MTConnect® and OPC/UA Companion Specification

Draft Version 2.0

Prepared for: MTConnect Institute and OPC Foundation
Prepared by: List of authors. . .
Prepared on: September 29, 2018

September 29, 2018

OPC Foundation and MTConnect® Institute

AGREEMENT OF USE

Foundation, 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ, 85260-1830

COMPLIANCE

The combination of the MTConnect Institute and OPC Foundation shall at all times be the sole entities that may authorize developers, suppliers and sellers of hardware and software to use certification marks, trademarks or other special designations to indicate compliance with these materials as specified within this document. Products developed using this specification may claim compliance or conformance with this specification if and only if the software satisfactorily meets the certification requirements set by the MTConnect Institute or the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

TRADEMARKS

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of Germany. This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

Table of Contents

List of Figures

1 Introduction

1.1 Background

In September 2010, the OPC Foundation and the MTConnect Institute signed a memorandum of understanding to provide a mechanism for OPC and MTConnect to collaborate to extend the reach of the existing manufacturing data exchange standards and implementation technologies in order to:

- Evolve the existing standards for each organization to provide complete manufacturing technology interoperability.
- Provide the mechanism for continuous improvement of standards and specifications overseen by each body.
- Work directly with the end users and suppliers of technology and manufacturing.
- Provide a coordinating function to exchange insights, identify overlaps, and harmonize work where appropriate.
- Facilitate clear communication and education for users and others concerning possible overlaps and the ways the standards and specifications can be used.
- Provide a solid foundation to develop and deliver specifications, technology and processes to facilitate adoption of the technology into real products.

The outcome of that agreement was an initial companion specification called MTConnect-OPC UA Version 1.2.0. MTConnect-OPC UA companion specification describes an architecture for exchanging information for interoperability and consistency between MTConnect specifications and the OPC Unified Architecture (UA) specifications, as well as describing the manufacturing technology equipment, devices, software or other products that may implement those standards.

This document, OPC Unified Architecture for MTConnect Companion Specification Draft Version 2.0, provides an update to the original companion specification to include the latest capabilities and functionality of the standards provided by the MTConnect Institute and the OPC Foundation.

1.2 MTConnect-OPC UA Goals

The OPC Unified Architecture for MTConnect Companion Specification is designed with the following goals in mind, in the interest of wide and rapid adoption by vendors of equipment and software:

- Incremental adoption:the technical barrier to MTConnect-OPC UA enablement will be greatly reduced with this companion specification and the source code and binaries available in the MTConnect-OPC UA reference port.
- Evolution: MTConnect and OPC UA can incrementally evolve without jeopardizing backwards compatibility of previous MTConnect-OPC UA versions.
- Customizability: MTConnect-OPC UA's extensibility enables integrators to create value-added software and tools that are machine-specific or installation-specific, without jeopardizing compatibility with other equipment or software.
- Non-proprietary: built on open standards, backed by both the OPC Foundation and the MTConnect Institute which represents hundreds of companies, individuals, government organizations and non-profits all working toward the goal of increased productivity in the manufacturing arena.

1.3 Who Will Find Benefit from this Specification?

To adopt the OPC Unified Architecture for MTConnect Companion Specification one will need to have a clear understanding of both MTConnect and OPC UA. From the technical side, we will discuss MTConnect-OPC UA from:

- The backend or OPC UA Server and MTConnect agent/adapter architecture.

- The client or software application side, we will discuss how one develops an application that is MTConnect-OPC UA enabled.
- Applying MTConnect semantics to devices containing an embedded OPC UA Server.

From the business side, we will reference a companion business MTConnect-OPC UA white paper that addresses the concerns from the owners and top management of the business as well as the operations and engineering management. It is the objective of this white paper to provide information primarily to MTConnect and OPC UA software developers. We do not make assumptions about the level of programming expertise beyond what would be considered to be "reasonable" level of expertise. It is for this reason that we include enough details about both MTConnect and OPC UA to provide the ability to implement this companion specification without having references back to other documents. However, the OPC and MTConnect standards are critical and become much more meaningful with the appropriate overview from this document.

1.4 References

1.4.1 OPC Foundation

The following specifications from the OPC foundation are referenced by this specification.

[UA Part 1] OPC UA Specification: Part 1 – Concepts
http://www.opcfoundation.org/UA/Part1/

[UA Part 2] OPC UA Specification: Part 2 – Security Model
http://www.opcfoundation.org/UA/Part2/

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model
http://www.opcfoundation.org/UA/Part3/

[UA Part 4] OPC UA Specification: Part 4 – Services
http://www.opcfoundation.org/UA/Part4/

[UA Part 5] OPC UA Specification: Part 5 – Information Model
http://www.opcfoundation.org/UA/Part5/

[UA Part 6] OPC UA Specification: Part 6 – Mappings
http://www.opcfoundation.org/UA/Part6/

[UA Part 7] OPC UA Specification: Part 7 – Profiles
http://www.opcfoundation.org/UA/Part7/

[UA Part 8] OPC UA Specification: Part 8 – Data Access
http://www.opcfoundation.org/UA/Part8/

[UA Part 9] OPC UA Specification: Part 9 – Alarms and Conditions
http://www.opcfoundation.org/UA/Part9/

[UA Part 10] OPC UA Specification: Part 10 – Programs
http://www.opcfoundation.org/UA/Part10/

[UA Part 11] OPC UA Specification: Part 11 – Historical Access
http://www.opcfoundation.org/UA/Part11/

[UA Part 13] OPC UA Specification: Part 13 – Aggregates
http://www.opcfoundation.org/UA/Part13/

1.4.2 MTConnect® Institute

[MT Part 1.0] MTConnect® Standard: Part 1.0 – Overview and Fundamentals, Version 1.4.0
https://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/5acb81f96d2a73d3a01281c5/1523286521969/MTC_Part1_0_OverviewAndFundamentals1_4_0.pdf

[MT Part 2.0] MTConnect® Standard: Part 2.0 – Devices Information Model, Version 1.4.0
https://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/5acb822e352f53a44f10534b/1523286575786/MTC_Part2_0_Devices_1_4_0.pdf

[MT Part 3.0] MTConnect® Standard: Part .0 – Streams Information Model, Version 1.4.0
https://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/5acb651e88251b53486ed7f5/1523279135088/MTC_Part3_0_StreamsInformationModel_1_4_0.pdf

[MT Part 4.0 MTConnect® Standard: Part 4 – Assets Information Model, Version 1.4.0
https://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/5acb824a8a922dc773e19caf/1523286602677/MTC_Part4_0_AssetsInformationModel_1_4_0.pdf

[MT Part 4.1] MTConnect® Standard: Part 4.1 – Cutting Tools, Version 1.4.0
https://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/5acb825a03ce649b2a64b282/1523286619088/MTC_Part4_1_CuttingTools_1_4_0.pdf

[MT Part 5.0] MTConnect® Standard: Part 5.0 – Interfaces, Version 1.4.0
https://static1.squarespace.com/static/54011775e4b0bc1fe0fb8494/t/5acb826a352f53a44f10606f/1523286635455/MTC_Part5_0_Interfaces_1_4_0.pdf

1.5 Abbrevations

The following abbreviation are used in this document:

- ERP – Enterprise Resource Planning
- HMI – Human Machine Interface
- HTTP – Hyper Text Transport Protocol
- MES – Management Execution Systems
- PLC – Programmable Logic Controller
- PMS – Production Management Systems

- SCADA – Supervisory Control And Data Acquisition
- TCP/IP – Transmission Control Protocol/Internet Protocol
- XML – eXtensible Mark-up Language

2 Use Cases

2.1 Overview

Before delving into the details of the specification it is useful to identify some of the key use cases for the technology. The use cases defined here are not an exhaustive list; however, they should help demonstrate how this specification is expected to be used and to help illustrate the benefits of a common information model.

2.2 Device Maker

The use case, shown in **??**_use_casefigure:mfg_use_caseters on the manufacturer of a piece of equipment or device that needs to provide connectivity to other systems. In some cases, the device manufacturer will be targeting markets other than equipment (Machine Tool) and would benefit from a more generic specification like OPC UA. On the other hand, the standardized semantics of MTConnect are extremely important to standardized communications on the manufacturing shop floor. The MTConnect-OPC UA specification and the resulting standard information model allows the device manufacturers to standardize on OPC UA as the network interface while making their information accessible to software applications that includes the enhanced meaning and structure provided by applying the MTConnect semantics. Figure 1 shows several clients developed for different purposes that can access information produced by the device via OPC UA.

3 Types

3.1 Components

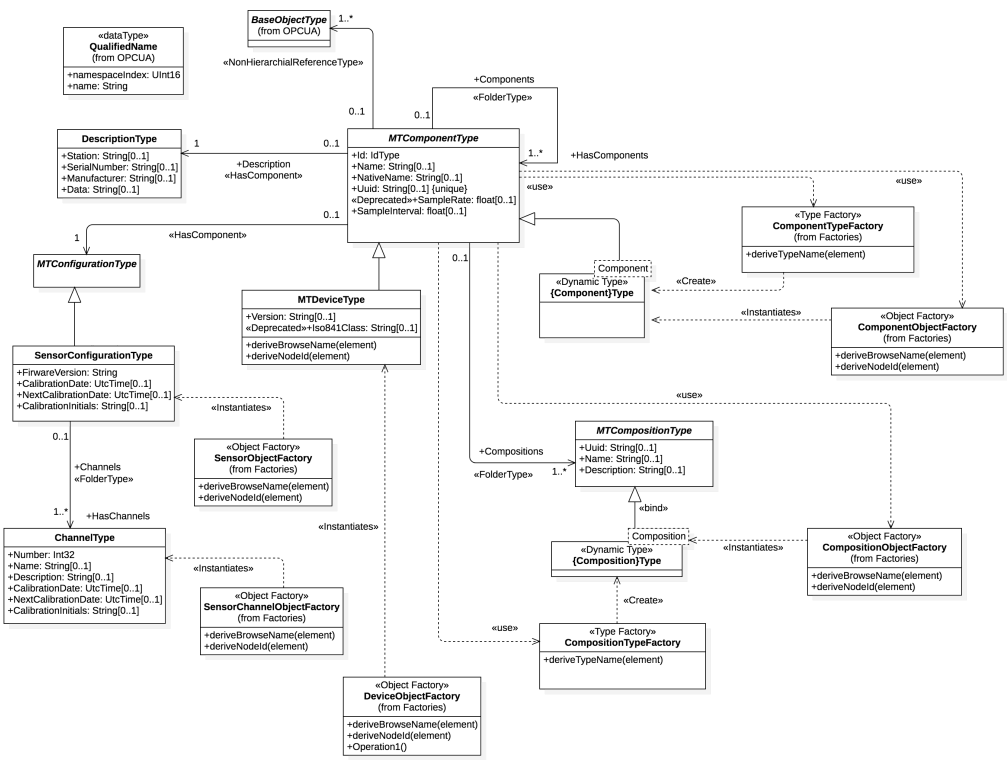

Figure 1: Components Diagram

The Components documents the Component models and the owned objects.

3.1.1 Defintion of `ChannelType`

An MTConnect `Channel` is a single data stream associated with a sensor. Each stream of data can be calibrated separately and allows for the specification of the meta information and descriptive information. The only required property of the `Channel` is the number which is the unique identifier.

The channels will be created by the `SensorChannelObjectFactory` that composes the `BrowseName` and the `NodeId` for each object. (See 3.4.12).

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	Mandatory
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

3.1.2 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

3.1.2.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  "Description"
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  concat(self.parent.NodeId, BrowseName)
  ```

3.1.3 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 subtypes 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	Mandatory
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

3.1.3.1 Constraints

- Constraint `node_id`:

```
context Component::NodeId : String
derive: concat(self.getDevice().uuid, self.getAttributes().id)
```

  Documentation: The NodeId SHALL be derrived from the combination of the device UUID and the id of the component.

3.1.4 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.

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

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`.

3.1.5 Defintion of `MTConfigurationType`

The abstract `MTConfigurationType` currently has only one sub-type, `SensorConfigurationType` (see 3.1.7). In the future the configurations will also contain component and device configuration information as sub-types.

Table 5: `MTConfigurationType` Definition

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

3.1.6 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 3.1.3)					
HasProperty	Variable	Version	String	PropertyType	Optional
HasProperty	Variable	Iso841Class	String	PropertyType	Optional

3.1.6.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  self.name
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  self.uuid
  ```

3.1.6.2 Constraints

- Constraint `uuid_not_empty`:

  ```
  uuid->notEmpty()
  ```

  Documentation: The UUID SHALL be provided.

- Constraint `name_not_empty`:

  ```
  name->notEmpty()
  ```

  Documentation: The name of the Device SHALL be given.

3.1.7 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 3.1.5)					
HasProperty	Variable	FirwareVersion	String	PropertyType	Mandatory
HasProperty	Variable	CalibrationDate	UtcTime	PropertyType	Optional
HasProperty	Variable	NextCalibrationDate	UtcTime	PropertyType	Optional
HasProperty	Variable	CalibrationInitials	String	PropertyType	Optional
Organizes	Object	Channels	ChannelType	FolderType	Optional

3.1.8 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 3.1.3)					

3.1.9 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 3.1.4)					

3.2 Data Items

Items.png Items.png

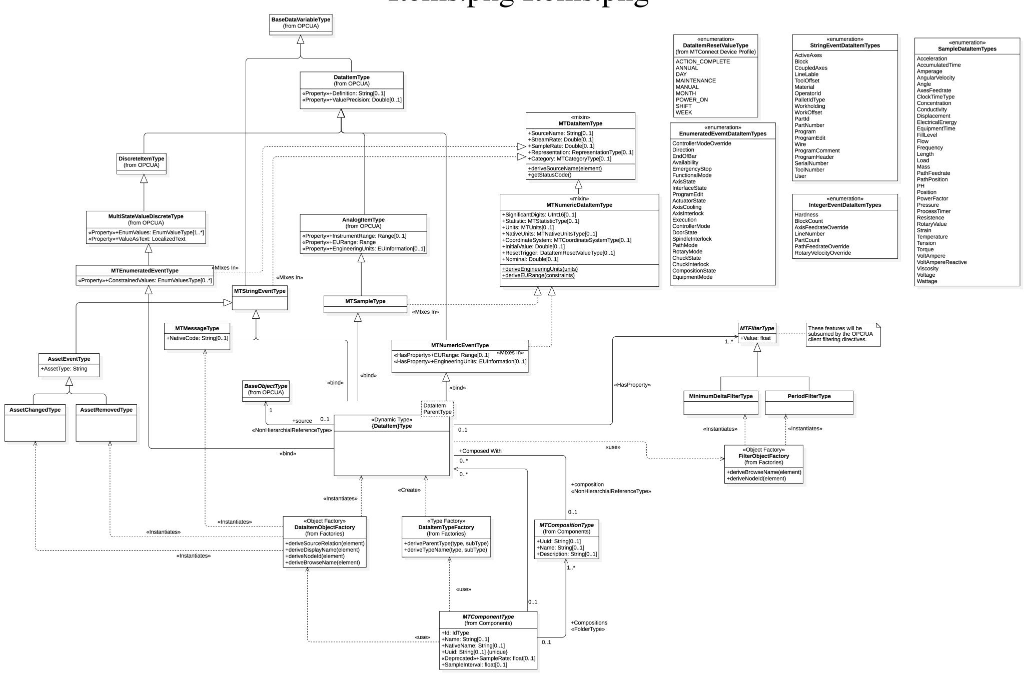

Figure 2: Data Items Diagram

3.2.1 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 3.2.2)					

3.2.2 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 3.2.11)					
HasProperty	Variable	AssetType	String	PropertyType	Mandatory

3.2.3 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 3.2.2)					

3.2.4 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	Mandatory
HasProperty	Variable	<Dynamic>	MTFilterType	<Dynamic>	Optional
HasComponent	Object	source		BaseObjectType	Optional

3.2.4.1 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.

3.2.5 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	Mandatory

3.2.6 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	Mandatory

3.2.6.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  concat(parent.BrowseName, pascalCase(element.type))
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  concat(parent.NodeId, pascalCase(element.type))
  ```

3.2.7 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 3.2.11)					
HasProperty	Variable	NativeCode	String	PropertyType	Optional

3.2.8 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 3.2.4)					
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

3.2.8.1 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.

3.2.9 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

3.2.10 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)					

3.2.11 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)					

3.2.11.1 Constraints

- Constraint `set_data_type`:

```
derive: DataType <-String
```

3.2.12 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 3.2.6)					

3.2.13 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 3.2.6)					

3.2.14 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 3.2.9)					

3.3 Conditions

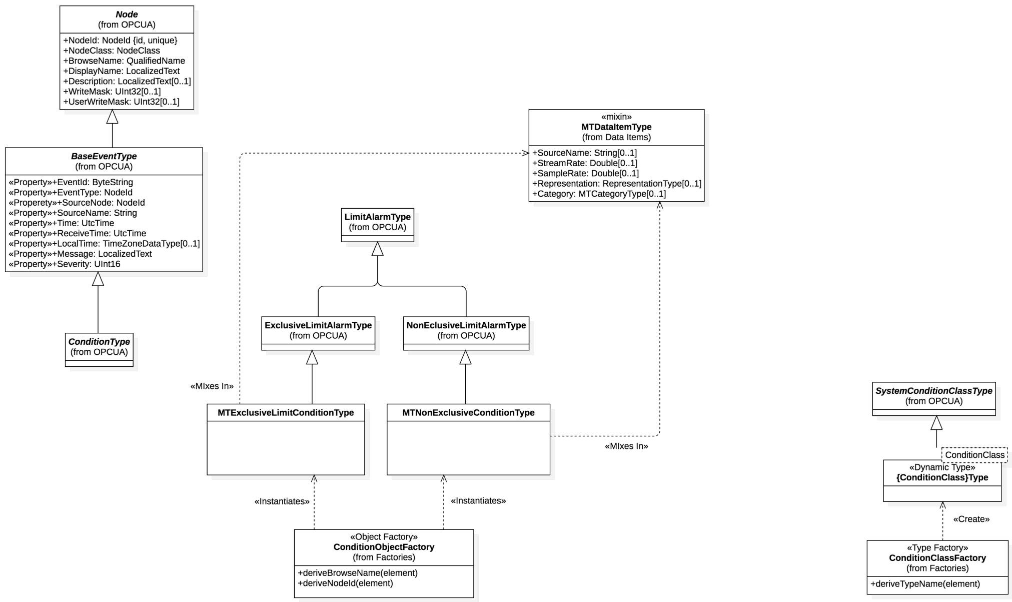

Figure 3: Conditions Diagram

3.3.1 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)					

3.3.2 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)					

3.3.3 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)					

3.4 Factories

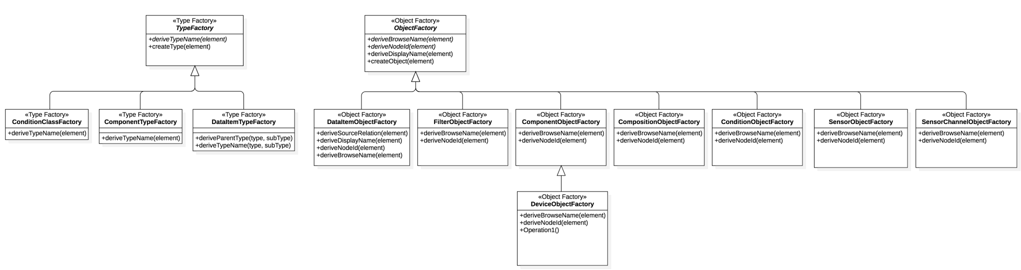

Figure 4: Factories Diagram

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.

3.4.1 Defintion of «Object Factory» `ComponentObjectFactory`

3.4.1.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  concat(element.QName, (if self.name.notEmpty() then concat('[', self.name
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  concat(self.findDevice().uuid, element.id)
  ```

3.4.2 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‘.

3.4.2.1 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.

3.4.3 Defintion of «Object Factory» `CompositionObjectFactory`

3.4.3.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  concat(pascalCase(element.type), (if self.name.notEmpty() then concat('['
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  concat(self.findDevice().uuid, element.id)
  ```

3.4.4 Defintion of «Type Factory» `CompositionTypeFactory`

3.4.4.1 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.

3.4.5 Defintion of «Type Factory» `ConditionClassFactory`

3.4.5.1 Operations

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

3.4.6 Defintion of «Object Factory» `ConditionObjectFactory`

3.4.6.1 Operations

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

3.4.7 Defintion of «Object Factory» `DataItemObjectFactory`

3.4.7.1 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`

3.4.8 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.

3.4.8.1 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`.

3.4.9 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.

3.4.9.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  derive: element.name
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  derive: element.uuid
  ```

3.4.10 Defintion of «Object Factory» `FilterObjectFactory`

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

3.4.10.1 Operations

- `deriveBrowseName(element)`

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

3.4.11 Defintion of «Object Factory» `ObjectFactory`

3.4.11.1 Operations

- `deriveBrowseName(element)`

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

  ```
  deriveBrowseName(element)
  ```

- `createObject(element)`

3.4.12 Defintion of «Object Factory» `SensorChannelObjectFactory`

3.4.12.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  concat('Channel', self.number)
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  concat(self.parent.NodeId, BrowseName)
  ```

3.4.13 Defintion of «Object Factory» `SensorObjectFactory`

3.4.13.1 Operations

- `deriveBrowseName(element)`
  Specification:

  ```
  element.QName
  ```

- `deriveNodeId(element)`
  Specification:

  ```
  concat(self.parent.NodeId, BrowseName)
  ```

Device Profile.png Device Profile.png

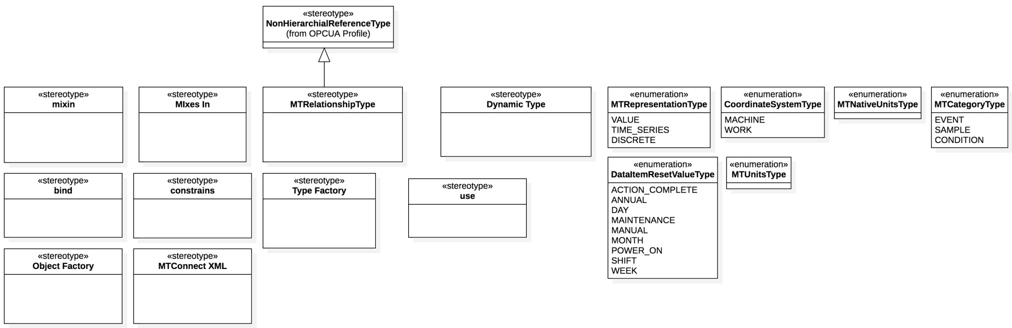

Figure 5: MTConnect Device Profile Diagram

3.4.14 Defintion of «Type Factory» `TypeFactory`

3.4.14.1 Operations

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

3.5 MTConnect Device Profile

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.

3.5.1 Defintion of `Dynamic Type`

3.5.2 Defintion of `MTConnect XML`

3.5.3 Defintion of `MTRelationshipType`

3.5.4 Defintion of `Mixes In`

This stereotype is associated with the dependency between a type and a mixin. See Section 3.5.9 for a complete description of the mixin.

3.5.5 Defintion of `Object Factory`

3.5.6 Defintion of `Type Factory`

3.5.7 Defintion of `bind`

When a dynamic type (See Section 3.5.1) 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.

3.5.8 Defintion of `constrains`

3.5.9 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.

3.5.10 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.