MTConnect® Standard

Guide: MTConnect and OPC/UA Companion Specification

Version 2.0

Prepared for: MTConnect Institute
Prepared by: William Sobel
Prepared on: September 29, 2018

September 29, 2018

MTConnect® Specification and Materials

Table of Contents

List of Figures

1 Introduction

The following conventions will be used throughout the document to provide a clear and consistent understanding of the use of each type of data and information used to define the MTConnect® standard and associated data.

1.1 Overview

Overview of the standards...

2 Types

2.1 Components

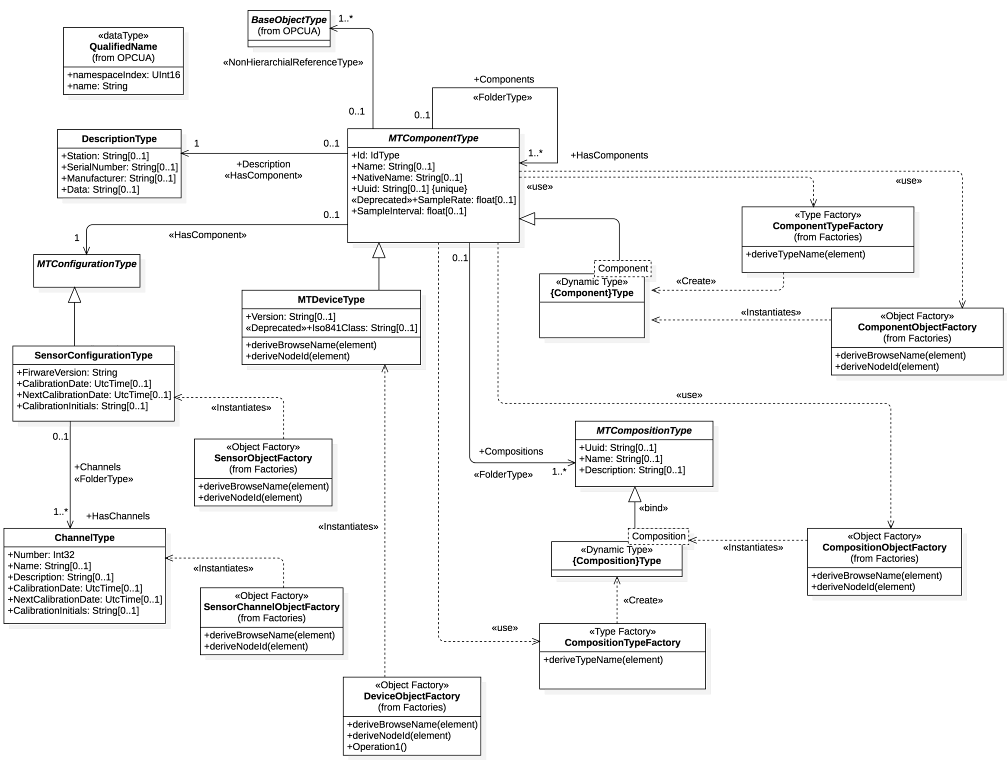

Figure 1: Components Diagram

The Components documents the Component models and the owned objects.

2.1.1 Defintion of `ChannelType`

Table 1: `ChannelType` Definition

Attribute	**Value**				
BrowseName	ChannelType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of BaseObjectType (See OPCUA Documentation)					
HasProperty	Variable	Number	Int32	PropertyType	Manditory
HasProperty	Variable	Name	String	PropertyType	Optional
HasProperty	Variable	MTDescription	String	PropertyType	Optional
HasProperty	Variable	CalibrationDate	UtcTime	PropertyType	Optional
HasProperty	Variable	NextCalibrationDate	UtcTime	PropertyType	Optional
HasProperty	Variable	CalibrationInitials	String	PropertyType	Optional

2.1.2 Defintion of `DescriptionType`

The desription provides some general information about the manufacture and serial number of the component. In the XML, the `CDATA` is freeform text that is represented in the `Data` Property of the Description Object. The description is related to the component with the OPC/UA `HasComponent` relationship.

Table 2: `DescriptionType` Definition

Attribute	**Value**				
BrowseName	DescriptionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of BaseObjectType (See OPCUA Documentation)					
HasProperty	Variable	Station	String	PropertyType	Optional
HasProperty	Variable	SerialNumber	String	PropertyType	Optional
HasProperty	Variable	Manufacturer	String	PropertyType	Optional
HasProperty	Variable	Data	String	PropertyType	Optional

2.1.2.1 Operations

- `deriveBrowseName(element)`
  Specification: `"Description"`
- `deriveNodeId(element)`
  Specification: `concat(self.parent.NodeId, BrowseName)`

2.1.3 Defintion of `MTComponentType`

The base Component Type from which all MTConnect Components are derived. The component type factory is used to create the specific OPC/UA Types as sub-types of the MTConnect `MTComponentType`. The component types will be created once for all Component objects of that type based on the `QName` of the MTConnect XML element.

The object factory will instantiate the Component Objects and insert them into the Components folder with a browse name of the Component QName and the `name` element if specified surrounded by square brackets, `[]`. For example if the MTConnect Element is:

```
<Linear name='X'>...</...>
```

The OPC/UA Object with browse name `Linear[X]` will be created with the HasTypeDefinition referencing the `Linear` OPC/UA type.

The meta data for the component and it's relationships are static. The dynamic data will be represented using the *OPC/UA Part 8*.

Table 3: `MTComponentType` Definition

Attribute	Value				
BrowseName	MTComponentType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
HasProperty	Variable	XmlId	IdType	PropertyType	Manditory
HasProperty	Variable	Name	String	PropertyType	Optional
HasProperty	Variable	NativeName	String	PropertyType	Optional
HasProperty	Variable	Uuid	String	PropertyType	Optional
HasProperty	Variable	SampleRate	float	PropertyType	Optional
HasProperty	Variable	SampleInterval	float	PropertyType	Optional
HasComponent	Object	Description		DescriptionType	Optional
HasComponent	Object	Configuration		MTConfigurationType	Optional
Organizes	Object	Components	MTComponentType	FolderType	Optional
Organizes	Object	Compositions	MTCompositionType	FolderType	Optional
HasProperty	Variable	<Dynamic>	BaseObjectType	<Dynamic>	Optional
Organizes	Object	Conditions	AlarmConditionType	FolderType	Optional
HasProperty	Variable	<Dynamic>	DataItemType	<Dynamic>	Optional

2.1.4 Defintion of `MTCompositionType`

The `MTCompositionType` is the abstract supertype of the dynamically generated composition types based on the attribute `type` of the `Composition` element of the MTConnect `Component`. The `Composition` is then related to the DataItems that reference the `Composition`'s id in their `compositionId` attribute.

The data items are added to the relationship where the `DataItem` to `Composition` relationship is represented by the `BrowseName` Composition property of the data item the data items are added by their browse names to the `Composition`.

Table 4: `MTCompositionType` Definition

Attribute	Value				
BrowseName	MTCompositionType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of BaseObjectType (See OPCUA Documentation)					
HasProperty	Variable	Uuid	String	PropertyType	Optional
HasProperty	Variable	Name	String	PropertyType	Optional
HasProperty	Variable	MTDescription	String	PropertyType	Optional
NonHierarchialReferenceType	Object	<DataItem>	DataItemType	NonHierarchialReferenceType	Optional

2.1.5 Defintion of `MTConfigurationType`

Table 5: `MTConfigurationType` Definition

Attribute	Value				
BrowseName	MTConfigurationType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of BaseObjectType (See OPCUA Documentation)					

2.1.6 Defintion of `MTDeviceType`

The MTDevice is a special type whose object will be the root of the device graph. The Device uses the component type factory and the component object factories to create each of the first level components.

The compositions, relationships, and data items are then recursively created as one decendes the MTConnect informaiton model.

Table 6: `MTDeviceType` Definition

Attribute	**Value**				
BrowseName	MTDeviceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTComponentType (see section 2.1.3)					
HasProperty	Variable	Version	String	PropertyType	Optional
HasProperty	Variable	Iso841Class	String	PropertyType	Optional

2.1.6.1 Operations

- `deriveBrowseName(element)`
  Specification: `self.name`
- `deriveNodeId(element)`
  Specification: `self.uuid`

2.1.7 Defintion of `SensorConfigurationType`

The SensorConfiguration browse name will be created as an Object relationship with the parent component.

Table 7: `SensorConfigurationType` Definition

Attribute	Value				
BrowseName	SensorConfigurationType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTConfigurationType (see section 2.1.5)					
HasProperty	Variable	FirwareVersion	String	PropertyType	Manditory
HasProperty	Variable	CalibrationDate	UtcTime	PropertyType	Optional
HasProperty	Variable	NextCalibrationDate	UtcTime	PropertyType	Optional
HasProperty	Variable	CalibrationInitials	String	PropertyType	Optional
Organizes	Object	Channels	ChannelType	FolderType	Optional

2.1.8 Defintion of `{Component}Type`

Table 8: `{Component}Type` Definition

Attribute	Value				
BrowseName	ComponentType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTComponentType (see section 2.1.3)					

2.1.9 Defintion of `{Composition}Type`

Table 9: `{Composition}Type` Definition

Attribute	**Value**				
BrowseName	CompositionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTCompositionType (see section 2.1.4)					

2.2 Data Items

Items.png Items.png

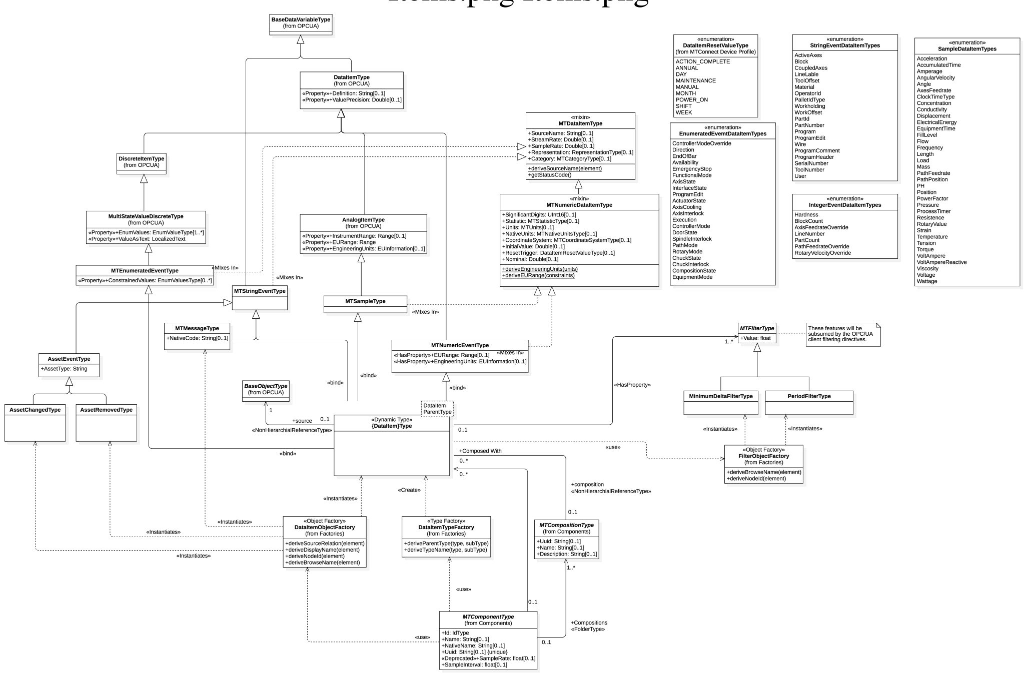

Figure 2: Data Items Diagram

2.2.1 Defintion of **`AssetChangedType`**

Table 10: `AssetChangedType` Definition

Attribute	**Value**				
BrowseName	AssetChangedType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of AssetEventType (see section 2.2.2)					

2.2.2 Defintion of **`AssetEventType`**

Table 11: `AssetEventType` Definition

Attribute	**Value**				
BrowseName	AssetEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTStringEventType (see section 2.2.11)					
HasProperty	Variable	AssetType	String	PropertyType	Manditory

2.2.3 Defintion of `AssetRemovedType`

Table 12: `AssetRemovedType` Definition

Attribute	**Value**				
BrowseName	AssetRemovedType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of AssetEventType (see section 2.2.2)					

2.2.4 Defintion of `MTDataItemType`

The data item mixin will inject the properties and the methods into the related classes. This facility is similar to the Ruby module mixin or the Scala traits.

Table 13: `MTDataItemType` Definition

Attribute	Value				
BrowseName	MTDataItemType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
HasProperty	Variable	SourceName	String	PropertyType	Optional
HasProperty	Variable	StreamRate	Double	PropertyType	Optional
HasProperty	Variable	SampleRate	Double	PropertyType	Optional
HasProperty	Variable	Representation	RepresentationType	PropertyType	Optional
HasProperty	Variable	Category	MTCategoryType	PropertyType	Manditory
HasProperty	Variable	<Dynamic>	MTFilterType	<Dynamic>	Optional
HasComponent	Object	source		BaseObjectType	Optional

2.2.4.1 Operations

- `deriveSourceName(element)`
  Specification: `self.Source.CDATA`
  Documentation: Derive the source name from the Source element CDATA. This will represent the alternative long name for the data item's source.

- `getStatusCode()`
  Documentation: The OPC/UA status code will be created using the following process:

  - If the value of the data item is `UNAVAILABLE` a status code of `Uncertain_NoCommunicationLastUsable`
  - When a reset trigger is specified, new `Good_` status codes will be created. See `ResetTrigger` enumeration.

2.2.5 Defintion of `MTEnumeratedEventType`

All Data Items with Category EVENT having a Controlled Vocabularies will be of this type. Otherwise, MTString

Table 14: `MTEnumeratedEventType` Definition

Attribute	**Value**				
BrowseName	MTEnumeratedEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MultiStateValueDiscreteType (See OPCUA Documentation)					
HasProperty	Variable	ConstrainedValues	EnumValuesType	PropertyType	Manditory

2.2.6 Defintion of `MTFilterType`

These features will be subsumed by the OPC/UA client filtering directives.

Table 15: `MTFilterType` Definition

Attribute	**Value**				
BrowseName	MTFilterType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
HasProperty	Variable	Value	float	PropertyType	Manditory

2.2.6.1 Operations

- `deriveBrowseName(element)`
  Specification: `concat(parent.BrowseName, pascalCase(element.type))`
- `deriveNodeId(element)`
  Specification: `concat(parent.NodeId, pascalCase(element.type))`

2.2.7 Defintion of `MTMessageType`

Table 16: `MTMessageType` Definition

Attribute	**Value**				
BrowseName	MTMessageType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTStringEventType (see section 2.2.11)					
HasProperty	Variable	NativeCode	String	PropertyType	Optional

2.2.8 Defintion of `MTNumericDataItemType`

These are the additional attributes that are relevent to numeric data items. The factory will evaluate these values and will set the engineering units and the range associated with the parent entity.

Table 17: `MTNumericDataItemType` Definition

Attribute	Value				
BrowseName	MTNumericDataItemType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTDataItemType (see section 2.2.4)					
HasProperty	Variable	SignificantDigits	UInt16	PropertyType	Optional
HasProperty	Variable	Statistic	MTStatisticType	PropertyType	Optional
HasProperty	Variable	Units	MTUnits	PropertyType	Optional
HasProperty	Variable	NativeUnits	MTNativeUnitsType	PropertyType	Optional
HasProperty	Variable	CoordinateSystem	MTCoordinateSystemType	PropertyType	Optional
HasProperty	Variable	InitialValue	Double	PropertyType	Optional
HasProperty	Variable	ResetTrigger	DataItemResetValueType	PropertyType	Optional
HasProperty	Variable	Nominal	Double	PropertyType	Optional

2.2.8.1 Operations

- `deriveEngineeringUnits(units)`
  Specification: `EngineeringUnits <- self.units`

- `deriveEURange(constraints)`
  Specification: `EURange.Low <- self.Constraints.Minimum EURange.High <- self.Constraints.Maximum`
  Documentation: Uses the MTConnect Constraints element if present to derive the minimum and maximum values for the numeric values. This applies to both the Numeric Event and the Sample types.

2.2.9 Defintion of `MTNumericEventType`

All data items with category EVENT and a numeric value.

Table 18: `MTNumericEventType` Definition

Attribute	**Value**				
BrowseName	MTNumericEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of DataItemType (See OPCUA Documentation)					
HasProperty	Variable	EURange	Range	PropertyType	Optional
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Optional

2.2.10 Defintion of `MTSampleType`

Data Items with category SAMPLE

Table 19: `MTSampleType` Definition

Attribute	**Value**				
BrowseName	MTSampleType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of AnalogItemType (See OPCUA Documentation)					

2.2.11 Defintion of `MTStringEventType`

All data items with category EVENT where the data is freeform text. The set_data_type constraint derives makes the data type a string for this type.

Table 20: `MTStringEventType` Definition

Attribute	**Value**				
BrowseName	MTStringEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of BaseDataVariableType (See OPCUA Documentation)					

2.2.12 Defintion of `MinimumDeltaFilterType`

Table 21: `MinimumDeltaFilterType` Definition

Attribute	**Value**				
BrowseName	MinimumDeltaFilterType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTFilterType (see section 2.2.6)					

2.2.13 Defintion of **`PeriodFilterType`**

Table 22: `PeriodFilterType` Definition

Attribute	**Value**				
BrowseName	PeriodFilterType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTFilterType (see section 2.2.6)					

2.2.14 Defintion of `{DataItem}Type`

For each DataItem the Sub Type, and the Type will be composed to be the `HasTypeDefinition` relationship of the object. The `BrowseName` will also include the Composition Type if a composition Id is provided.

Table 23: `{DataItem}Type` Definition

Attribute	**Value**				
BrowseName	DataItemType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of MTNumericEventType (see section 2.2.9)					

2.3 Conditions

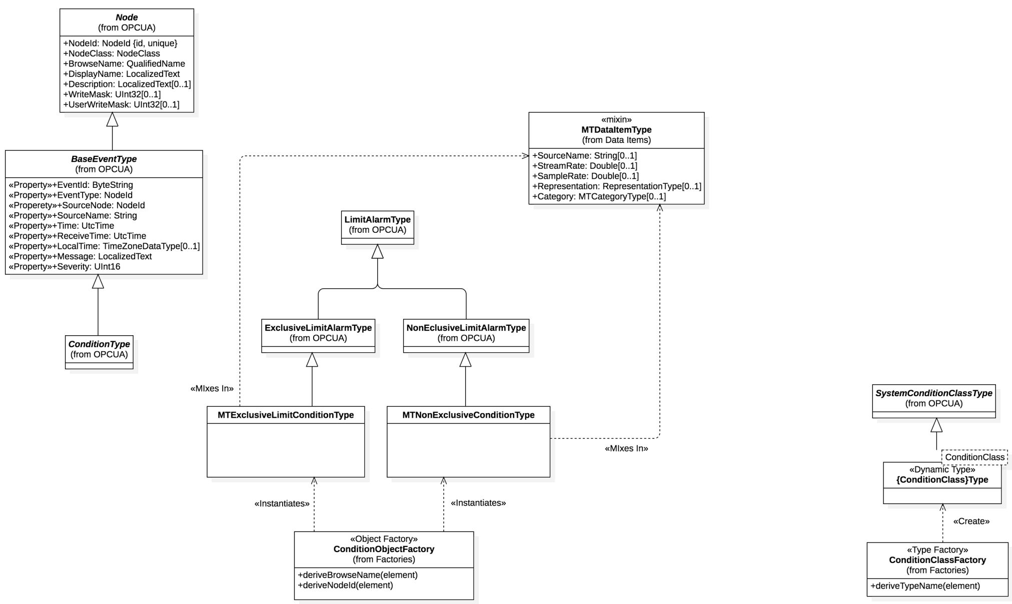

Figure 3: Conditions Diagram

2.3.1 Defintion of `MTExclusiveLimitConditionType`

Table 24: `MTExclusiveLimitConditionType` Definition

Attribute	**Value**				
BrowseName	MTExclusiveLimitConditionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of ExclusiveLimitAlarmType (See OPCUA Documentation)					

2.3.2 Defintion of **`MTNonExclusiveConditionType`**

Table 25: `MTNonExclusiveConditionType` Definition

Attribute	**Value**				
BrowseName	MTNonExclusiveConditionType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of NonEclusiveLimitAlarmType (See OPCUA Documentation)					

2.3.3 Defintion of `{ConditionClass}Type`

Table 26: `{ConditionClass}Type` Definition

Attribute	Value				
BrowseName	ConditionClassType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modeling Rule**
Subtype of SystemConditionClassType (See OPCUA Documentation)					

2.4 Factories

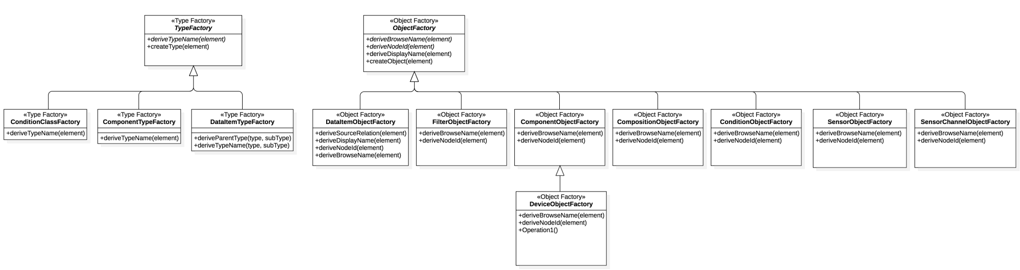

Figure 4: Factories Diagram

The factories are not part of the OPC/UA information model. They are a set of helper classes that are used to create dynamic types and objects. Since the MTConnect information model can be layered on top of the OPC/UA abstrations, the factories provide the rules for creating the browse and display names for each type.

The factories also create dynamic objects when requried for variables of various classes when they are required, such as the Data Items and the Components. Some of the relationships are more complex since they require a dynamic super-type relationship that relies on the correct placement of the MTConnect elements to be correctly represented using the OPC/UA base types.

This is especially evident when mapping the DataItems and the Conditions to the MTConnect Information Models and providing sufficent definition to allow for unambiguous implementation.

2.4.1 Defintion of «Object Factory» `ComponentObjectFactory`

2.4.1.1 Operations

- `deriveBrowseName(element)`
  Specification: `concat(element.QName, (if self.name.notEmpty() then concat(’[’, self.name, ’]’)) else ” endif))`

- `deriveNodeId(element)`
  Specification: `concat(self.findDevice().uuid, element.id)`

2.4.2 Defintion of «Type Factory» `ComponentTypeFactory`

The ‘ComponentTypeFactory‘ creates component types using the MTConnect XML element as an input. The factory takes the ‘QName‘ (or qualified name) of the XML element and then appends ‘Type‘. For example an ‘<Controller id=’...’></...>‘ element will create an OPC/UA ‘ControllerType‘ type definition as an extension of the base ‘MTControllerType‘.

Currently there is no additional abstractions or super types required by the companion specification. The types will be a single level where each Component is a sub-type of the base ‘MTComponentType‘.

2.4.2.1 Operations

- `deriveTypeName(element)`
  Specification: `derive: Component <- element.QName`
  Documentation: The QName of the element for the component will be used to derive the type of the node.

2.4.3 Defintion of «Object Factory» `CompositionObjectFactory`

2.4.3.1 Operations

- `deriveBrowseName(element)`
  Specification: `concat(pascalCase(element.type), (if self.name.notEmpty() then concat(’[’, self.name, ’]’)) else ” endif))`
- `deriveNodeId(element)`
  Specification: `concat(self.findDevice().uuid, element.id)`

2.4.4 Defintion of «Type Factory» `CompositionTypeFactory`

2.4.4.1 Operations

- `deriveTypeName(element)`
  Specification: `derive: Composition <- pascalCase(element.type)`
  Documentation: The type for the composition will be created using the pascal case of the ‘type‘ from the composition element.

2.4.5 Defintion of «Type Factory» `ConditionClassFactory`

2.4.5.1 Operations

- `deriveTypeName(element)`
  Documentation: Create condition classes based on the OPC/UA three condition types.

2.4.6 Defintion of «Object Factory» `ConditionObjectFactory`

2.4.6.1 Operations

- `deriveBrowseName(element)`
- `deriveNodeId(element)`

2.4.7 Defintion of «Object Factory» `DataItemObjectFactory`

2.4.7.1 Operations

- `deriveSourceRelation(element)`
  Documentation: Use the source composition, component id, or data item id to locate the source node id for this relationship. If one exists, add an object with browse name "source" that relates to the entity referenced by the id.

  The most specific identity should be used in the following order:

  - `DataItemId`
  - `CompositionId`
  - `ComponentId`

  Since the data item implies composition and component and the composition implies component, there should only be one attribute given for the source.

- `deriveDisplayName(element)`
  Documentation: Same as the BrowseName.

- `deriveNodeId(element)`
  Documentation: The nodeId will be given by the device uuid and the DataItem id attribute.

- `deriveBrowseName(element)`
  Documentation: The browse name will be composed of the following parts of the model:

  1. If the compositionId is present, the compositionId will be resolved the the Composition element and the pascal case of the type attribute will be placed first.
  2. If the subType is present, the pascal case of the subType will be placed next.
  3. The pascal case of the type will be placed last.

  For example, for a data item with the following attributes:

  - type: `TEMPERATURE`
  - composition type: `STORAGE_BATTERY`

  will have the following browse name: `StorageBatteryTemperature`

  For the data item with the following attributes:

  - type: `ANGLE`
  - subType: `ACTUAL`
  - composition type: `ENCODER`

  will have the following browse name: `EncoderActualAngle`

2.4.8 Defintion of «Type Factory» **`DataItemTypeFactory`**

Based on the data item category, type, and subType, this class creates a new OPC/UA type and also provides the template parameter for the ParentType from which this type is derived.

2.4.8.1 Operations

- `deriveParentType(type, subType)`
  Documentation: The parent type is derived from the category as follows:

  - `SAMPLE` -> `SampleType`
  - `EVENT` ->
    * Enumerated Value -> `MTEnumeratedEventType`
    * Integer Value -> `MTNumericEventType`
    * Otherwise -> `MTStringEventType`

- `deriveTypeName(type, subType)`
  Specification: `concat(pascalCase(subType), pascalCase(type))`
  Documentation: Used to derive the class name for creating a pascal case name from the sub type and the type. For example type `ROTARY_VELOCITY` and subType `ACTUAL` will become `ActualRotaryVelocity`.

2.4.9 Defintion of «Object Factory» **`DeviceObjectFactory`**

The model instantiation for MTConnect begins with the ‘Device‘ MTConnect element and then recursively traverses the sub-elements. The device will the capabilities in the component factory to generate all the data items and component types.

2.4.9.1 Operations

- `deriveBrowseName(element)`
  Specification: `derive: element.name`

- `deriveNodeId(element)`
  Specification: `derive: element.uuid`

2.4.10 Defintion of «Object Factory» `FilterObjectFactory`

Creates filters based on the type attribute of the Filter element.

2.4.10.1 Operations

- `deriveBrowseName(element)`
- `deriveNodeId(element)`
  Documentation: The node id is composed of the data item id and the browse name.

2.4.11 Defintion of «Object Factory» `ObjectFactory`

2.4.11.1 Operations

- `deriveBrowseName(element)`
- `deriveNodeId(element)`
- `deriveDisplayName(element)`
  Specification: `deriveBrowseName(element)`
- `createObject(element)`

2.4.12 Defintion of «Object Factory» `SensorChannelObjectFactory`

2.4.12.1 Operations

- `deriveBrowseName(element)`
  Specification: `concat('Channel', self.number)`
- `deriveNodeId(element)`
  Specification: `concat(self.parent.NodeId, BrowseName)`

2.4.13 Defintion of «Object Factory» `SensorObjectFactory`

2.4.13.1 Operations

- `deriveBrowseName(element)`
  Specification: `element.QName`
- `deriveNodeId(element)`
  Specification: `concat(self.parent.NodeId, BrowseName)`

2.4.14 Defintion of «Type Factory» `TypeFactory`

2.4.14.1 Operations

- `deriveTypeName(element)`
- `createType(element)`

2.5 MTConnect Device Profile

Device Profile.png Device Profile.png

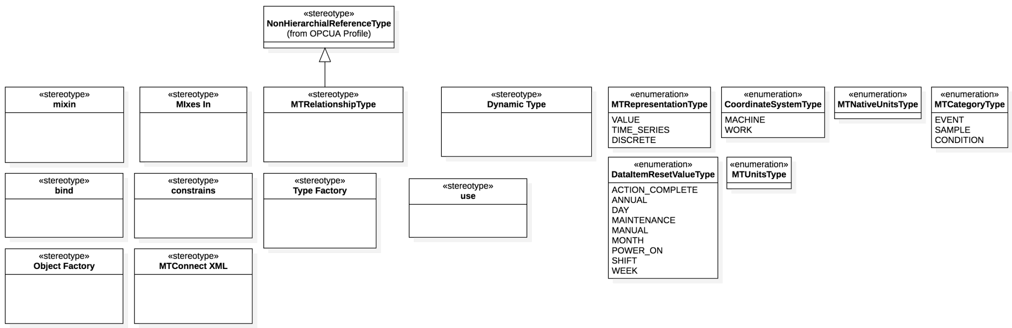

Figure 5: MTConnect Device Profile Diagram

The device profile documents the common data types and stereotypes that are used to construct the model. A stereotype is a design or modeling pattern that provides additional information about the type or the relationship between types.

It can also identify the behavior of a property or the role the type or relation will play in the model.

Stereotypes are used throughout the model to provide additional information that will halp provide context and definition to aid in better understanding the data model.

2.5.1 Defintion of `Dynamic Type`

2.5.2 Defintion of `MTConnect XML`

2.5.3 Defintion of `MTRelationshipType`

2.5.4 Defintion of `Mixes In`

This stereotype is associated with the dependency between a type and a mixin. See Section 2.5.9 for a complete description of the mixin.

2.5.5 Defintion of `Object Factory`

2.5.6 Defintion of `Type Factory`

2.5.7 Defintion of `bind`

When a dynamic type (See Section 2.5.1) creates an instance where the super-type can be associated based on the data item category and type, the `Type Factory` will specify which supertype is to be referenced.

The `bind` stereotype indicates the relationship between the dynamic sub-type and the parent type are resolved baed on the MTConnect DataItem meta data.

2.5.8 Defintion of `constrains`

2.5.9 Defintion of `mixin`

The mixin pattern injects the properties and operations into the types that are related to the using the `Mixes In` dependency. Mixins allow for lightweight multiple inheritance. Since OPC/UA does not allow for multiple inheritance and the MTConnect types require the same set of properties when they are sub-typed from existing OPC/UA types, this mechanism allows for this relationship to be expressed.

2.5.10 Defintion of `use`

The use stereotype indicates that one class uses as a helper to perform a specific operation or activity. This stereotype is mainly used to indicate that a specific factory is being employed by another type to create dynamic properties or relationships.