

ParameterDefinitions

```
/// Components of a bound track parameters vector.
///
/// To be used to access components by named indices instead of just numbers.
/// This must be a regular `enum` and not a scoped `enum class` to allow
/// implicit conversion to an integer. The enum value are thus visible directly
/// in `namespace Acts` and are prefixed to avoid naming collisions.
enum BoundParametersIndices : unsigned int {
    // Local position on the reference surface.
    // This is intentionally named different from the position components in
    // the other data vectors, to clarify that this is defined on a surface
    // while the others are defined in free space.
    eBoundLoc0 = 0,
    eBoundLoc1 = 1,
    // Direction angles
    eBoundPhi = 2,
    eBoundTheta = 3,
    // Global inverse-momentum-like parameter, i.e. q/p or 1/p
    // The naming is inconsistent for the case of neutral track parameters where
    // the value is interpreted as 1/p not as q/p. This is intentional to avoid
    // having multiple aliases for the same element and for lack of an acceptable
    // common name.
    eBoundQOverP = 4,
    eBoundTime = 5,
    // Last uninitialized value contains the total number of components
    eBoundParametersSize,
    // The following aliases without prefix exist for historical reasons
    // Generic spatial coordinates on the local surface
    eLOC_0 = eBoundLoc0,
    eLOC_1 = eBoundLoc1,
    // Spatial coordinates on a disk in polar coordinates
    eLOC_R = eLOC_0,
    eLOC_PHI = eLOC_1,
    // Spatial coordinates on a disk in Cartesian coordinates
    eLOC_X = eLOC_0,
    eLOC_Y = eLOC_1,
    // Spatial coordinates on a cylinder
    eLOC_RPHI = eLOC_0,
    eLOC_Z = eLOC_1,
    // Closest approach coordinates on a virtual perigee surface
    eLOC_D0 = eLOC_0,
    eLOC_Z0 = eLOC_1,
    // Direction angles
    ePHI = eBoundPhi,
    eTHETA = eBoundTheta,
    eQOP = eBoundQOverP,
    eT = eBoundTime,
};
```

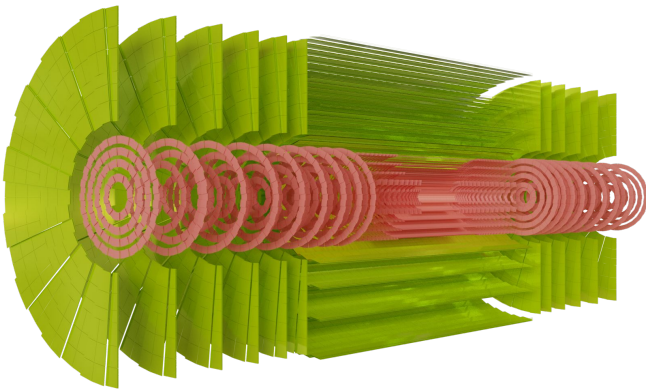
```
/// Components of a free track parameters vector.
///
/// To be used to access components by named indices instead of just numbers.
/// This must be a regular `enum` and not a scoped `enum class` to allow
/// implicit conversion to an integer. The enum value are thus visible directly
/// in `namespace Acts` and are prefixed to avoid naming collisions.
enum FreeParametersIndices : unsigned int {
    // Spatial position
    // The spatial position components must be stored as one continuous block.
    eFreePos0 = 0u,
    eFreePos1 = eFreePos0 + 1u,
    eFreePos2 = eFreePos0 + 2u,
    // Time
    eFreeTime = 3u,
    // (Unit) direction
    // The direction components must be stored as one continuous block.
    eFreeDir0 = 4u,
    eFreeDir1 = eFreeDir0 + 1u,
    eFreeDir2 = eFreeDir0 + 2u,
    // Global inverse-momentum-like parameter, i.e. q/p or 1/p
    // See BoundParametersIndices for further information
    eFreeQOverP = 7u,
    // Last uninitialized value contains the total number of components
    eFreeParametersSize,
};
```

- 2 enums for different parametrisation
- Corresponding integers represent different dimensions

ParameterSet

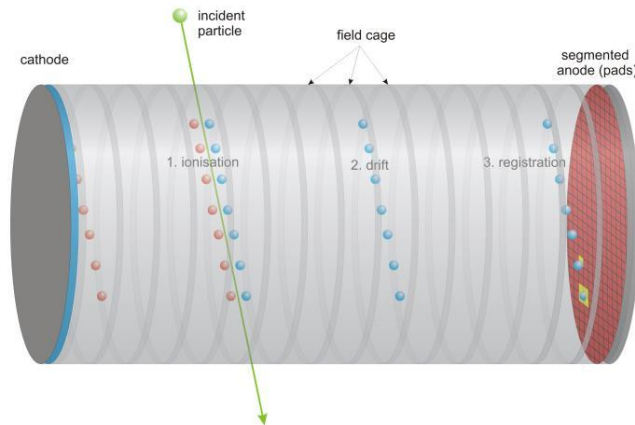
Old:

```
template <ParID_t... params>
class ParameterSet {
```



New:

```
template <typename parameter_indices_t, parameter_indices_t... params>
class ParameterSet {
```



- This class stores the parameters for an arbitrary (sub)set of parameter indices
- Old design was fixed for BoundParametersIndices
 - Parameters are always represented on a surface
- New design allows custom specification of set of indices
 - But enforces consistency in used set of indices

Measurement

Current master:

```
template <typename source_link_t, ParID_t... params>
class Measurement {
```

Same setup as for ParameterSet...

private:

```
ParSet_t m_oParameters; ///< measured parameter set
std::shared_ptr<const Surface>
    m_pSurface; ///< surface at which the measurement took place
```

...because Measurement builds a ParameterSet

```
source_link_t m_sourceLink; ///< link to the source for this measurement
```

#272:

```
template <typename source_link_t, typename parameter_indices_t,
          parameter_indices_t... params>
```

```
class Measurement {
```

Same solution as for ParameterSet

private:

```
ParamSet m_oParameters; ///< measured parameter set
std::shared_ptr<const RefObject> m_pReferenceObject =
    nullptr;              ///< object which corresponds to the measurement
source_link_t m_sourceLink; ///< link to the source for this measurement
```

ReferenceObject

Current master:

private:

```
ParSet_t m_oParameters; ///< measured parameter set
std::shared_ptr<const Surface>
    m_pSurface; ///< surface at which the measurement took place

source_link_t m_sourceLink; ///< link to the source for this measurement
```

#272:

private:

```
ParamSet m_oParameters; ///< measured parameter set
std::shared_ptr<const RefObject> m_pReferenceObject =
    nullptr; ///< object which corresponds to the measurement
source_link_t m_sourceLink; ///< link to the source for this measurement
```

/// @brief Deduction of the measuring geometry object based on the used indices

```
template <typename T>
struct ReferenceObject {};
template <>
struct ReferenceObject<BoundParametersIndices> {
    using type = Surface;
};
template <>
struct ReferenceObject<FreeParametersIndices> {
    using type = Volume;
};
```

- Master: A measurement is always bound to a surface
- #272: A measurement is either bound to a surface or a volume
- Decision is based on the indices of the measurement
- Measurement is templated on indices
→ Decision at compile time