range<float> EnergyRange;
```

- Similar to gantry classes, these are not intended to be used as (general purpose) getters, instead:

```
// getter methods
EnergyRange energyRange() const;
double photonFlux() const;
```

- Spectral properties are defined by a so-called spectrum model

```
// virtual methods
virtual IntervalDataSeries spectrum(uint nbSamples) const;
virtual uint spectrumDiscretizationHint() const;
virtual void setSpectrumModel(AbstractXraySpectrumModel* model);
```

SOURCE

Base class

AbstractSource

```
// virtual methods
virtual IntervalDataSeries spectrum(uint nbSamples) const;
virtual uint spectrumDiscretizationHint() const;
virtual void setSpectrumModel(AbstractXraySpectrumModel* model);
```

- `setSpectrumModel()` – setter for the spectrum model (can be restricted to certain types in reimplementations)
- `spectrum()` – returns a data series with values sampled from the spectrum model covering the full energy range with a given number of samples (note: relative photon counts!)
- `spectrumDiscretizationHint()` – returns a hint for a reasonable number of samples to use when sampling the spectrum (default 10)

- Some convenience getters:

```
float meanEnergy() const;
bool hasSpectrumModel() const;
```

self-explanatory

```
IntervalDataSeries spectrum(EnergyRange range, uint nbSamples) const;
```

- returns a data series with values sampled from the spectrum model covering only the specified energy range with a given number of samples

SOURCE

Base class

AbstractSource

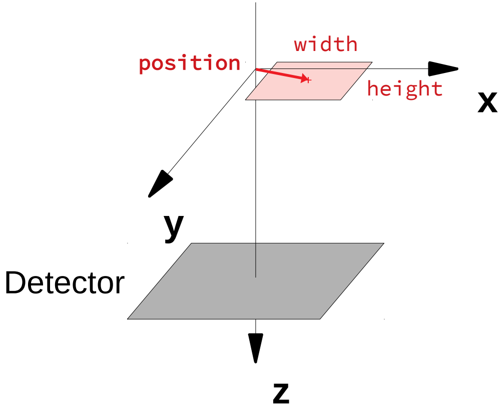
- More general specifications:

```
// setter methods
void setFocalSpotSize(const QSizeF& size);
void setFocalSpotSize(double width, double height); // convenience alternative
void setFocalSpotPosition(const Vector3x1& position);
void setFocalSpotPosition(double x, double y, double z); // convenience alternative

void setFluxModifier(double modifier);
void setEnergyRangeRestriction(const EnergyRange& window);
```

- flux modifier: global (multiplicative) manipulation of the photon flux
- energy range restriction: restricts the energy range to a narrower interval

- focal spot specifications given in CT coordinate system



GenericSource

- Parameters can be set directly:

```
// setter methods
void setEnergyRange(const EnergyRange& range);
void setSpectrum(const IntervalDataSeries& spectrum, bool updateFlux = false);
void setPhotonFlux(double flux);
```

`setSpectrum()`:

- data in `spectrum` is put into a `TabulatedDataModel` and used whenever a spectrum is requested from the component
- also updates the energy range to the smallest/largest value found in `spectrum`
- if `updateFlux==true`, integral over `spectrum` is used as new photon flux

→ useful, for example, to use externally generated spectra in the CTL

- another useful convenience feature:

```
static void setPhotonCountInSystem(SimpleCTSystem* system, double photonsPerPixel);
```

→ adjusts the photon flux in `system` such that each detector pixel in the current configuration receives (on average) `photonsPerPixel` photons (`system` must contain a `GenericSource` !)

SOURCE

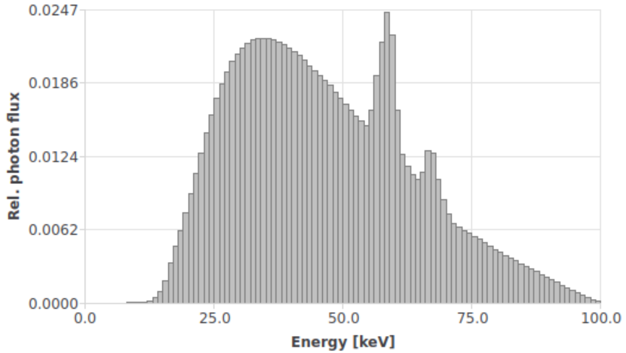
Specific

XrayTube

- Represents a typical X-ray tube with parameters
  - Tube voltage → determines energy range and spectrum (rel. photon counts per energy)
  - Milliampere seconds (mAs) → determines photon flux

```
// setter methods
void setTubeVoltage(double voltage);
void setMilliampereSeconds(double mAs);
```

- Internally uses a spectrum model according to the TASMIP model [1]
  → valid for tube voltages from 30 to 140 kV



[1] Boone and Seibert, Med Phys. 1997 Nov;24(11):1661-70. doi: 10.1118/1.597953.

SOURCE

Specific

XrayLaser

- Source component that emits a photons of a single energy

```
// setter
void setPhotonEnergy(double energy);
void setRadiationOutput(double output);
```

→ total energy emitted to 1cm² in 1m distance (in mWs)

- Photon flux is computed from output power and photon energy

```
radiationOutput() / (ELEC_VOLT * photonEnergy() * 1.0e3)
```

→ changing photon energy will change the flux, but keeps total emitted energy constant

- Energy range is [`photonEnergy()`, `photonEnergy()`]
- `spectrumDisctretizationHint()` is 1