MTConnect® Standard

Version 1.4.0

ANSI/MTC1.4-2018 (12/7/2018)

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

CONTENTS

MTConnect® Standard
Part 1.0 - Overview and Fundamentals

Version 1.4.0

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

MTConnect® Specification and Materials

Table of Contents

Table of Figures

1 Overview of MTConnect®

MTConnect® is a data and information exchange standard that is based on a *data dictionary* of terms describing information associated with manufacturing operations. The standard also defines a series of *semantic data models* that provide a clear and unambiguous representation of how that information relates to a manufacturing operation. The MTConnect Standard has been designed to enhance the data acquisition capabilities from equipment in manufacturing facilities, to expand the use of data driven decision making in manufacturing operations, and to enable software applications and manufacturing equipment to move toward a plug-and-play environment to reduce the cost of integration of manufacturing software systems.

The MTConnect standard supports two primary communications methods – *Request/Response* and *Publish/Subscribe* type of communications. The *Request/Response* communications structure is used throughout this document to describe the functionality provided by MTConnect. See *Section 8.3.6 – Data Streaming* for details describing the functionality of the *Publish/Subscribe* communications structure available from an *MTConnect Agent*.

Although the MTConnect Standard has been defined to specifically meet the requirements of the manufacturing industry, it can also be readily applied to other application areas as well.

The MTConnect Standard is an open, royalty free standard – meaning that it is available for anyone to download, implement, and utilize in software systems at no cost to the implementer.

The *semantic data models* defined in the MTConnect Standard provide the information required to fully characterize data with both a clear and unambiguous meaning and a mechanism to directly relate that data to the manufacturing operation where the data originated. Without a semantic data model, client software applications must apply an additional layer of logic to raw data to convey this same level of meaning and relationship to manufacturing operations. The approach provided in the MTConnect Standard for modeling and organizing data allows software applications to easily interpret data from a wide variety of data sources which reduces the complexity and effort to develop applications.

The data and information from a broad range of manufacturing equipment and systems are addressed by the MTConnect Standard. Where the *data dictionary* and *semantic data models* are insufficient to define some information within an implementation, an implementer may extend the *data dictionary and semantic data models* to address their specific requirements. See *Section 6.7* for guidelines related to extensibility of the MTConnect Standard.

To assist in implementation, the MTConnect Standard is built upon the most prevalent standards in the manufacturing and software industries. This maximizes the number of software tools available for implementation and provides the highest level of interoperability with other standards, software applications, and equipment used throughout manufacturing operations.

Current MTConnect implementations are based on HTTP as a transport protocol and XML as a language for encoding each of the *semantic data models* into electronic documents. All software examples provided in the various MTConnect Standard documents are based on these two core technologies.

The base functionality defined in the MTConnect Standard is the *data dictionary* describing manufacturing information and the *semantic data models*. The transport protocol and the programming language used to represent or transfer the information provided by the *semantic data models* are not restricted in the standard to HTTP and XML. Therefore, other protocols and programming languages may be used to represent the semantic models and/or transport the information provided by these data models between an *MTConnect Agent* (server) and a client software application as may be required by a specific implementation.

Note: The term "document" is used with different meanings in the MTConnect Standard:

- Meaning 1: The MTConnect Standard itself is comprised of multiple documents each addressing different aspects of the Standard. Each document is referred to as a *Part* of the Standard.
- Meaning 2: In an MTConnect implementation, the electronic documents that are published from a data source and stored by an *MTConnect Agent*.
- Meaning 3: In an MTConnect implementation, the electronic documents generated by an *MTConnect Agent* for transmission to a client software application.

The following will be used throughout the MTConnect Standard to distinguish between these different meanings for the term "document":

- MTConnect Document(s) or Document(s) shall be used to refer to printed or electronic document(s) that represent a *Part(s)* of the MTConnect Standard.
- All reference to electronic documents that are received from a data source and stored in an *MTConnect Agent* shall be referred to as "*Document(s)*" and are typically provided with a prefix identifier; e.g. *Asset Document*.
- All references to electronic documents generated by an *MTConnect Agent* and sent to a client software application shall be referred to as a "*Response Document*".

When used with no additional descriptor, the form "document" shall be used to refer to any printed or electronic document.

Manufacturing software systems implemented utilizing MTConnect can be represented by a very simple structure:

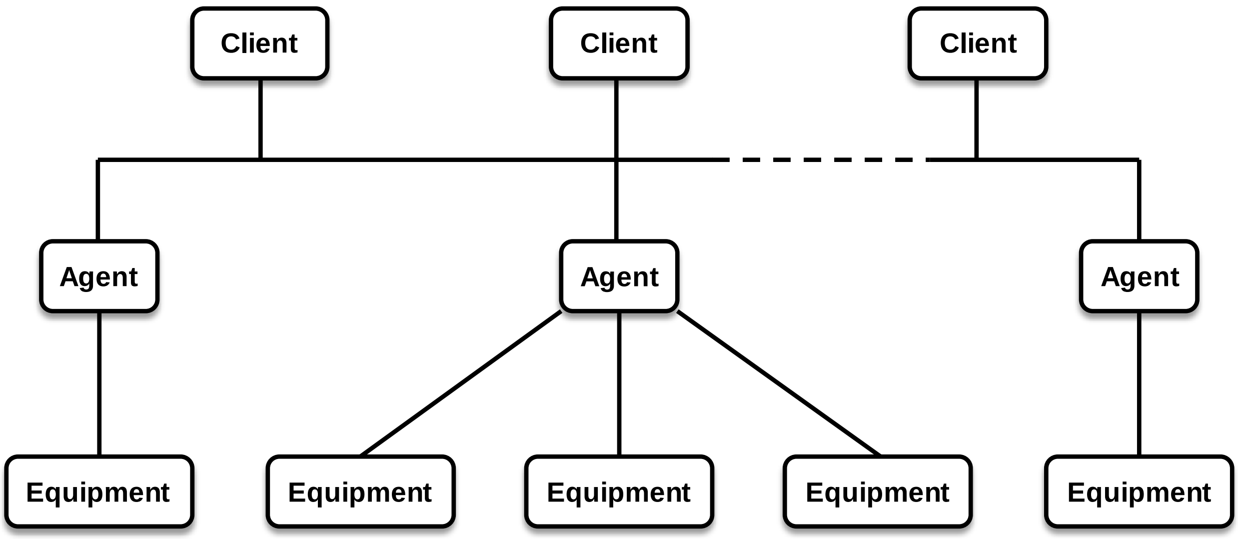


Figure 1: Basic MTConnect Implementation Structure

The three basic modules that comprise a software system implemented using MTConnect are:

Equipment: Any data source. In the MTConnect Standard, equipment is defined as any tangible property that is used to equip the operations of a manufacturing facility. Examples of equipment are machine tools, ovens, sensor units, workstations, software applications, and bar feeders.

MTConnect Agent: Software that collects data published from one or more piece(s) of equipment, organizes that data in a structured manner, and responds to requests for data from client software systems by providing a structured response in the form of a *Response Document* that is constructed using the *semantic data models* defined in the Standard.

Note: The *MTConnect Agent* may be fully integrated into the piece of equipment or the *Agent* may be independent of the piece of equipment. Implementation of an *Agent* is the responsibility of the supplier of the piece of equipment and/or the implementer of the *MTConnect Agent*.

Client Software Application: Software that requests data from *MTConnect Agents* and processes that data in support of manufacturing operations.

Based on *Figure 1* above, it is important to understand that the MTConnect Standard only addresses the following functionality and behavior of an *MTConnect Agent*:

- the method used by a client software application to request information from an *MTConnect Agent*.
- the response that a *MTConnect Agent* provides to a client software application.
- a *data dictionary* used to provide consistency in understanding the meaning of data reported by a data source.
- the description of the *semantic data models* used to structure *Response Documents* provided by a *MTConnect Agent* to a client software application.

These functions are the primary building blocks that define the *Base Functional Structure* of the MTConnect Standard.

There are a wide variety of data sources (equipment) and data consumption systems (client software systems) used in manufacturing operations. There are also many different uses for the data associated with a manufacturing operation. No single approach to implementing a data communication system can address all data exchange and data management functions typically required in the data driven manufacturing environment. MTConnect has been uniquely designed to address this diversity of data types and data usages by providing different *semantic data models* for different data application requirements:

Data Collection: The most common use of data in manufacturing is the collection of data associated with the production of products and the operation of equipment that produces those products. The MTConnect Standard provides comprehensive *semantic data models* that represent data collected from manufacturing operations. These *semantic data models* are detailed in *Part 2.0 – Devices Information Model* and *Part 3.0 – Streams Information Model* of the MTConnect Standard.

Inter-operations Between Pieces of Equipment: The MTConnect Standard provides an *Interaction Model* that structures the information required to allow multiple pieces of equipment to coordinate actions required to implement manufacturing activities. This *Interaction Model* is an implementation of a *Request/Response Messaging Structure*. This *Interaction Model* is called `Interfaces` which is detailed in *Part 5.0 - Interfaces* of the MTConnect Standard.

Shared Data: Certain information used in a manufacturing operation is commonly shared amongst multiple pieces of equipment and/or software applications. This information is not typically "owned" by any one manufacturing resource. The MTConnect Standard represents this information through a series of *semantic data models* – each describing different types of information used in the manufacturing environment. Each type of information is called an *MTConnect Asset*. *MTConnect Assets* are detailed in *Part 4.0 – Assets Information Model*, and its sub-*Parts*, of the MTConnect Standard.

2 Purpose of This Document

This document, *Part 1.0 – Overview and Functionality* of the MTConnect® Standard, addresses two major topics relating to the MTConnect Standard. The first sections of the document define the organization of the documents used to describe the MTConnect Standard; including the terms and terminology used throughout the Standard. The balance of the document defines the following:

- Operational concepts describing how an *MTConnect Agent* should organize and structure data that has been collected from a data source.
- Definition and structure of the *Response Documents* supplied by an *MTConnect Agent*.
- The protocol used by a client software application to communicate with an *MTConnect Agent*.

3 Terminology

The definitions for terms and terminology as used to describe the features and functions within the MTConnect Standard are provided below.

Term	Definition as Used in the MTConnect Standard
Abstract Element	An element that defines a set of common characteristics that are shared by a group of elements. An abstract element cannot appear in a document. In a specific implementation of a schema, an abstract element is replaced by a derived element that is itself not an abstract element. The characteristics for the derived element are inherited from the abstract element. Appears in the documents in the following form: abstract.
Adapter	An optional piece of hardware or software that transforms information provided by a piece of equipment into a form that can be received by an *MTConnect Agent*. Appears in the documents in the following form: adapter.
Agent	Refers to an *MTConnect Agent*. Software that collects data published from one or more piece(s) of equipment, organizes that data in a structured manner, and responds to requests for data from client software systems by providing a structured response in the form of a *Response Document* that is constructed using the *semantic data models* defined in the Standard. Appears in the documents in the following form: *MTConnect Agent* or *Agent*.
Application Programming Interface (API)	A set of methods to provide communications between software applications. The API defined in the MTConnect Standard describes the methods for providing the *Request/Response Information Exchange* between an *MTConnect Agent* and client software applications. Appears in the documents in the following forms: Application Programming Interface or API.

Term	Definition as Used in the MTConnect Standard
Archetype	**General Description of an *MTConnect Asset*:** Archetype is a class of *MTConnect Assets* that provides the requirements, constraints, and common properties for a type of *MTConnect Asset*. Appears in the documents in the following form: Archetype. **Used as an XML term describing an *MTConnect Asset*:** In an XML representation of the *Assets Information Model*, `Archetype` is an abstract element that is replaced by a specific type of *Asset* Archetype. Appears in the documents in the following form: `Archetype`.
Asset	**General meaning:** Typically referred to as an *MTConnect Asset*. An *MTConnect Asset* is something that is used in the manufacturing process, but is not permanently associated with a single piece of equipment, can be removed from the piece of equipment without compromising its function, and can be associated with other pieces of equipment during its lifecycle. **Used to identify a storage area in an *MTConnect Agent*:** See description of Buffer. **Used as an *Information Model***: Used to describe an *Information Model* that contains the rules and terminology that describe information that may be included in electronic documents representing *MTConnect Assets*. The *Assets Information Model* defines the structure for the *Assets Response Document.* Individual *Information Models* describe the structure of the *Asset Documents* represent each type of *MTConnect Asset*. Appears in the documents in the following form: *Assets Information Model* or *(asset type) Information Model.*

Term	Definition as Used in the MTConnect Standard
Asset (cont.)	**<u>Used when referring to an *MTConnect Asset*:</u>** Refers to the information related to an *MTConnect Asset* or a group of *MTConnect Assets*. Appears in the documents in the following form: *Asset* or *Assets*. **<u>Used as an XML container or element:</u>** • When used as an XML container that consists of one or more types of `Asset` XML elements. Appears in the documents in the following form: `Assets`. • When used as an abstract XML element. It is replaced in the XML document by types of `Asset` elements representing individual *Asset* entities. Appears in the documents in the following form: `Asset`. **<u>Used to describe information stored in an *MTConnect Agent*:</u>** Identifies an electronic document published by a data source and stored in the *assets buffer* of an *MTConnect Agent*. Appears in the documents in the following form: *Asset Document.* **<u>Used as an XML representation of an *MTConnect Response document*:</u>** Identifies an electronic document encoded in XML and published by an *MTConnect Agent* in response to a *Request* for information from a client software application relating to *MTConnect Assets*. Appears in the documents in the following form: `MTConnectAssets`. **<u>Used as an *MTConnect Request*:</u>** Represents a specific type of communications request between a client software application and an *MTConnect Agent* regarding *MTConnect Assets*. Appears in the documents in the following form: *Asset Request*. **<u>Used as part of an *HTTP Request*:</u>** Used in the path portion of an *HTTP Request Line*, by a client software application, to initiate an *Asset Request* to an *MTConnect Agent* to publish an `MTConnectAssets` document. Appears in the documents in the following form: `asset`.

Term	Definition as Used in the MTConnect Standard
Attribute	A term that is used to provide additional information or properties for an element. Appears in the documents in the following form: attribute.
Base Functional Structure	A consistent set of functionalities defined by the MTConnect Standard. This functionality includes the protocol(s) used to communicate data to a client software application, the *semantic data models* defining how that data is organized into *Response Documents*, and the encoding of those *Response Documents*. Appears in the documents in the following form: *Base Functional Structure*.
Buffer	**General meaning:** A section of an *MTConnect Agent* that provides storage for information published from pieces of equipment. **Used relative to *Streaming Data*:** A section of an *MTConnect Agent* that provides storage for information relating to individual pieces of *Streaming Data*. Appears in the documents in the following form: *buffer*. **Used relative to *MTConnect Assets*:** A section of an *MTConnect Agent* that provides storage for *Asset Documents*. Appears in the documents in the following form: *assets buffer*.
CDATA	**General meaning:** An abbreviation for **C**haracter **Data**. CDATA is used to describe a value (text or data) published as part of an XML element. For example, "*`This is some text`*" is the CDATA in the XML element: 1. `<Message ...>This is some text</Message>` Appears in the documents in the following form: CDATA.
Child Element	A portion of a data modeling structure that illustrates the relationship between an element and the higher-level *Parent Element* within which it is contained. Appears in the documents in the following form: *Child Element*.

<table>
<tr><th>Term</th><th>Definition as Used in the MTConnect Standard</th></tr>
<tr><td>Client</td><td>A process or set of processes that send Requests for information to an MTConnect Agent; e.g. software applications or a function that implements the Request portion of an Interface Interaction Model.

Appears in the documents in the following form: client.</td></tr>
<tr><td>Component</td><td><u>General meaning:</u>

A Structural Element that represents a physical or logical part or sub-part of a piece of equipment.

Appears in the documents in the following form: Component.

<u>Used in Information Models:</u>

A data modeling element used to organize the data being retrieved from a piece of equipment.
When used as an XML container to organize Lower Level <code>Component</code> elements.

Appears in the documents in the following form: <code>Components</code>.
When used as an abstract XML element. <code>Component</code> is replaced in a data model by a type of Component element. <code>Component</code> is also an XML container used to organize Lower Level <code>Component</code> elements, Data Entities, or both.

Appears in the documents in the following form: <code>Component</code>.</td></tr>
</table>

<table>
<tr><th>Term</th><th>Definition as Used in the MTConnect Standard</th></tr>
<tr><td>Composition</td><td><u>General meaning:</u>

Data modeling elements that describe the lowest level basic structural or functional building blocks contained within a <code>Component</code> element.

Appears in the documents in the following form: Composition Element.

<u>Used in Information Models:</u>

• When used as an XML container to organize <code>Composition</code> elements.

Appears in the documents in the following form: <code>Compositions</code>.

• When used as an abstract XML element. <code>Composition</code> is replaced in a data model by a type of Composition Element.

Appears in the documents in the following form: <code>Composition</code>.</td></tr>
</table>

<table>
<tr><th>Term</th><th>Definition as Used in the MTConnect Standard</th></tr>
<tr><td>Condition</td><td><u>General meaning:</u>
An indicator of the health of a piece of equipment or a Component and its ability to function.
<u>Used as a modeling element:</u>
A data modeling element used to organize and communicate information relative to the health of a piece of equipment or Component.
Appears in the documents in the following form: Condition or as Condition Element(s).
<u>Used in Information Models:</u>
An XML element used to represent Condition Elements.
• When used as an XML container to organize Lower Level <code>Condition</code> elements.
Appears in the documents in the following form: <code>Condition</code>.
• When used as a Lower Level element, the form <code>Condition</code> is an abstract type XML element. This Lower Level element is a Data Entity. <code>Condition</code> is replaced in a data model by type of Condition element.
Appears in the documents in the following form: <code>Condition</code>.
Note: The form <code>Condition</code> is used to represent both above uses.</td></tr>
<tr><td>Controlled Vocabulary</td><td>A restricted set of values that may be published as the Valid Data Value for a Data Entity.
Appears in the documents in the following form: Controlled Vocabulary.</td></tr>
</table>

Term	Definition as Used in the MTConnect Standard
Current	**General meaning:** Meaning 1: A term describing the most recent occurrence of something. Meaning 2: A term used to describe movement; e.g. electric current or air current. Appears in the documents in the following form: current **Used in reference to an *MTConnect Agent*:** A reference to the most recent information available to an *MTConnect Agent*. Appears in the documents in the following form: current. **Used as an *MTConnect Request*:** A specific type of communications request between a client software application and an *MTConnect Agent* regarding *Streaming Data*. Appears in the documents in the following form: *Current Request.* **Used as part of an *HTTP Request*:** Used in the path portion of an *HTTP Request Line*, by a client software application, to initiate a *Current Request* to an *MTConnect Agent* to publish an `MTConnectStreams` document. Appears in the documents in the following form: `current`.
Data Dictionary	Listing of standardized terms and definitions used in *MTConnect Information Models*. Appears in the documents in the following form: *data dictionary*.
Data Entity	A primary data modeling element that represents all elements that either describe data items that may be reported by an *MTConnect Agent* or the data items that contain the actual data published by an *Agent*. Appears in the documents in the following form: *Data Entity*.

Term	Definition as Used in the MTConnect Standard
Data Item	**General meaning:** Descriptive information or properties and characteristics associated with a *Data Entity*. Appears in the documents in the following form: data item. **Used in an XML representation of a *Data Entity*:** • When used as an XML container to organize `DataItem` elements. Appears in the documents in the following form: `DataItems`. • When used to represent a specific *Data Entity*, the form `DataItem` is an XML element. Appears in the documents in the following form: `DataItem`.
Data Source	Any piece of equipment that can produce data that is published to an *MTConnect Agent*. Appears in the documents in the following form: data source.
Data Streaming	A method for an *MTConnect Agent* to provide a continuous stream of information in response to a single *Request* from a client software application. Appears in the documents in the following form: *Data Streaming*.
Deprecated	An indication that specific content in an *MTConnect Document* is currently usable but is regarded as being obsolete or superseded. It is recommended that deprecated content should be avoided. Appears in the documents in the following form: **DEPRECATED**.
Deprecation Warning	An indicator that specific content in an *MTConnect Document* may be changed to **DEPRECATED** in a future release of the standard. Appears in the documents in the following form: **DEPRECATION WARNING**.
Devices Information Model	A set of rules and terms that describes the physical and logical configuration for a piece of equipment and the data that may be reported by that equipment. Appears in the documents in the following form: *Devices Information Model*.

<table>
<tr><th>Term</th><th>Definition as Used in the MTConnect Standard</th></tr>
<tr><td>Device</td><td>A part of an information model representing a piece of equipment.
<u>Used in an XML representation of a Response Document:</u>
• When used as an XML container to organize <code>Device</code> elements.
Appears in the documents in the following form: <code>Devices</code>.
• When used as an XML container to represent a specific piece of equipment and is composed of a set of Structural Elements that organize and provide relevance to data published from that piece of equipment.
Appears in the documents in the following form: <code>Device</code>.</td></tr>
<tr><td>Document</td><td><u>General meaning:</u>
A piece of written, printed, or electronic matter that provides information.
<u>Used to represent an MTConnect Document:</u>
Refers to printed or electronic document(s) that represent a Part(s) of the MTConnect Standard.
Appears in the documents in the following form: MTConnect Document.
<u>Used to represent a specific representation of an MTConnect Document:</u>
Refers to electronic document(s) associated with an MTConnect Agent that are encoded using XML; Response Documents or Asset Documents.
Appears in the documents in the following form: MTConnect XML Document.
<u>Used to describe types of information stored in an MTConnect Agent:</u>
In an implementation, the electronic documents that are published from a data source and stored by an MTConnect Agent.
Appears in the documents in the following form: Asset Document.
<u>Used to describe information published by an MTConnect Agent:</u>
A document published by an MTConnect Agent based upon one of the semantic data models defined in the MTConnect Standard in response to a request from a client.
Appears in the documents in the following form: Response Document.</td></tr>
</table>

Term	Definition as Used in the MTConnect Standard
Document Body	The portion of the content of an *MTConnect Response Document* that is defined by the relative *MTConnect Information Model*. The *Document Body* contains the *Structural Elements* and *Data Entities* reported in a *Response Document*. Appears in the documents in the following form: *Document Body*.
Document Header	The portion of the content of an *MTConnect Response Document* that provides information from an *MTConnect Agent* defining version information, storage capacity, protocol, and other information associated with the management of the data stored in or retrieved from the *Agent*. Appears in the documents in the following form: *Document Header*.
Element	Refers to an XML element. An XML element is a logical portion of an XML document or schema that begins with a `start-tag` and ends with a corresponding `end-tag`. The information provided between the `start-tag` and `end-tag` may contain attributes, other elements (sub-elements), and/or CDATA. Note: Also, an XML element may consist of an `empty-element tag.` Refer to *Appendix B* for more information on element tags. Appears in the documents in the following form: element.
Element Name	A descriptive identifier contained in both the `start-tag` and `end-tag` of an XML element that provides the name of the element. Appears in the documents in the following form: element name. **Used to describe the name for a specific XML element:** Reference to the name provided in the `start-tag`, `end-tag`, or `empty-element tag` for an XML element. Appears in the documents in the following form: *Element Name*.
Equipment	Represents anything that can publish information and is used in the operations of a manufacturing facility shop floor. Examples of equipment are machine tools, ovens, sensor units, workstations, software applications, and bar feeders. Appears in the documents in the following form: equipment or piece of equipment.

<table>
<tr><th>Term</th><th>Definition as Used in the MTConnect Standard</th></tr>
<tr><td>Error Information Model</td><td>The rules and terminology that describes the Response Document returned by an MTConnect Agent when it encounters an error while interpreting a Request for information from a client software application or when an Agent experiences an error while publishing the Response to a Request for information.

Appears in the documents in the following form: Error Information Model.</td></tr>
<tr><td>Event</td><td><u>General meaning:</u>

The occurrence of something that happens or takes place.

Appears in the documents in the following form: event.

<u>Used as a type of Data Entity:</u>

An identification that represents a change in state of information associated with a piece of equipment or an occurrence of an action. Event also provides a means to publish a message from a piece of equipment.

Appears in the documents in the following form: Event.

<u>Used as a <code>category</code> attribute for a Data Entity:</u>

Used as a value for the <code>category</code> attribute for an XML <code>dataItem</code> element.

Appears in the documents in the following form: <code>EVENT</code>.

<u>Used as an XML container or element:</u>
When used as an XML container that consists of one or more types of <code>Event</code> XML elements.

Appears in the documents in the following form: <code>Events</code>.
When used as an abstract XML element. It is replaced in the XML document by types of <code>Event</code> elements.

Appears in the documents in the following form: <code>Event</code>.</td></tr>
<tr><td>Extensible</td><td>The ability for an implementer to extend MTConnect Information Models by adding content not currently addressed in the MTConnect Standard.</td></tr>
<tr><td>Fault State</td><td>In the MTConnect Standard, a term that indicates the reported status of a Condition category Data Entity.

Appears in the documents in the following form: Fault State.</td></tr>
</table>

Term	Definition as Used in the MTConnect Standard
Heartbeat	**<u>General meaning:</u>** A function that indicates to a client application that the communications connection to an *MTConnect Agent* is still viable during times when there is no new data available to report – often referred to as a "keep alive" message. Appears in the documents in the following form: *heartbeat*. **<u>When used as part of an *HTTP Request*:</u>** The form `heartbeat` is used as a parameter in the `query` portion of an *HTTP Request Line.* Appears in the documents in the following form: `heartbeat`.
HTTP	**H**yper-**T**ext **T**ransport **P**rotocol. The protocol used by all web browsers and web applications. Note: HTTP is an IETF standard and is defined in RFC 7230. See https://tools.ietf.org/html/rfc7230 for more information.
HTTP Error Message	In the MTConnect Standard, a response provided by an *MTConnect Agent* indicating that an *HTTP Request* is incorrectly formatted or identifies that the requested data is not available from the *Agent.* Appears in the documents in the following form: *HTTP Error Message*.
HTTP Header	In the MTConnect Standard, the ccontent of the *Header* portion of either an *HTTP Request* from a client software application or an *HTTP Response* from an *MTConnect Agent*. Appears in the documents in the following form: *HTTP Header*.
HTTP Method	In the MTConnect Standard, a portion of a command in an *HTTP Request* that indicates the desired action to be performed on the identified resource; often referred to as verbs.
HTTP Request	In the MTConnect Standard, a communications command issued by a client software application to an *MTConnect Agent* requesting information defined in the *HTTP Request Line*. Appears in the documents in the following form: *HTTP Request*.

Term	Definition as Used in the MTConnect Standard
HTTP Request Line	In the MTConnect Standard, the first line of an *HTTP Request* describing a specific *Response Document* to be published by an *MTConnect Agent*. Appears in the documents in the following form: *HTTP Request Line*.
HTTP Response	In the MTConnect Standard, the information published from an *MTConnect Agent* in reply to an *HTTP Request*. An *HTTP Response* may be either a *Response Document* or an *HTTP Error Message*. Appears in the documents in the following form: *HTTP Response*.
HTTP Server	In the MTConnect Standard, a software program that accepts *HTTP Requests* from client software applications and publishes *HTTP Responses* as a reply to those *Requests*. Appears in the documents in the following form: *HTTP Server*.
HTTP Status Code	In the MTConnect Standard, a numeric code contained in an *HTTP Response* that defines a status category associated with the *Response* – either a success status or a category of an HTTP error. Appears in the documents in the following form: *HTTP Status Code*.
id	<u>**General meaning:**</u> An identifier used to distinguish a piece of information. Appears in the documents in the following form: id. <u>**Used as an XML attribute:**</u> When used as an attribute for an XML element - *Structural Element, Data Entity*, or *Asset*. `id` provides a unique identity for the element within an XML document. Appears in the documents in the following form: `id`.
Implementation	A specific instantiation of the MTConnect Standard.
Information Model	The rules, relationships, and terminology that are used to define how information is structured. For example, an information model is used to define the structure for each *MTConnect Response Document*; the definition of each piece of information within those documents and the relationship between pieces of information. Appears in the documents in the following form: *Information Model*.

<table>
<tr><th>Term</th><th>Definition as Used in the MTConnect Standard</th></tr>
<tr><td>Instance</td><td>Describes a set of Streaming Data in an MTConnect Agent. Each time an Agent is restarted with an empty buffer, data placed in the buffer represents a new instance of the Agent.

Appears in the documents in the following form: instance.</td></tr>
<tr><td>Interaction Model</td><td>The definition of information exchanged to support the interactions between pieces of equipment collaborating to complete a task.

Appears in the documents in the following form: Interaction Model.</td></tr>
<tr><td>Interface</td><td><u>General meaning:</u>

The exchange of information between pieces of equipment and/or software systems.

Appears in the documents in the following form: interface.

<u>Used as an Interaction Model:</u>

An Interaction Model that describes a method for inter-operations between pieces of equipment.

Appears in the documents in the following form: Interface.

<u>Used as an XML container or element:</u>
When used as an XML container that consists of one or more types of <code>Interface</code> XML elements.

Appears in the documents in the following form: <code>Interfaces</code>.
When used as an abstract XML element. It is replaced in the XML document by types of <code>Interface</code> elements.

Appears in the documents in the following form: <code>Interface</code>.</td></tr>
</table>

Term	Definition as Used in the MTConnect Standard
Message	**<u>General meaning:</u>** The content of a communication process. Appears in the documents in the following form: message. **<u>Used relative to an *MTConnect Agent*:</u>** Describes the information that is exchanged between an *MTConnect Agent* and a client software application. A *Message* may contain either a *Request* from a client software application or a *Response* from an *MTConnect Agent*. Appears in the documents in the following form: *Message*. **<u>Used as a type *of Data Entity*:</u>** Describes a type of *Data Entity* in the *Devices Information Model* that can contain any text string of information or native code to be transferred from a piece of equipment. Appears in the documents in the following form: `MESSAGE`. **<u>Used as an *Element Name*:</u>** An *Element Name* for a *Data Entity* in the *Streams Information Model* that can contain any text string of information or native code to be transferred from a piece of equipment. Appears in the documents in the following form: `Message`.
Metadata	Data that provides information about other data. For example, *Equipment Metadata* defines both the *Structural Elements* that represent the physical and logical parts and sub-parts of each piece of equipment, the relationships between those parts and sub-parts, and the definitions of the *Data Entities* associated with that piece of equipment. Appears in the documents in the following form: *Metadata* or *Equipment Metadata*.
MTConnect Agent	See definition for *Agent*.
MTConnectAssets Response Document	An electronic document published by an *MTConnect Agent* in response to a *Request* for information from a client software application relating to *MTConnect Assets*. Appears in the documents in the following form: *MTConnectAssets Response Document*.

Term	Definition as Used in the MTConnect Standard
MTConnectDevices Response Document	An electronic document published by an *MTConnect Agent* in response to a *Request* for information from a client software application that includes *metadata* for one or more pieces of equipment. Appears in the documents in the following form: *MTConnectDevices Response Document*.
MTConnectErrors Response Document	An electronic document published by an *MTConnect Agent* whenever it encounters an error while interpreting a *Request* for information from a client software application or when an *Agent* experiences an error while publishing the *Response* to a *Request* for information. Appears in the documents in the following form: *MTConnectErrors Response Document*.
MTConnect Request	A communication request for information issued from a client software application to an *MTConnect Agent*. Appears in the documents in the following form: *MTConnect Request*.
MTConnectStreams Response Document	An electronic document published by an *MTConnect Agent* in response to a *Request* for information from a client software application that includes *Streaming Data* from the *Agent*. Appears in the documents in the following form: *MTConnectStreams Response Document*.
NMTOKEN	The data type for XML identifiers. Note: The identifier must start with a letter, an underscore "_" or a colon. The next character must be a letter, a number, or one of the following ".", "-", "_", ":". The identifier must not have any spaces or special characters. Appears in the documents in the following form: NMTOKEN.
Parameter	**General Meaning:** A variable that must be given a value during the execution of a program or a communications command. **When used as part of an *HTTP Request*:** Represents the content (keys and associated values) provided in the *Query* portion of an *HTTP Request Line* that identifies specific information to be returned in a *Response Document*. Appears in the documents in the following form: parameter.

Term	Definition as Used in the MTConnect Standard
Parent Element	An XML element used to organize *Lower Level* child elements that share a common relationship to the *Parent Element*. Appears in the documents in the following form: *Parent Element*.
Persistence	A method for retaining or restoring information.
Probe	**General meaning of a physical entity:** An instrument commonly used for measuring the physical geometrical characteristics of an object. • **Used to describe a measurement device:** The form probe is used to define a measurement device that provides position information. Appears in the documents in the following form: probe. • **Used within a *Data Entity*:** The form `PROBE` is used to designate a subtype for the *Data Entity* `PATH_POSITION` indicating a measurement position relating to a probe unit. Appears in the documents in the following form: `PROBE`. **General meaning for communications with an *MTConnect Agent*:** Probe is used to define a type of communication request. • **Used as a type of communication request:** The form *Probe Request* represents a specific type of communications request between a client software application and an *MTConnect Agent* regarding *metadata* for one or more pieces of equipment. Appears in the documents in the following form: *Probe Request*. • **Used in an *HTTP Request Line*:** The form `probe` is used to designate a *Probe Request* in the `<Path>` portion of an *HTTP Request Line*. Appears in the documents in the following form: `probe`.
Protocol	A set of rules that allow two or more entities to transmit information from one to the other.

Term	Definition as Used in the MTConnect Standard
Publish/Subscribe	In the MTConnect Standard, a communications messaging pattern that may be used to publish *Streaming Data* from an *MTConnect Agent*. When a *Publish/Subscribe* communication method is established between a client software application and an *MTConnect Agent*, the *Agent* will repeatedly publish a specific *MTConnectStreams* document at a defined period. Appears in the documents in the following form: *Publish/Subscribe*.
Query	**General Meaning:** A portion of a request for information that more precisely defines the specific information to be published in response to the request. Appears in the documents in the following form: *Query*. **Used in an *HTTP Request Line*:** The form `query` includes a string of parameters that define filters used to refine the content of a *Response Document* published in response to an *HTTP Request*. Appears in the documents in the following form: `query`.
Request /Response Messaging Structure	A communications pattern that supports the transfer of information between an *MTConnect Agent* and a client software application. In a *Request/Response* information exchange, a client software application requests specific information from an *MTConnect Agent*. An *MTConnect Agent* responds to the *Request* by publishing a *Response Document*. Appears in the documents in the following form: *Request/Response Messaging Structure*.
Request	A communications method where a client software application transmits a message to an *MTConnect Agent*. That message instructs the *Agent* to respond with specific information. Appears in the documents in the following form: *Request*.
Requester	An entity that initiates a *Request* for information in a communications exchange. Appears in the documents in the following form: *Requester*.
Responder	An entity that responds to a *Request* for information in a communications exchange. Appears in the documents in the following form: *Responder*.

Term	Definition as Used in the MTConnect Standard
Response Document	See definition of Document.
REST	Stands for **RE**presentational **S**tate **T**ransfer: A software architecture where a client software application and server move through a series of state transitions based solely on the request from the client and the response from the server. Appears in the documents in the following form: REST.
Root Element	The first *Structural Element* provided in a *Response Document* encoded using XML. The *Root Element* is an XML container and is the *Parent Element* for all other XML elements in the document. The *Root Element* appears immediately following the *XML Declaration*. Appears in the documents in the following form: *Root Element*.

<table>
<tr><th>Term</th><th>Definition as Used in the MTConnect Standard</th></tr>
<tr><td>Sample</td><td><u>General meaning:</u>

The collection of one or more pieces of information.

<u>Used when referring to the collection of information:</u>

When referring to the collection of a piece of information from a data source.

Appears in the documents in the following form: sample.

<u>Used as an MTConnect Request:</u>

When representing a specific type of communications request between a client software application and an MTConnect Agent regarding Streaming Data.

Appears in the documents in the following form: Sample Request.

<u>Used as part of an HTTP Request:</u>

Used in the <code>path</code> portion of an HTTP Request Line, by a client software application, to initiate a Sample Request to an MTConnect Agent to publish an <code>MTConnectStreams</code> document.

Appears in the documents in the following form: <code>sample</code>.

<u>Used to describe a Data Entity:</u>

Used to define a specific type of Data Entity. A Sample type Data Entity reports the value for a continuously variable or analog piece of information.

Appears in the documents in the following form: Sample or Samples.

<u>Used as an XML container or element:</u>

• When used as an XML container that consists of one or more types of <code>Sample</code> XML elements.

Appears in the documents in the following form: <code>Samples</code>.

• When used as an abstract XML element. It is replaced in the XML document by types of <code>Sample</code> elements representing individual Sample type of Data Entity.

Appears in the documents in the following form: <code>Sample</code>.</td></tr>
</table>

Term	Definition as Used in the MTConnect Standard
Schema	**General meaning:** The definition of the structure, rules, and vocabularies used to define the information published in an electronic document. Appears in the documents in the following form: schema. **Used in association with an *MTConnect Response Document*:** Identifies a specific schema defined for an *MTConnect Response Document*. Appears in the documents in the following form: *schema*.
Semantic Data Model	A methodology for defining the structure and meaning for data in a specific logical way. It provides the rules for encoding electronic information such that it can be interpreted by a software system. Appears in the documents in the following form: *semantic data model*.
Sequence Number	The primary key identifier used to manage and locate a specific piece of *Streaming Data* in an *MTConnect Agent*. *Sequence number* is a monotonically increasing number within an *instance* of an *MTConnect Agent*. Appears in the documents in the following form: *sequence number*.
Standard	**General meaning:** A document established by consensus that provides rules, guidelines, or characteristics for activities or their results (as defined in ISO/IEC Guide 2:2004). **Used when referring to the MTConnect Standard.** The MTConnect Standard is a standard that provides the definition and semantic data structure for information published by pieces of equipment. Appears in the documents in the following form: Standard or MTConnect Standard.
Streaming Data	The values published by a piece of equipment for the *Data Entities* defined by the *Equipment Metadata*. Appears in the documents in the following form: *Streaming Data*.

Term	Definition as Used in the MTConnect Standard
Streams Information Model	The rules and terminology (*semantic data model*) that describes the *Streaming Data* returned by an *MTConnect Agent* from a piece of equipment in response to a *Sample Request* or a *Current Request*. Appears in the documents in the following form: *Streams Information Model*.
Structural Element	**General meaning:** An XML element that organizes information that represents the physical and logical parts and sub-parts of a piece of equipment. Appears in the documents in the following form: *Structural Element*. **Used to indicate hierarchy of Components:** When used to describe a primary physical or logical construct within a piece of equipment. Appears in the documents in the following form: *Top Level Structural Element*. When used to indicate a *Child Element* which provides additional detail describing the physical or logical structure of a *Top Level Structural Element.* Appears in the documents in the following form: *Lover Level Structural Element*.
Subtype	**General meaning:** A secondary or subordinate type of categorization or classification of information. In software and data modeling, a subtype is a type of data that is related to another higher-level type of data. Appears in the documents in the following form: subtype. **Used as an attribute for a *Data Entity*:** Used as an attribute that provides a sub-categorization for the `type` attribute for a piece of information. Appears in the documents in the following form: `subType`.

Term	Definition as Used in the MTConnect Standard
Time Stamp	**General meaning:** The best available estimate of the time that the value(s) for published or recorded information was measured or determined. Appears in the documents as "time stamp". **Used as an attribute for recorded or published data:** An attribute that identifies the time associated with a *Data Entity* as stored in an *MTConnect Agent*. Appears in the documents in the following form: `timestamp`.
Type	**General meaning:** A classification or categorization of information. In software and data modeling, a type is a grouping function to identify pieces of information that share common characteristics. Appears in the documents in the following form: type. **Used as an attribute for a *Data Entity*:** Used as an attribute that provides a categorization for piece of information that share common characteristics. Appears in the documents in the following form: `type`.
URI	Stands for **U**niversal **R**esource **I**dentifier. See http://www.w3.org/TR/uri-clarification/#RFC3986
URL	Stands for **U**niform **R**esource **L**ocator. See http://www.w3.org/TR/uri-clarification/#RFC3986
URN	Stands for **U**niform **R**esource **N**ame. See http://www.w3.org/TR/uri-clarification/#RFC3986
UTC/GMT	Stands for **C**oordinated **U**niversal **T**ime/**G**reenwich **M**ean **T**ime. UTC/GMT is the primary time standard by which the world regulates clocks and time. The time stamp for all information reported in an *MTConnect Response* document is provided in UTC/GMT format.

Term	Definition as Used in the MTConnect Standard
UUID	<u>**General meaning:**</u> Stands for **U**niversally **U**nique **I**dentifier. (Can also be referred to as a GUID in some literature – Globally Unique Identifier). Note: Defined in RFC 4122 of the IETF. See https://www.ietf.org/rfc/rfc4122.txt for more information. Appears in the documents in the following form: UUID. <u>**Used as an attribute for an XML element:**</u> Used as an attribute that provides a unique identity for a piece of information reported by an *MTConnect Agent*. Appears in the documents in the following form: `uuid`.
Valid Data Values	One or more acceptable values or constrained values that can be reported for a *Data Entity*. Appears in the documents in the following form: *Valid Data Value(s)*.
W3C	Stands for **W**orld **W**ide **W**eb **C**onsortium. W3C is an international community of organizations and the public work together to develop internet standards. W3C Standards are used as a guide within the MTConnect Standard.
WARNING	**General Meaning:** A statement or action that indicates a possible danger, problem, or other unexpected situation. **Used relative to changes in an *MTConnect Document*:** Used to indicate that specific content in an *MTConnect Document* may be changed in a future release of the standard. Appears in the documents in the following form: **WARNING**. **Used as a *Valid Data Value* for a *Condition*:** Used as a *Valid Data Value* for a *Condition* type *Data Entity*. Appears in the documents in the following form: `WARNING`. **Used as an *Element Name* for a *Data Entity*:** Used as the *Element Name* for a *Condition* type *Data Entity* in an *MTConnectStreams Response Document*. Appears in the documents in the following form: `Warning`.

Term	Definition as Used in the MTConnect Standard
XML	Stands for E**X**tensible **M**arkup **L**anguage. XML defines a set of rules for encoding documents that both a human-readable and machine-readable. XML is the language used for all code examples in the MTConnect Standard. Refer to http://www.w3.org/XML for more information about XML.
XML Container	In the MTConnect Standard, a type of XML element. An XML container is used to organize other XML elements that are logically related to each other. A container may have either *Data Entities* or other *Structural Elements* as *Child Elements*.
XML Document	An XML document is a structured text file encoded using XML. An XML document is an instantiation of an XML schema. It has a single root XML element, conforms to the XML specification, and is structured based upon a specific schema. *MTConnect Response Documents* may be encoded as an XML document.
XML Schema	In the MTConnect Standard, an instantiation of a schema defining a specific document encoded in XML.
XPATH	**General meaning:** XPATH is a command structure that describes a way for a software system to locate information in an XML document. XPATH uses an addressing syntax based on a path through the document's logical structure. See http://www.w3.org/TR/xpath for more information on XPATH. Appears in the documents in the following form: XPATH.

4 MTConnect Standard

The MTConnect® Standard is organized in a series of documents (also referred to as MTConnect Documen*ts*) that each address a specific set of requirements defined by the Standard. Each MTConnect Document will be referred to as a *Part* of the Standard; e.g., *Part 1.0 - Functionality and Overview*. Together, these documents describe the *Base Functional Structure* specified in the MTConnect Standard.

Implementation of any manufacturing data management system may utilize information from any number of these documents. However, it is not necessary to realize all information contained in these documents for any one specific implementation.

4.1 MTConnect Documents Organization

The MTConnect specification is organized into the following documents:

Part 1.0 – Overview and Functionality: Provides an overview of the MTConnect Standard and defines the terminology and structure used throughout all documents associated with the Standard. Additionally, *Part 1.0* describes the functions provided by an *MTConnect Agent* and the protocol used to communicate with an *MTConnect Agent*.

Part 2.0 – Devices Information Model: Defines the *semantic data model* that describes the data that can be supplied by a piece of equipment. This model details the XML elements used to describe the structural and logical configuration for a piece of equipment. It also describes each type of data that may be supplied by a piece of equipment in a manufacturing operation.

Part 3.0 – Streams Information Model: Defines the *semantic data model* that organizes the data that is collected from a piece of equipment and transferred to a client software application from an *MTConnect Agent*.

Part 4.0 – Assets Information Model: Provides an overview of *MTConnect Assets* and the functions provided by an *MTConnect Agent* to communicate information relating to *Assets*. The various *semantic data models* describing each type of *MTConnect Asset* are defined in sub-*Part* documents (*Part 4.x*) of the MTConnect Standard.

Part 5.0 – Interfaces: Defines the MTConnect implementation of the *Interaction Model* used to coordinate actions between pieces of equipment used in manufacturing systems.

4.2 MTConnect Document Versioning

The MTConnect Standard will be periodically updated with new and expanded functionality. Each new release of the Standard will include additional content adding new functionality and/or extensions to the *semantic data models* defined in the Standard.

The MTConnect Standard uses a three-digit version numbering system to identify each release of the Standard that indicates the progression of enhancements to the Standard. The format used to identify the documents in a specific version of the MTConnect Standard is:

major.minor.revision

major – Identifier representing a consistent set of functionalities defined by the MTConnect Standard. This functionality includes the protocol(s) used to communicate data to a client software application, the *semantic data models* defining how that data is organized into *Response Documents*, and the encoding of those *Response Documents*. This set of functionalities is referred to as the *Base Functional Structure*.

When a release of the MTConnect Standard removes or modifies any of the protocol(s), *semantic data models*, or encoding of the *Response Documents* included in the *Base Functional Structure* in such a way that it breaks backward compatibility and a client software application can no longer communicate with an *MTConnect Agent* or cannot interpret the information provided by an *MTConnect Agent*, the *major* version identifier for the Documents in the release is revised to a successively higher number.

See *Section 4.6 – Backwards Compatibility* for details regarding the interaction between a client software application and versions of the MTConnect Standard.

minor – Identifier representing a specific set of functionalities defined by the MTConnect Standard. Each release of the Standard (with a common *major* version identifier) includes new and/or expanded functionality – protocol extensions, new or extended *semantic data models*, and/or new programming languages. Each of these releases of the Standard is indicated by a successively higher *minor* version identifier.

If a new *major* version of the MTConnect Standard is released, the *minor* version identifier will be reset to 0.

revision – A supplemental identifier representing only organizational or editorial changes to a *minor* version document with no changes in the functionality described in that document.

New releases of a specific document are indicated by a successively higher *revision* version identifier.

If a new *minor* version of a document is released, the *revision* identifier will be reset to 0.

An example of the Version identifier for a specific document would be:

Version *M.N.R*

4.2.1 Document Releases

A *major* revision change represents a substantial change to the MTConnect Standard. At the time of a *major* revision change, all documents representing the MTConnect Standard will be updated and released together.

A *minor* revision change represents some level of extended functionality supported by the MTConnect Standard. At the time of a *minor* version release, MTConnect Documents representing the changes or enhancements to the Standard will be updated as required. However, all documents, whether updated or not, will be released together with a new *minor* version number. Providing all documents at a common *major* and *minor* version makes it easier for implementers to manage the compatibility and upgrade of the different software tools incorporated into a manufacturing software system.

Since a *revision* represents no functional changes to the MTConnect Standard and includes only editorial or descriptive changes that enhance the understanding of the functionality supported by the Standard, individual documents within the Standard may be released at any time with a new *revision* and that release does not impact any other documents associated with the MTConnect Standard.

The latest released version of each document provided for the MTConnect Standard, and historical releases of those documents, are provided at http://www.mtconnect.org .

4.3 MTConnect Document Naming Convention

MTConnect Documents are identified as follows:

4.3.1 Document Title

Each MTConnect Document **MUST** be identified as follows:

MTConnect® Standard

Part #.# - *Title*

Version *M.N.R*

The following keys are used to distinguish different *Parts* of the MTConnect Standard and the version of the MTConnect Document:

#.# – Identifier of the specific *Par*t and sub-Part of the MTConnect Standard

Title – Description of the type of information contained in the MTConnect Document

M – Indicator of the major version of the MTConnect Document

N– Indicator of the minor version of the MTConnect Document

R – Indicator of the revision of the MTConnect Document

For example, a release of *Part 2.0 – Devices Information Model* would be:

MTConnect® Standard

Part 2.0 – Devices Information Model

Version 1.2.0

4.3.2 Electronic Document File Naming

Electronic versions of the MTConnect Documents will be provided in PDF format. The naming convention of the electronic files representing each document will be identified as follows:

MTC_Part_#.#_*Title*_*M.N.R*.pdf

The same keys are used to distinguish the electronic documents as are defined above for the document title.

The electronic version of the same release of *Part 2.0 – Devices Information Model* would be:

MTC_Part_2.0_Devices Information Model_1.2.0.pdf

4.4 Document Conventions

Additional information regarding specific content in the MTConnect Standard is provided in the sections below.

4.4.1 Use of MUST, SHOULD, and MAY

These words convey specific meaning in the MTConnect Standard when presented in capital letters, Times New Roman font, and a Bold font style.

- The word **MUST** indicates content that is mandatory to be provided in an implementation where indicated.
- The word **SHOULD** indicates content that is recommended, but the exclusion of which will not invalidate an implementation.
- The word **MAY** indicates content that is optional. It is up to the implementer to decide if the content is relevant to an implementation.
- The word **NOT** may be added to the words **MUST** or **SHOULD** to negate the requirement.

4.4.2 Text Conventions

The following conventions will be used throughout the MTConnect Documents to provide a clear and consistent understanding of the use of each type of information used to define the MTConnect Standard.

These conventions are:

- Standard text is provided in Times New Roman font.
- References to documents, sections or sub-sections of a document, or figures within a document are *italicized*; e.g., *Part 2.0 – Devices Information Model.*
- Terms with a specific meaning in the MTConnect Standard will be *italicized*; e.g., *major* indicating a version of the Standard.
- When these same terms are used within the text without specific reference to their function within the MTConnect Standard, they will be provided as non-italicized font; e.g., major indicating a descriptor of another term.
- Terms representing content of an MTConnect *semantic data model* or the protocol used in MTConnect will be provided in fixed size, Courier New font; e.g., `component`, `probe`, `current`.

  When these same terms are used within the text without specific reference to their function within the MTConnect Standard, they will be provided as Times New Roman font.
- All *Valid Data Values* that are restricted to a limited or controlled vocabulary will be provided in upper case Courier New font with an _ (underscore) separating words. For example: `ON`, `OFF`, `ACTUAL`, `COUNTER_CLOCKWISE`, etc.
- All descriptive attributes associated with each piece of data defined in a *Response Document* will be provided in Courier New font and camel case font style. For example: `nativeUnits`.

4.4.3 Code Line Syntax and Conventions

The following conventions will be used throughout the MTConnect Documents to describe examples of software code produced by an *MTConnect Agent* or commands provided to an *Agent* from a client software application.

All examples are provided in fixed size Courier New font with line numbers.

These conventions are:

- XML Code examples:

```
<MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mtconnect.com:MTConnectStreams:1.1"
```

- HTTP URL examples:
  - http://<authority>/<path>[?<query>]When a portion of a URL is enclosed in angle brackets ("<" and ">"), that section of the URL is a place holder for specific information that will replace the term between the angle brackets.

    Note: The angle brackets in a URL do not relate to the angle brackets used as the `tag` elements in an XML example.
  - A portion of a URL that is enclosed in square brackets "[" and "]" indicates that the enclosed content is optional.
  - All other characters in the URL are literal.

4.4.4 *Semantic Data Model* Content

For each of the *semantic data models* defined in the MTConnect Standard, there are tables describing pieces of information provided in the data models. Each table has a column labeled *Occurrence*. *Occurrence* defines the number of times the content defined in the tables **MAY** be provided in the usage case specified.

- If the *Occurrence* is 1, the content **MUST** be provided.
- If the *Occurrence* is 0..1, the content **MAY** be provided and if provided, at most, only one occurrence of the content **MUST** be provided.
- If the *Occurrence* is 0..INF, the content **MAY** be provided and any number of occurrences of the content **MAY** be provided.
- If the *Occurrence* is 1..INF, one or more occurrences of the content **MUST** be provided.
- If the *Occurrence* is a number, e.g., 2, exactly that number of occurrences of the content **MUST** be provided.

4.4.5 Referenced Standards and Specifications

Other standards and specifications may be used to describe aspects of the protocol, *data dictionary*, or *semantic data models* defined in the MTConnect Standard. When a specific standard or specification is referenced in the MTConnect Standard, the name of the standard or specification will be provided in *italicized* font.

See *Appendix A: Bibliography* for a complete listing of standards and specifications used or referenced in the MTConnect Standard.

4.4.6 Deprecation and Deprecation Warnings

When the MTConnect Institute adds new functionality to the MTConnect Standard, the new content may supersede some of the functionality of existing content or significantly enhance one of the *semantic data models*. When this occurs, existing content may no longer be valid for use in the new version of the Standard.

4.4.6.1 Deprecation

In cases when new content supersedes the functionality of the existing content, the original content **MUST** no longer be included in future implementations – only the new content should be used.

The superseded content is identified by striking through the original content (~~original content~~) and marking the content with the words "**DEPRECATED** in *Version M.N"*.

The deprecated content must remain in all future *minor* versions of the document. The content may be removed when a *major* version update is released. This provides implementers guidance on how to interpret data that may be provided from equipment utilizing an older version of the Standard. This content provides the information required for implementers to develop software applications that support backwards compatibility with older versions of the standard.

A software application may be designed to be compliant with any specific *minor* version of the standard. That software application may be collecting data from many different pieces of equipment. Each of these pieces of equipment may be providing data defined by the current version or any of the previous *minor* versions of the standard. To maintain compatibility with existing pieces of equipment, software applications should be implemented to interpret data defined in the current release of the MTConnect Standard, as well as all deprecated content associated with earlier versions of the Standard.

4.4.6.2 Deprecation Warning

When new content provides improved alternatives for defining the *semantic data models*, the MTConnect Institute may determine that the original content could possibly be deprecated in the future. When this occurs, a content will be marked with the words "**DEPRECATION WARNING"** to identify the content that may be deprecated in the future. This provides advanced notice to implementers that they should choose to utilize the improved alternatives when developing new products or software systems to avoid the possibility that the original content may be deprecated in a future version of the Standard.

4.5 Document Version Management

The MTConnect Institute establishes a balanced approach to determining when, or if, to release an updated version of the MTConnect Standard. New versions of the MTConnect Standard will be released periodically to extend the functionality defined by the Standard. It is a strategic objective of the MTConnect Institute that new releases of the Standard must not occur too frequently since each release may disrupt existing products and software systems. Decisions on the timing and content of new versions of the Standard are determined by the MTConnect Technical Advisory Group (TAG).

Any MTConnect Document designated with a new *major* and *minor* version number that includes substantive changes requires a 90-day review of the new content in the document by the TAG prior to the release of that document. This review period allows the TAG time to comment on the recommended changes and to determine that the additional content provided in each version is clearly defined. Additionally, the TAG review includes an assessment that the new content is free from known intellectual property, patent, and copyright infringements.

If the TAG review identifies a need for additional substantive changes to any MTConnect Document, that Document will be again updated and submitted for an additional 30-day review period by the TAG. This process is repeated until a voting majority of the TAG approves each Document to be considered as a release candidate for a new version of the MTConnect Standard.

If only editorial changes are made to an MTConnect Document, then a review of that document is not required. However, upon the discretion of the Technical Steering Committee, a 30-day review of the changed content may be requested.

Once all Documents associated with a planned release are reviewed and approved, the MTConnect Institute will then seek approval for the release of the new version of the Standard from the MTConnect Board of Trustees. After that, there will be a formal announcement of the availability of a new release of the MTConnect Standard.

4.6 Backwards Compatibility

MTConnect Documents with a different *major* version identifier represent a significant change in the *Base Functional Structure* of the MTConnect Standard. This means that the schema or protocol defined by the Standard may have changed in ways that will require software applications to change how they request and/or interpret data received from an *MTConnect Agent*. Software applications should be fully version aware since no assumption of backwards compatibility should be assumed at the time of a *major* revision change to the MTConnect Standard.

The MTConnect Institute strives to maintain version compatibility through all *minor* revisions of the MTConnect Standard. New *minor* versions may introduce extensions to existing *semantic data models*, extend the protocol used to communicate to the *MTConnect Agent*, and/or add new *semantic data models* to extend the functionality of the Standard. Client software applications may be designed to be compliant with any specific *minor* version of the MTConnect Standard. Additionally, software applications should be capable of interpreting information from an *MTConnect Agent* providing data based upon a lower *minor* version identifier. It should also be capable of interpreting information from an *MTConnect Agent* providing data based upon a higher *minor* version identifier of the MTConnect Standard than the version supported by the client, even though the client may ignore or not be capable of interpreting the extended content provided by the *MTConnect Agent*.

A *revision* version of any MTConnect Document provides only editorial changes requiring no changes to an *MTConnect Agent* or a client application.

5 MTConnect Fundamentals

The MTConnect® Standard defines the functionality of an *MTConnect Agent*. In an MTConnect installation, pieces of equipment publish information to an *MTConnect Agent*. Client software applications request information from the *Agent* using a communications protocol. Based on the specific information that the client software application has requested from the *Agent*, the *Agent* forms a *Response Document* based upon one of the *semantic data models* defined in the MTConnect Standard and then transmits that document to the client software application.

Figure 2 below illustrates the architecture of a typical MTConnect installation.

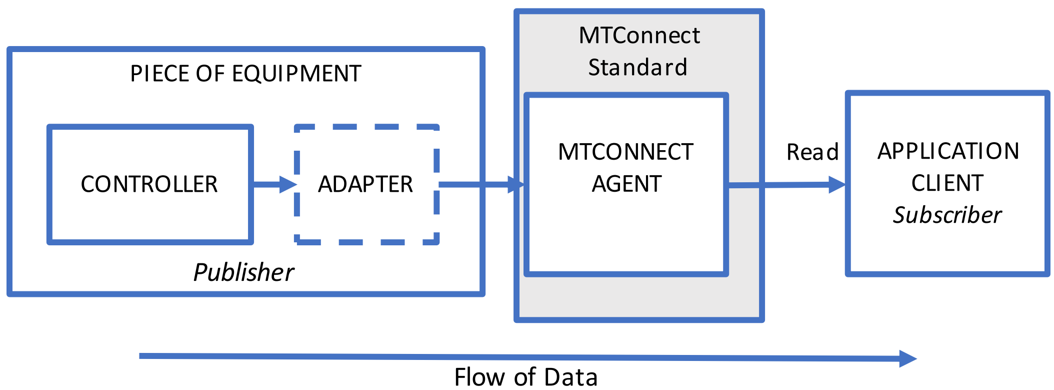


Figure 2: *MTConnect Architecture* Model

Note: In each implementation of a communication system based on the MTConnect Standard, there **MUST** be a *schema* defined that encodes the rules and terminology defined for each of the *semantic data models*. These *schemas* **MAY** be used by client software applications to validate the content and structure of the *Response Documents* published by an *MTConnect Agent*.

5.1 *MTConnect Agent*

An *MTConnect Agent* is the centerpiece of an MTConnect implementation. It provides two primary functions:

- Organizes and manages individual pieces of information published by one or more pieces of equipment.
- Publishes that information in the form of a *Response Document* to client software applications.

The MTConnect Standard addresses the behavior of an *MTConnect Agent* and the structure and meaning of the data published by an *Agent*. It is the responsibility of the implementer of an *MTConnect Agent* to determine the means by which the behavior is achieved for a specific *Agent*.

An *MTConnect Agent* is software that may be installed as part of a piece of equipment or it may be installed separately. When installed separately, an *Agent* may receive information from one or more pieces of equipment.

Some pieces of equipment may be able to communicate directly to an *MTConnect Agent*. Other pieces of equipment may require an *Adapter* to transform the information provided by the equipment into a form that can be sent to an *Agent*. In either case, the method of transmitting information from the piece of equipment to an *MTConnect Agent* is implementation dependent and is not addressed as part of the MTConnect Standard.

One function of an *MTConnect Agent* is to store information that it receives from a piece of equipment in an organized manner. A second function of an *MTConnect Agent* is to receive *Requests* for information from one or many client software applications and then respond to those *Requests* by publishing a *Response Document* that contains the requested information.

There are three types of information stored by an *MTConnect Agent* that **MAY** be published in a *Response Document*. These are:

- *Equipment Metadata* defines the *Structural Elements* that represent the physical and logical parts and sub-parts of each piece of equipment that can publish data to the *Agent*, the relationships between those parts and sub-parts, and the *Data Entities* associated with each of those *Structural Elements*. This *Equipment Metadata* is provided in an *MTConnectDevices Response Document*. See *Part 2, Devices Information Model* for more information on *Equipment Metadata*.

- *Streaming Data* provides the values published by pieces of equipment for the *Data Entities* defined by the *Equipment Metadata*. *Streaming Data* is provided in an *MTConnectStreams Response Document*. See *Part 2, Streams Information Model* for more information on *Streaming Data*.

- *MTConnect Assets* represent information used in a manufacturing operation that is commonly shared amongst multiple pieces of equipment and/or software applications. *MTConnect Assets* are provided in an *MTConnectAssets Response Document*. See *Part 4, Assets Information Model* for more information on *MTConnect Assets*.

The exchange between an *MTConnect Agent* and a client software application is a *Request* and *Response* information exchange mechanism. See *Section 5.4* for details on this *Request/Response* information exchange mechanism.

5.1.1 Instance of an *MTConnect Agent*

As described above, an *MTConnect Agent* collects and organizes values published by pieces of equipment. As with any piece of software, an *MTConnect Agent* may be periodically restarted. When an *MTConnect Agent* restarts, it **MUST** indicate to client software applications whether the information available in the *buffer* represents a completely new set of data or if the *buffer* includes data that had been collected prior to the restart of the *Agent*.

Any time an *MTConnect Agent* is restarted and begins to collect a completely new set of *Streaming Data*, that set of data is referred to as an *instance* of the *Agent*. The *MTConnect Agent* **MUST** maintain a piece of information called `instanceId` that represents the specific *instance* of the *Agent.*

`instanceId` is represented by a 64-bit integer. The `instanceId` **MAY** be implemented using any mechanism that will guarantee that the value for `instanceId` will be unique each time the *MTConnect Agent* begins collecting a new set of data.

When an *MTConnect Agent* is restarted and it provides a method to recover all, or some portion, of the data that was stored in the *buffer* before it stopped operating, the *Agent* **MUST** use the same `instanceId` that was defined prior to the restart.

5.1.2 Storage of *Equipment Metadata* for a Piece of Equipment

An *MTConnect Agent* **MUST** be capable of publishing *Equipment Metadata* for each piece of equipment that publishes information through the *Agent*. *Equipment Metadata* is typically a static file defining the *Structural Elements* associated with each piece of equipment reporting information through the *Agen*t and the *Data Entities* that can be associated with each of these *Structural Elements*. See details on *Structural Elements* and *Data Entities* in *Part 2 - Devices Information Model*.

The MTConnect Standard does not define the mechanism to be used by an *MTConnect Agent* to acquire, maintain, or store the *Equipment Metadata*. This mechanism **MUST** be defined as part of the implementation of a specific *MTConnect Agent*.

5.1.3 Storage of *Streaming Data*

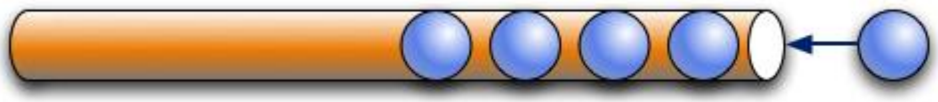
Streaming Data that is published from a piece(s) of equipment to an *MTConnect Agent* is stored by the *Agent* based upon the sequence upon which each piece of data is received. As described below, the order in which data is stored by the *Agent* is one of the factors that determines the data that may be included in a specific *MTConnectStreams Response Document*.

5.1.3.1 Management of *Streaming Data* Storage

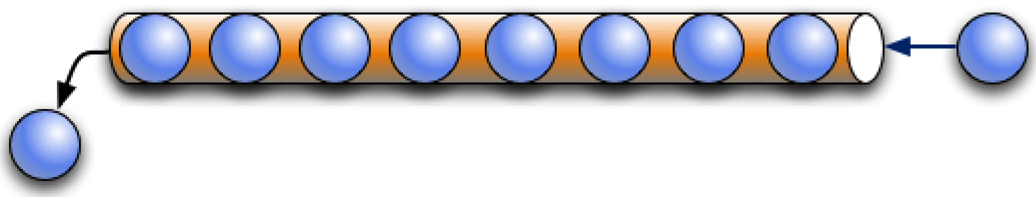
An *MTConnect Agent* stores a fixed amount of data. The amount of data stored by an *Agent* is dependent upon the implementation of a specific *MTConnect Agent*. The examples below demonstrate how discrete pieces of data received from pieces of equipment are stored.

The method for storing *Streaming Data* in an *MTConnect Agent* can be thought of as a tube that can hold a finite set of balls. Each ball represents the occurrence of a *Data Entity* published by a piece of equipment. This data is pushed in one end of the tube until there is no more room for additional balls. At that point, any new data inserted will push the oldest data out the back of the tube. The data in the tube will continue to shift in this manner as new data is received.

This tube is referred to as a *buffer* in an *MTConnect Agent*.



In the example below, the maximum number of *Data Entities* that can be stored in the *buffer* of the *MTConnect Agent* is 8. The maximum number of *Data Entities* that can be stored in the *buffer* is represented by a value called `bufferSize`. This example illustrates that when the *buffer* fills up, the oldest piece of data falls out the other end.



This process constrains the memory storage requirements for an *MTConnect Agent* to a fixed maximum size since the MTConnect Standard only requires an *Agent* to store a finite number of pieces of data.

As an implementation guideline, the *buffer* **SHOULD** be sized large enough to provide storage for a reasonable amount of information received from all pieces of equipment that are publishing information to that *MTConnect Agent*. The implementer should also consider the impact of a temporary loss of communications between a client software application and an *MTConnect Agent* when determining the size for the buffer. A larger buffer will allow a client software application more time to reconnect to an *Agent* without losing data.

5.1.3.2 *Sequence Numbers*

In an *MTConnect Agent,* each occurrence of a *Data Entity* in the *buffer* will be assigned a monotonically increasing *sequence number* as it is inserted into the *buffer*. The *sequence number* is a 64-bit integer and the values assigned as *sequence numbers* will never wrap around or be exhausted; at least within the next 100,000 years based on the size of a 64-bit number.

Sequence number is the primary key identifier used to manage and locate a specific piece of data in an *MTConnect Agent*. The *sequence number* associated with each *Data Entity* reported by an *MTConnect Agent* is identified with an attribute called `sequence`.

The *sequence number* for each piece of data **MUST** be unique for an *instance* of an *MTConnect Agent* (see *Section 5.1.1* for information on *instances* of an *MTConnect Agent*). If data is received from more than one piece of equipment, the sequence numbers are based on the order in which the data is received regardless of which piece of equipment produced that data. The *sequence number* **MUST** be a monotonically increasing number that spans all pieces of equipment publishing data to an *Agent.* This allows for multiple pieces of equipment to publish data through a single *MTConnect Agent* with no *sequence number* collisions and unnecessary protocol complexity.

The *sequence number* **MUST** be reset to one (1) each time an *MTConnect Agent* is restarted and begins to collect a fresh set of data; i.e., each time `instanceId` is changed.

The following example demonstrates the relationship between `instanceId` and `sequence` when an *MTConnect Agent* stops and restarts and begins collecting a new set of data. In this case, the `instanceId` is changed to a new value and value for `sequence` resets to one (1):

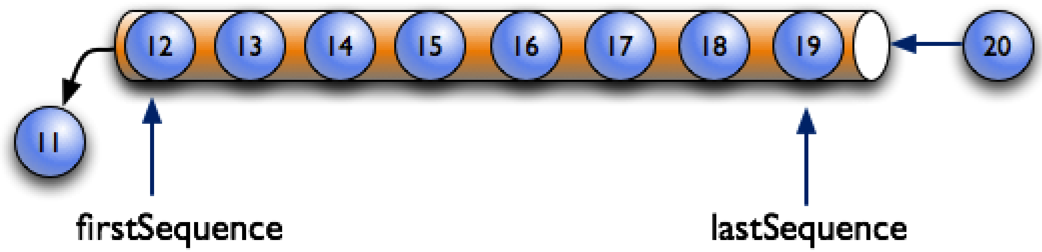
instanceId	**sequence**
234556	**234**
	235
	236
	237
	238
Agent Stops and Restarts	
234557	**1**
	2
	3
	4
	5

Figure 3: `instanceId` and `sequence`

The example below also shows two additional pieces of information defined for an *MTConnect Agent*:

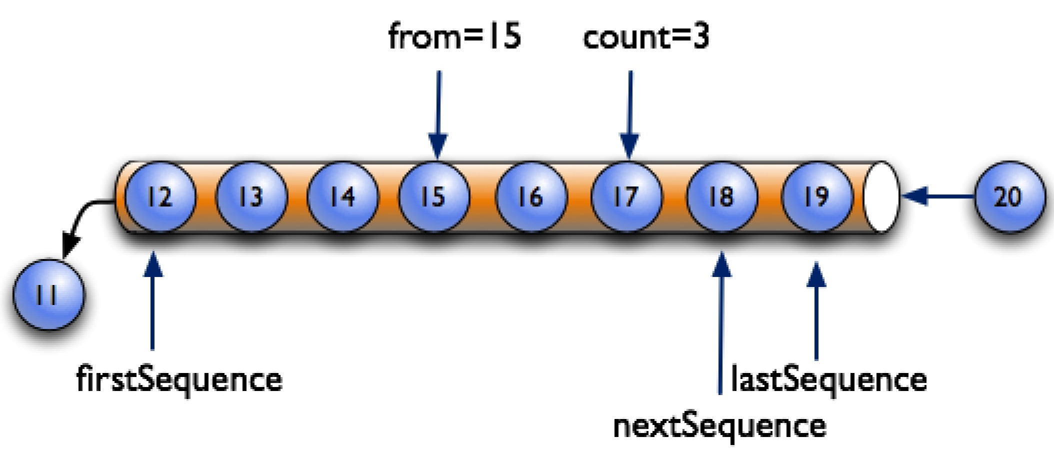
- `firstSequence` – the oldest piece of data contained in the *buffer*; i.e., the next piece of data to be moved out of the *buffer*
- `lastSequence` – the newest data added to the *buffer*

`firstSequence` and `lastSequence` provide guidance to a software application identifying the range of data available that may be requested from an *MTConnect Agent*.



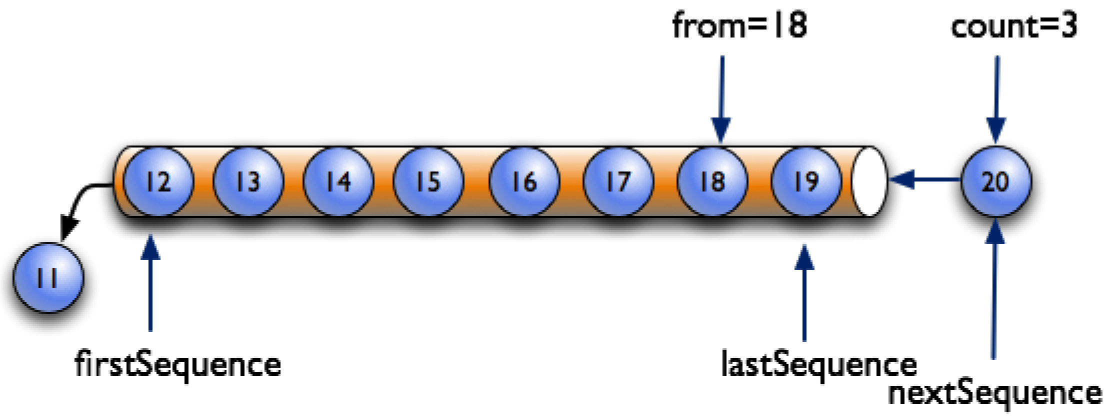
When a client software application requests data from an *MTConnect Agent*, it can specify both the *sequence number* of the first piece of data (`from`) that **MUST** be included in the Response Document and the total number (`count`) of pieces of data that **SHOULD** be included in that document.

In the example below, the request specifies that the data to be returned starts at *sequence number* 15 (`from`) and includes a total of three items (`count`).



Once a *Response* to a *Request* has been completed, the value of `nextSequence` will be established. `nextSequence` is the *sequence number* of the next piece of data available in the *buffer*. In the above example, the next *sequence number* (`nextSequence`) will be 18.

As shown in the example below, the combination of `from` and `count` defined by the *Request* indicates a *sequence number* for data that is beyond that which is currently in the *buffer*. In this case, `nextSequence` is set to a value of `lastSequence` + 1.



5.1.3.3 *Buffer* Data Structure

The information in the *buffer* of an *MTConnect Agent* can be thought of as a four-column table of data. Each column in the table represents:

- The first column is the *sequence number* associated with each *Data Entity* - `sequence`.
- The second column is the time that the data was published by a piece of equipment. This time is defined as the `timestamp` associated with that *Data Entity*. See *Section 5.1.3.4* for details on `timestamp`.
- The third column, `dataItemId`, refers to the identity of Data Entities as they will appear in the `MTConnectStreams` *Response Document*. See *Section 5* of *Part 3.0 – Streams Information Model* for details on `dataItemId` for a *Data Entity* and how that identify relates to the `id` attribute of the corresponding *Data Entity* in the *Devices Information Model*.
- The fourth column is the value associated with each Data Entity.

The following is an example demonstrating the concept of how data may be stored in an *MTConnect Agent*:

AGENT

Seq	Time	dataItemId	Value
101	2016-12-13T09:44:00.2221	AVAIL-28277	UNAVAILABLE
102	2016-12-13T09:54:00.3839	AVAIL-28277	AVAILABLE
103	2016-12-13T10:00:00.0594	POS-Y-28277	25.348
104	2016-12-13T10:00:00.0594	POS-Z-28277	13.23
105	2016-12-13T10:00:03.2839	SS-28277	0
106	2016-12-13T10:00:03.2839	POS-X-73746	11.195
107	2016-12-13T10:00:03.2839	POS-Y-73746	24.938
108	2016-12-13T10:01:37.8594	POS-Z-73746	1.143
109	2016-12-13T10:02:03.2617	SS-28277	1002

Figure 4: Data Storage Concept

The storage mechanism for the data, the internal representation of the data, and the implementation of the *MTConnect Agent* itself is not part of the MTConnect Standard. The implementer can choose both the amount of data to be stored in the *Agent* and the mechanism for how the data is stored. The only requirement is that an *MTConnect Agent* publish the *Response Documents* in the required format.

5.1.3.4 Time Stamp

Each piece of equipment that publishes information to an *MTConnect Agent* **SHOULD** provide a time stamp indicating when each piece of information was measured or determined. If no time stamp is provided, the *Agent* **MUST** provide a time stamp for the information based upon when that information was received at the *Agent*.

The timestamp associated with each piece of information is reported by an *MTConnect Agent* as `timestamp`. `timestamp` **MUST** be reported in UTC (Coordinated Universal Time) format; e.g., "2010-04-01T21:22:43Z".

Note: Z refers to UTC/GMT time, not local time.

Client software applications should use the value of `timestamp` reported for each piece of information as the means for ordering when pieces of information were generated as opposed to using `sequence` for this purpose.

> Note: It is assumed that `timestamp` provides the best available estimate of the time that the value(s) for the published information was measured or determined.

If two pieces of information are measured or determined at the exact same time, they **MUST** be reported with the same value for `timestamp`. Likewise, all information that is recorded in the *buffer* with the same value for `timestamp` should be interpreted as having been recorded at the same point in time; even if that data was published by more than one piece of equipment.

5.1.3.5 Recording Occurrences of Streaming Data

An *MTConnect Agent* **MUST** record data in the *buffer* each time the value for that specific piece of data changes. If a piece of equipment publishes multiple occurrences of a piece of data with the same value, the *Agent* **MUST NOT** record multiple occurrence for that Data Entity.

> Note: There is one exception to this rule. Some Data Entities may be defined with a `representation` attribute value of `DISCRETE` (See *Section 7.2.2.12* of *Part 2.0 Devices Information Model* for details on `representation`.) In this case, each occurrence of the data represents a new and unique piece of information. The *MTConnect Agent* **MUST** then record each occurrence of the *Data Entity* that is published by a piece of equipment.

The value for each piece of information reported by an *MTConnect Agent* must be considered by a client software application to be valid until such a time that another occurrence of that piece of information is published by the *Agent*.

5.1.3.6 Maintaining Last Value for Data Entities

An *MTConnect Agent* **MUST** retain a copy of the last available value associated with each *Data Entity* known to the *Agent*; even if an occurrence of that *Data Entity* is no longer in the *buffer*. This function allows an *MTConnect Agent* to provide a software application a view of the last known value for each *Data Entity* associated with a piece of equipment.

The *MTConnect Agent* **MUST** also retain a copy of the last value associated with each *Data Entity* that has flowed out of the *buffer*. This function allows an *MTConnect Agent* to provide a software application a view of the last known value for each *Data Entity* associated with a *Current Request* with an `@` parameter in the `query` portion of its *HTTP Request Line* (See *Section 8.3.2* for details on *Current Request*).

5.1.3.7 Unavailability of Data

An *MTConnect Agent* **MUST** maintain a list of *Data Entities* that **MAY** be published by each piece of equipment providing information to the *Agent*. This list of *Data Entities* is derived from the *Equipment Metadata* stored in the *Agent* for each piece of equipment.

Each time an *MTConnect Agent* is restarted, the *Agent* **MUST** place an occurrence of every *Data Entity* in the *buffer*. The value reported for each of these *Data Entities* **MUST** be set to `UNAVAILABLE` and the `timestamp` for each **MUST** be set to the time that the last piece of data was collected by the *Agent* prior to the.

If at any time an *MTConnect Agent* loses communications with a piece of equipment, or the *Agent* is unable to determine a valid value for all, or any portion, of the *Data Entities* published by a piece of equipment, the *Agent* **MUST** place an occurrence of each of these *Data Entities* in the *buffer* with its value set to `UNAVAILABLE`. This signifies that the value is currently indeterminate and no assumptions of a valid value for the data is possible.

Since an *MTConnect Agent* may receive information from multiple pieces of equipment, it **MUST** consider the validity of the data from each of these pieces of equipment independently.

There is one exception to the rules above. Any *Data Entity* that is constrained to a constant data value **MUST** be reported with the constant value and the *MTConnect Agent* **MUST NOT** set the value of that *Data Entity* to `UNAVAILABLE`.

> Note: The schema for the *Devices Information Model* (defined in *Part 2.0 - Devices Information Model*) defines how the value reported for an individual piece of data may be constrained to one or more specific values.

5.1.3.8 Data Persistence and Recovery

The implementer of an *MTConnect Agent* must decide on a strategy regarding the storage of *Streaming Data* in the *buffer* of the *Agent*.

In the simplest form, an *MTConnect Agent* can hold the *buffer* information in volatile memory where no data is persisted when the *Agent* is stopped. In this case, the *Agent* **MUST** update the value for `instanceId` when the *Agent* restarts to indicate that the *Agent* has begun to collect a new set of data.

If the implementation of an *MTConnect Agent* provides a method of persisting and restoring all or a portion of the information in the *buffer* of the *Agent* (*sequence numbers*, *timestamps*, identify, and values), the *Agent* **MUST NOT** change the value of the `instanceId` when the *Agent* restarts. This will indicate to a client software application that it does not need to reset the value for `nextSequence` when it requests the next set of data from the *Agent*.

When an implementer chooses to provide a method to persist the information in an *MTConnect Agent*, they may choose to store as much data as is practical in a recoverable storage system. Such a method may also include the ability to store historical information that has previously been pushed out of the *buffer*.

5.1.3.9 Heartbeat

An *MTConnect Agent* **MUST** provide a function that indicates to a client application that the HTTP connection is still viable during times when there is no new data available to report in a *Response Document*. This function is defined as *heartbeat*.

Heartbeat represents the amount of time after a *Response Document* has been published until a new *Response Document* **MUST** be published, even when no new data is available.

See *Section 8.3.2.2* for more details on configuring the *heartbeat* function.

5.1.4 Storage *of* Documents for *MTConnect Assets*

An *MTConnect Agent* also stores information associated with *MTConnect Assets*.

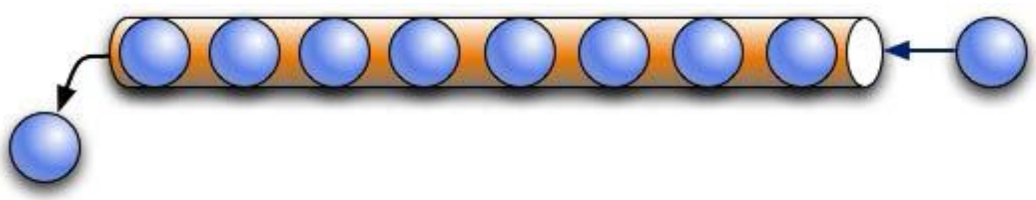
When a piece of equipment publishes a document that represents information associated with an *MTConnect Asset*, an *MTConnect Agent* stores that document in a *buffer*. This *buffer* is called the *assets buffer*. The document is called an *Asset Document*.

The *assets buffer* **MUST** be a separate *buffer* from the one where the *Streaming Data* is stored.

The *Assets Document* that is published by the piece of equipment **MUST** be organized based upon one of the applicable *Asset Information Models* defined in one of the *Parts 4.x* of the MTConnect Standard.

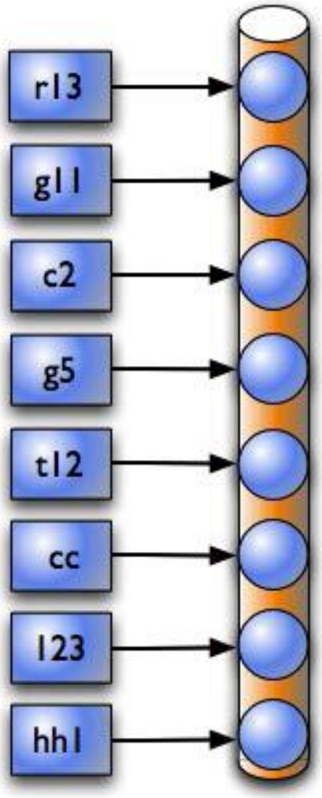
An *MTConnect Agent* will only retain a limited number of *Asset Documents* in the *assets buffer*. The *assets buffer* functions similar to the *buffer* for *Streaming Data*; i.e., when the *assets buffer* is full, the oldest *Assets Document* is pushed from the *buffer*.

The figure below demonstrates the oldest *Assets Document* being pushed from the *assets buffer* when a new *Assets Document* is added and the *assets buffer* is full:



Within an *MTConnect Agent*, the management *of Asset Documents* behave like a key/value storage in a database. In the case of *MTConnect Assets*, the key is an identifier for an *Asset* (see details on `assetId` in *Part 4.0 - Assets Information Model*) and the value is the *Asset Document* that was published by the piece of equipment.

The figure below demonstrates the relationship between the key (`assetId`) and the stored *Asset Documents*:



> Note: The key (`assetId`) is independent of the `order` of the *Asset Documents* stored in the *assets buffer*.

When an *MTConnect Agent* receives a new *Asset Document* representing an *MTConnect Asset*, it must determine whether this document represents an *MTConnect Asset* that is not currently represented in the *assets buffer* or if the document represents new information for an *MTConnect Asset* that is already represented in the *assets buffer*. When a new *Asset Document* is received, one of the following **MUST** occur:

- If the *Asset Document* represents an *MTConnect Asset* that is not currently represented in the assets *buffer*, the *Agent* **MUST** add the new document to the front of the *assets buffer*. If the *assets buffer* is full, the oldest *Asset Document* will be removed from the *assets buffer*.

- If the *Asset Document* represents an *MTConnect Asset* that is already represented in the *assets buffer,* the *Agent* **MUST** remove the existing *Assets Document* representing that *MTConnect Asset* from the *assets buffer* and add the new *Assets Document* to the front of the *assets buffer*.

The MTConnect Standard does not specify the maximum number of *Asset Documents* that may be stored in the *assets buffer*; that limit is determined by the implementation of a specific *MTConnect Agent*. The number of *Asset Documents* that may be stored in an *MTConnect Agent* is defined by the value for `assetBufferSize` (See *Section 6.5, Document Header* for more information on `assetBufferSize`.). A value of 4,294,967,296 or 2^{32} can be provided for `assetBufferSize` to indicate unlimited storage.

There is no requirement for an *MTConnect Agent* to provide persistence for the *Asset Documents* stored in the *assets buffer*. If an *MTConnect Agent* should fail, all *Asset Documents* stored in the *assets buffer* **MAY** be lost. It is the responsibility of the implementer to determine if *Asset Documents* stored in an *MTConnect Agent* may be restored or if those *Asset Documents* are retained by some other software application.

Additional details on how an *MTConnect Agent* organizes and manages information associated with *MTConnect Assets* are provided in *Part 4.0 – Assets Information Model*.

5.2 *Response Documents*

Response Documents are electronic documents generated and published by an *MTConnect Agent* in response to a *Request* for data.

The *Response Documents* defined in the MTConnect Standard are:

- *MTConnectDevices*: An electronic document that contains the information published by an *MTConnect Agent* describing the data that can be published by one or more piece(s) of equipment. The structure of the *MTConnectDevices* document is based upon the requirements defined by the *Devices Information Model*. See *Part 2.0 – Devices Information Model* for details on this information model.

- *MTConnectStreams*: An electronic document that contains the information published by an *MTConnect Agent* that contains the data that is published by one or more piece(s) of equipment. The structure of the *MTConnectStreams* document is based upon the requirements defined by the *Streams Information Model*. See *Part 3.0 – Streams Information Model* for details on this information model.

- *MTConnectAssets*: An electronic document that contains the information published by an *MTConnect Agent* that **MAY** include one or more *Asset Documents*. The structure of the *MTConnectAssets* document is based upon the requirements defined by the *Assets Information Model*. See *Part 4.0 – Assets Information Model* for details on this information model.

- *MTConnectError*: An electronic document that contains the information provided by an *MTConnect Agent* when an error has occurred when trying to respond to a *Request* for data. The structure of the *MTConnectError* document is based upon the requirements defined by the *Errors Information Model*. See *Section 9* of this document for details on this information model.

Response Documents may be represented by any document format supported by an *MTConnect Agent*. No matter what document format is used to structure these documents, the requirements for representing the data and other information contained in those documents **MUST** adhere to the requirements defined in the *Information Models* associated with each document.

5.2.1 XML Documents

XML is currently the only document format supported by the MTConnect Standard for encoding *Response Documents*. Other document formats may be supported in the future.

Since XML is the document format supported by the MTConnect Standard for encoding documents, all examples demonstrating the structure of the *Response Documents* provided throughout the MTConnect Standard are based on XML. These documents will be referred to as *MTConnect XML Documents* or *XML Documents*.

Section 6, XML Representation of Response Documents defines how each document is structured as an XML document.

5.3 *Semantic Data Models*

A *semantic data model* is a software engineering method for representing data where the context and the meaning of the data is constrained and fully defined.

Each of the *semantic data models* defined by the MTConnect Standard include:

- The types of information that may be published by a piece of equipment,
- The meaning of that information and units of measure, if applicable,
- Structural information that defines how different pieces of information relate to each other, and
- Structural information that defines how the information relates to where the information was measured or generated by the piece of equipment.

As described previously, the content of the *Response Documents* provided by an *MTConnect Agent* are each defined by a specific *semantic data model*. The details for the *semantic data model* used to define each of the *Response Documents* are detail as follows:

- MTConnectDevices: *Part 2.0 - Devices Information Model*.
- MTConnectStreams: *Part 3.0 - Streams Information Model*.
- MTConnectAssets: *Part 4.0 - Assets Information Model* and its sub-Parts.
- MTConnectError: *Part 1.0 - Overview and Fundamentals*, *Section 9, Errors Information Model*.

Without semantics, a single piece of data does not convey any relevant meaning to a person or a client software application. However, when that piece of data is paired with some semantic context, the data inherits significantly more meaning. The data can then be more completely interpreted by a client software application without human intervention.

The MTConnect *semantic data models* allows the information published by a piece of equipment to be transmitted to client software application with a full definition of the meaning of that information and in full context defining how that information relates to the piece of equipment that measured or generated the information.

5.4 *Request/Response* Information Exchange

The transfer of information between an *MTConnect Agent* and a client software application is based on a *Request/Response* information exchange approach. A client software application requests specific information from an *MTConnect Agent*. An *MTConnect Agent* responds to the *Request* by publishing a *Response Document*.

In normal operation, there are four types of *MTConnect Requests* that can be issued by a client software application that will result in different *Responses* by an *MTConnect Agent*. These *Requests* are:

- *Probe Request*– A client software application requests the *Equipment Metadata* for each piece of equipment that **MAY** publish information through an *MTConnect Agent*. The *Agent* publishes a *MTConnectDevices Response Document* that contains the requested information. A *Probe Request* is represented by the term `probe` in a *Request* from a client software application.

- *Current Request* – A client software application requests the current value for each of the data types that have been published from a piece(s) of equipment to an *MTConnect Agent*. The *Agent* publishes a *MTConnectStreams Response Document* that contains the requested information. A *Current Request* is represented by the term `current` in a *Request* from a client software application.

- *Sample Request* – A client software application requests a series of data values from the *buffer* in an *MTConnect Agent* by specifying a range of *sequence numbers* representing that data. The *Agent* publishes a *MTConnectStreams Response Document* that contains the requested information. A *Sample Request* is represented by the term `sample` in a *Request* from a client software application.

- *Asset Request* – A client software application requests information related to *MTConnect Assets* that has been published to an *MTConnect Agent*. The *Agent* publishes an *MTConnectAssets Response Document* that contains the requested information. An *Asset Request* is represented by the term `asset` in a *Request* from a client software application.

Note: If an *MTConnect Agent* is unable to respond to the request for information or the request includes invalid information, the *Agent* will publish an *MTConnectError Response Document*. See *Section 9* for information regarding *MTConnect Error Information Model*.

The specific format for the *Request* for information from an *MTConnect Agent* will depend on the *Protocol* implemented as part of the *Request/Response Information Exchange* mechanism deployed in a specific implementation. See *Section 7, Protocol* for details on implementing the *Request/Response Information Exchange*.

Also, the specific format for the *Response Documents* may also be implementation dependent. See *Section 6, XML Representation of Response Document Structure* for details on the format for the *Response Documents* encoded with XML.

5.5 Accessing Information from an *MTConnect Agent*

Each of the *Requests* defined for the *Request/Response Information Exchange* requires an *MTConnect Agent* to respond with a specific view of the information stored by the *Agent*. The following describes the relationships between the information stored by an *Agent* and the contents of the *Response Documents*.

5.5.1 Accessing *Equipment Metadata* from an *MTConnect Agent*

The *Equipment Metadata* associated with each piece of equipment that publishes information to an *MTConnect Agent* is typically static information that is maintained by the *Agent*. The MTConnect Standard does not define how the *Agent* captures or maintains that information. The only requirement that the MTConnect Standard places on an *MTConnect Agent* regarding this *Equipment Metadata* is that the *Agent* properly store this information and then configure and publish a *MTConnectDevices Response Document* in response to a *Probe Request*.

All issues associated with the capture and maintenance of the *Equipment Metadata* is the responsibility of the implementer of a specific *MTConnect Agent*.

5.5.2 Accessing *Streaming Data* from the *Buffer* of an *MTConnect Agent*

There are two *Requests* defined for the *Request/Response Information Exchange* that require an *MTConnect Agent* to provide different views of the information stored in the buffer of the *Agent*. These *Requests* are `current` and `sample`.

The example below demonstrates how an *MTConnect Agent* interprets the information stored in the *buffer* to provide the content that is published in different versions of the *MTConnectStreams Response Document* based on the specific *Request* that is issued by a client software application.

For this example, we are demonstrating an *MTConnect Agent* with a *buffer* that can hold up to eight (8) *Date Entities*; i.e., the value for `bufferSize` is 8. This *Agent* is collecting information for two pieces of data – `Pos` representing a position and `Line` representing a line of logic or commands in a control program.

In this *buffer*, the value for `firstSequence` is 12 and the value for `lastSequence` is 19. There are five (5) different values for `Pos` and three (3) different values for `Line`.

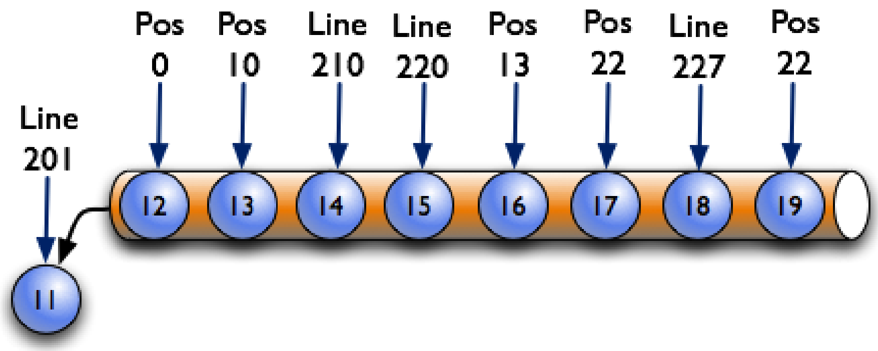


Figure 5: Example *Buffer*

If an *MTConnect Agent* receives a *Sample Request* from a client software application, the *Agent* **MUST** publish an *MTConnectStreams Response Document* that contains a range of data values. The range of values are defined by the `from` and `count` parameters that must be included as part of the *Sample Request*. If the value of `from` is 14 and the value of `count` is 5, the *Agent* **MUST** publish an *MTConnectStreams Response Document* that includes five (5) pieces of data represented by *sequence numbers* 14, 15, 16, 17, and 18 – three (3) occurrences of `Line` and two (2) occurrences of `Pos`. In this case, `nextSequence` will also be returned with a value of 19.

Likewise, if the same *MTConnect Agent* receives a *Current Request* from a client software application, the *Agent* **MUST** publish an *MTConnectStreams Response Document* that contains the most current information available for each of the types of data that is being published to the *Agent*. In this case, the specific data that **MUST** be represented in the *MTConnectStreams Response Document* is `Pos` with a value of 22 and a *sequence number* of 19 and `Line` with a value of 227 and a *sequence number* of 18.

There is also a derivation of the *Current Request* that will cause an *Agent* to publish an *MTConnectStreams Response Document* that contains a set of data relative to a specific *sequence number*. The *Current Request* **MAY** include an additional parameter called `at`. When the `at` parameter, along with an `instanceId`, is included as part of a *Current Request*, an *MTConnect Agent* **MUST** publish an *MTConnectStreams Response Document* that contains the most current information available for each of the types of *Data Entities* that are being published to the *Agent* that occur immediately at or before the *sequence number* specified with the `at` parameter.

For example, if the *Request* is `current?at=15`, an *MTConnect Agent* **MUST** publish a *MTConnectStreams Response Document* that contains the most current information available for each of the *Data Entities* that are stored in the *buffer* of the *Agent* with a *sequence number* of 15 or lower. In this case, the specific data that **MUST** be represented in the *MTConnectStreams Response Document* is `Pos` with a value of 10 and a *sequence number* of 13 and `Line` with a value of 220 and a *sequence number* of 15.

If a `current` *Request* is received for a *sequence number* of 11 or lower, an *MTConnect Agent* **MUST** return an `OUT_OF_RANGE` *MTConnectError Response Document*. The same *HTTP Error Message* **MUST** be given if a *sequence number* is requested that is greater than the end of the *buffer*. See *Section 9* for more information on *MTConnect Error Response Document*.

5.5.3 Accessing MTConnect Assets Information from an MTConnect Agent

When an *MTConnect Agent* receives an A*sset Request*, the *Agent* **MUST** publish an `MTConnectAssets` document that contains information regarding the *Asset Documents* that are stored in the *Agent*.

See *Part 4.0 - Assets Information Model* for details on *MTConnect Assets*, *Asset Requests*, and the *MTConnectAssets Response Document*.

6 XML Representation of *Response Documents*

As defined in *Section 5.2.1*, XML is currently the only language supported by the MTConnect® Standard for encoding *Response Documents*.

Response Documents must be valid and conform to the *schema* defined in the *semantic data model* defined for that document. The schema for each *Response Document* **MUST** be updated to correlate to a specific version of the MTConnect Standard. Versions, within a *major version*, of the MTConnect Standard will be defined in such a way to best maintain backwards compatibility of the *semantic data models* through all *minor* revisions of the Standard. However, new *minor* versions may introduce extensions or enhancements to existing *semantic data models*.

To be valid, a *Response Document* must be well-formed; meaning that, amongst other things, each element has the required XML *start-tag* and *end-tag* and that the document does not contain any illegal characters. The validation of the document may also include a determination that required elements and attributes are present, they only occur in the appropriate location in the document, and they appear only the correct number of times. If the document is not well-formed, it may be rejected by a client software application. The *semantic data model* defined for each *Response Document* also specifies the elements and *Child Elements* that may appear in a document. XML elements may contain *Child Elements*, CDATA, or both. The *semantic data model* also defines the number of times each element and *Child Element* may appear in the document.

Each *Response Document* encoded using XML consists of the following primary sections:

- XML Declaration
- Root Element
- Schema and Namespace Declaration
- Document Header
- Document Body

The following will provide details defining how each of the *Response Documents* are encoded using XML.

Note: See *Section 3, Terminology* for the definition of XML related terms used in the MTConnect Standard.

6.1 Fundamentals of Using XML to Encode *Response Documents*

The MTConnect Standard follows industry conventions for formatting the elements and attributes included in an XML document. The general guidelines are as follows:

- All element names **MUST** be specified in Pascal case (first letter of each word is capitalized). For example: `<PowerSupply/>`.

- The name for an attribute **MUST** be Camel case; similar to Pascal case, but the first letter will be lower case. For example: `<MyElement nativeName="bob"/>` where `MyElement` is the *Element Name* and `nativeName` is an attribute.

- All CDATA values that are defined with a limited or controlled vocabulary **MUST** be in upper case with an _ (underscore) separating words. For example: `ON`, `OFF`, `ACTUAL`, and `COUNTER_CLOCKWISE`.

- The values provided for a date and/or a time **MUST** follow the W3C ISO 8601 format with an arbitrary number of decimals representing fractions of a second. Refer to the following specification for details on the format for dates and times: http://www.w3.org/TR/NOTE-datetime.

  The format for the value describing a date and a time will be YYYY-MM-DDThh:mm:ss.ffff. An example would be: 2017-01-13T13:01.213415Z.

  Note: Z refers to UTC/GMT time, not local time.

  The accuracy and number of decimals representing fractions of a second for a `timestamp` **MUST** be determined by the capabilities of the piece of equipment publishing information to an *MTConnect Agent*. All time values **MUST** be provided in UTC (GMT).

- XML element names **MUST** be spelled out and abbreviations are not permitted. See the exclusion below regarding the use of the suffix `Ref`.

- XML attribute names **SHOULD** be spelled out and abbreviations **SHOULD** be avoided. The exception to this rule is the use of `id` when associated with an identifier. See the exclusion below regarding the use of the suffix `Ref`.

- The abbreviation `Ref` for `Reference` is permitted as a suffix to element names of either a *Structural Element* or a *Data Entity* to provide an efficient method to associate information defined in another location in a *Data Model* without duplicating that original data or structure. See *Section 4.8 in Part 2, Devices Information Model* for more information on `Reference`.

6.2 XML Declaration

The first section of a *Response Document* encoded with XML **SHOULD** be the *XML Declaration*. The declaration is a single element.

An example of an XML Declaration would be:

```
<?xml version="1.0" encoding="UTF-8"?>
```

This element provides information regarding how the XML document is encoded and the character type used for that encoding. See the W3C website for more details on the XML declaration.

6.3 *Root Element*

Every *Response Document* **MUST** contain only one root element. The MTConnect Standard defines `MTConnectDevices`, `MTConnectStreams`, `MTConnectAssets`, and `MTConnectError` as *Root Elements*.

The *Root Element* specifies a specific *Response Document* and appears at the top of the document immediately following the *XML Declaration*.

6.3.1 `MTConnectDevices` *Root Element*

`MTConnectDevices` is the *Root XML Element* for the *MTConnectDevices Response Document*.

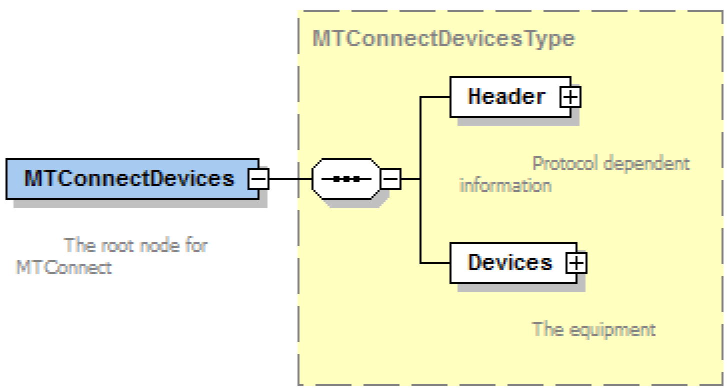


Figure 6: `MTConnectDevices` Structure

`MTConnectDevices` **MUST** contain two *Child Elements* - `Header` and `Devices`. Details for `Header` are defined in *Section 6.5, Document Header*.

`Devices` is an XML container that represents the *Document Body* for an *MTConnectDevices Response Document* – see *Section 6.6*. Details for the *semantic data model* describing the contents for `Devices` are defined in *Part 2.0 - Devices Information Model*.

MTConnectDevices also has a number of attributes. These attributes are defined in *Section 6.4, Schema and Namespace Declaration*.

6.3.1.1 MTConnectDevices Elements

An MTConnectDevices element **MUST** contain a Header and a Devices element.

Element	Description	Occurrence
Header	An XML container in an *MTConnect Response Document* that provides information from an *MTConnect Agent* defining version information, storage capacity, and parameters associated with the data management within the *Agent*.	1
Devices	The XML container in an *MTConnectDevices Response Document* that provides the *Equipment Metadata* for each of the pieces of equipment associated with an *MTConnect Agent*.	1

6.3.2 MTConnectStreams *Root Element*

MTConnectStreams is the *Root Element* for the *MTConnectStreams Response Document*.

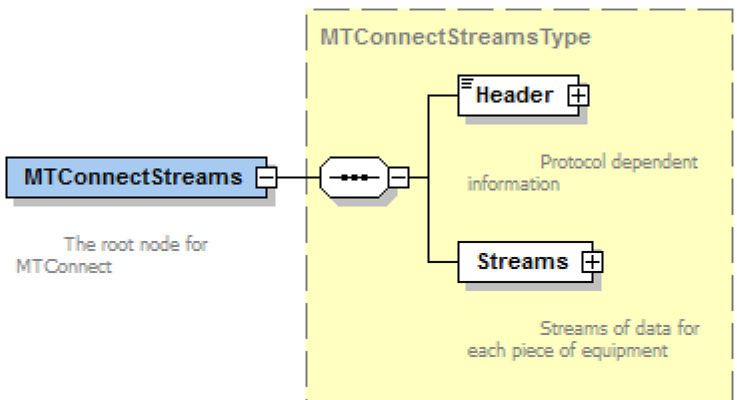


Figure 7: MTConnectStreams Structure

MTConnectStreams **MUST** contain two *Child Elements* - Header and Streams.

Details for Header are defined in *Section 6.5, Document Header*.

Streams is an XML container that represents the *Document Body* for a *MTConnectStreams Response Document* – see *Section 6.6*. Details for the *semantic data model* describing the contents for Streams are defined in *Part 3.0 - Streams Information Model*.

MTConnectStreams also has a number of attributes. These attributes are defined in *Section 6.4, Schema and Namespace Declaration*.

6.3.2.1 `MTConnectStreams` Elements

An `MTConnectStreams` element **MUST** contain a `Header` and a `Streams` element.

Element	Description	Occurrence
`Header`	An XML container in an *MTConnect Response Document* that provides information from an *MTConnect Agent* defining version information, storage capacity, and parameters associated with the data management within the *Agent*.	1
`Streams`	The XML container for the information published by an *MTConnect Agent* in a *MTConnectStreams Response Document*.	1

6.3.3 `MTConnectAssets` *Root Element*

`MTConnectAssets` is the *Root Element* for the *MTConnectAssets Response Document*.

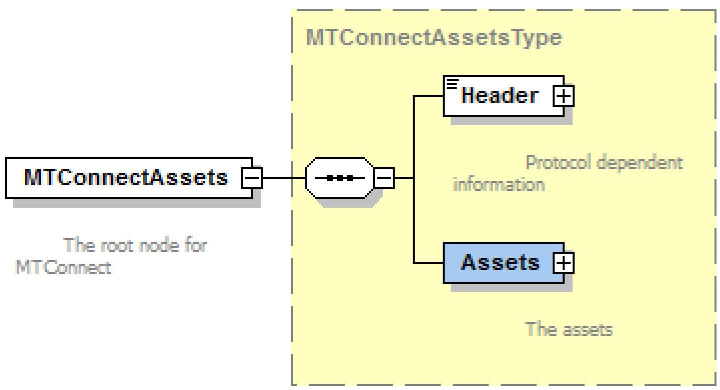


Figure 8: `MTConnectAssets` Structure

`MTConnectAssets` **MUST** contain two *Child Elements* - `Header` and `Assets`.

Details for `Header` are defined in *Section 6.5, Document Header*.

`Assets` is an XML container that represents the *Document Body* for an *MTConnectAssets Response Document* – see *Section 6.6*. Details for the *semantic data model* describing the contents for `Assets` are defined in *Part 4.0 - Assets Information Model*.

`MTConnectAssets` also has a number of attributes. These attributes are defined in *Section 6.4, Schema and Namespace Declaration*.

6.3.3.1 `MTConnectAssets` Elements

An `MTConnectAssets` element **MUST** contain a `Header` and an `Assets` element.

Element	Description	Occurrence
`Header`	An XML container in an *MTConnect Response Document* that provides information from an *MTConnect Agent* defining version information, storage capacity, and parameters associated with the data management within the *Agent*.	1
`Assets`	The XML container in an *MTConnectAssets Response Document* that provides information for *MTConnect Assets* associated with an *MTConnect Agent*.	1

6.3.4 `MTConnectError` *Root Element*

`MTConnectError` is the *Root Element* for the *MTConnectError Response Document*.

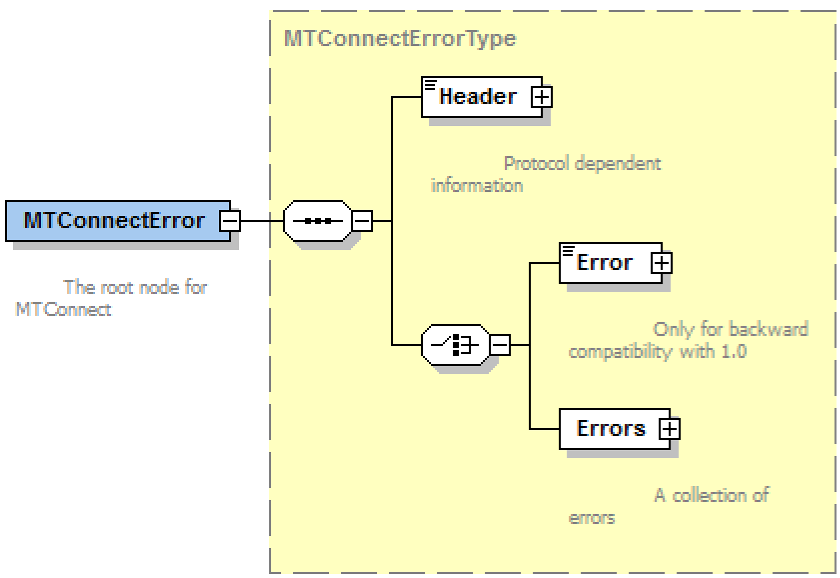


Figure 9: `MTConnectError` Structure

`MTConnectError` **MUST** contain two *Child Elements* - `Header` and `Errors`.

> Note: When compatibility with *Version 1.0.1* and earlier of the MTConnect Standard is required for an implementation, the *MTConnectErrors Response Document* contains only a single `Error` *Data Entity* and the `Errors` *Child Element* **MUST NOT** appear in the document.

Details for `Header` are defined in *Section 6.5, Document Header*.

`Errors` is an XML container that represents the *Document Body* for an *MTConnectError Response Document* – See *Section 6.6*. Details for the semantic data model describing the contents for `Errors` are defined in *Section 9, Errors Information Model*.

`MTConnectError` also has a number of attributes. These attributes are defined in *Section 6.4, Schema and Namespace Declaration*.

6.3.4.1 `MTConnectError` Elements

An `MTConnectError` element **MUST** contain a `Header` and an `Errors` element.

Element	Description	Occurrence
`Header`	An XML container in an *MTConnect Response Document* that provides information from an *MTConnect Agent* defining version information, storage capacity, and parameters associated with the data management within the *Agent*.	1
`Errors`	The XML container in an *MTConnectErrors Response Document* that provides information associated with errors encountered by an *MTConnect Agent*.	1

6.4 Schema and Namespace Declaration

XML provides standard methods for declaring the *schema* and *namespace* associated with a document encoded by XML. The declaration of the *schema* and *namespace* for *MTConnect Response Documents* **MUST** be structured as attributes in the *Root Element* of the document. XML defines these attributes as pseudo-attributes since they provide additional information for the entire document and not just specifically for the *Root Element* itself.

> Note: If a *Response Document* contains sections that utilize different schemas and/or *namespaces*, additional pseudo-attributes should appear in the document as declared using standard conventions as defined be W3C.

For further information on declarations refer to *Appendix C*.

6.5 *Document Header*

The *Document Header* is an XML container in an *MTConnect Response Document* that provides information from an *MTConnect Agent* defining version information, storage capacity, and parameters associated with the data management within the *Agent*. This XML element is called `Header`.

`Header` **MUST** be the first XML element following the *Root Element* of any *Response Document*. The `Header` XML element **MUST NOT** contain any *Child Elements*.

The content of the `Header` element will be different for each type of *Response Document*.

6.5.1 Header for `MTConnectDevices`

The `Header` element for an *MTConnectDevices Response Document* defines information regarding the creation of the document and the data storage capability of the *MTConnect Agent* that generated the document.

6.5.1.1 XML *Schema* Structure for `Header` for `MTConnectDevices`

The following XML *schema* represents the structure of the `Header` XML element that **MUST** be provided for an *MTConnectDevices Response Document*.

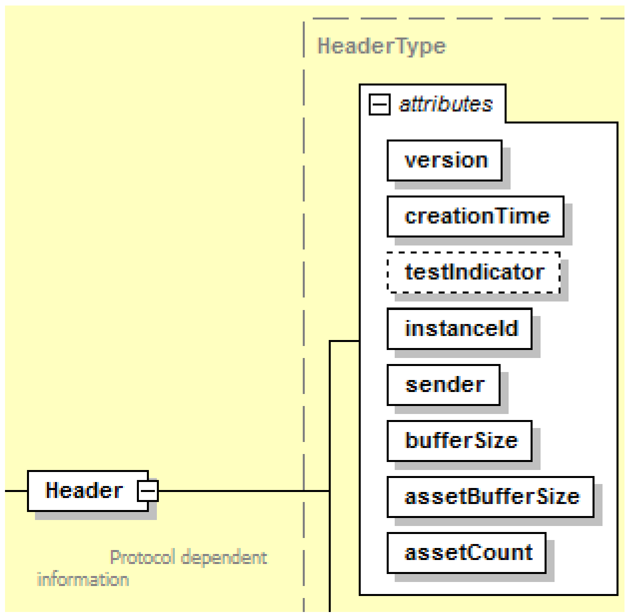


Figure 10: `Header` *Schema* Diagram for `MTConnectDevices`

6.5.1.2 Attributes for `Header` for `MTConnectDevices`

The following table defines the attributes that may be used to provide additional information in the `Header` element for an *MTConnectDevices Response Document*.

Attribute	Description	Occurrence
`version`	The *major*, *minor*, *and revision* number of the MTConnect Standard that defines the *semantic data model* that represents the content of the *Response Document*. It also includes the revision number of the *schema* associated with that specific *semantic model*. The value reported for `version` **MUST** be a series of four numeric values, separated by a decimal point, representing a *major*, *minor*, *and revision* number of the MTConnect Standard and the revision number of a specific *schema*. As an example, the value reported for `version` for a *Response Document* that was structured based on *schema* revision 10 associated with Version 1.4.0 of the MTConnect Standard would be: 1.4.0.10 `version` is a required attribute.	1
`creationTime`	`creationTime` represents the time that an *MTConnect Agent* published the *Response Document*. `creationTime` **MUST** be reported in UTC (Coordinated Universal Time) format; e.g., "2010-04-01T21:22:43Z". Note: Z refers to UTC/GMT time, not local time. `creationTime` is a required attribute.	1
`testIndicator`	A flag indicating that the *MTConnect Agent* that published the *Response Document* is operating in a test mode. The contents of the *Response Document* may not be valid and **SHOULD** be used for testing and simulation purposes only. The values reported for `testIndicator` are: - `TRUE`: The Agent is functioning in a test mode. - `FALSE`: The Agent is not function in a test mode. If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `FALSE`. `testIndicator` is an optional attribute.	0..1

Attribute	Description	Occurrence
`instanceId`	A number indicating a specific instantiation of the *buffer* associated with the *MTConnect Agent* that published the *Response Document*. The value reported for `instanceId` **MUST** be a unique unsigned 64-bit integer. The value for `instanceId` **MUST** be changed to a different unique number each time the *buffer* is cleared and a new set of data begins to be collected. `instanceId` is a required attribute.	1
`sender`	An identification defining where the *MTConnect Agent* that published the *Response Document* is installed or hosted. The value reported for `sender` **MUST** be either an IP Address or Hostname describing where the *MTConnect Agent* is installed or the URL of the *MTConnect Agent*; e.g., `http://<address>[:port]/`. Note: The port number need not be specified if it is the default HTTP port 80. `sender` is a required attribute.	1
`bufferSize`	A value representing the maximum number of *Data Entities* that **MAY** be retained in the *MTConnect Agent* that published the *Response Document* at any point in time. The value reported for `bufferSize` **MUST** be a number representing an unsigned 32-bit integer. `bufferSize` is a required attribute. Note 1: `bufferSize` represents the maximum number of *sequence numbers* that **MAY** be stored in the *MTConnect Agent*. Note 2: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `bufferSize`.	1
`assetBufferSize`	A value representing the maximum number of *Asset Documents* that can be stored in the *MTConnect Agent* that published the *Response Document*. The value reported for `assetBufferSize` **MUST** be a number representing an unsigned 32-bit integer. `assetBufferSize` is a required attribute. Note: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `assetBufferSize`.	1

Attribute	Description	Occurrence
assetCount	A number representing the current number of *Asset Documents* that are currently stored in the *MTConnect Agent* as of the creationTime that the *Agent* published the *Response Document*. The value reported for assetCount **MUST** be a number representing an unsigned 32-bit integer and **MUST NOT** be larger than the value reported for assetBufferSize. assetCount is a required attribute.	1

The following is an example of a Header XML element for an *MTConnectDevices Response Document*:

```
1.  <Header creationTime="2017-02-16T16:44:27Z" sender="MyAgent"
2.    instanceId="1268463594" bufferSize="131072"
3.    version="1.4.0.10" assetCount="54" assetBufferSize="1024"/>
```

6.5.2 Header for MTConnectStreams

The Header element for an *MTConnectStreams Response Document* defines information regarding the creation of the document and additional information necessary for an application to interact and retrieve data from the *MTConnect Agent*.

6.5.2.1 XML *Schema* Structure for `Header` for `MTConnectStreams`

The following XML *schema* represents the structure of the `Header` XML element that **MUST** be provided for an *MTConnectStreams Response Document*.

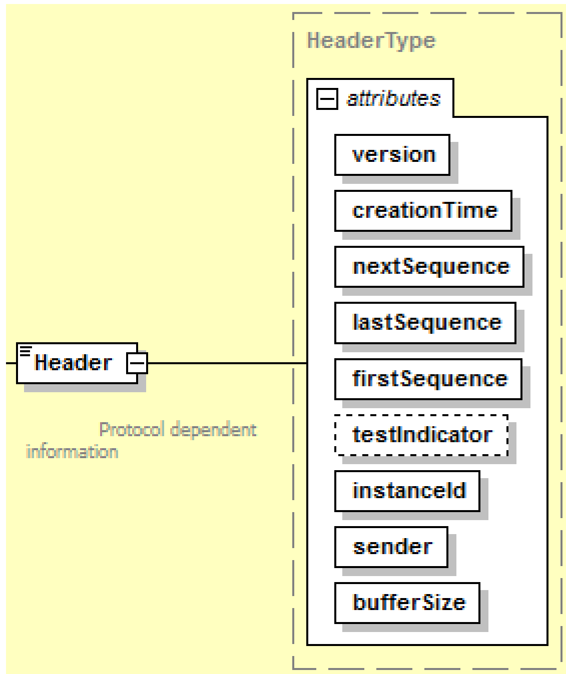


Figure 11: `Header` *Schema* Diagram for `MTConnectStreams`

6.5.2.2 Attributes for MTConnectStreams Header

The following table defines the attributes that may be used to provide additional information in the `Header` element for an *MTConnectStreams Response Document*.

Attribute	Description	Occurrence
`version`	The *major*, *minor*, *and revision* number of the MTConnect Standard that defines the *semantic data model* that represents the content of the *Response Document*. It also includes the revision number of the *schema* associated with that specific *semantic model*. The value reported for `version` **MUST** be a series of four numeric values, separated by a decimal point, representing a *major*, *minor*, *and revision* number of the MTConnect Standard and the revision number of a specific *schema*. As an example, the value reported for `version` for a *Response Document* that was structured based on *schema* revision 10 associated with Version 1.4.0 of the MTConnect Standard would be: 1.4.0.10 `version` is a required attribute.	1

Attribute	Description	Occurrence
`creationTime`	`creationTime` represents the time that an *MTConnect Agent* published the *Response Document*. `creationTime` **MUST** be reported in UTC (Coordinated Universal Time) format; e.g., "2010-04-01T21:22:43Z". Note: Z refers to UTC/GMT time, not local time. `creationTime` is a required attribute.	1
`nextSequence`	A number representing the *sequence number* of the piece of *Streaming Data* that is the next piece of data to be retrieved from the *buffer* of the *MTConnect Agent* that was not included in the *Response Document* published by the *Agent*. If the *Streaming Data* included in the *Response Document* includes the last piece of data stored in the *buffer* of the *MTConnect Agent* at the time that the document was published, then the value reported for `nextSequence` **MUST** be equal to `lastSequence` + 1. The value reported for `nextSequence` **MUST** be a number representing an unsigned 64-bit integer. `nextSequence` is a required attribute.	1
`lastSequence`	A number representing the *sequence number* assigned to the last piece of *Streaming Data* that was added to the *buffer* of the *MTConnect Agent* immediately prior to the time that the *Agent* published the *Response Document*. The value reported for `lastSequence` **MUST** be a number representing an unsigned 64-bit integer. `lastSequence` is a required attribute.	1
`firstSequence`	A number representing the *sequence number* assigned to the oldest piece of *Streaming Data* stored in the *buffer* of the *MTConnect Agent* immediately prior to the time that the *Agent* published the *Response Document*. The value reported for `firstSequence` **MUST** be a number representing an unsigned 64-bit integer. `firstSequence` is a required attribute.	1

Attribute	Description	Occurrence
`testIndicator`	A flag indicating that the *MTConnect Agent* that published the *Response Document* is operating in a test mode. The contents of the *Response Document* may not be valid and **SHOULD** be used for testing and simulation purposes only. The values reported for `testIndicator` are: - `TRUE`: The *Agent* is functioning in a test mode. - FALSE: The *Agent* is not functioning in a test mode. If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `FALSE`. `testIndicator` is an optional attribute.	0..1
`instanceId`	A number indicating a specific instantiation of the *buffer* associated with the *MTConnect Agent* that published the *Response Document*. The value reported for `instanceId` **MUST** be a unique unsigned 64-bit integer. The value for `instanceId` **MUST** be changed to a different unique number each time the *buffer* is cleared and a new set of data begins to be collected. `instanceId` is a required attribute.	1
`sender`	An identification defining where the *MTConnect Agent* that published the *Response Document* is installed or hosted. The value reported for `sender` **MUST** be either an IP Address or Hostname describing where the *MTConnect Agent* is installed or the URL of the *MTConnect Agent*; e.g., `http://<address>[:port]/`. Note: The port number need not be specified if it is the default HTTP port 80. `sender` is a required attribute.	1
`bufferSize`	A value representing the maximum number of *Data Entities* that **MAY** be retained in the *MTConnect Agent* that published the *Response Document* at any point in time. The value reported for `bufferSize` **MUST** be a number representing an unsigned 32-bit integer. `bufferSize` is a required attribute. Note 1: `bufferSize` represents the maximum number of *sequence numbers* that **MAY** be stored in the *MTConnect Agent*. Note 2: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `bufferSize`.	1

The following is an example of a `Header` XML element for an *MTConnectStreams Response Document*:

```
<Header creationTime="2017-02-16T16:44:27Z" sender="MyAgent"
  instanceId="1268463594" bufferSize="131072"
  version="1.4.0.10" assetCount="54" assetBufferSize="1024"/>
```

6.5.3 Header for MTConnectAssets

The `Header` element for an *MTConnectAssets Response Document* defines information regarding the creation of the document and the storage of *Asset Documents* in the *MTConnect Agent* that generated the document.

6.5.3.1 XML *Schema* Structure for Header for MTConnectAssets

The following XML *schema* represents the structure of the `Header` XML element that **MUST** be provided for an *MTConnectAssets Response Document*.

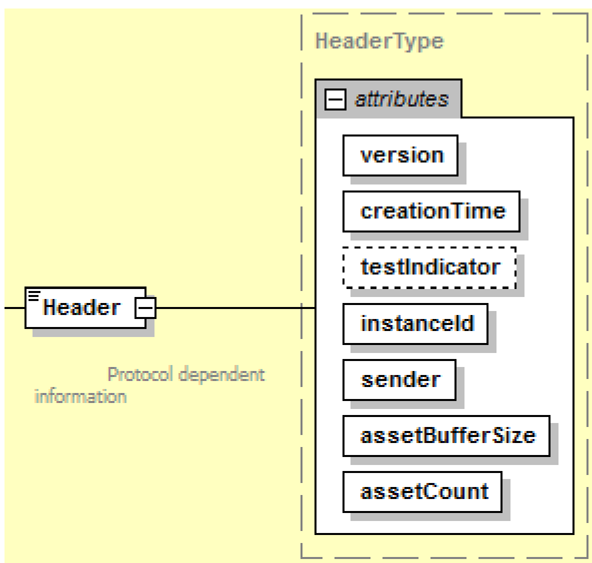


Figure 12: Header *Schema* Diagram for MTConnectAssets

6.5.3.2 `Attributes` for `Header` for `MTConnectAssets`

The following table defines the attributes that may be used to provide additional information in the `Header` element for an *MTConnectAssets Response Document*.

Attribute	Description	Occurrence
`version`	The *major*, *minor*, *and revision* number of the MTConnect Standard that defines the *semantic data model* that represents the content of the *Response Document*. It also includes the revision number of the *schema* associated with that specific *semantic model*. The value reported for `version` **MUST** be a series of four numeric values, separated by a decimal point, representing a *major*, *minor*, *and revision* number of the MTConnect Standard and the revision number of a specific *schema*. As an example, the value reported for `version` for a *Response Document* that was structured based on *schema* revision 10 associated with Version 1.4.0 of the MTConnect Standard would be: 1.4.0.10 `version` is a required attribute.	1
`creationTime`	`creationTime` represents the time that an *MTConnect Agent* published the *Response Document*. `creationTime` **MUST** be reported in UTC (Coordinated Universal Time) format; e.g., "2010-04-01T21:22:43Z". Note: Z refers to UTC/GMT time, not local time. `creationTime` is a required attribute.	1
`testIndicator`	A flag indicating that the *MTConnect Agent* that published the *Response Document* is operating in a test mode. The contents of the *Response Document* may not be valid and **SHOULD** be used for testing and simulation purposes only. The values reported for `testIndicator` are: - `TRUE`: The *Agent* is functioning in a test mode. - `FALSE`: The *Agent* is not functioning in a test mode. If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `FALSE`. `testIndicator` is an optional attribute.	0..1

Attribute	Description	Occurrence
`instanceId`	A number indicating a specific instantiation of the *buffer* associated with the *MTConnect Agent* that published the *Response Document*. The value reported for `instanceId` **MUST** be a unique unsigned 64-bit integer. The value for `instanceId` **MUST** be changed to a different unique number each time the *buffer* is cleared and a new set of data begins to be collected. `instanceId` is a required attribute.	1
`sender`	An identification defining where the *MTConnect Agent* that published the *Response Document* is installed or hosted. The value reported for `sender` **MUST** be either an IP Address or Hostname describing where the *MTConnect Agent* is installed or the URL of the *MTConnect Agent*; e.g., `http://<address>[:port]/`. Note: The port number need not be specified if it is the default HTTP port 80. `sender` is a required attribute.	1
`assetBufferSize`	A value representing the maximum number of *Asset Documents* that **MAY** be retained in the *MTConnect Agent* that published the *Response Document* at any point in time. The value reported for `bufferSize` **MUST** be a number representing an unsigned 32-bit integer. `bufferSize` is a required attribute. Note 1: `bufferSize` represents the maximum number of *sequence numbers* that **MAY** be stored in the *MTConnect Agent*. Note 2: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `bufferSize`.	1
`assetCount`	A number representing the current number of *Asset Documents* that are currently stored in the *MTConnect Agent* as of the `creationTime` that the *Agent* published the *Response Document*. The value reported for `assetCount` **MUST** be a number representing an unsigned 32-bit integer and **MUST NOT** be larger than the value reported for `assetBufferSize`. `assetCount` is a required attribute.	1

The following is an example of a `Header` XML element for an *MTConnectAssets Response Document*:

```
<Header creationTime="2017-02-16T16:44:27Z" sender="MyAgent"
  instanceId="1268463594" version="1.4.0.10" assetCount="54"
  assetBufferSize="1024"/>
```

6.5.4 `Header` for `MTConnectError`

The `Header` element for an *MTConnectError Response Document* defines information regarding the creation of the document and the data storage capability of the *MTConnect Agent* that generated the document.

6.5.4.1 XML *Schema* Structure for `Header` for `MTConnectError`

The following XML *schema* represents the structure of the `Header` XML element that **MUST** be provided for an *MTConnectError Response Document*.

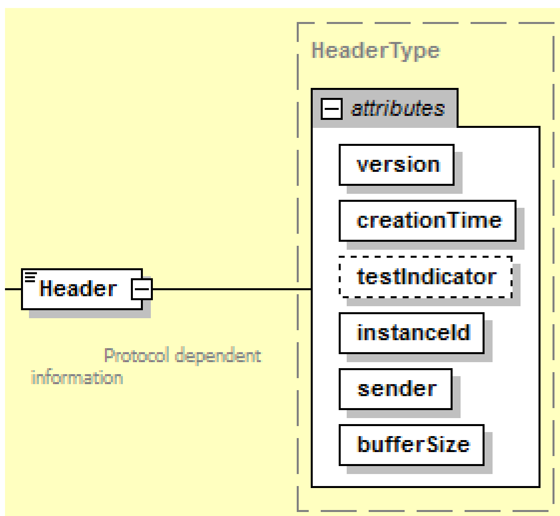


Figure 13: `Header` *Schema* Diagram for `MTConnectError`

6.5.4.2 Attributes for `Header` for `MTConnectError`

The following table defines the attributes that may be used to provide additional information in the `Header` element for an *MTConnectError Response Document*.

Attribute	Description	Occurrence
`version`	The *major*, *minor*, *and revision* number of the MTConnect Standard that defines the *semantic data model* that represents the content of the *Response Document*. It also includes the revision number of the *schema* associated with that specific *semantic model*. The value reported for `version` **MUST** be a series of four numeric values, separated by a decimal point, representing a *major*, *minor*, *and revision* number of the MTConnect Standard and the revision number of a specific *schema*. As an example, the value reported for `version` for a *Response Document* that was structured based on *schema* revision 10 associated with Version 1.4.0 of the MTConnect Standard would be: 1.4.0.10 `version` is a required attribute.	1
`creationTime`	`creationTime` represents the time that an *MTConnect Agent* published the *Response Document*. `creationTime` **MUST** be reported in UTC (Coordinated Universal Time) format; e.g., "2010-04-01T21:22:43Z". Note: Z refers to UTC/GMT time, not local time. `creationTime` is a required attribute.	1
`testIndicator`	A flag indicating that the *MTConnect Agent* that published the *Response Document* is operating in a test mode. The contents of the *Response Document* may not be valid and **SHOULD** be used for testing and simulation purposes only. The values reported for `testIndicator` are: - `TRUE`: The *Agent* is functioning in a test mode. - `FALSE`: The *Agent* is not functioning in a test mode. If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `FALSE`. `testIndicator` is an optional attribute.	0..1

instanceId	A number indicating a specific instantiation of the *buffer* associated with the *MTConnect Agent* that published the *Response Document*. The value reported for instanceId **MUST** be a unique unsigned 64-bit integer. The value for instanceId **MUST** be changed to a different unique number each time the *buffer* is cleared and a new set of data begins to be collected. instanceId is a required attribute.	1
sender	An identification defining where the *MTConnect Agent* that published the *Response Document* is installed or hosted. The value reported for sender **MUST** be either an IP Address or Hostname describing where the *MTConnect Agent* is installed or the URL of the *MTConnect Agent*; e.g., http://<address>[:port]/. Note: The port number need not be specified if it is the default HTTP port 80. sender is a required attribute.	1
bufferSize	A value representing the maximum number of *Data Entities* that **MAY** be retained in the *MTConnect Agent* that published the *Response Document* at any point in time. The value reported for bufferSize **MUST** be a number representing an unsigned 32-bit integer. bufferSize is a required attribute. Note 1: bufferSize represents the maximum number of *sequence numbers* that **MAY** be stored in the *MTConnect Agent*. Note 2: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the bufferSize.	1

The following is an example of a Header XML element for an *MTConnectError Response Document*:

```
<Header creationTime="2017-02-16T16:44:27Z" sender="MyAgent"
  instanceId="1268463594" bufferSize="131072" version="1.4.0.10"/>
```

6.6 *Document Body*

The *Document Body* contains the information that is published by an *MTConnect Agent* in response to a *Request* from a client software application. Each *Response Document* has a different XML element that represents the *Document Body*.

The structure of the content of the XML element representing the *Document Body* is defined by the *semantic data models* defined for each *Response Document*.

The following table defines the relationship between each of the *Response Documents*, the XML element that represents the *Document Body* for each document, and the *semantic data model* that defines the structure for the content of each of the *Response Documents*:

Response Document	XML Element for *Document Body*	*Semantic Data Model*
MTConnectDevices	`Devices`	*Devices Information Model*, MTConnect Standard – Part 2.0
MTConnectStreams	`Streams`	*Streams Information Model*, MTConnect Standard – Part 3.0
MTConnectAssets	`Assets`	*Assets Information Model*, MTConnect Standard – Part 4.0, and its sub-*Parts*
MTConnectError	`Errors` Note: `Errors` **MUST NOT** be used when backwards compatibility with MTConnect Standard Version 1.0.1 and earlier is required.	*Errors Information Model*, MTConnect Standard – Part 1.0, *Section 9*

6.7 Extensibility

MTConnect is an extensible standard, which means that implementers **MAY** extend the *Data Models* defined in the various sections of the MTConnect Standard to include information required for a specific implementation. When these *Data Models* are encoded using XML, the methods for extending these *Data Models* are defined by the rules established for extending any XML schema (see the W3C website for more details on extending XML data models).

The following are typical extensions that **MAY** be considered in the MTConnect *Data Models*:

- Additional `type` and `subType` values for *Data Entities.*
- Additional *Structural Elements* as containers.
- Additional Composition elements.
- New *Asset* types that are sub-typed from the *Abstract Asset* type.
- *Child Elements* that may be added to specific XML elements contained within the *MTConnect Information Models*. These extended elements **MUST** be identified in a separate *namespace*.

When extending an MTConnect *Data Model*, there are some basic rules restricting changes to the MTConnect *Data Models*.

When extending an *MTConnect Data Model*, an implementer:

- **MUST NOT** add new value for `category` for *Data Entities,*
- **MUST NOT** add new *Root Elements,*
- **SHOULD NOT** add new *Top Level Components, and*
- **MUST NOT** add any new attributes or include any sub-elements to `Composition`.

> Note: Throughout the documents additional information is provided where extensibility may be acceptable or unacceptable to maintain compliance with the MTConnect Standard.

When a *schema* representing a *Data Model* is extended, the *Schema and Namespace Declaration* at the beginning of the corresponding *Response Document* **MUST** be updated to reflect the new *schema* and *namespace* so that a client software application can properly validate the *Response Document*.

An XML example of a *Schema and Namespace Declaration*, including an extended *schema* and *namespace*, would be:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.    <MTConnectDevices
3.      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
4.      xmlns="urn:mtconnect.org:MTConnectDevices:1.3"
5.      xmlns:m="urn:mtconnect.org:MTConnectDevices:1.3"
6.      xmlns:x="urn:MyLocation:MyFile:MyVersion"
7.      xsi:schemaLocation="urn:MyLocation:MyFile:MyVersion
8.        /schemas/MyFileName.xsd">
```

In this example:

- `xmlns:x` is added in Line 6 to identify the *XML Schema Instance* for the extended *schema*. *Element Names* identified with an "`x`" prefix are associated with this specific *XML Schema Instance*.

  Note: The "x" prefix **MAY** be replaced with any prefix that the implementer chooses for identifying the extended *schema* and *namespace*.

- `xsi:schemaLocation` is modified in Line 7 to associate the namespace URN with the URL specifying the location of schema file.

- *`MyLocation`*, *`MyFile`*, *`MyVersion`*, *and `MyFileName`* in Lines 6 and 7 **MUST** be replaced by the actual name, version, and location of the extended *schema*.

When an extended *schema* is implemented, each *Structural Element*, *Data Entity*, and *MTConnect Asset* defined in the extended *schema* **MUST** be identified in each respective *Response Document* by adding a prefix to the XML *Element Name* associated with that *Structural Element*, *Data Entity*, or *MTConnect Asset*. The prefix identifies the *schema* and *namespace* where that XML Element is defined.

7 Protocol and Messaging

An *MTConnect*® *Agent* performs two major communications tasks. It collects information from pieces of equipment and it publishes *MTConnect Response Documents* in response to *Requests* from client software applications.

The MTConnect Standard does not address the method used by an *MTConnect Agent* to collect information from a piece of equipment. The relationship between the *Agent* and a piece of equipment is implementation dependent. The *Agent* may be fully integrated into the piece of equipment or the *Agent* may be independent of the piece of equipment. Implementation of the relationship between a piece of equipment and an *MTConnect Agent* is the responsibility of the supplier of the piece of equipment and/or the implementer of the *MTConnect Agent*.

The communications mechanism between an *MTConnect Agent* and a client software application requires the following primary components:

- *Physical Connection*: The network transmission technologies that physically interconnect an *MTConnect Agent* and a client software application. Examples of a *Physical Connection* would be an Ethernet network or a wireless connection.

- *Transport Protocol*: A set of capabilities that provide the rules and procedures used to transport information between an *MTConnect Agent* and a client software application through a *Physical Connection*.

- *Application Programming Interface (API)*: The *Request* and *Response* interactions that occur between an *MTConnect Agent* and a client software application.

- *Message*: The content of the information that is exchanged. The *Message* includes both the content of the *MTConnect Response Document* and any additional information required for the client software application to interpret the *Response Document*.

Note: The *Physical Connections*, *Transport Protocols*, and *Application Programming Interface (API)* supported by an *MTConnect Agent* are independent of the *Message* itself; i.e., the information contained in the *MTConnect Response Documents* is not changed based on the methods used to transport those documents to a client software application.

An *MTConnect Agent* **MAY** support multiple methods for communicating with client software applications. The MTConnect Standard specifies one methodology for communicating that **MUST** be supported by every *MTConnect Agent*. This methodology is a *REpresentational State Transfer* (REST) interface, which defines a stateless, client-server communications architecture. This REST interface is the architectural pattern that specifies the exchange of information between an *MTConnect Agent* and a client software application. REST dictates that a server has no responsibility for tracking or coordinating with a client software application regarding which information or how much information the client software application may request from a server. This removes the burden for a server to keep track of client sessions. An *MTConnect Agent* **MUST** be implemented as a server supporting the RESTful interface.

8 *HTTP Messaging* Supported by an *MTConnect Agent*

This section describes the application of *HTTP Messaging* applied to a REST interface that **MUST** be supported by an *MTConnect Agent* to realize the *MTConnect Request/Response Information Exchange* functionality

8.1 REST Interface

An *MTConnect Agent* **MUST** provide a REST interface that supports HTTP version 1.0 to communicate with client applications. This interface **MUST** support HTTP (RFC7230) and use URIs (RFC3986) to identify specific information requested from an *Agent.* HTTP is most often implemented on top of the Transmission Control Protocol (TCP) that provides an ordered byte stream of data and the Internet Protocol (IP) that provides unified addressing and routing between computers. However, additional interfaces to an *MTConnect Agent* may be implemented in conjunction with any other communications technologies.

The REST interface supports an *Application Programming Interface (API)* that adheres to the architectural principles of a stateless, uniform interface to retrieve data and other information related to either pieces of equipment or *MTConnect Assets*. The API allows for access, but not modification of data stored within the *MTConnect Agent* and is nullipotent, meaning it will not produce any side effects on the information stored in an *MTConnect Agent* or the function of the *Agent* itself.

HTTP Messaging is comprised of two basic functions – an *HTTP Request* and an *HTTP Response*. A client software application forms a *Request* for information from an *MTConnect Agent* by specifying a specific set of information using an *HTTP Request*. In response, an *MTConnect Agent* provides either an *HTTP Response* or replies with an *HTTP Error Message* as defined below.

8.2 *HTTP Request*

The MTConnect Standard defines that an *MTConnect Agent* **MUST** support the `HTTP GET` verb – no other HTTP methods are required to be supported.

An *HTTP Request* **MAY** include three sections:

- an *HTTP Request Line*
- *HTTP Header Fields*
- an *HTTP Body*

The MTConnect Standard defines that an *HTTP Request* issued by a client application **SHOULD** only have two sections:

- an *HTTP Request Line*
- *Header Fields.*

The *HTTP Request Line* identifies the specific information being requested by the client software application. If an *MTConnect Agent* receives any information in an *HTTP Request* that is not specified in the MTConnect Standard, the *Agent* **MAY** ignore it.

The structure of an *HTTP Request Line* consists of the following portions:

- *HTTP Request Method*: GET
- *HTTP Request URL*: http://<authority>/<path>[?<query>]
- *HTTP Version*: HTTP/1.0

For the following discussion, the *HTTP Request URL* will only be considered since the *Method* will always be GET and the MTConnect Standard only requires HTTP/1.0.

8.2.1 authority Portion of an *HTTP Request Line*

The authority portion consists of the DNS name or IP address associated with an *MTConnect Agent* and an optional TCP port number [:port] that the *Agent* is listening to for incoming *Requests* from client software applications. If the port number is the default Port 80, Port is not required.

Example forms for authority are:

- http://machine/
- http://machine:5000/
- http://192.168.1.2:5000/

8.2.2 **path** Portion of an *HTTP Request Line*

The <Path> portion of the *HTTP Request Line* has the follow segments:

- /<name or uuid>/<request>

In this portion of the *HTTP Request Line*, name or uuid designates that the information to be returned in a *Response Document* is associated with a specific piece of equipment that has published data to the *MTConnect Agent*. See *Part 2 - Devices Information Model* for details on name or uuid for a piece of equipment.

> Note: If name or uuid are not specified in the *HTTP Request Line*, an *MTConnect Agent* **MUST** return the information for all pieces of equipment that have published data to the *Agent* in the *Response Document*.

In the <Path> portion of the *HTTP Request Line,* <request> designates one of the *Requests* defined in *Section 5.4*. The value for <request> **MUST** be probe, current, sample, or asset(s) representing the *Probe Request*, *Current Request*, *Sample Request*, and *Asset Request* respectively.

8.2.3 **query** Portion of an *HTTP Request Line*

The [?<query>]portion of the *HTTP Request Line* designates an *HTTP Query*. *Query* is a string of parameters that define filters used to refine the content of a *Response Document* published in response to an *HTTP Request*.

8.3 MTConnect *Request/Response Information Exchange* Implemented with HTTP

An *MTConnect Agent* **MUST** support *Probe Requests, Current Requests, Sample Requests, and Asset Requests*.

The following sections define how the *HTTP Request Line* is structured to support each of these types of *Requests* and the information that an *MTConnect Agent* **MUST** provide in response to these *Requests*.

8.3.1 *Probe Request* Implemented Using HTTP

An *MTConnect Agent* responds to a *Probe Request* with an *MTConnectDevices Response Document* that contains the *Equipment Metadata* for pieces of equipment that are requested and currently represented in the *Agent*.

There are two forms of the *Probe Request*:

- The first form includes an *HTTP Request Line* that does not specify a specific path portion (name or uuid). In response to this *Request,* the *MTConnect Agent* returns an *MTConnectDevices Response Document* with information for all pieces of equipment represented in the *MTConnect Agent.*
    1. http://<authority>/probe
- The second form includes an *HTTP Request Line* that specifies a specific path portion that defines either a name or uuid. In response to this *Request*, the *MTConnect Agent* returns an *MTConnectDevices Response Document* with information for only the one piece of equipment associated with that name or uuid.
    1. http://<authority>/<name or uuid>/probe

8.3.1.1 **Path Portion of the *HTTP Request Line* for a *Probe Request***

The following segments of path **MUST** be supported in an *HTTP Request Line* for a *Probe Request*:

Path Segments	Description
name or UUID	If present, specifies that only the *Equipment Metadata* for the piece of equipment *represented by* the name or UUID will be published If not present, *Metadata* for all pieces of equipment associated with the *MTConnect Agent* will be published.
<request>	Designates one of the following *Requests*: probe, current, sample, or asset(s). probe **MUST** be provided.

8.3.1.2 **Query Portion of the *HTTP Request Line* for a *Probe Request***

The *HTTP Request Line* for a *Probe Request* **SHOULD NOT** contain a Query. If the *Request* does contain a Query, the *Agent* **MUST** ignore the Query.

8.3.1.3 ***Response* to a *Probe Request***

The *Response* to a *Probe Request* **SHOULD** be an *MTConnectDevices Response Document* for one or more pieces of equipment as designated by the path portion of the *Request*.

The *Response Document* returned in response to a *Probe Request* **MUST** always provide the most recent information available to an *MTConnect Agent*.

The *Response* **MUST** also include an *HTTP Status Code*. If problems are encountered by an *MTConnect Agent* while responding to a *Probe Request*, the *Agent* **MUST** also publish an *Error Response Document*.

8.3.1.4 *HTTP Status Codes* for a *Probe Request*

The following *HTTP Status Codes* **MUST** be supported as possible responses to a *Probe Request*:

HTTP Status Code	Code Name	Description
200	OK	The *Request* was handled successfully.
400	Bad Request	The *Request* could not be interpreted. The *MTConnect Agent* **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies either `INVALID_URI` or `INVALID_REQUEST` as the `errorCode`.
404	Not Found	The *Request* could not be interpreted. The *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `NO_DEVICE` as the `errorCode.`
405	Method Not Allowed	A method other than `GET` was specified in the *Request* or the piece of equipment specified in the *Request* could not be found. The *MTConnect Agent* **MUST** return a 405 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode.`
406	Not Acceptable	The *HTTP Accept Header* in the *Request* was not one of the supported representations. The *MTConnect Agent* **MUST** return a 406 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode.`
431	Request Header Fields Too Large	The fields in the *HTTP Request* exceed the limit of the implementation of the *MTConnect Agent.* The *MTConnect Agent* **MUST** return a 431 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INVALID_REQUEST` as the `errorCode.`
500	Internal Server Error	There was an unexpected error in the *MTConnect Agent* while responding to a *Request.* The *MTConnect Agent* **MUST** return a 500 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INTERNAL_ERROR` as the `errorCode.`

8.3.2 *Current Request* Implemented Using HTTP

An *MTConnect Agent* responds to a *Current Request* with an *MTConnectStreams Response Document* that contains the current value of *Data Entities* associated with each piece of *Streaming Data* available from the *Agent,* subject to any filtering defined in the *Request*.

There are two forms of the *Current Request*:

- The first form is given without a specific path portion (name or uuid). In response to this *Request*, the *MTConnect Agent* returns an *MTConnectStreams Response Document* with information for all pieces of equipment represented in the *buffer* of the *Agent.*

    1. http://<authority>/current[?query]

- The second form includes a specific path portion that defines either a name or uuid. In response to this *Request*, the *MTConnect Agent* returns an *MTConnectStreams Response Document* with information for only the one piece of equipment associated with the name or uuid defined in the *Request*.

    1. http://<authority>/<name or uuid>/current[?query]

8.3.2.1 **Path** Portion of the *HTTP Request Line* for a *Current Request*

The following segments of path **MUST** be supported for an *HTTP Request Line* for a *Current Request*:

Path Segments	Description
name or UUID	If present, specifies that only the *Data Entities* for the piece of equipment represented by the name or UUID will be published. If not present, *Data Entities* for all pieces of equipment associated with the *Agent* will be published.
<request>	Designates one of the following *Requests*: probe, current, sample, or asset(s). current **MUST** be provided.

8.3.2.2 **Query** Portion of the *HTTP Request Line* for a *Current Request*

A *Query* may be used to more precisely define the specific information to be included in a *Response Document*. Multiple parameters may be used in a *Query* to further refine the information to be included. When multiple parameters are provided, each parameter is separated by an ampersand (&) character and each parameter appears only once in the *Query*. The parameters within the *Query* may appear in any sequence.

The following `query` parameters **MUST** be supported in an *HTTP Request Line* for a *Current Request*:

Query Parameters	Description
`path`	An XPATH that defines specific information or a set of information to be included in an *MTConnectStreams Response Document*. The value for the XPATH is the location of the information defined in the *MTConnectDevices Information Model* that represents the *Structural Element(s)* and/or the specific *Data Entities* to be included in the *MTConnectStreams Response Document.*
`at`	Requests that the *MTConnect Response Document* **MUST** include the current value for all *Data Entities* relative to the time that a specific *sequence number* was recorded. The value associated with the `at` parameter references a specific *sequence number*. The value **MUST** be an unsigned 64-bit value. The `at` parameter **MUST NOT** be used in conjunction with the `interval` parameter since this would cause an M*TConnect Agent* to repeatedly return the same data. If the value provided for the `at` parameter is a negative number or is not a, the *Request* **MUST** be determined to be invalid. The *MTConnect Agent* **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies an `INVALID_REQUEST` `errorCode`. If the value provided for the `at` parameter is either lower than the value of `firstSequence` or greater than the value of `lastSequence`, the *Request* **MUST** be determined to be invalid. The *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. The *Agent* **MUST** also publish an *Error Response Document* that identifies an `OUT_OF_RANGE` `errorCode`. Note: Some information stored in the *buffer* of an *MTConnect Agent* may not be returned for a *Current Request* with a *Query* containing an `at` parameter if the *sequence number* associated with the most current value for that information is greater than the *sequence number* specified in the *Query*.
`interval`	When a *Current Request* includes a *Query* with the `interval` parameter, an *MTConnect Agent* **MUS**T respond to this *Request* by repeatedly publishing the required *Response Document* at the time interval (period) defined by the value provided for the `interval` parameter. The value provided for `interval` **MUST** be expressed in milliseconds and **MUST** be a positive value greater than 0. The `interval` parameter **MUST NOT** be used in conjunction with the `at` parameter since this would cause an M*TConnect Agent* to repeatedly return the same data. If a *Request* contains a *Query* with an `interval` parameter, it **MUST** remain in effect until the client software application terminates its connection to the *Agent*.

8.3.2.3 *Response* to a *Current Request*

The *Response* to a *Current Request* **SHOULD** be an *MTConnectStreams Response Document* for one or more pieces of equipment designated by the `path` portion of the *Request*.

The *Response* to a *Current Request* **MUST** always provide the most recent information available to an *MTConnect Agent* or, when the `at` parameter is specified, the value of the data at the given *sequence number*.

The *Data Entities* provided in the *MTConnectStreams Response Document* will be limited to those specified in the combination of the `path` segment of the *Current Request* and the value of the XPATH defined for the `path` attribute provided in the `query` segment of that *Request*.

8.3.2.4 *HTTP Status Codes* for a *Current Request*

The following *HTTP Status Codes* **MUST** be supported as possible responses to a *Current Request*:

HTTP Status Code	Code Name	Description
200	OK	The *Request* was handled successfully.
400	Bad Request	If the *Request* could not be interpreted, the MTConnect Agent **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies either an `INVALID_URI`, `INVALID_REQUEST`, or `INVALID_XPATH` as the `errorCode`. If the `query` parameters do not contain a valid value or include an invalid parameter, the MTConnect Agent **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `QUERY_ERROR` as the `errorCode`.
404	Not Found	If the *Request* could not be interpreted, the *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `NO_DEVICE` as the `errorCode`. If the value of the `at` parameter was greater than the *last sequence number* or is less than the *first sequence number*, the *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `OUT_OF_RANGE` as the `errorCode`.
405	Method Not Allowed	A method other than `GET` was specified in the *Request* or the piece of equipment specified in the *Request* could not be found. The *MTConnect Agent* **MUST** return a 405 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode`.

HTTP Status Code	Code Name	Description
406	Not Acceptable	The *HTTP Accept Header* in the *Request* was not one of the supported representations. The *MTConnect Agent* **MUST** return a 406 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode`.
431	Request Header Fields Too Large	The fields in the *Request* exceed the limit of the implementation of the *MTConnect Agent.* The *MTConnect Agent* **MUST** return a 431 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INVALID_REQUEST` as the `errorCode`.
500	Internal Server Error	There was an unexpected error in the *MTConnect Agent* while responding to a *Request*. The *MTConnect Agent* **MUST** return a 500 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INTERNAL_ERROR` as the `errorCode`.

8.3.3 *Sample Request* Implemented Using HTTP

An *MTConnect Agent* responds to a *Sample Request* with an *MTConnectStreams Response Document* that contains a set of values for *Data Entities* currently available for *Streaming Data* from the *Agent,* subject to any filtering defined in the *Request*.

There are two forms to the *Sample Request*:

- The first form is given without a specific `path` portion (`name` or `uuid`). In response to this *Request*, the *MTConnect Agent* returns an *MTConnectStreams Response Document* with information for all pieces of equipment represented in the *Agent.*

```
1.  http://<authority>/sample[?query]
```

- The second form includes a specific `path` portion that defines either a `name` or `uuid`. In response to this *Request*, the *MTConnect Agent* returns an *MTConnectStreams Response Document* with information for only the one piece of equipment associated with the `name` or `uuid` defined in the *Request*.

```
1.  http://<authority>/<name or uuid>/sample?query
```

8.3.3.1 `Path` Portion of the *HTTP Request Line* for a *Sample Request*

The following segments of `path` **MUST** be supported in the *HTTP Request Line* for a *Sample Request*:

Path Segments	Description
`name` or `UUID`	If present, specifies that only the *Data Entities* for the piece of equipment represented by the `name` or `UUID` will be published. If not present, *Data Entities* for all pieces of equipment associated with the *Agent* will be published.
`<request>`	Designates one of the following *Requests*: `probe`, `current`, `sample`, or `asset(s).` `sample` **MUST** be provided.

8.3.3.2 `Query` Portion of the *HTTP Request Line* for a *Sample Request*

A *Query* may be used to more precisely define the specific information to be included in a *Response Document*. Multiple parameters may be used in a *Query* to further refine the information to be included. When multiple parameters are provided, each parameter is separated by an `&` character and each parameter appears only once in the *Query*. The parameters within the *Query* may appear in any sequence.

The following `query` parameters **MUST** be supported in an *HTTP Request Line* for a *Sample Request*:

Query Parameters	Description
`path`	An XPATH that defines specific information or a set of information to be included in an *MTConnectStreams Response Document*. The value for the XPATH is the location of the information defined in the *MTConnectDevices Information Model* that represents the *Structural Element(s)* and/or the specific *Data Entities* to be included in the *MTConnectStreams Response Document*.

Query Parameters	Description
`from`	The `from` parameter designates the *sequence number* of the first *Data Entity* in the *buffer* of the *MTConnect Agent* that **MUST** be included in the *Response Document*. The value for `from` **MUST** be an unsigned 64-bit integer. The `from` parameter is typically provided in conjunction with the `count` parameter. However, this is not required. If the *sequence number* provided as the value for the `from` parameter is 0, the information provided in the *Response Document* **MUST** be provided starting with the information located in the buffer of an *MTConnect Agent* defined by `firstSequence.` If no *sequence number* is provided as the value for the `from` parameter, the information provided in the *Response Document* **MUST** be provided starting with the information located in the *buffer* of an *MTConnect Agent* defined by `firstSequence.` If the *sequence number* provided as the value for the `from` parameter is a negative number, the request **MUST** be determined to be invalid and the *MTConnect Agent* **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies an `INVALID_REQUEST errorCode`. If the value provided for the `from` parameter is either lower than the value of `firstSequence` or greater than the value of `lastSequence`, the request **MUST** be determined to be invalid and the *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies an `OUT_OF_RANGE errorCode`.
`interval`	When a *Sample Request* includes a *Query* with the `interval` parameter, an *MTConnect Agent* **MUST** respond to this *Request* by repeatedly publishing the required *Response Document* at the time interval (period) defined by the value provided for the `interval` parameter. The value provided for `interval` **MUST** be expressed in milliseconds. The `interval` parameter **MUST NOT** be used in conjunction with the `at` parameter since this would cause an M*TConnect Agent* to repeatedly return the same data. If the value for the `interval` parameter is 0, the M*TConnect Agent* **MUST** provide successive *Response Documents* at the fastest rate that the *Agent* can support. If a `count` parameter is not provided in conjunction with an `interval` parameter, an *MTConnect Agent* **SHOULD** use a default value of 100 for `count`. If a *Request* contains a *Query* with an `interval` parameter, it **MUST** remain in effect until the client software application terminates its connection to the *Agent*. An *MTConnect Agent* **MUST NOT** publish a *Response Document* if no new data associated with the *Response Document* is available in the *buffer*. However, if new data associated with the *Response Document* is received by the *Agent* at a point in time after the value of the interval parameter is exceeded, the *Agent* **MUST** then publish a new version of the *Response Document* immediately.

Query Parameters	Description
`count`	The `count` parameter designates the total number of *Data Entities* to be published from the *buffer* of the *MTConnect Agent* in the *Response Document.* The `count` parameter is typically provided in conjunction with the `from` parameter. However, this is not required. If the value provided for the `count` parameter defines information located in the *buffer* of an *MTConnect Agent* that would be a *sequence number* greater than the value of `lastSequence`, the information provided **MUST** be limited only to the information available in the *buffer*. If no value is provided for the `count` parameter, the information provided in the *Response Document* **MUST** default to `count=100.` If the value provided for the `count` parameter is 0 or a negative number, the request **MUST** be determined to be invalid. The *MTConnect Agent* must return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies an `INVALID_REQUEST` `errorCode`.
`heartbeat`	Sets the time period for the *heartbeat* function in an *MTConnect Agent*. The value for `heartbeat` represents the amount of time after a *Response Document* has been published until a new *Response Document* **MUST** be published, even when no new data is available. The value for `heartbeat` is defined in milliseconds. If no value is defined for heartbeat, the value **SHOULD** default to 10 seconds. `heartbeat` **MUST** only be specified if interval is also specified.

8.3.3.3 ***Response* to a *Sample Request***

The *Response* to a *Sample Request* **SHOULD** be an *MTConnectStreams Response Document* for one or more pieces of equipment designated by the `path` portion of the *Request*.

The *Response* to a *Sample Request* **MUST** always provide the most recent information available to an *MTConnect Agent* or, when the `at` parameter is specified, the value of the data at the given *sequence number*.

The *Data Entities* provided in the *MTConnectStreams Response Document* will be limited to those specified in the combination of the `path` segment of the *Sample Request* and the value of the XPATH defined for the `path` attribute provided in the `query` segment of that *Request*.

When the value of `from` references the value of the next sequence number (`nextSequence`) and there are no additional *Data Entities* available in the *buffer*, the response document will have an empty `<Streams/>` element in the *MTConnectStreams* document to indicate no data is available at the point in time that the *Agent* published the *Response Document*.

8.3.3.4 *HTTP Status Codes* for a *Sample Request*

The following *HTTP Status Codes* **MUST** be supported as possible responses to a *Sample Request*:

HTTP Status Code	Code Name	Description
200	OK	The *Request* was handled successfully.
400	Bad Request	If the *Request* could not be interpreted, the *MTConnect Agent* **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies either an `INVALID_URI`, `INVALID_REQUEST`, or `INVALID_XPATH` as the `errorCode`. If the `query` parameters do not contain a valid value or include an invalid parameter, The *MTConnect Agent* **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `QUERY_ERROR` as the `errorCode`.
404	Not Found	If the *Request* could not be interpreted, the *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `NO_DEVICE` as the `errorCode`. If the value of the `at` query parameter was greater than the last sequence number or less than the first sequence number, the *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `OUT_OF_RANGE` as the `errorCode`.
405	Method Not Allowed	A method other than `GET` was specified in the *Request* or the piece of equipment specified in the *Request* could not be found. The *MTConnect Agent* **MUST** return a 405 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode`.
406	Not Acceptable	The *HTTP Accept Header* in the Request was not one of the supported representations. The *MTConnect Agent* **MUST** return a 406 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode`.
431	Request Header Fields Too Large	The fields in the *Request* exceed the limit of the implementation of the *MTConnect Agent.* The *MTConnect Agent* **MUST** return a 431 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INVALID_REQUEST` as the `errorCode`.

HTTP Status Code	Code Name	Description
500	Internal Server Error	There was an unexpected error in the *MTConnect Agent* while responding to a *Current Request*. The *MTConnect Agent* **MUST** return a 500 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INTERNAL_ERROR` as the `errorCode`.

8.3.4 *Asset Request* Implemented Using HTTP

An *MTConnect Agent* responds to an *Asset Request* with an *MTConnectAssets Response Document* that contains information for *MTConnect Assets* from the *Agent,* subject to any filtering defined in the *Request*.

There are multiple forms to the *Asset Request*:

- The first form is given without a specific `path` portion (`name` or `uuid`). In response to this *Request*, the *MTConnect Agent* returns an *MTConnetAssets Response Document* that contains information for all *Asset Document* represented in the *Agent.*

```
1.   http://<authority>/assets
```

- The second form includes a specific `path` portion that defines the identity (`asset_id`) for one or more specific *Asset Documents*. In response to this *Request*, the *MTConnect Agent* returns an *MTConnetAssets Response Document* that contains information for the specific *Assets* represented in the *Agent* and defined by each of the `asset_id` values provided in the *Request*. Each `asset_id` is separated by a ";".

```
1.   http://<authority>/asset/asset_id;asset_id;asset_id....
```

Note: An *HTTP Request Line* may include combinations of `path` and `query` to achieve the desired set of *Asset Documents* to be included in a specific *MTConnectAssets Response Document*.

8.3.4.1 `Path` Portion of the *HTTP Request Line* for an *Asset Request*

The following segments of `path` **MUST** be supported in the *HTTP Request Line* for an *Asset Request*:

Path Segments	Description
`<request>`	Designates one of the following *Requests*: `probe`, `current`, `sample`, or `asset(s)`. `asset` or `assets` **MUST** be provided.
`asset_id`	Identifies the `id` attribute of an *MTConnect Asset* to be provided by an *MTConnect Agent*.

8.3.4.2 `Query` Portion of the *HTTP Request Line* for an *Asset Request*

A *Query* may be used to more precisely define the specific information to be included in a *Response Document*. Multiple parameters may be used in a *Query* to further refine the information to be included. When multiple parameters are provided, each parameter is separated by an `&` character and each parameter appears only once in the *Query*. The parameters within the *Query* may appear in any sequence.

The following `query` parameters **MUST** be supported in an *HTTP Request Line* for an *Asset Request*:

Query Parameters	Description
`type`	Defines the type of *MTConnect Asset* to be returned in the *MTConnectAssets Response Document*. The type for an *Asset* is the term used in the *MTConnect Assets Information Model* to describe different types of *Assets*. It is the term that is substituted for the `Asset` container and describes the highest-level element in the *Asset* hierarchy. See *Part 4.0, Section 3.2.3* for more information on the type of an *Asset*.
`removed`	*Assets* can have an attribute that indicates whether the *Asset* has been removed from a piece of equipment. The valid values for `removed` are `true` or `false.` If the value of the `removed` parameter in the `query` is `true`, then *Asset Documents for Asset*s that have been marked as removed from a piece of equipment will be included in the *Response Document*. If the value of the `removed` parameter in the `query` is `false`, then *Asset Documents* for *Assets* that have been marked as removed from a piece of equipment will not be included in the *Response Document*. If `removed` is not defined in a `query`, the default value for `removed` **MUST** be determined to be `false`.
`count`	Defines the maximum number of *Asset Documents* to return in an *MTConnectAssets Response Document*. If `count` is not defined in the `query`, the default vale for `count` **MUST** be determined to be 100.

8.3.4.3 *Response* to an *Asset Request*

The *Response* to an *Asset Request* **SHOULD** be an *MTConnectAssets Response Document* containing information for one or more *Asset Documents* designated by the *Request*.

The *Response* to an *Asset Request* **MUST** always provide the most recent information available to an *MTConnect Agent*.

The *Asset Documents* provided in the *MTConnectAssets Response Document* will be limited to those specified in the combination of the `path` segment of the *Asset Request* and the parameters provided in the `query` segment of that *Request*.

If the `removed` query parameter is not provided with a value of `true`, *Asset Documents* for *Assets* that have been marked as removed will not be provided in the response.

8.3.4.4 *HTTP Status Codes* for a *Sample Request*

The following *HTTP Status Codes* **MUST** be supported as possible responses to an *Asset Request*:

HTTP Status Code	Code Name	Description
200	OK	The *Request* was handled successfully.
400	Bad Request	If the *Request* could not be interpreted, the *MTConnect Agent* **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies either an `INVALID_URI` or `INVALID_REQUEST` as the `errorCode`. If the `query` parameters do not contain a valid value or include an invalid parameter, The *MTConnect Agent* **MUST** return a 400 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `QUERY_ERROR` as the `errorCode`.
404	Not Found	If the *Request* could not be interpreted, the *MTConnect Agent* **MUST** return a 404 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `NO_DEVICE` or `ASSET_NOT FOUND` as the `errorCode`.
405	Method Not Allowed	A method other than `GET` was specified in the *Request* or the piece of equipment specified in the *Request* could not be found. The *MTConnect Agent* **MUST** return a 405 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode`.
406	Not Acceptable	The *HTTP Accept Header* in the Request was not one of the supported representations. The *MTConnect Agent* **MUST** return a 406 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `UNSUPPORTED` as the `errorCode`.
431	Request Header Fields Too Large	The fields in the *Request* exceed the limit of the implementation of the *MTConnect Agent*. The *MTConnect Agent* **MUST** return a 431 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INVALID_REQUEST` as the `errorCode`.

HTTP Status Code	Code Name	Description
500	Internal Server Error	There was an unexpected error in the *MTConnect Agent* while responding to a *Current Request*. The *MTConnect Agent* **MUST** return a 500 *HTTP Response Code*. Also, the *Agent* **MUST** publish an *Error Response Document* that identifies `INTERNAL_ERROR` as the `errorCode`.

8.3.5 HTTP Errors

When an *MTConnect Agent* receives an *HTTP Request* that is incorrectly formatted or is not supported by the *Agent*, the *Agent* **MUST** publish an *HTTP Error Message* which includes a specific status code from the tables above indicating that the *Request* could not be handled by the *Agent*.

Also, if the *MTConnect Agent* experiences an internal error and is unable to provide the requested *Response Document*, it **MUST** publish an *HTTP Error Message* that includes a specific status code from the table above.

When an *MTConnect Agent* encounters an error in interpreting or responding to an *HTTP Request*, the *Agent* **MUST** also publish an `MTConnectError` *Response Document* that provides additional details about the error. See *Section 9.0 – Error Information Model* for details on the `MTConnectError` *Response Document*.

8.3.6 Data Streaming

Since an *MTConnect Agent* **MUST** support a REST interface and it **MUST** support HTTP *Messaging*, it **MUST** also support *HTTP Data Streaming*. *HTTP Data Streaming* is a method for a server to provide a continuous stream of information in response to a single *Request* from a client software application. *Data Streaming* is a version of a *Publish/Subscribe* method of communications.

For an *MTConnect Agent*, a *Data Streaming Request* is initiated by a client software application by making an *HTTP Request* to the *Agent* that includes a *Query* with an `interval` parameter.

When an *MTConnect Agent* receives this *Request*, the *Agent* **MUST** respond by repeatedly publishing the appropriate *MTConnect Response Document*. Each version of the requested *Response Document* is published based on the time period defined by the *value* provided for the `interval` parameter included in the *Request*.

Once initiated, a *Data Streaming Request* continues until either the *Agent* or the client software application terminates the connection between the *Agent* and the client.

If no new information is available in the *buffer* of the *MTConnect Agent* associated with the requested *Response Document* and the time since the previous document was sent exceeds the value of the `interval` parameter, the *Agent* **MUST NOT** publish a *Response Document*. However, if new data associated with the *Response Document* is received by the *Agent* at a point in time after the value of the *period* for the `interval` parameter is exceeded, the *Agent* **MUST** then publish a new *Response Document* immediately.

An *MTConnect Agent* **SHOULD** support any number of simultaneous and asynchronous *Data Streaming Requests* with a single client or any number of client software application.

8.3.6.1 *Heartbeat*

When *Streaming Data* is requested from a *Sample Request*, an *MTConnect Agent* **MUST** support a *heartbeat* to indicate to a client application that the HTTP connection is still viable during times when there is no new data available to be published. The *heartbeat* is indicated by an *MTConnect Agent* by sending an *MTConnect Response Document* with an empty *Steams* container (See *Part 3, Section 4.1 Streams* for more details on the *Streams* container) to the client software application.

The *heartbeat* **MUST** occur on a periodic basis given by the optional `heartbeat` query parameter or **MUST** default to 10 seconds. An *MTConnect Agent* **MUST** maintain a separate *heartbeat* for each client application for which the *Agent* is responding to a *Data Streaming Request*.

An *MTConnect Agent* **MUST** begin calculating the interval for the time-period of the *heartbeat* for each client application immediately after a *Response Document* is published to that specific client application.

The *heartbeat* remains in effect for each client software application until the *Data Streaming Request* is terminated by either the *MTConnect Agent* or the client application.

9 *Error Information Model*

The *Error Information Model* establishes the rules and terminology that describes the *Response Document* returned by an *MTConnect Agent* when it encounters an error while interpreting a *Request* for information from a client software application or when an *Agent* experiences an error while publishing the *Response* to a *Request* for information.

An *MTConnect Agent* provides the information regarding errors encountered when processing a *Request* for information by publishing an *MTConnectError Response Document* to the client software application that made the *Request* for information.

9.1 *MTConnectError Response Document*

The *MTConnectError Response Document* is comprised of two sections: `Header` and `Errors`.

The `Header` section contains information defining the creation of the document and the data storage capability of the *MTConnect Agent* that generated the document. (See *Section 6.5.4* above.)

The `Errors` section of the *MTConnectError Response Document* is a *Structural Element* that organizes *Data Entities* describing each of the errors reported by an *MTConnect Agent*.

9.1.1 *Structural Element* for `MTConnectError`

Structural Elements are XML elements that form the logical structure for an XML document. The *MTConnectError Response Document* has only one *Structural Element*. This *Structural Element* is `Errors`. `Errors` is an XML container element that organizes the information and data associated with all errors relevant to a specific *Request* for information.

The following XML schema represents the structure of the `Errors` XML element.

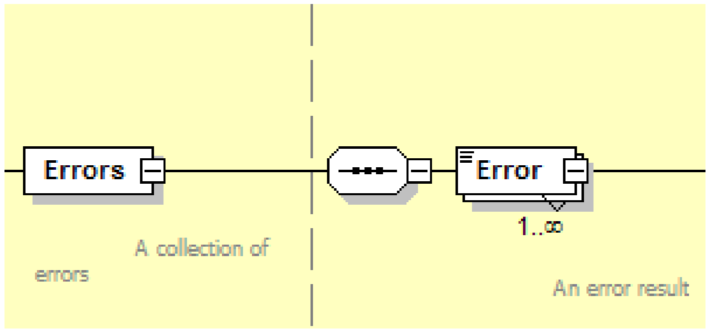


Figure 14: `Errors` *Schema* Diagram

Element	Description	Occurrence
`Errors`	An XML container element in an `MTConnectError` *Response Document* provided by an *MTConnect Agent* when an error is encountered associated with a *Request* for information from a client software application. There **MUST** be only one `Errors` element in an `MTConnectError` *Response Document*. The `Errors` element **MUST** contain at least one `Error` Data Entity element.	1

Note: When compatibility with *Version 1.0.1* and earlier of the MTConnect Standard is required for an implementation, the *MTConnectErrors Response Document* contains only a single `Error` *Data Entity* and the `Errors` *Structural Element* **MUST NOT** appear in the document.

9.1.2 `Error` *Data Entity*

When an *MTConnect Agent* encounters an error when responding to a *Request* for information from a client software application, the information describing the error(s) is reported as a *Data Entity* in an MTConnectError Re*sponse Document*. *Data Entities* are organized in the `Errors` XML container.

There is only one type of *Data Entity* defined for an *MTConnectError Response Document*. That *Data Entity* is called `Error`.

The following is an illustration of the structure of an XML document demonstrating how `Error` *Data Entities* are reported in an *MTConnectError Response Document*:

```
<MTConnectError>
  <Header/>
  <Errors>
    <Error/>
    <Error/>
    <Error/>
  </Errors>
</MTConnectError>
```

The `Errors` element **MUST** contain at least one *Data Entity*. Each *Data Entity* describes the details for a specific error reported by an *MTConnect Agent* and is represented by the XML element named `Error`.

`Error` XML elements **MAY** contain both attributes and CDATA that provide details further defining a specific error. The CDATA **MAY** provide the complete text provided by an *MTConnect Agent* for the specific error.

9.1.2.1 XML *Schema* Structure for `Error`

The following XML schema represents the structure of an `Error` XML element showing the attributes defined for `Error`.

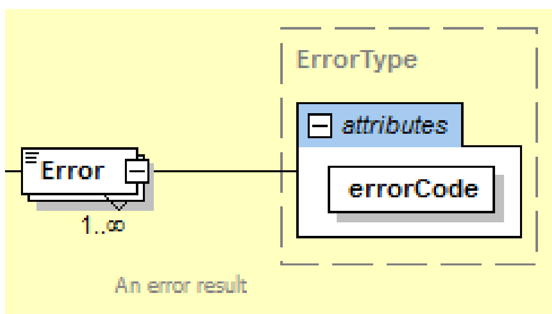


Figure 15: `Error` Schema Diagram

9.1.2.2 Attributes for `Error`

`Error` has one attribute. The following table defines this attribute that provides additional information for an `Error` XML element.

Attribute	Description	Occurrence
`errorCode`	Provides a descriptive code that indicates the type of error that was encountered by an *MTConnect Agent* when attempting to respond to a *Request* for information. `errorCode` is a required attribute.	1

9.1.2.3 Values for `errorCode`

There is a limited vocabulary defined for `errorCode`. The value returned for `errorCode` **MUST** be one of the following:

Value for `errorCode`	Description
`ASSET_NOT_FOUND`	The *Request* for information specifies an *MTConnect Asset* that is not recognized by the *MTConnect Agent*.
`INTERNAL_ERROR`	The *MTConnect Agent* experienced an error while attempting to published the requested information.
`INVALID_REQUEST`	The *Request* contains information that was not recognized by the *MTConnect Agent*.
`INVALID_URI`	The URI provided was incorrect.
`INVALID_XPATH`	The XPATH identified in the *Request* for information could not be parsed correctly by the *MTConnect Agent*. This could be caused by an invalid syntax or the XPATH did not match a valid identify for any information stored in the *Agent*.
`NO_DEVICE`	The identity of the piece of equipment specified in the *Request* for information is not associated with the *MTConnect Agent*.
`OUT_OF_RANGE`	The *Request* for information specifies *Steaming Data* that includes sequence number(s) for pieces of data that are beyond the end of the *buffer*.
`QUERY_ERROR`	The MTConnect Agent was unable to interpret the `Query`. The `Query` parameters do not contain valid values or include an invalid parameter.
`TOO_MANY`	The `count` parameter provided in the *Request* for information requires either of the following: – *Steaming Data* that includes more pieces of data than the *MTConnect Agent* is capable of organizing in an *MTConnectStreams Response Document.* – *Assets* that include more *Asset Documents* in an *MTConnectAssets Response Document* than the *MTConnect Agent* is capable of handling.
`UNAUTHORIZED`	The *Requestor* does not have sufficient permissions to access the requested information.
`UNSUPPORTED`	A valid *Request* was provided, but the *MTConnect Agent* does not support the feature or type of *Request.*

9.1.2.4 CDATA for Error

The CDATA for `Error` contains a textual description of the error and any additional information an *MTConnect Agent* is capable of providing regarding a specific error. The *Valid Data Value* returned for `Error` **MAY** be any text string.

9.1.3 Examples for `MTConnectError`

The following is an example demonstrating the structure of an *MTConnectError Response Document*:

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectError xmlns="urn:mtconnect.org:MTConnectError:1.4"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="urn:mtconnect.org:MTConnectError:1.4
    /schemas/MTConnectError_1.4.xsd">
  <Header creationTime="2010-03-12T12:33:01Z"
    sender="MyAgent" version="1.4.1.10" bufferSize="131000"
    instanceId="1383839" />
  <Errors>
    <Error errorCode="OUT_OF_RANGE" >Argument was out of
      range</Error>
    <Error errorCode="INVALID_XPATH" >Bad path</Error>
  </Errors>
</MTConnectError>
```

The following is an example demonstrating the structure of an *MTConnectError Response Document* when backward compatibility with *Version 1.0.1* and earlier of the MTConnect Standard is required. In this case, the *Document Body* contains only a single `Error` *Data Entity* and the `Errors` *Structural Element* **MUST NOT** appear in the document.

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectError xmlns="urn:mtconnect.org:MTConnectError:1.1"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="urn:mtconnect.org:MTConnectError:1.1
    /schemas/MTConnectError_1.1.xsd">
  <Header creationTime="2010-03-12T12:33:01Z"
    sender="MyAgent" version="1.1.0.10" bufferSize="131000"
    instanceId="1383839" />
  <Error errorCode="OUT_OF_RANGE" >Argument was out of
    range</Error>
</MTConnectError>
```

Appendix A

Bibliography

- Engineering Industries Association. *EIA Standard* - *EIA-274-D*, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.
- ISO TC 184/SC4/WG3 N1089. *ISO/DIS 10303-238*: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.
- International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.
- International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.
- International Organization for Standardization. *ISO 6983/1* – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.
- Electronic Industries Association. *ANSI/EIA-494-B-1992*, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.
- National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.
- International Organization for Standardization. *ISO 10303-11*: 1994, Industrial automation systems and integration Product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.
- International Organization for Standardization. *ISO 10303-21*: 1996, Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.
- H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.
- International Organization for Standardization. *ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature.* Geneva, Switzerland, 2001.

- *ASME B5.59-2 Version 9c: Data Specification for Properties of Machine Tools for Milling and Turning. 2005.*
- *ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers. 2005.*
- OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.*
- View the following site for RFC references: http://www.faqs.org/rfcs/ .

Appendix B

Fundamentals of Using XML to Encode *Response Documents*

The MTConnect Standard specifies the structures and constructs that are used to encode *Response Documents*. When these *Response Documents* are encoded using XML, there are additional rules defined by the XML standard that apply for creating an XML compliant document. An implementer should refer to the W3C website for additional information on XML documentation and implementation details - http://www.w3.org/XML .

The following provides specific terms and guidelines referenced in the MTConnect Standard for forming *Response Documents* with XML:

- `Tag`: A tag is an XML construct that forms the foundation for an XML expression. It defines the scope (beginning and end) of an XML expression. The main types of tags are:
- `start-tag`: Designates the beginning on an XML element; e.g., <*Element Name*>
- `end-tag`: Designates the end on an XML element; e.g., </*Element Name*>.

  Note: If an element has no *Child Elements* or CDATA, the `end-tag` may be shortened to `/>`.

- `Element`: An element is an XML statement that is the primary building block for a document encoded using XML. An element begins with a `start-tag` and ends with a matching `end-tag`. The characters between the `start-tag` and the `end-tag` are the element's content. The content may contain attributes, CDATA, and/or other elements. If the content contains additional elements, these elements are called *Child Elements*.

  An example would be: <*Element Name*>Content of the Element</*Element Name*>.

- *Child Element*: An XML element that is contained within a higher-level *Parent Element*. A *Child Element* is also known as a sub-element. XML allows an unlimited hierarchy of *Parent-Child Element* relationships that establishes the structure that defines how the various pieces of information in the document relate to each other. A *Parent Element* may have multiple associated *Child Elements*.
- *Element Name*: A descriptive identifier contained in both the `start-tag` and `end-tag` that provides the name of an XML element.
- `Attribute`: A construct consisting of a name–value pair that provides additional information about that XML element. The format for an attribute is `name="value"`; where the value for the attribute is enclosed in a set of quotation (") marks. An XML attribute **MUST** only have a single value and each attribute can appear at most once in each element. Also, each attribute **MUST** be defined in a *schema* to either be required or optional.

- An example of attributes for an XML element are:

```
<DataItem category="SAMPLE" id="S1load"nativeUnits="PERCENT"
  type="LOAD" units="PERCENT"/>
```

In this example, `DataItem` is the `Element Name`. `category`, `id`, `nativeUnits`, `type`, and `units` are the names of the attributes. "`SAMPLE`", "`S1load`", "`PERCENT`", "`LOAD`, and "`PERCENT`" are the values for each of the respective attributes.

- CDATA: CDATA is an XML term representing *Character Data*. *Character Data* contains a value(s) or text that is associated with an XML element. CDATA can be restricted to certain formats, patterns, or words.

An example of CDATA associated with an XML element would be:

```
<Message id="M1">This is some text</Message>
```

In this example, `Message` is the `Element Name` and `This is some text` is the CDATA.

- *namespace*: An XML *namespace* defines a unique vocabulary for named elements and attributes in an XML document. An XML document may contain content that is associated with multiple *namespaces*. Each *namespace* has its own unique identifier.

Elements and attributes are associated with a specific *namespace* by placing a prefix on the name of the element or attribute that associates that name to a specific *namespace*; e.g., `x:MyTarget` associates the element name `MyTarget` with the *namespace* designated by `x:` (the prefix).

namespaces are used to avoid naming conflicts within an XML document. The naming convention used for elements and attributes may be associated with either the default *namespace* specified in the *header* of an XML document or they may be associated with one or more alternate namespaces. All elements or attributes associated with a *namespace* that is not the default namespace, must include a prefix (e.g., x:) as part of the name of the element or attribute to associate it with the proper *namespace*. See *Appendix C* for details on the structure for XML *headers*.

The names of the elements and attributes declared in a *namespace* may be identified with a different prefix than the prefix that signifies that specific *namespace*. These prefixes are called *namespace aliases*. As an example, MTConnect Standard specific *namespaces* are designated as `m:` and the names of the elements and attributes defined in that *namespace* have an *alias* prefix of `mt:` which designates these names as MTConnect Standard specific vocabulary; e.g., `mt:MTConnectDevices`.

XML documents are encoded with a hierarchy of elements. In general, XML elements may contain *Child Elements*, CDATA, or both. However, in the MTConnect Standard, an element **MUST NOT** contain mixed content; meaning it cannot contain both *Child Elements* and CDATA.

The *semantic data model* defined for each *Response Document* specifies the elements and *Child Elements* that may appear in a document. The *semantic data model* also defines the number of times each element and *Child Element* may appear in the document.

The following example demonstrates the hierarchy of XML elements and *Child Elements* used to form an XML document:

1. <*Root Level*> (*Parent Element*)
2. <*First Level*> (*Child Element* to *Root Level* and *Parent Element* to *Second Level*)
3. <*Second Level*> (*Child Element* to *First Level* and *Parent Element* to *Third Level*)
4. <*Third Level* name="N1"></*Third Level*> (*Child Element* to *Second Level*)
5. <*Third Level* name="N2"></*Third Level*> (*Child Element* to *Second Level*)
6. <*Third Level* name="N3"></*Third Level*> (*Child Element* to *Second Level*)
7. </*Second Level*> (`end-tag` for *Second Level*)
8. </*First Level*> (`end-tag` for *First Level*)
9. </*Root Level*> (`end-tag` for *Root Level*)

In the above example, *Root Level* and *First Level* have one *Child Element* (sub-elements) each and *Second Level* has three *Child Elements*; each called *Third Level*. Each *Third Level* element has a different `name` attribute. Each level in the structure is an element and each lower level element is a *Child Element*.

Appendix C

Schema and Namespace Declaration Information

There are four pseudo-attributes typically included in the *Header* of a *Response Document* that declare the *schema* and *namespace* for the document. Each of these pseudo-attributes provides specific information for a client software application to properly interpret the content of the *Response Document*.

The pseudo-attributes include:

- `xmlns:xsi` – The `xsi` portion of this attribute name stands for *XML Schema Instance*. An *XML Schema Instance* provides information that may be used by a software application to interpret XML specific information within a document. See the W3C website for more details on `xmlns:xsi`.

- `xmlns` – Declares the default *namespace* associated with the content of the *Response Document*. The default *namespace* is considered to apply to all elements and attributes whenever the name of the element or attribute does not contain a prefix identifying an alternate *namespace*.

  The value of this attribute is an URN identifying the name of the file that defines the details of the *namespace* content. This URN provides a unique identify for the *namespace*.

- `xmlns:m` – Declares the MTConnect specific *namespace* associated with the content of the *Response Document*. There may be multiple *namespaces* declared for an XML document. Each may be associated to the default *namespace* or it may be totally independent. The `:m` designates that this is a specific MTConnect *namespace* which is directly associated with the default *namespace*.

  Note: See *Section 6.7, Extensibility* for details regarding extended *namespaces*.

  The value associated with this attribute is an URN identifying the name of the file that defines the details of the *namespace* content.

- `xsi:schemaLocation` - Declares the name for the *schema* associated with the *Response Document* and the location of the file that contains the details of the *schema* for that document.

  The value associated with this attribute has two parts:

  - A URN identifying the name of the specific *XML Schema Instance* associated with the *Response Document*.

  - The path to the location where the file describing the specific *XML Schema Instance* is located. If the file is located in the same root directory where the *MTConnect Agent* is installed, then the local path **MAY** be declared. Otherwise, a fully qualified URL must be declared to identify the location of the file.

  Note: In the format of the value associated with `xsi:schemaLocation`, the URN and the path to the *schema* file **MUST** be separated by a "space".

In the following example, the first line is the *XML Declaration*. The second line is a *Root Element* called `MTConnectDevices`. The remaining four lines are the pseudo-attributes of `MTConnectDevices` that declare the XML *schema* and *namespace* associated with an *MTConnectDevices Response Document*.

```
<?xml version="1.0" encoding="UTF-8"?>
  <MTConnectDevices
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns="urn:mtconnect.org:MTConnectDevices:1.3"
    xmlns:m="urn:mtconnect.org:MTConnectDevices:1.3"
    xsi:schemaLocation="urn:mtconnect.org:
      MTConnectDevices:1.3 /schemas/MTConnectDevices_1.3.xsd">
```

The format for the values provided for each of the pseudo-attributes **MUST** reference the *semantic data model* (e.g., `MTConnectDevices`, `MTConnectStreams`, `MTConnectAssets`, or `MTConnectError`) and the version (i.e.; `1.1`, `1.2`, `1.3`, etc.) of the MTConnect Standard that depict the *schema* and *namespace(s)* associated with a specific *Response Document*.

When an implementer chooses to extend an *MTConnect Data Model* by adding custom data types or additional *Structural Elements*, the `schema` and `namespace` for that *Data Model* should be updated to reflect the additional content. When this is done, the `namespace` and `schema` information in the *Header* should be updated to reflect the URI for the extended `namespace` and `schema`.

MTConnect® Standard
Part 2.0 - Devices Information Model

Version 1.4.0

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

Table of Contents

Table of Figures

1 Purpose of This Document

This document, *Part 2.0 – Devices Information Model* of the MTConnect® Standard, establishes the rules and terminology to be used by designers to describe the function and operation of a piece of equipment and to define the data that is provided by an *MTConnect Agent* from the equipment. The *Devices Information Model* also defines the structure for the XML document that is returned from a *MTConnect Agent* in response to a *Probe Request*.

In the MTConnect Standard, *equipment* represents any tangible property that is used in the operations of a manufacturing facility. Examples of *equipment* are machine tools, ovens, sensor units, workstations, software applications, and bar feeders.

Note: See *Part 3.0 – Streams Information Model* of the MTConnect Standard for details on the XML documents that are returned from a *MTConnect Agent* in response to a *Sample* or *Current* Request.

2 Terminology and Conventions

Refer to *Section 2* of *Part 1.0 – Overview and Functionality* for a dictionary of terms, reserved language, and document conventions used in the MTConnect Standard.

3 Devices Information Model

The *Devices Information Model* represents the physical and logical configuration for a piece of equipment used for a manufacturing process or for any other purpose. It also provides the definition of data that may be reported by that equipment.

Using information defined in the *Devices Information Model*, a software application can determine the configuration and reporting capabilities of a piece of equipment. To do this, the software application issues a *Probe Request* (defined in *Section 8.1.1* of *Part 1.0 – Overview and Functionality* of the MTConnect Standard) to a *MTConnect® Agent* associated with a piece of equipment. A *MTConnect Agent* responds to the *Probe Request* with an `MTConnectDevices` XML document that contains information describing both the physical and logical structure of the piece of equipment and a detailed description of each *Data Entity* that can be reported by the *Agent* associated with the piece of equipment. This information allows the client software application to interpret the document and to extract the data with the same meaning, value, and context that it had at its original source.

The `MTConnectDevices` XML document is comprised of two sections: `Header` and `Devices.`

The `Header` section contains protocol related information as defined in *Section 6.5.1* of *Part 1.0 – Overview and Functionality* of the MTConnect Standard.

The `Devices` section of the `MTConnectDevices` document contains a `Device` XML container for each piece of equipment described in the document. Each `Device` container is comprised of two primary types of XML elements – *Structural Elements* and *Data Entities*.

Structural Elements are defined as XML elements that organize information that represents the physical and logical parts and sub-parts of a piece of equipment (See *Section 4* of this document for more details).

Data Entities are defined as XML elements that describe data that can be reported by a piece of equipment. In the *Devices Information Model*, *Data Entities* are defined as `DataItem` elements (See *Section 7* and *8* of this document).

The *Structural Elements* and *Data Entities* in the `MTConnectDevices` document provide information representing the physical and logical structure for a piece of equipment and the types of data that the piece of equipment can report relative to that structure. The `MTConnectDevices` document does not contain values for the data types reported by the piece of equipment. The `MTConnectStreams` document defined in *Part 3.0 – Streams Information Model* provides the data values that are reported by the piece of equipment. As such, most *Structural Elements* and *Data Entities* in the `MTConnectDevices` document do not contain `CDATA`. XML elements that provide values or information in the CDATA will be specifically identified in *Sections 4*, *7*, and *9* of this document.

Note: The MTConnect Standard also defines the information model for `Assets`. An *Asset* is something that is used in the manufacturing process, but is not permanently associated with a single piece of equipment, can be removed from the piece of equipment without compromising its function, and can be associated with other pieces of equipment during its lifecycle. See *Part 4.0 – Assets* of the MTConnect Standard for more details on *Assets*.

4 *Structural Elements* for `MTConnectDevices`

Structural Elements are XML elements that form the logical structure for the `MTConnectDevices` XML document. These elements are used to organize information that represents the physical and logical architecture of a piece of equipment. Refer to *Figure 1* below for an overview of the *Structural Elements* used in an `MTConnectDevices` document.

A variety of *Structural Elements* are defined to describe a piece of equipment. Some of these elements **MUST** always appear in the `MTConnectDevices` XML document, while others are optional and **MAY** be used, as required, to provide additional structure.

The first, or highest level, *Structural Element* in a `MTConnectDevices` XML document is `Devices`. `Devices` is a container type XML element used to group one or more pieces of equipment into a single XML document. `Devices` **MUST** always appear in the `MTConnectDevices` document.

`Device` is the next *Structural Element* in the `MTConnectDevices` XML document. `Device` is also a container type XML element. A separate `Device` container is used to identify each piece of equipment represented in the `MTConnectDevices` document. Each `Device` container provides information on the physical and logical structure of the piece of equipment and the data associated with that equipment. `Device` can also represent any logical grouping of pieces of equipment that function as a unit or any other data source that provides data through a *MTConnect Agent*.

One or more `Device` element(s) **MUST** always appear in an `MTConnectDevices` document.

`Components` is the next *Structural Element* in the `MTConnectDevices` XML document. `Components` is also a container type XML element. `Components` is used to group information describing *Lower Level* physical parts or logical functions of a piece of equipment.

If the `Components` container appears in the XML document, it **MUST** contain one or more `Component` type XML elements.

`Component` is the next level of *Structural Element* in the `MTConnectDevices` XML document. `Component` is both an abstract type XML element and a container type element.

As an abstract type element, `Component` will never appear in the XML document describing a piece of equipment and will be replaced by a specific `Component` type defined in *Section 5*. Each `Component` type is also a container type element. As a container, the `Component` type element is used to organize information describing *Lower Level Structural Elements* or *Data Entities* associated with the `Component`.

If *Lower Level Structural Elements* are described, these elements are by definition child `Component` elements of a parent `Component`. At this next level, the *Lower Level* child `Component` elements are grouped into an XML container called `Components`.

This *Lower Level* `Components` container is comprised of one or more child `Component` XML elements representing the sub-parts of the parent `Component`. Just like the parent `Component` element, the child `Component` element is an abstract type XML element and will never appear in the XML document – only the different *Lower Level* child `Component` types will appear.

This parent-child relationship can continue to any depth required to fully define a piece of equipment.

The following example is an XML document structure that demonstrates the relationship between a parent `Component` and *Lower Level* child components:

```
<Devices>
  <Device>
    <Components>
      <Axes>   (Parent component)
        <Components>
          <Rotary>   (Child component to Axes and Parent component to
                              Lower Level components)
            <Components>
              <Chuck>   (Child component to Rotary)
```

The following XML Tree demonstrates the various *Structural Elements* provided to describe a piece of equipment and the relationship between these elements.

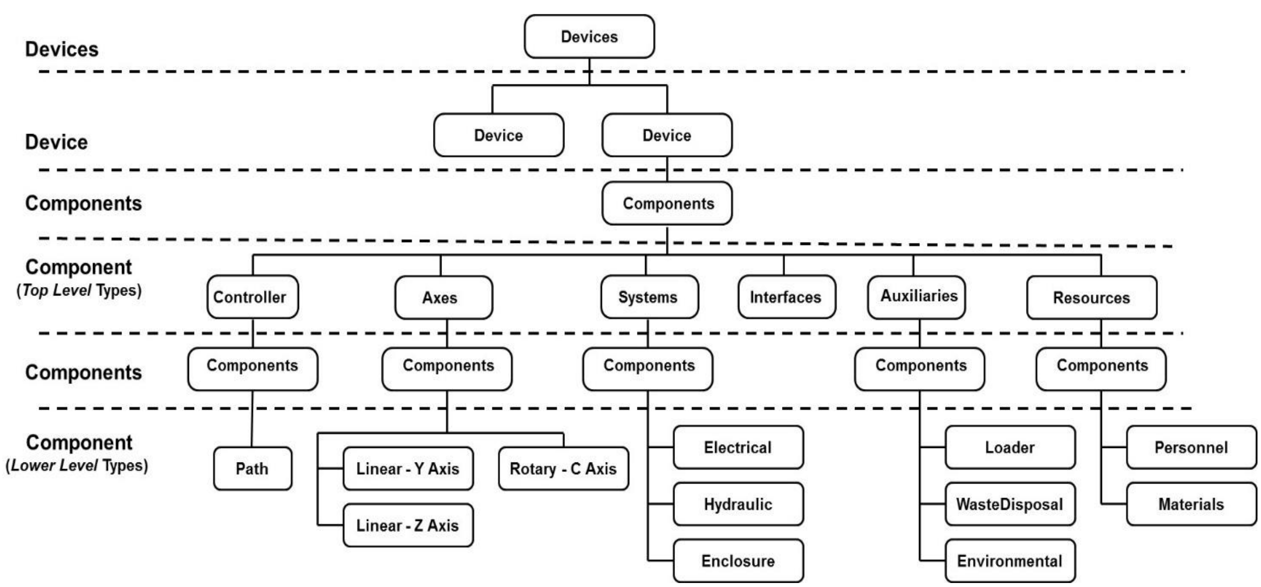


Figure 1: Example `Device` *Structural Elements*

Component type XML elements **MAY** be further decomposed into Composition type XML elements. Composition elements describe the lowest level basic structural or functional building blocks contained within a Component. Any number of Composition elements **MAY** be used. Data provided for a Component provides more specific meaning when it is associated with one of the Composition elements of the Component. The different Composition types that **MAY** appear in the XML document are defined in *Section 6*.

The Composition elements are organized into a Compositions container. The Compositions container **MAY** appear in the XML document further describing a Component. If one or more Composition element(s) is provided to describe a Component, a Compositions container **MUST** be defined for the Component.

The following illustration represents an XML document structure that demonstrates the relationship between a parent Component and its Composition elements:

```
<Devices>
 <Device>
   <Components>
    <Axes> (Component)
     <Components>
       <Linear> (Component)
        <Compositions>
          <Composition>
          <Composition>
          <Composition>
```

The following XML Tree demonstrates this relationship between a `Component` and some of its potential `Composition` elements.

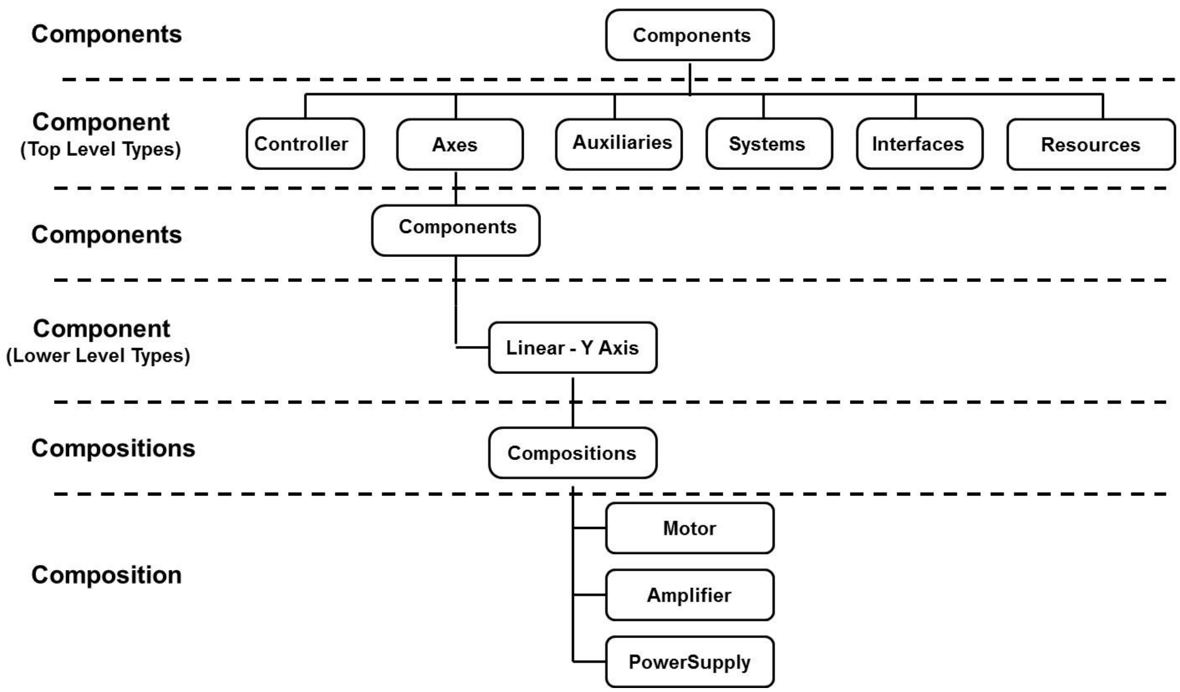


Figure 2: Example `Composition` *Structural Elements*

4.1 Devices

`Devices` is a container type XML element that **MUST** contain only `Device` elements. `Devices` **MUST** contain at least one `Device` element, but **MAY** contain multiple `Device` elements. *Data Entities* **MAY NOT** be directly associated with the `Devices` container.

Element	Description	Occurrence
`Devices`	The first, or highest level, *Structural Element* in a `MTConnectDevices` document. `Devices` is a container type XML element.	1

4.2 Device

`Device` is an XML container type element that organizes the *Structural Elements* and *Data Entities* associated with a piece of equipment. *Data Entities* **MAY** be directly associated with the `Device` container. `Device` **MUST** provide the data item `AVAILABILITY`, which represents the *Agent's* ability to communicate with the data source.

In the `MTConnectDevices` XML document, `Device` is a unique type of *Structural Element*. `Device` carries all of the properties of a `Component` (see *Section 4.4*). Additionally, `Device` **MUST** have a `uuid` attribute that uniquely identifies the piece of equipment. The value for the `uuid` **SHOULD NOT** change over time. The value for `uuid` **MUST** be universally unique and **MUST** only appear once in any MTConnect installation. All *Structural Elements* and *Data Entities* associated with a piece of equipment are therefore uniquely identified through their association with the `Device` container.

Element	Description	Occurrence
`Device`	The primary container element for each piece of equipment. `Device` is organized within the `Devices` container. There **MAY** be multiple `Device` elements in an XML document.	1..INF

Note: Some data sources may not be integral to a specific piece of equipment. These data sources may function independently or produce data that is not relevant to a specific piece of equipment. An example would be a temperature sensor installed in a plant to monitor the ambient air temperature. In such a case, these individual data sources, if they singularly or together perform a unique function, **MAY** be modeled in a MTConnect XML document as a `Device`. When modeled as a `Device`, these data sources **MUST** provide all of the data and capabilities defined for a `Device`.

It is possible for a piece of equipment to be defined as both a `Component` of a `Device` and simultaneously function independently as a separate `Device` reporting data directly through a *MTConnect Agent* using its own `uuid`. An example would be a temperature monitoring system that is defined as a `Device` reporting data about the environment within a facility and simultaneously reporting data for a `Component` of another piece of equipment that it is monitoring.

4.2.1 XML Schema Structure for `Device`

The following XML schema represents the structure of the `Device` XML element showing the attributes defined for `Device` and the elements that may be associated with `Device`.

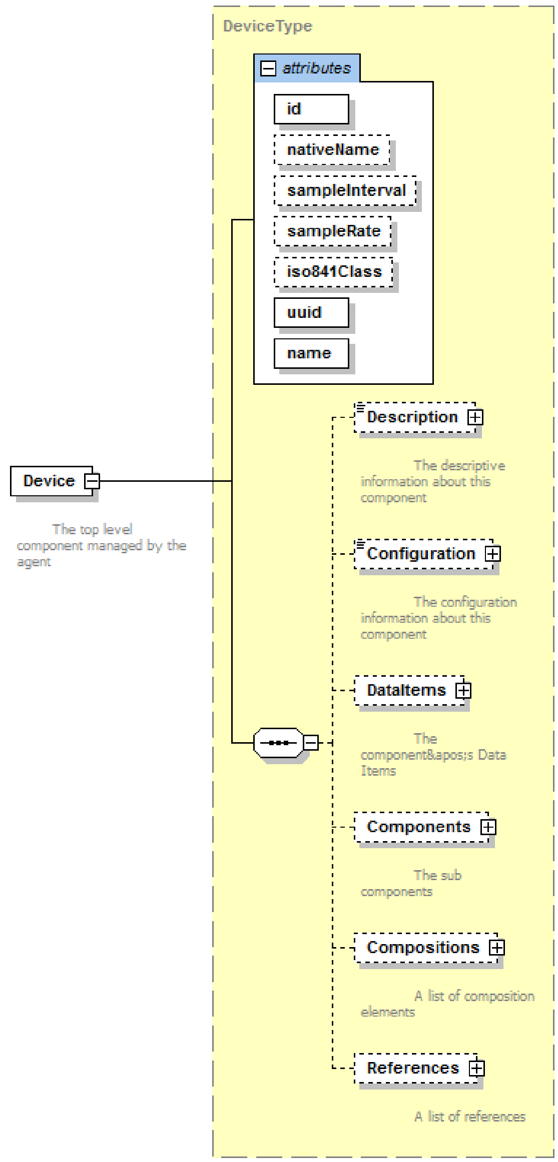


Figure 3: `Device` Schema Diagram

4.2.2 Attributes for `Device`

The following table defines the attributes that may be used to provide additional information for a `Device` type element.

Attribute	Description	Occurrence
`id`	The unique identifier for this XML element. `id` is a required attribute. An `id` **MUST** be unique across all the `id` attributes in the document. An XML ID-type.	1
`nativeName`	The common name normally associated with this piece of equipment. `nativeName` is an optional attribute.	0..1
`sampleInterval`	An optional attribute that is an indication provided by a piece of equipment describing the interval in milliseconds between the completion of the reading of the data associated with the `Device` element until the beginning of the next sampling of that data. This indication is reported as the number of milliseconds between data captures. This information may be used by client software applications to understand how often information from a piece of equipment is expected to be refreshed. The refresh rate for all data from the piece of equipment will be the same as for the `Device` element unless specifically overridden by another `sampleInterval` provided for a `Component` of the `Device` element. If the value of `sampleInterval` is less than one millisecond, the value will be represented as a floating-point number. For example, an interval of 100 microseconds would be 0.1.	0..1**
~~`sampleRate`~~	**DEPRECATED** in *MTConnect Version. 1.2*. Replaced by `sampleInterval`.	0..1***
~~`iso841Class`~~	**DEPRECATED** in *MTConnect Version 1.1.*	0..1***

Attribute	Description	Occurrence
`uuid`	A unique identifier for this XML element. `uuid` is a required attribute. The `uuid` **MUST** be unique amongst all `uuid` identifiers used in an MTConnect installation. For example, this may be a combination of the manufacturer's code and serial number. The `uuid` **SHOULD** be alphanumeric and not exceed 255 characters. An NMTOKEN XML type.	1*
`name`	The name of the piece of equipment represented by the `Device` element. `name` is a required attribute. This name **MUST** be unique for each `Device` XML element defined in the `MTConnectDevices` document. An NMTOKEN XML type.	1

Notes:* A `uuid` **MUST** be provided for each `Device` element. It is optional for all other Structural Elements.

** The sampleInterval is used to aid a client software application in interpreting values provided by some Data Entities. This is the desired sample interval and may vary depending on the capabilities of the piece of equipment.

*** Remains in schema for backwards compatibility.

4.2.3 Elements for `Device`

The following table lists the elements defined to provide additional information for a `Device` element. These elements are organized in the `Device` container.

Element	Description	Occurrence
`Description`	An XML element that can contain any descriptive content.	0..1
`Configuration`	An XML element that contains technical information about a piece of equipment describing its physical layout or functional characteristics.	0..1
`DataItems`	A container for the *Data Entities* (See *Section 7 and 8* of this document for more details) provided by this `Device` element.	1 *
`Components`	A container for the `Component` elements associated with this `Device` element.	0..1
`Compositions`	A container for the `Composition` elements associated with this `Device` element.	0..1
`References`	A container for the `Reference` elements associated with this `Device` element.	0..1

Note: * `DataItems` **MUST** be provided since every piece of equipment **MUST** report `AVAILABILITY`.

4.2.3.1 `Description` for `Device`

The following XML schema represents the structure of the `Description` XML element showing the attributes defined for `Description`. `Description` can contain any descriptive content for this piece of equipment. This element is defined to contain mixed content and additional XML elements (indicated by the `any` element in the schema below) **MAY** be added to extend the schema for `Description`.

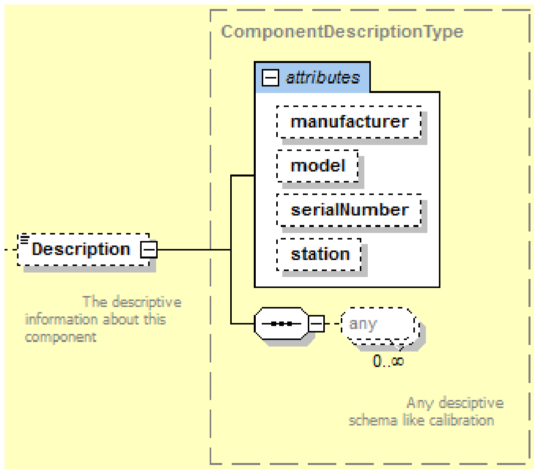


Figure 4: `Description` Schema Diagram

The following table lists the attributes defined for the Description XML element.

Attribute	Description	Occurrence
manufacturer	The name of the manufacturer of the piece of equipment represented by the Device element. manufacturer is an optional attribute.	0..1
model	The model description of the piece of equipment represented by the Device element. model is an optional attribute.	0..1
serialNumber	The serial number associated with piece of equipment represented by the Device element. serialNumber is an optional attribute.	0..1
station	The station where the equipment represented by the Device element is located when it is part of a manufacturing unit or cell with multiple stations. station is an optional attribute.	0..1

The content of Description **MAY** include any additional descriptive information the implementer chooses to include regarding a piece of equipment. This content **SHOULD** be limited to information not included elsewhere in the MTConnectDevices XML document.

An example of a Description is as follows:

```
<Description manufacturer="Example Co" serialNumber="A124FFF"
  station="2"> Example Co Simulated Vertical 3 Axis Machining center.
</Description>
```

4.2.3.2 Configuration for Device

The Configuration XML element contains technical information about a piece of equipment. Configuration **MAY** include any information describing the physical layout or functional characteristics of the piece of equipment, such as capabilities, testing, installation, operation, calibration, or maintenance.

Not all types of equipment support `Configuration`. When `Configuration` is supported, details on the schema for `Configuration` will be included in the applicable sections of the MTConnect Standard.

Element	Description	Occurrence
`Configuration`	An XML element that contains technical information about a piece of equipment describing its physical layout or functional characteristics.	0..1

Configuration data for `Device` is structured in the `MTConnectDevices` XML document as shown below. `AbstractConfiguration` is an abstract type XML element. It will never appear in the XML document representing a piece of equipment. When `Configuration` is supported for a type of equipment, that configuration will appear in the XML document. Currently, `Sensor` is the only type of equipment that supports `Configuration`. `SensorConfiguration` is described in detail in *Section 9.4*.

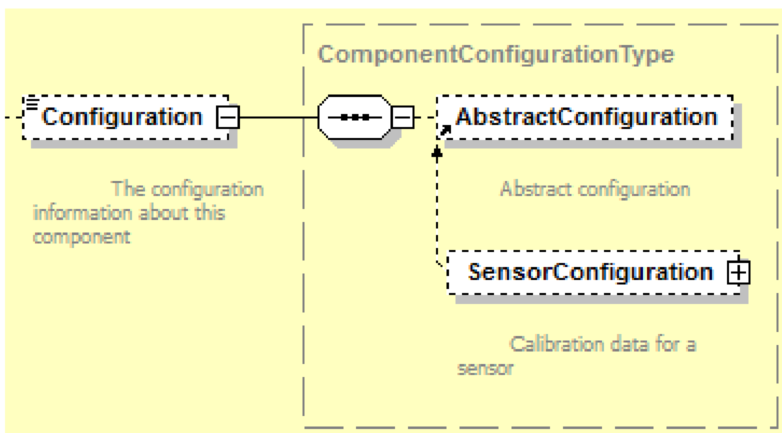


Figure 5: `Configuration` Schema Diagram

4.2.3.3 `DataItems` for `Device`

`DataItems` is an XML container that provides structure for organizing the data reported by a piece of equipment that is associated with the `Device` element.

`DataItems` **MUST** be provided since every piece of equipment **MUST** report the data item `AVAILABILITY`.

See *Sections 7 and 8* of this document for details on the `DataItems` XML element.

4.2.3.4 `Components` within `Device`

The use of the XML container `Components` within a `Device` element provides the ability to break down the structure of a `Device` element into *Top Level* and *Lower Level* physical and logical sub-parts. If a `Components` XML element is provided, then only one `Components` element **MUST** be defined for a `Device` element.

4.2.3.5 `Compositions` for `Device`

`Compositions` is an XML container used to organize `Composition` elements associated with a `Device` element. See *Section 4.5* for details on `Compositions`.

4.2.3.6 `References` for `Device`

`References` is an XML container used to organize `Reference` elements associated with a `Device` element. See *Section 4.7* for details on `References`.

4.3 Components

`Components` is an XML container used to group information describing physical parts or logical functions of a piece of equipment. `Components` contains one or more `Component` XML elements.

Element	Description	Occurrence
`Components`	XML container that consists of one or more types of `Component` XML elements. If a `Components` XML element is provided, then only one `Components` element **MUST** be defined for a `Device` element.	0..1

4.4 Component

A Component XML element is a container type XML element used to organize information describing a physical part or logical function of a piece of equipment. It also provides structure for describing the *Lower Level Structural Elements* associated with the Component. Component is an abstract type XML element and will never appear directly in the MTConnect XML document. As an abstract type XML element, Component will be replaced in the XML document by specific Component types. XML elements representing Component are described in *Section 5* and include elements such as Axes, Controller, and Systems.

Element	Description	Occurrence
Component	An abstract XML element. Replaced in the XML document by types of Component elements representing physical parts and logical functions of a piece of equipment. There can be multiple types of Component XML elements in the document.	1..INF

4.4.1 XML Schema Structure for `Component`

The following XML schema represents the structure of a `Component` XML element showing the attributes defined for `Component` and the elements that **MAY** be associated with `Component`.

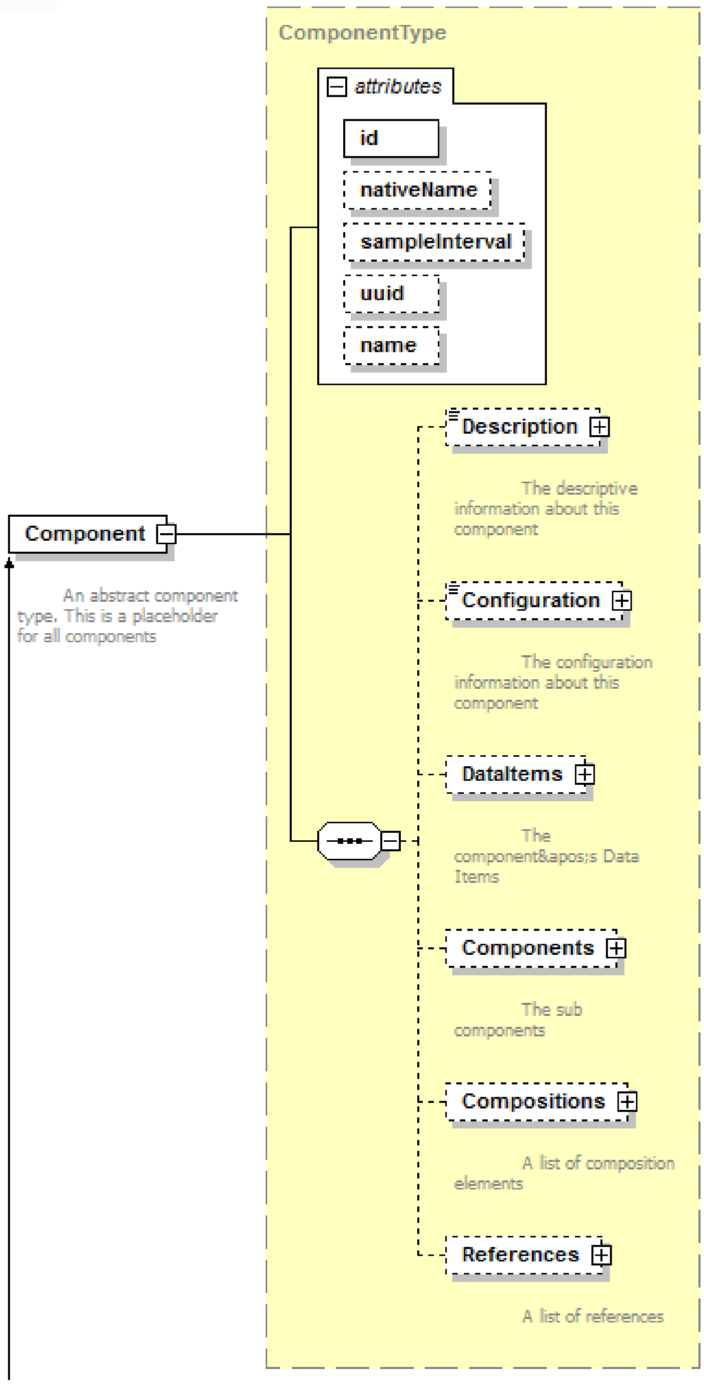


Figure 6: `Component` Schema

4.4.2 Attributes for `Component`

The following table defines the attributes that may be used to provide additional information for a `Component` type XML element.

Attribute	Description	Occurrence
`id`	The unique identifier for this XML element. `id` is a required attribute. An `id` **MUST** be unique across all the `id` attributes in the document. An XML ID-type.	1
`nativeName`	The common name normally associated with a specific physical or logical part of a piece of equipment. `nativeName` is an optional attribute.	0..1
`sampleInterval`	An optional attribute that is an indication provided by a piece of equipment describing the interval in milliseconds between the completion of the reading of the data associated with the `Component` element until the beginning of the next sampling of that data. This indication is reported as the number of milliseconds between data captures. This information may be used by client software applications to understand how often information from a piece of equipment for a specific `Component` element is expected to be refreshed. The refresh rate for data from all *Lower Level* `Component` elements will be the same as for the parent `Component` element unless specifically overridden by another `sampleInterval` provided for the *Lower Level* `Component` element. If the value of `sampleInterval` is less than one millisecond, the value will be represented as a floating-point number. For example, an interval of 100 microseconds would be 0.1.	0..1**
~~`sampleRate`~~	**DEPRECATED** in *MTConnect Version 1.2*. Replaced by `sampleInterval`.	0..1***
`uuid`	A unique identifier for this XML element. `uuid` is an optional attribute. The `uuid` **MUST** be unique amongst all `uuid` identifiers used in an MTConnect installation. For example, this may be a combination of the manufacturer's code and serial number. The `uuid` **SHOULD** be alphanumeric and not exceed 255 characters. An NMTOKEN XML type.	0..1*

Attribute	Description	Occurrence
`name`	The name of the `Component` element. `name` is an optional attribute. However, if there are multiple *Lower Level* components that have the same parent and are of the same component type (example `Linear`), then the `name` attribute **MUST** be provided for all *Lower Level* components of the same element type to differentiate between the similar components. When provided, `name` **MUST** be unique for all *Lower Level* components of a parent `Component`. An NMTOKEN XML type.	0..1

Notes: * While `uuid` **MUST** be provided for the Device element, it is optional for Component elements.

** The sampleInterval is used to aid a client software application in interpreting values provided by some *Data Entities*. This is the desired sample interval and may vary depending on the capabilities of the piece of equipment.

***Remains in schema for backwards compatibility.

4.4.3 Elements of `Component`

The following table lists the elements defined to provide additional information for a `Component` type XML element.

Element	Description	Occurrence
`Description`	An element that can contain any descriptive content.	0..1
`Configuration`	An XML element that contains technical information about a component describing its physical layout or functional characteristics.	0..1
`DataItems`	A container for the *Data Entities* (defined in *Section 8*) associated with this `Component` element.	0..1*
`Components`	A container for *Lower Level* `Component` XML elements associated with this parent `Component`.	0..1*
`Compositions`	A container for the `Composition` elements (defined in *Section 6*) associated with this `Component` element.	0..1
`References`	A container for the `Reference` elements associated with this `Component` element.	0..1*

Notes: *At least one of `Components`, `DataItems`, or `References` **MUST** be provided.

4.4.3.1 Description for Component

The following XML schema represents the structure of the `Description` XML element showing the attributes defined for `Description`. `Description` can contain any descriptive content of this `Component`. This element is defined to contain mixed content and additional XML elements (indicated by the `any` element in the schema below) **MAY** be added to extend the schema for `Description`.

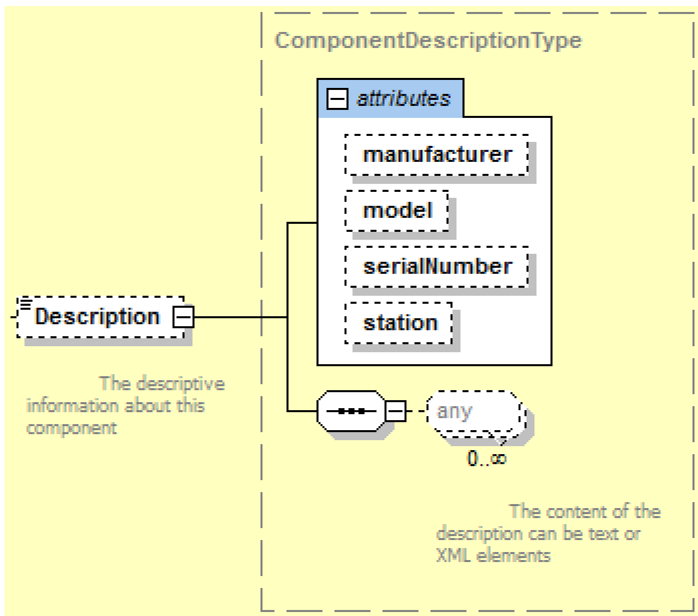


Figure 7: Schema for `Description` of `Component`

The following table lists the attributes defined for the `Description` XML element.

Attribute	Description	Occurrence
`manufacturer`	The name of the manufacturer of the physical or logical part of a piece of equipment represented by the `Component` element. `manufacturer` is an optional attribute.	0..1
`model`	The model description of the physical part or logical function of a piece of equipment represented by the `Component` element. `model` is an optional attribute.	0..1
`serialNumber`	The serial number associated with the physical part or logical function of a piece of equipment represented by the `Component` element. `serialNumber` is an optional attribute.	0..1
`station`	The station where the physical part or logical function of a piece of equipment represented by the `Component` element is located when it is part of a manufacturing unit or cell with multiple stations. `station` is an optional attribute.	0..1

The content of `Description` **MAY** include any additional descriptive information the implementer chooses to include regarding the `Component` element. This content **SHOULD** be limited to information not included elsewhere in the `MTConnectDevices` XML document.

An example of a `Description` element is as follows:

```
<Description manufacturer="Example Co"
  serialNumber="EXCO-TT-099PP-XXXX"> Advanced Pulse watt-hour transducer
  with pulse output
</Description>
```

4.4.3.2 Configuration for Component

The `Configuration` XML element contains technical information about a component. `Configuration` **MAY** include any information describing the physical layout or functional characteristics of a component, such as capabilities, testing, installation, operation, calibration, or maintenance.

Not all `Component` types support `Configuration`. When `Configuration` is supported, details on the schema for `Configuration` will be included in the applicable sections of the MTConnect Standard.

Element	Description	Occurrence
`Configuration`	An XML element that contains technical information about a component describing its physical layout or functional characteristics.	0..1

Configuration data for `Component` is structured in the `MTConnectDevices` XML document as shown below. `AbstractConfiguration` is an abstract type XML element. It will never appear in the XML document for a device. When `Configuration` is supported for a `Component` type, that configuration will appear in the XML document. Currently, `Sensor` is the only component type that supports `Configuration`. `SensorConfiguration` is described in detail in *Section 9.4*.

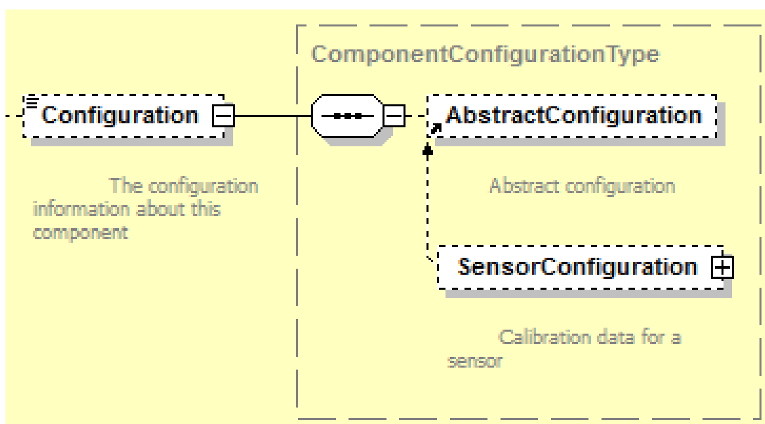


Figure 8: `Component` Configuration Schema

4.4.3.3 `DataItems` for `Component`

`DataItems` is an XML container that provides structure for organizing the data reported by a piece of equipment that is associated with the `Component`.

See *Section 7* of this document for details on the `DataItems` XML element.

4.4.3.4 `Components` within `Component`

The use of the XML container `Components` within a `Component` element provides the ability to further break down the structure of a `Component` element into even *Lower Level* physical and logical sub-parts. These *Lower Level* elements can add more clarity and granularity to the physical or logical structure of a piece of equipment and the data associated with that equipment.

This parent-child relationship can be extended down to any level necessary to fully describe a piece of equipment. These *Lower Level* `Component` elements use the same XML structure as `Component` defined in *Section 4.4.1* of this document.

A parent `Component` and the *Child Elements* are represented in a XML document as follows:

```
<Devices>
  <Device>
    <Components>
      <Axes> (Component)
        <Components>
          <Linear> (Component)
            <Components>
              <Etc. > (Component)
```

4.4.3.5 Compositions for Component

`Compositions` is an XML container used to organize the lowest level structural building blocks contained within a `Component` as defined below.

4.4.3.6 References for Component

`References` is an XML container used to organize `Reference` elements associated with a `Component` element. See *Section 4.7* for details on `References`.

4.5 Compositions

`Compositions` is an XML container that defines the lowest level structural building blocks contained within a `Component` element.

`Compositions` contains one or more `Composition` XML elements.

Element	Description	Occurrence
`Compositions`	XML Container consisting of one or more types of `Composition` XML elements. Only one `Compositions` container **MAY** appear for a `Component` element.	0..1

4.6 Composition

`Composition` XML elements are used to describe the lowest level physical building blocks of a piece of equipment contained within a `Component`.

Like `Component` elements, `Composition` elements provide the ability to organize information describing *Lower Level* sub-parts of a higher-level `Component` element. However, unlike `Component`, `Composition` **MUST NOT** be further sub-divided and *Data Entities* **MUST NOT** be assigned to `Composition` elements.

`Composition` elements are used to add more clarity and granularity to the data being retrieved from a piece of equipment. The meaning of the data associated with a `Component` may be enhanced by designating a specific `Composition` element associated with that data.

An example of the additional detail provided when using `Composition` elements would be:

> A `TEMPERATURE` associated with a `Linear` type axis may be further clarified by referencing the `MOTOR` or `AMPLIFIER` type `Composition` element associated with that axis, which differentiates the temperature of the motor from the temperature of the amplifier.

`Composition` is a typed XML element and will always define a specific type of structural building block contained within a `Component`. XML elements representing the types of `Composition` elements are described in *Section 6* of this document and include elements describing such basic building blocks as motors, amplifiers, filters, and pumps.

A parent `Component` and child `Composition` elements are represented in an XML document as follows:

```
<Devices>
 <Device>
   <Components>
     <Axes> (Component)
       <Components>
         <Linear> (Component)
           <Compositions>
             <Composition>
             <Composition>
             <Composition>
```

Element	Description	Occurrence
`Composition`	An XML element used to describe the lowest level structural building blocks contained within a `Component` element. `Composition` is a typed XML element. There can be multiple types of `Composition` XML elements defined for a `Component` element.	1..INF

4.6.1 XML Schema Structure for `Composition`

The following XML schema represents the structure of a `Composition` XML element showing the attributes defined for `Composition` and the elements that may be associated with `Composition` type XML elements.

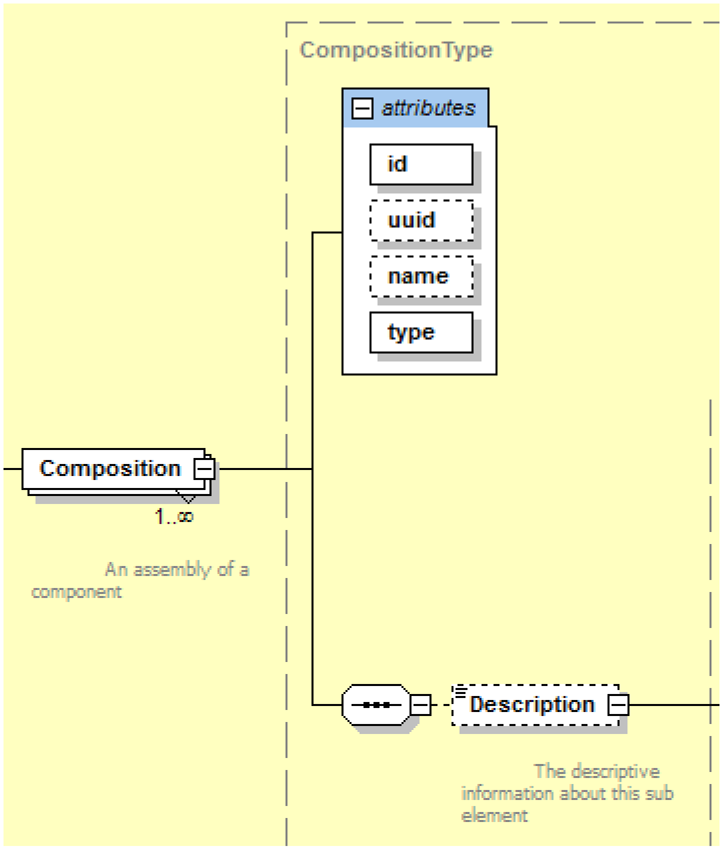


Figure 9: `Composition` Schema

4.6.2 Attributes for Composition

The following table defines the attributes that may be used to provide additional information for a Composition type XML element.

Attribute	Description	Occurrence
id	The unique identifier for this XML element. id is a required attribute. An id **MUST** be unique across all the id attributes in the document. An XML ID-type.	1
uuid	A unique identifier for this XML element. uuid is an optional attribute. The uuid **MUST** be unique amongst all uuid identifiers used in an MTConnect installation. For example, this may be a combination of the manufacturer's code and serial number. The uuid **SHOULD** be alphanumeric and not exceed 255 characters. An NMTOKEN XML type.	0..1
name	The name of the Composition element. name is an optional attribute. If provided, name **MUST** be unique within a Component element. An NMTOKEN XML type.	0..1
type	The type of Composition element. type is a required attribute. Examples of types are MOTOR, FILTER, PUMP, and AMPLIFIER. Refer to *Section 6* for a list of currently defined types.	1

4.6.3 Elements of `Composition`

The following table lists the elements defined to provide additional information for a `Composition` type XML element.

Element	Description	Occurrence
`Description`	An element that can contain any descriptive content.	0..1

4.6.3.1 `Description` for `Composition`

The following XML schema represents the structure of the `Description` XML element showing the attributes defined for `Description`. `Description` can contain any descriptive content for this `Composition` element. This element is defined to contain mixed content and additional XML elements (indicated by the `any` element in the schema below) **MAY** be added to extend the schema for `Description`.

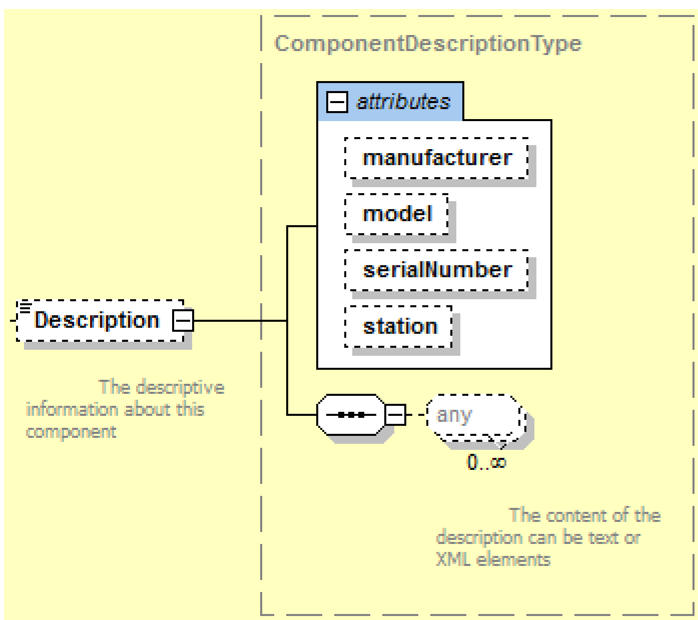


Figure 10: Schema for `Description` of `Composition`

The following table lists the attributes defined for the `Description` XML element.

Attribute	Description	Occurrence
`manufacturer`	The name of the manufacturer of the physical part of a piece of equipment represented by the `Composition` element. `manufacturer` is an optional attribute.	0..1
`model`	The model description of the physical part of a piece of equipment represented by the `Composition` element. `model` is an optional attribute.	0..1
`serialNumber`	The serial number associated with the physical part of a piece of equipment represented by the `Composition` element. `serialNumber` is an optional attribute.	0..1
`station`	The station where the physical part of a piece of equipment represented by the `Composition` element is located when it is part of a manufacturing unit or cell with multiple stations. `station` is an optional attribute.	0..1

The content of `Description` **MAY** include any additional descriptive information the implementer chooses to include regarding the `Composition` element. This content **SHOULD** be limited to information not included elsewhere in the `MTConnectDevices` XML document.

An example of a `Description` element is as follows:

```
<Description manufacturer="Example Co" serialNumber="A124FFF"
station="2"> Spindle motor associated with Path 2.
</Description>
```

4.7 References

`References` is an XML container that organizes pointers to information defined elsewhere within the XML document for a piece of equipment.

`References` may be modeled as part of a `Device`, `Component` or `Interface` type *Structural Element*.

References contains one or more Reference XML elements.

Element	Description	Occurrence
References	XML Container consisting of one or more types of Reference XML elements. Only one References container **MUST** appear for a Device, Component, or Interface element.	0..1

4.8 Reference

Reference is a pointer to information that is associated with another *Structural Element* defined elsewhere in the XML document for a piece of equipment. That information may be data from the other element or the entire structure of that element.

Reference is an efficient method to associate information with an element without duplicating any of the data or structure. For example, a Bar Feeder System may make a request for the BarFeederInterface and receive all the relevant data for the interface and the associated spindle (ROTARY element) that is referenced as part of the BarFeederInterface.

Reference is an abstract type XML element and will never appear directly in the MTConnect XML document. As an abstract type XML element, Reference will be replaced in the XML document by a specific Reference type. The current supported types of Reference are DataItemRef and ComponentRef XML elements.

The following XML schema represents the structure of the `Reference` XML element.

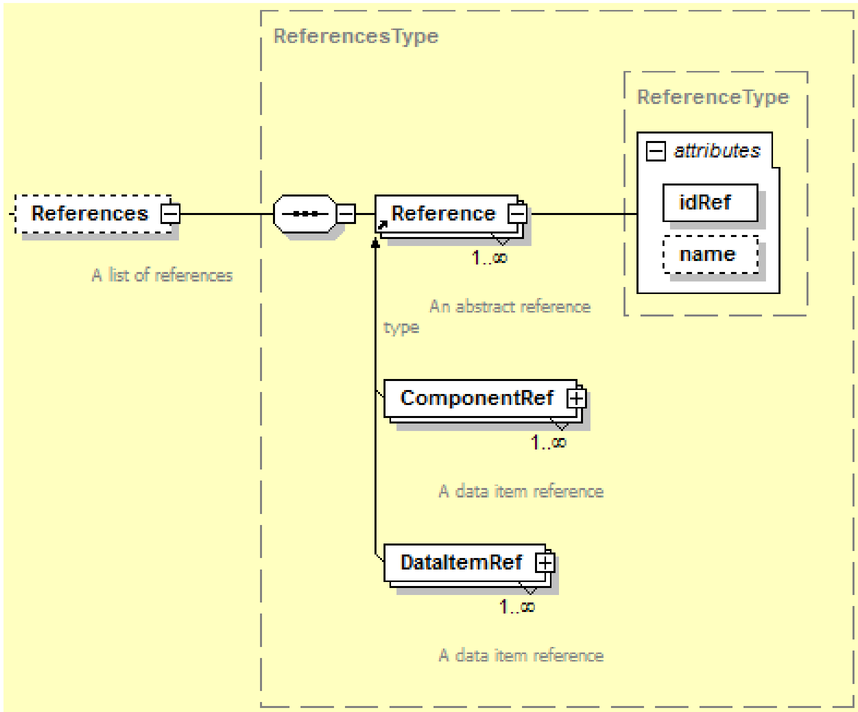


Figure 11: `Reference` Schema Diagram

4.8.1 `ComponentRef`

`ComponentRef` XML element is a pointer to all of the information associated with another *Structural Element* defined elsewhere in the XML document for a piece of equipment. `ComponentRef` allows all of the information (*Lower Level* `Components` and all *Data Entities*) that is associated with the other *Structural Element* to be directly associated with this XML element.

The following XML schema represents the structure of a `ComponentRef` XML element showing the attributes defined for `ComponentRef`.

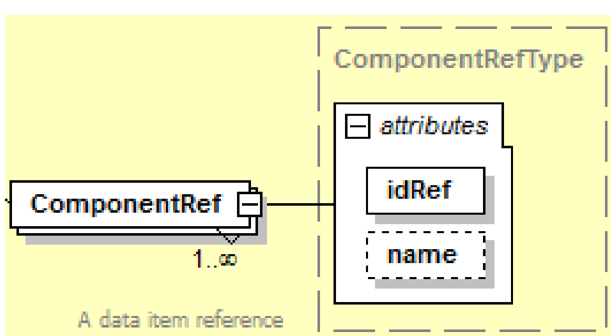


Figure 12: `ComponentRef` Schema Diagram

The following table lists the `attributes` defined for the `ComponentRef` element.

Attribute	Description	Occurrence
`idRef`	A pointer to the `id` attribute of the `Component` that contains the information to be associated with this XML element. `idRef` is a required attribute.	1
`name`	The name of the `ComponentRef` element. `name` is an optional attribute. However, if there are multiple `ComponentRef` elements defined for a component, the `name` attribute **MUST** be provided for all `ComponentRef` elements to differentiate between the similar elements. When provided, `name` **MUST** be unique for all `ComponentRef` elements associated with the *Parent Element*. An NMTOKEN XML type.	0..1

4.8.2 DataItemRef

`DataItemRef` XML element is a pointer to a *Data Entity* associated with another *Structural Element* defined elsewhere in the XML document for a piece of equipment. `DataItemRef` allows the data associated with a data item defined in another *Structural Element* to be directly associated with this XML element.

The following XML schema represents the structure of a `DataItemRef` XML element showing the attributes defined for `DataItemRef`.

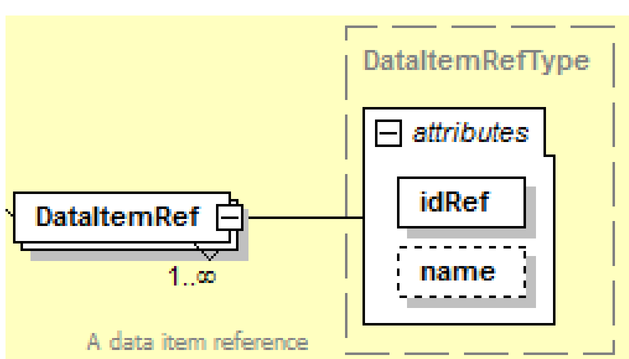


Figure 13: `DataItemRef` Schema Diagram

The following table lists the `attributes` defined for the `DataItemRef` element.

Attribute	Description	Occurrence
`idRef`	A pointer to the `id` attribute of the `DataItem` that contains the information to be associated with this XML element. `idRef` is a required attribute.	1
`name`	The name of the `DataItemRef` element. `name` is an optional attribute. However, if there are multiple `DataItemRef` elements defined for a component, the `name` attribute **MUST** be provided for all `DataItemRef` elements to differentiate between the similar elements. When provided, `name` **MUST** be unique for all `DataItemRef` elements associated with the *Parent Element*. An NMTOKEN XML type.	0..1

5 `Component` *Structural Elements*

`Component` *Structural Elements* are XML containers used to represent physical parts or logical functions of a piece of equipment.

`Component` *Structural Elements* are defined into two major categories:

- *Top Level* `Component` elements are used to group the *Structural Elements* representing the most significant physical or logical functions of a piece of equipment. The *Top Level* `Component` elements provided in an `MTConnectDevices` document **SHOULD** be restricted to those defined in the table below. However, these *Top Level* `Component` elements **MAY** also be used as *Lower Level* `Component` elements; as required.

- *Lower Level* `Component` elements are used to describe the sub-parts of the parent `Component` to provide more clarity and granularity to the physical or logical structure of the *Top Level* `Component` elements.

This section (*Section 5*) of the *Devices Information Model* provides guidance for the most common relationships between *Top Level* `Component` elements and *Lower Level* child components. However, all `Component` elements **MAY** be used in any configuration, as required, to fully describe a piece of equipment.

As described in *Section 4* above, `Component` is an abstract type *Structural Element* within the *Devices Information Model* and will never appear directly in the `MTConnectDevices` XML document. As abstract type XML elements, `Component` will be replaced in the XML document by a specific `Component` type defined below.

The following table defines the *Top Level* `Component` elements available to describe a piece of equipment.

Top Level Component* Element *	**Description**
`Axes`	An XML container used to organize the *Structural Elements* of a piece of equipment that perform linear or rotational motion.
`Controller`	An XML container used to organize information about an intelligent or computational function within a piece of equipment.
`Systems`	An XML container used to organize information for *Lower Level* elements representing the major sub-systems that are permanently integrated into a piece of equipment.
`Auxiliaries`	An XML container used to organize information for *Lower Level* elements representing functional sub-systems that provide supplementary or extended capabilities for a piece of equipment, but they are not required for the basic operation of the equipment.

Top Level Component Element **	Description
`Resources`	An XML container used to organize information for *Lower Level* elements representing types of items, materials, and personnel that support the operation of a piece of equipment or work to be performed at a location. `Resources` also represents materials or other items consumed or transformed by a piece of equipment for production of parts or other types of goods.
`Interfaces`	An XML container that organizes information used to coordinate actions and activities between pieces of equipment that communicate information between each other.

** Note: The following components have been relocated or redefined since they are not classified as restricted *Top Level* components:

- `Power` was **DEPRECATED** in *MTConnect Version 1.1* and was replaced by the *Data Entity* called `AVAILABILITY`.

- `Door has` been redefined as a *Lower Level* component of a parent `Component` element or as a `Composition` element.

- `Actuator`, due to its uniqueness, has been redefined as a piece of equipment with the ability to be represented as a *Lower Level* component of a parent `Component` element or as a `Composition` element.

- `Sensor`, due to its uniqueness, has been redefined as a piece of equipment with the ability to be represented as a *Lower Level* component of a parent `Component` element (See *Section 9* for further detail).

- `Stock` has been redefined as a *Lower Level* component of the `Resources` *Top Level* `Component` element.

The common relationship between the *Top Level* `Component` elements and the *Lower Level* child `Component` elements are described below. It should be noted that as the MTConnect Standard evolves, more `Component` types will be added to organize information for new types of equipment and/or new physical or logical sub-parts of equipment.

5.1 Axes

`Axes` is a *Top Level* `Component` element. It is a container that organizes information representing the *Structural Elements* that perform linear or rotational motion for a piece of equipment.

`Axes` organizes information for the individual physical axes into `Component` types of `Linear` and `Rotary` based on the type of motion performed by each axis. `Axes` **MUST** contain at least one `Linear` or one `Rotary` type axis.

The following diagram defines the relationship between the `Axes` container and the individual axis type *Structural Elements*.

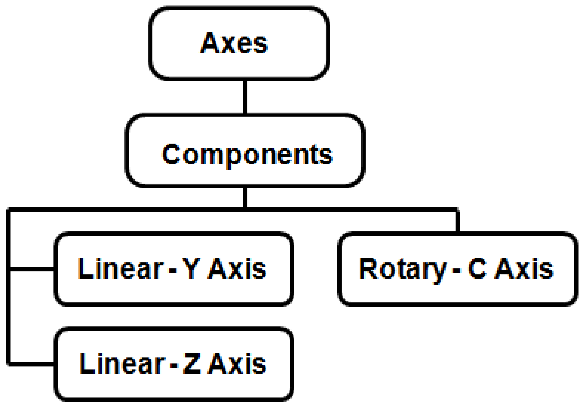


Figure 14: `Axes` Example with Two Linear Axes and One Rotary Axis

5.1.1 `Linear`

A `Linear` axis represents the movement of a physical piece of equipment, or a portion of the equipment, in a straight line.

Movement may be either in a positive or negative direction.

`Linear` type axes **MUST** be identified using a value for the `name` attribute as X, Y, or Z with numbers appended for additional axes in the same plane. Additional linear axes are often referred to as U, V, and W. However, MTConnect defines the secondary axes to X, Y, and Z as X2, Y2, and Z2.

If the piece of equipment is unable to provide information associated with the `name` attribute, then the `nativeName` attribute **MUST** be included to identify the axis.

5.1.2 `Rotary`

A `Rotary` axis represents any non-linear or rotary movement of a physical piece of equipment or a portion of the equipment.

`Rotary` type axes **MUST** be identified using a value for the `name` attribute as A, B, and C for axes that rotate around the X, Y, and Z axes respectively. As with the `Linear` axes, a number **MUST** be appended for additional axes in the same plane (C, C2, C3, C4, …).

If the piece of equipment is unable to provide information associated with the `name` attribute, then the `nativeName` attribute **MUST** be included to identify the axis.

An axis whose function is to provide rotary motion may function as a continuous rotation (`SPINDLE` mode), continuous-path contour rotary motion (`CONTOUR` mode), or positioning (`INDEX` mode) to discrete rotary positions. As such, a `Rotary` type axis **SHOULD** specify a `ROTARY_MODE` data item identifying the operating mode of the axis: `SPINDLE`, `INDEX`, or `CONTOUR`.

5.1.2.1 Chuck

`Chuck` is an XML container that provides the information about a mechanism that holds a part or stock material in place. It may also represent the information about any other type mechanism that holds items in place within a piece of equipment.

The operation of a `Chuck` when represented as a `Component` element is defined by `CHUCK_STATE`. The value of `CHUCK_STATE` **MAY** be `OPEN`, `CLOSED`, or `UNLATCHED`.

`Chuck` may be used in the `MTConnectDevices` document as either a *Lower Level* component or as a `Composition` element of a parent `Component` element.

5.2 Controller

`Controller` is a *Top Level* container that organizes information for an intelligent part of a piece of equipment that monitors and calculates information to alter the operating conditions of the equipment. Typical types of controllers for a piece of equipment include CNC (Computer Numerical Control), PAC (Programmable Automation Control), IPC (Industrialized Computer), or IC (Imbedded Computer).

`Controller` provides information regarding the execution of a control program(s), the mode of operation of the piece of equipment, and fault information regarding the operation of the equipment.

> Note: *MTConnect Version 1.1.0* and later implementations **SHOULD** use a *Lower Level* `Component` element called `Path` to represent an individual tool path or other independent function within a `Controller` element. When the `Controller` element is capable of executing more than one simultaneous and independent programs, the implementation **MUST** specify a *Lower Level* `Path` element representing each of the independent functions of the `Controller`.

5.2.1 Path

`Path` is an XML container that represents the information for an independent operation or function within a `Controller`. For many types of equipment, `Path` represents a set of `Axes`, one or more `Program` elements, and the data associated with the motion of a control point as it moves through space. However, it **MAY** also represent any independent function within a `Controller` that has unique data associated with that function.

`Path` **SHOULD** provide an `EXECUTION` data item to define the operational state of the `Controller` component of the piece of equipment.

If the Controller is capable of performing more than one independent operation or function simultaneously, a separate Path component **MUST** be used to organize the data associated with each independent operation or function.

5.3 Systems

Systems is a *Top Level* XML container that provides structure for the information describing one or more *Lower Level* functional systems that perform as discrete operating modules of the equipment or provide utility type services to support the operation of the equipment. These systems are required for the piece of equipment to perform its intended function and are permanently integrated into the piece of equipment.

Since these systems operate as separate functional units, they are represented in the MTConnectDevices XML document as individual *Lower Level* Component elements of Systems based on the function or service provided.

5.3.1 Hydraulic System

Hydraulic is an XML container that represents the information for a system comprised of all the parts involved in moving and distributing pressurized liquid throughout the piece of equipment.

5.3.2 Pneumatic System

Pneumatic is an XML container that represents the information for a system comprised of all the parts involved in moving and distributing pressurized gas throughout the piece of equipment.

5.3.3 Coolant System

Coolant is an XML container that represents the information for a system comprised of all the parts involved in distribution and management of fluids that remove heat from a piece of equipment.

5.3.4 Lubrication System

Lubrication is an XML container that represents the information for a system comprised of all the parts involved in distribution and management of fluids used to lubricate portions of the piece of equipment.

5.3.5 Electric System

Electric is an XML container that represents the information for the main power supply for device piece of equipment and the distribution of that power throughout the equipment. The electric system will provide all the data with regard to electric current, voltage, frequency, etc. that applies to the piece of equipment as a functional unit. Data regarding electric power that is specific to a Component will be reported as *Data Entities* for that specific Component.

5.3.6 Enclosure System

Enclosure is an XML container that represents the information for a structure used to contain or isolate a piece of equipment or area. The Enclosure system may provide information regarding access to the internal components of a piece of equipment or the conditions within the enclosure. For example, Door may be defined as a *Lower Level* Component or Composition element of the Enclosure system.

5.3.7 Protective System

Protective is an XML container that represents the information for those functions that detect or prevent harm or damage to equipment or personnel. Protective does not include the information relating to the Enclosure system.

5.3.8 ProcessPower System

ProcessPower is an XML container that represents the information for a power source associated with a piece of equipment that supplies energy to the manufacturing process separate from the Electric system. For example, this could be the power source for an EDM machining process, an electroplating line, or a welding system.

5.3.9 Feeder System

Feeder is an XML container that represents the information for a system that manages the delivery of materials within a piece of equipment. For example, this could describe the wire delivery system for an EDM or welding process; conveying system or pump and valve system distributing material to a blending station; or a fuel delivery system feeding a furnace.

5.3.10 Dielectric System

Dielectric is an XML container that represents the information for a system that manages a chemical mixture used in a manufacturing process being performed at that piece of equipment. For example, this could describe the dielectric system for an EDM process or the chemical bath used in a plating process.

5.4 Auxiliaries

Auxiliaries is a *Top Level* XML container that provides structure for the information describing one or more *Lower Level* functional systems that provide supplementary or additional capabilities for the operation of a piece of equipment. These systems extend the capabilities of a piece of equipment, but are not required for the equipment to function.

Since these systems operate as independent units or are only temporarily associated with a piece of equipment, they are represented in the MTConnectDevices XML document as individual *Lower Level* Component elements of Auxiliaries based on the function or service provided to the equipment.

5.4.1 `Loader` System

`Loader` is an XML container that represents the information for a unit comprised of all the parts involved in moving and distributing materials, parts, tooling, and other items to or from a piece of equipment.

5.4.2 `WasteDisposal` System

`WasteDisposal` is an XML container that represents the information for a unit comprised of all the parts involved in removing manufacturing byproducts from a piece of equipment.

5.4.3 `ToolingDelivery` System

`ToolingDelivery` is an XML container that represents the information for a unit involved in managing, positioning, storing, and delivering tooling within a piece of equipment.

5.4.4 `BarFeeder` System

`BarFeeder` is an XML container that represents the information for a unit involved in delivering bar stock to a piece of equipment.

5.4.5 `Environmental` System

`Environmental` is an XML container that represents the information for a unit or function involved in monitoring, managing, or conditioning the environment around or within a piece of equipment.

5.4.6 `Sensor` System

`Sensor` is a XML container that represents the information for a piece of equipment that responds to a physical stimulus and transmits a resulting impulse or value from a sensing unit. When modeled as a component of `Auxiliaries`, `sensor` **SHOULD** represent an integrated *sensor unit* system that provides signal processing, conversion, and communications. A *sensor unit* may have multiple *sensing elements*; each representing the data for a variety of measured values. See *Section 9.2* for more details on *sensor unit*.

> Note: If modeling an individual sensor, then `sensor` should be associated with the component that the measured value is most closely associated. See *Section 5.7.3*.

5.5 Resources

`Resources` is a *Top Level* XML container that groups items that support the operation of a piece of equipment. `Resources` also represents materials or other items consumed, transformed, or used for production of parts, materials, or other types of goods by a piece of equipment.

5.5.1 Materials

`Materials` is an XML container that provides information about materials or other items consumed or used by the piece of equipment for production of parts, materials, or other types of goods. `Materials` also represents parts or part stock that are present at a piece of equipment or location to which work is applied to transform the part or stock material into a more finished state.

5.5.1.1 Stock

`Stock` is an XML container that represents the information for the material that is used in a manufacturing process and to which work is applied in a machine or piece of equipment to produce parts.

`Stock` may be either a continuous piece of material from which multiple parts may be produced or it may be a discrete piece of material that will be made into a part or a set of parts.

5.5.2 Personnel

`Personnel` is an XML container that provides information about an individual or individuals who either control, support, or otherwise interface with a piece of equipment.

5.6 Interfaces

`Interfaces` is a *Top Level* XML *Structural Element* in the `MTConnectDevices` XML document. `Interfaces` organizes the information provided by a piece of equipment used to coordinate activities with other pieces of equipment. As such, `Interfaces` represents the inter-device communication information between a piece of equipment and other pieces of equipment.

See Part 5.0 – Interfaces of the MTConnect Standard for detailed information on `Interfaces`.

5.7 Other Components

While most component elements **SHOULD** be modeled in a specific manner, there are some types of component elements that are used ubiquitously in equipment and **MAY** be associated with any number of different types of *parent* component elements.

These components **MAY** be modeled as *Lower Level* components of the *Parent Element*.

5.7.1 Actuator

`Actuator` is an XML container that represents the information for an apparatus for moving or controlling a mechanism or system. It takes energy usually provided by air, electric current, or liquid and converts the energy into some kind of motion.

5.7.2 Door

`Door` is an XML container that represents the information for a mechanical mechanism or closure that can cover, for example, a physical access portal into a piece of equipment. The closure can be opened or closed to allow or restrict access to other parts of the equipment.

When `Door` is represented as a `Component`, it **MUST** have a data item called `DOOR_STATE` to indicate if the door is `OPEN`, `CLOSED`, or `UNLATCHED`. A `Component` **MAY** contain multiple `Door` components.

5.7.3 Sensor

`Sensor` is a XML container that represents the information for a piece of equipment that responds to a physical stimulus and transmits a resulting impulse or value. If modeling individual sensors, then `sensor` should be associated with the component that the measured value is most closely associated.

See *Section 9* for more details on the use of `Sensor`.

6 `Composition` Type *Structural Elements*

`Composition` *Structural Elements* are used to describe the lowest level physical building blocks of a piece of equipment contained within a `Component`. By referencing a specific `Composition` element, further clarification and meaning to data associated with a specific `Component` can be achieved.

Both `Component` and `Composition` elements are *Lower Level* child `Component` XML elements representing the sub-parts of the parent `Component`. However, there are distinct differences between `Component` and `Composition` type elements.

`Component` elements may be further defined with *Lower Level* `Component` elements and may have associated *Data Entities*.

`Composition` elements represent the lowest level physical part of a piece of equipment. They **MUST NOT** be further defined with *Lower Level* `Component` elements and they **MUST NOT** have *Data Entities* directly associated with them. They do provide additional information that can be used to enhance the specificity of *Data Entities* associated with the parent `Component`.

The following table defines `Composition` type elements that are currently available to describe sub-parts of a `Component` element.

Element Type	Description
`ACTUATOR`	A mechanism for moving or controlling a mechanical part of a piece of equipment. It takes energy usually provided by air, electric current, or liquid and converts the energy into some kind of motion.
`AMPLIFIER`	An electronic component or circuit for amplifying power, electric current, or voltage.
`BALLSCREW`	A mechanical structure for transforming rotary motion into linear motion.
`BELT`	An endless flexible band used to transmit motion for a piece of equipment or to convey materials and objects.
`BRAKE`	A mechanism for slowing or stopping a moving object by the absorption or transfer of the energy of momentum, usually by means of friction, electrical force, or magnetic force.
`CHOPPER`	A mechanism used to break material into smaller pieces.
`CIRCUIT_BREAKER`	A mechanism for interrupting an electric circuit.

Element Type	Description
CHAIN	An interconnected series of objects that band together and are used to transmit motion for a piece of equipment or to convey materials and objects.
CHUCK	A mechanism that holds a part, stock material, or any other item in place.
CHUTE	An inclined channel for conveying material.
CLAMP	A mechanism used to strengthen, support, or fasten objects in place.
COMPRESSOR	A pump or other mechanism for reducing volume and increasing pressure of gases in order to condense the gases to drive pneumatically powered pieces of equipment.
DOOR	A mechanical mechanism or closure that can cover a physical access portal into a piece of equipment allowing or restricting access to other parts of the equipment.
DRAIN	A mechanism that allows material to flow for the purpose of drainage from, for example, a vessel or tank.
ENCODER	A mechanism used to measure rotary position.
FAN	Any mechanism for producing a current of air.
FILTER	Any substance or structure through which liquids or gases are passed to remove suspended impurities or to recover solids.
GRIPPER	A mechanism that holds a part, stock material, or any other item in place.
HOPPER	A chamber or bin in which materials are stored temporarily, typically being filled through the top and dispensed through the bottom.
MOTOR	A mechanism that converts electrical, pneumatic, or hydraulic energy into mechanical energy.
OIL	A viscous liquid.
PUMP	An apparatus raising, driving, exhausting, or compressing fluids or gases by means of a piston, plunger, or set of rotating vanes.
LINEAR_POSITION_FEEDBACK	A mechanism that measures linear motion or position.
POWER_SUPPLY	A unit that provides power to electric mechanisms.
PULLEY	A mechanism or wheel that turns in a frame or block and serves to change the direction of or to transmit force.

Element Type	Description
SENSING_ELEMENT	A mechanism that provides a signal or measured value.
STORAGE_BATTERY	A component consisting of one or more cells, in which chemical energy is converted into electricity and used as a source of power.
SWITCH	A mechanism for turning on or off an electric current or for making or breaking a circuit.
TANK	A receptacle or container for holding material.
TENSIONER	A mechanism that provides or applies a stretch or strain to another mechanism.
TRANSFORMER	A mechanism that transforms electric energy from a source to a secondary circuit.
VALVE	Any mechanism for halting or controlling the flow of a liquid, gas, or other material through a passage, pipe, inlet, or outlet.
WATER	A fluid.
WIRE	A string like piece or filament of relatively rigid or flexible material provided in a variety of diameters.

Note: As the MTConnect Standard evolves, more `Composition` types will be added.

7 *Data Entities* for `Device`

In the `MTConnectDevices` XML document, *Data Entities* are XML elements that describe data that can be reported by a piece of equipment and are associated with `Device` and `Component` *Structural Elements*. While the *Data Entities* describe the data that can be reported by a piece of equipment in the `MTConnectDevices` document, the actual data values are provided in the *Streams Information Model*. See *Part 3.0 – Streams Information Model* for the details on the reported values.

Each *Data Entity* **SHOULD** be modeled in the `MTConnectDevices` document such that it is associated with the *Structural Element* that the reported data directly applies.

When *Data Entities* are associated with a *Structural Element*, they are organized in a `DataItems` XML element. `DataItems` is a container type XML element. `DataItems` provides the structure for organizing individual `DataItem` elements that represent each *Data Enitity*. The `DataItems` container is comprised of one or more `DataItem` type XML element(s).

`DataItem` describes specific types of *Data Entities* that represent a numeric value, a functioning state, or a health status reported by a piece of equipment. `DataItem` provides a detailed description for each *Data Entity* that is reported; it defines the type of data being reported and an array of optional attributes that further describes that data. The different types of `DataItem` elements are defined in *Section 8*.

The following XML Tree demonstrates the relationship between *Data Entities* (`DataItem`) and the various *Structural Elements* in the `MTConnectDevices` XML document.

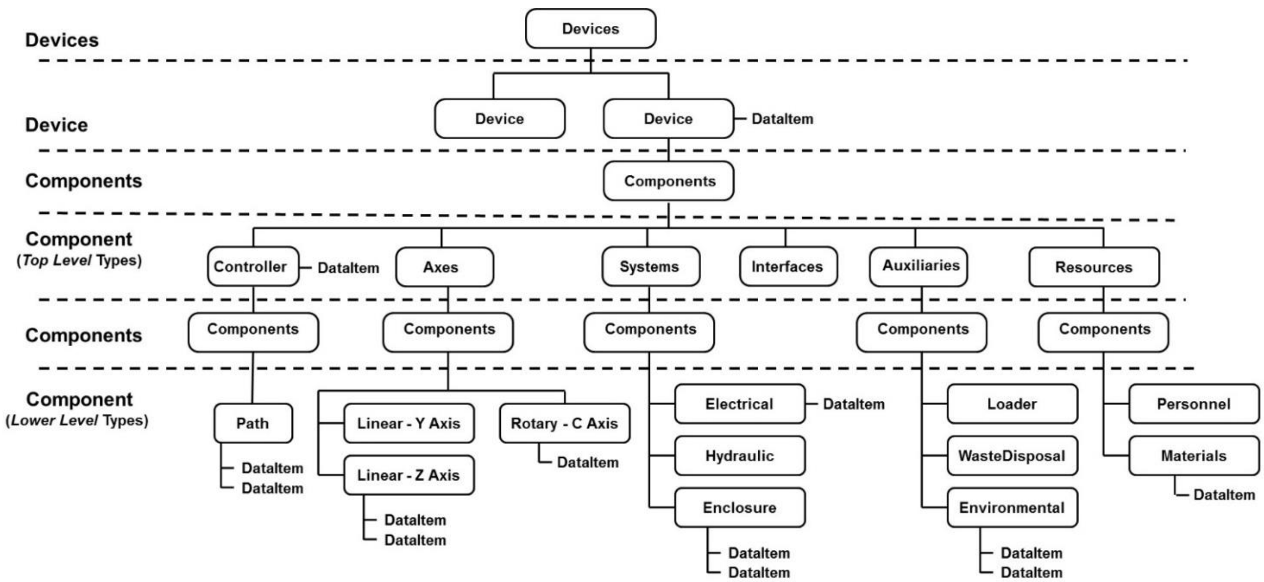


Figure 15: Example *Data Entities* for `Device` (`DataItem`)

7.1 DataItems

The `DataItems` XML element is the first, or highest, level container for the *Data Entities* associated with a `Device` or `Component` XML element. `DataItems` **MUST** contain only `DataItem` type elements. `DataItems` **MUST** contain at least one `DataItem` type element, but **MAY** contain multiple `DataItem` type elements.

Element	Description	Occurrence
`DataItems`	XML Container consisting of one or more types of `DataItem` XML elements. Only one `DataItems` container **MUST** appear for each *Structural Element* in the XML document.	0..1

7.2 DataItem

A DataItem XML element represents each *Data Entity* that **MAY** be reported by a piece of equipment through a *MTConnect Agent*. DataItem provides a detailed description for each *Data Entity* that is reported and defines the type of data being reported along with an array of optional attributes that further define that data. XML elements representing DataItem will include elements such as TEMPERATURE, PRESSURE, and VELOCITY.

Element	Description	Occurrence
DataItem	*Data Entity* describing a piece of information reported about a piece of equipment.	1..INF

7.2.1 XML Schema Structure for `DataItem`

The following XML schema represents the structure of a `DataItem` XML element showing the attributes defined for `DataItem` and the elements that may be associated with `DataItem` type XML elements.

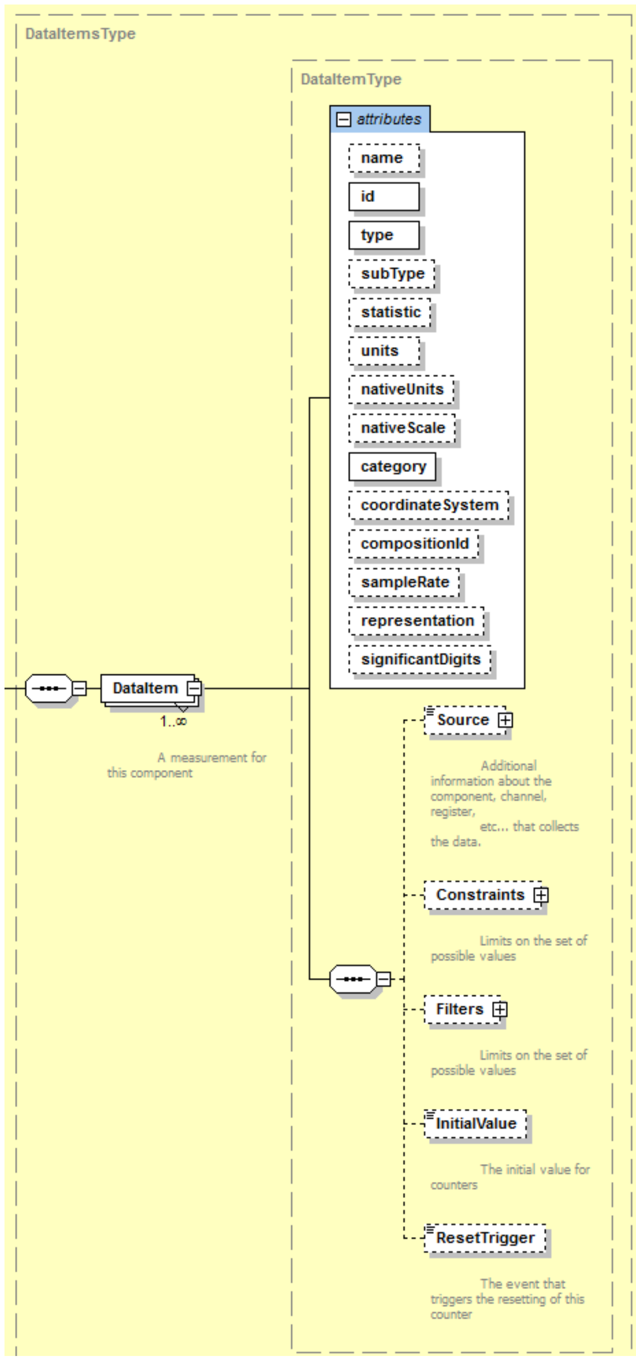


Figure 16: `DataItem` Schema Diagram

7.2.2 Attributes for DataItem

The following table lists the attributes defined to provide information for a DataItem type XML element.

DataItem **MUST** specify the type of data being reported, the id of the DataItem, and the category of the DataItem.

Attribute	Description	Occurrence
name	The name of the data item. name is provided as an additional human readable identifier for this data item in addition to the id. name is an optional attribute and will be implementation dependent. An NMTOKEN XML type.	0..1
id	The unique identifier for this data item. id is a required attribute. The id attribute **MUST** be unique within the MTConnectDevices document. An XML ID-type.	1
type	The type of data being measured. type is a required attribute. Examples of types are POSITION, VELOCITY, ANGLE, BLOCK, and ROTARY_VELOCITY.	1
subType	A sub-categorization of the data item type. subType is an optional attribute. For example, the subType of POSITION can be ACTUAL or COMMANDED. Not all type attributes have a subType.	0..1
statistic	Describes the type of statistical calculation performed on a series of data samples to provide the reported data value. statistic is an optional attribute. Examples of statistic are AVERAGE, MINIMUM, MAXIMUM, ROOT_MEAN_SQUARE, RANGE, MEDIAN, MODE, and STANDARD_DEVIATION.	0..1

Attribute	Description	Occurrence
`units`	The unit of measurement for the reported value of the data item. `units` is an optional attribute. Data items in the `Sample` category **MUST** report the standard units for the measured values. See *Section 7.2.2.7* for a list of available standard units identified in the MTConnect Standard.	0..1
`nativeUnits`	The native units of measurement for the reported value of the data item. `nativeUnits` is an optional attribute. See *Section 7.2.2.8* for a list of available native units identified in the MTConnect Standard	0..1
`nativeScale`	The `nativeUnits` may not be scaled to directly represent the original measured value. `nativeScale` **MAY** be used to convert the reported value to represent the original measured value. `nativeScale` is an optional attribute. As an example, the `nativeUnits` may be reported as `GALLON/MINUTE`. The measured value may actually be in 1000 `GALLON/MINTUE`. The value of the reported data **MAY** be divided by the `nativeScale` to convert the reported value to its original measured value and units. If provided, the value **MUST** be numeric.	0..1
`category`	Specifies the kind of information provided by a data item. `category` is a required attribute. The available options are `SAMPLE`, `EVENT`, or `CONDITION`.	1
`coordinateSystem`	For measured values relative to a coordinate system like `POSITION`, the coordinate system being used may be reported. `coordinateSystem` is an optional attribute. The available values for `coordinateSystem` are `WORK` and `MACHINE`.	0..1
`compositionId`	The identifier attribute of the `Composition` element that the reported data is most closely associated. `compositionID` is an optional attribute.	0..1

Attribute	Description	Occurrence
sampleRate	The rate at which successive samples of a data item are recorded by a piece of equipment. sampleRate is an optional attribute. sampleRate is expressed in terms of samples per second. If the sampleRate is smaller than one, the number can be represented as a floating point number. For example, a rate 1 per 10 seconds would be 0.1	0..1**
representation	Description of a means to interpret data consisting of multiple data points or samples reported as a single value. representation is an optional attribute. representation will define a unique format for each set of data. representation for TIME_SERIES, DISCRETE, and VALUE are defined below in *Section 7.2.2.12*. If representation is not specified, it **MUST** be determined to be VALUE.	0..1
significantDigits	The number of significant digits in the reported value. significantDigits is an optional attribute. This **SHOULD** be specified for all numeric values.	0..1

7.2.2.1 name Attribute for DataItem

The attribute name is provided as an additional human readable identifier for a data item. It is not required and is implementation dependent.

7.2.2.2 id Attribute for DataItem

Each DataItem element **MUST** be identified with an id. The id attribute **MUST** be unique across the entire MTConnectDevices document for a piece of equipment, including the identifiers for all *Structural Elements*. This unique id provides the information required by a client software application to uniquely identify each *Data Entity*.

For example, an XML document may provide three different *Data Entities* representing the position of the axes on a machine (x axis position, y axis position, and z axis position). All three may be modeled in the XML document as Position type data items for the Axes components. The unique id allows the client software application to distinguish the data for each of the axes.

7.2.2.3 type and subType Attributes for DataItem

The attribute type specifies the kind of data that is represented by the data item.

The attribute `type` **MUST** be specified for every data item.

A data item **MAY** further qualify the data being reported by specifying a `subType`. `subType` is required for certain data item `types`. For example, `POSITION` has the `subType` of `ACTUAL` and `PROGRAMMED`. Both data values can be represented in the document as two separate and different `DataItem` XML elements – `POSITION` with `subType` `ACTUAL` and `POSITION` with `subType` `PROGRAMMED`.

The `type` and `subType` **SHOULD** be used to further identify the meaning of the `DataItem` associated with a `Component` element when a `subType` is applicable. There **SHOULD NOT** be more than one `DataItem` with the same `type`, `subType`, and `compositionId` within a `Component` element.

Section 8 of this document provides a detailed listing of the data item `type` and `subType` elements defined for each `category` of data item available for a piece of equipment: `SAMPLE`, `EVENT`, and `CONDITION`.

7.2.2.4 `statistic` Attribute for `DataItem`

A piece of equipment may further process some data types using a statistical calculation like average, mean, or square root. In this case, the `statistic` attribute **MAY** be used to indicate how the data was processed.

`statistic` may be defined for any `SAMPLE` type `DataItem`. All `statistic` data is reported in the standard units of the `DataItem`.

`statistic` data is always the result of a calculation using data that has been measured over a specified period of time.

The value of `statistic` may be periodically reset. When a piece of equipment reports a `DataItem` with a value that is a `statistic`, the information provided in the XML document for that *Data Entity* **MUST** include an additional attribute called `duration`. The attribute `duration` defines the period of time over which the `statistic` has been calculated. Refer to *Part 3.0 – Streams Information Model* of the MTConnect Standard for more information about `duration`.

The following are the `statistic` calculations that can be defined for a `DataItem`.

Statistic	Description
`AVERAGE`	Mathematical Average value calculated for the data item during the calculation period.
`KURTOSIS`	A measure of the "peakedness" of a probability distribution; i.e., the shape of the distribution curve.

Statistic	Description
MAXIMUM	Maximum or peak value recorded for the data item during the calculation period.
MEDIAN	The middle number of a series of numbers.
MINIMUM	Minimum value recorded for the data item during the calculation period.
MODE	The number in a series of numbers that occurs most often.
RANGE	Difference between the Maximum and Minimum value of a data item during the calculation period. Also represents Peak-to-Peak measurement in a waveform.
ROOT_MEAN_SQUARE	Mathematical Root Mean Square (RMS) value calculated for the data item during the calculation period.
STANDARD_DEVIATION	Statistical Standard Deviation value calculated for the data item during the calculation period.

7.2.2.5 `units` Attribute for `DataItem`

The following table lists the units that are defined as the standard unit of measure for each type of `DataItem.` All `SAMPLE` type data items **MUST** report data values in standard units.

Units	Description
AMPERE	Amps
CELSIUS	Degrees Celsius
COUNT	A counted event
DECIBEL	Sound Level
DEGREE	Angle in degrees
DEGREE/SECOND	Angular degrees per second
DEGREE/SECOND^2	Angular acceleration in degrees per second squared
HERTZ	Frequency measured in cycles per second
JOULE	A measurement of energy.

Units	Description
`KILOGRAM`	Kilograms
`LITER`	Liters
`LITER/SECOND`	Liters per second
`MICRO_RADIAN`	Measurement of Tilt
`MILLIMETER`	Millimeters
`MILLIMETER/SECOND`	Millimeters per second
`MILLIMETER/SECOND^2`	Acceleration in millimeters per second squared
`MILLIMETER_3D`	A point in space identified by X, Y, and Z positions and represented by a space-delimited set of numbers each expressed in millimeters.
`NEWTON`	Force in Newtons
`NEWTON_METER`	Torque, a unit for force times distance.
`OHM`	Measure of Electrical Resistance
`PASCAL`	Pressure in Newtons per square meter
`PASCAL_SECOND`	Measurement of Viscosity
`PERCENT`	Percentage
`PH`	A measure of the acidity or alkalinity of a solution
`REVOLUTION/MINUTE`	Revolutions per minute
`SECOND`	A measurement of time.
`SIEMENS/METER`	A measurement of Electrical Conductivity
`VOLT`	Volts
`VOLT_AMPERE`	Volt-Ampere (VA)
`VOLT_AMPERE_REACTIVE`	Volt-Ampere Reactive (VAR)
`WATT`	Watts
`WATT_SECOND`	Measurement of electrical energy, equal to one Joule

7.2.2.6 `nativeUnits` Attribute for `DataItem`

The `nativeUnits` attribute provides additional information about the original measured value for a *Data Entity* reported by a piece of equipment. `nativeUnits` **MAY** be specified to provide additional information about the data if the units of the measured value supplied by the piece of equipment differ from the value provided for that data when converted to standard units.

The following table defines the `nativeUnits` currently supported by the `MTConnectDevices` XML document:

Native Units	Description
`CENTIPOISE`	A measure of Viscosity
`DEGREE/MINUTE`	Rotational velocity in degrees per minute
`FAHRENHEIT`	Temperature in Fahrenheit
`FOOT`	Feet
`FOOT/MINUTE`	Feet per minute
`FOOT/SECOND`	Feet per second
`FOOT/SECOND^2`	Acceleration in feet per second squared
`FOOT_3D`	A point in space identified by X, Y, and Z positions and represented by a space-delimited set of numbers each expressed in feet.
`GALLON/MINUTE`	Gallons per minute.
`INCH`	Inches
`INCH/MINUTE`	Inches per minute
`INCH/SECOND`	Inches per second
`INCH/SECOND^2`	Acceleration in inches per second squared
`INCH_3D`	A point in space identified by X, Y, and Z positions and represented by a space-delimited set of numbers each expressed in inches.
`INCH_POUND`	A measure of torque in inch pounds.
`KELVIN`	A measurement of temperature
`KILOWATT`	A measurement in kilowatt.

Native Units	Description
KILOWATT_HOUR	Kilowatt hours which is 3.6 mega joules.
LITER	Measurement of volume of a fluid
LITER/MINUTE	Measurement of rate of flow of a fluid
MILLIMETER/MINUTE	Velocity in millimeters per minute
POUND	US pounds
POUND/INCH^2	Pressure in pounds per square inch (PSI).
RADIAN	Angle in radians
RADIAN/SECOND	Velocity in radians per second
RADIAN/SECOND^2	Rotational acceleration in radian per second squared
RADIAN/MINUTE	Velocity in radians per minute.
REVOLUTION/SECOND	Rotational velocity in revolution per second
OTHER	Unsupported units

7.2.2.7 `nativeScale` Attribute for `DataItem`

The units of measure for some measured values may be different from the `nativeUnits` defined in *Section 7.2.2.8* above. In the cases where the units of measure use a different weighting or range than is provided by `nativeUnits`, the `nativeScale` attribute can be used to define the original units of measure.

As an example, a velocity measured in units of 100 ft/min can be represented as `nativeUnits="FEET/MINUTE"` and `nativeScale="100"`.

7.2.2.8 `category` Attribute for `DataItem`

Many `DataItem` types provide two forms of data, a value (reported as either a `SAMPLE` or `EVENT` category) and a health status (reported as a `CONDITION` category). Therefore, each occurrence of a `DataItem` in the XML document **MUST** report a `category` attribute. This `category` attribute provides the information required by a client software application to determine the specific meaning of the data provided.

Each *Data Entity* provided by a piece of equipment **MUST** be identified with one of the following:

`SAMPLE` A `SAMPLE` is the reading of the value of a continuously variable or analog data value. A continuous value can be measured at any point-in-time and will always produce a result. An example of a continuous data value is the position of the Linear X Axis.

The data provided for a `SAMPLE` category data item is always a floating point number or integers that have an infinite number of possible values. This is different from a state or discrete type data item that has a limited number of possible values. A data item of category `SAMPLE` **MUST** also provide the `units` attribute.

`EVENT` An `EVENT` is a data item representing a discrete piece of information from the piece of equipment. `EVENT` does not have intermediate values that vary over time, as does `SAMPLE`. An `EVENT` is information that, when provided at any specific point in time, represents the current state of the piece of equipment.

There are two types of `EVENT`: those representing state, with two or more discrete values, and those representing messages that contain plain text data.

An example of a state type `EVENT` is the value of the data item `DOOR_STATE`, which can be `OPEN`, `UNLATCHED`, or `CLOSED`. (Note: No other values are valid to represent the value of `DOOR_STATE`.)

An example of a message type `EVENT` is the value for a data item `PROGRAM`. The value representing `PROGRAM` can be any valid string of characters.

`CONDITION` A `CONDITION` is a data item that communicates information about the health of a piece of equipment and its ability to function. A valid value for a data item in the category `CONDITION` can be one of `NORMAL`, `WARNING`, or `FAULT.`

A data item of category `CONDITION` **MAY** report multiple values (`CONDITION`) at one time whereas a data item of category `SAMPLE` or `EVENT` can only have a single value at any one point in time.

7.2.2.9 `coordinateSystem` Attribute for `DataItem`

The values reported by a piece of equipment for some types of data will be associated to a specific positioning measurement system used by the equipment. The `coordinateSystem` attribute **MAY** be used to specify the coordinate system used for the measured value.

The `coordinateSystem` attribute is used by a client software application to interpret the spatial relationship between values reported by a piece of equipment.

If `coordinateSystem` is not provided, all values representing positional data for `Axes` **MUST** be interpreted using the `MACHINE` coordinate system and all values representing positional data for `Path` **MUST** be interpreted using the `WORK` coordinate system.

The following table defines the types of `coordinateSystem` currently supported by the `MTConnectDevices` XML document:

Coordinate System	Description
`MACHINE`	An unchangeable coordinate system that has machine zero as its origin.
`WORK`	The coordinate system that represents the working area for a particular workpiece whose origin is shifted within the `MACHINE` coordinate system. If the `WORK` coordinates are not currently defined in the piece of equipment, the `MACHINE` coordinates will be used.

7.2.2.10 `compositionId` Attribute for `DataItem`

`compositionId` attribute identifies the `id` of the `Composition` element where the reported data is most closely associated.

An example would be a `TEMPERATURE` associated with a `Linear` type axis may be further clarified by referencing the `MOTOR` or `AMPLIFIER` type `Composition` element associated with that axis, which differentiates the temperature of the motor from the temperature of the amplifier.

The `compositionId` attribute provides the information required by a client software application to interpret the data with a greater specificity and to disambiguate between multiple *Data Entities* of the same data `type` associated with a `Component` element.

7.2.2.11 `sampleRate` Attribute for `DataItem`

The value for some data types provided by a piece of equipment may be reported as a single set of data containing a series of values that have been recorded at a fixed sample rate. When such data is reported, the `sampleRate` defines the rate at which successive samples of data were recorded.

The `sampleRate` attribute provides the information required by a client software application to interpret the data and the sampling time relationship between successive values contained in the set of data.

`sampleRate` is expressed in terms of samples per second. If the sample rate is smaller than one, the number can be represented as a floating point number. For example, a rate 1 per 10 seconds would be 0.1

7.2.2.12 `representation` Attribute for `DataItem`

Some data types provide data that may consist of a series of values or a file of data, not a single value. Other data types provide a series of data values that may require additional information so that the data may be correctly understood by a client software application.

When such data is provided, the `representation` attribute **MUST** be used to define the format for the data provided.

The types of `representation` defined are provided in the table below.

> Note: See *Part 3.0 - Streams Information Model* of the MTConnect Standard for more information on the structure and format of each `representation`.

Representation	Description
`VALUE`	The measured value of the sampled data. If no representation is specified for a data item, the representation **MUST** be determined to be VALUE.
`TIME_SERIES`	A series of sampled data. The data is reported for a specified number of samples and each sample is reported with a fixed period.
`DISCRETE`	A *Data Entity* where each discrete occurrence of the data may have the same value as the previous occurrence of the data. There is no reported state change between occurrences of the data. In this case, duplicate occurrences of the same data value **SHOULD NOT** be suppressed. An example of a `DISCRETE` data type would be a parts counter that reports the completion of each part versus the accumulation of parts. Another example would be a `Message` that does not typically have a reset state and may re-occur each time a specific message is triggered.

7.2.2.13 `significantDigits` Attribute for `DataItem`

`significantDigits` is used to specify the level of precision (number of significant digits) for the value provided for a data item.

`significantDigits` attribute is not required for a data item, but it is recommended and **SHOULD** be used for any data item reporting a numeric value.

7.2.3 Elements for `DataItem`

The following table lists the elements defined to provide additional information for a `DataItem` type XML element.

Element	Description	Occurrence
`Source`	`Source` is an optional XML element that identifies the `Component`, `DataItem`, or `Composition` representing the part of the piece of equipment from which a measured value originates.	0..1
`Constraints`	`Constraints` is an optional container that provides a set of expected values that can be reported for this `DataItem`. `Constraints` are used by a software application to evaluate the validity of the reported data.	0..1
`Filters`	An optional container for the `Filter` elements associated with this `DataItem` element.	0..1
`InitialValue`	`InitialValue` is an optional XML element that defines the starting value for a data item as well as the value to be set for the data item after a reset event. Only one `InitialValue` element may be defined for a data item. The value will be constant and cannot change. If no `InitialValue` element is defined for a data item that is periodically reset, then the starting value for the data item **MUST** be a value of 0.	0..1
`ResetTrigger`	`ResetTrigger` is an optional XML element that identifies the type of event that may cause a reset to occur. It is additional information regarding the meaning of the data that establishes an understanding of the time frame that the data represents so that the data may be correctly understood by a client software application.	0..1

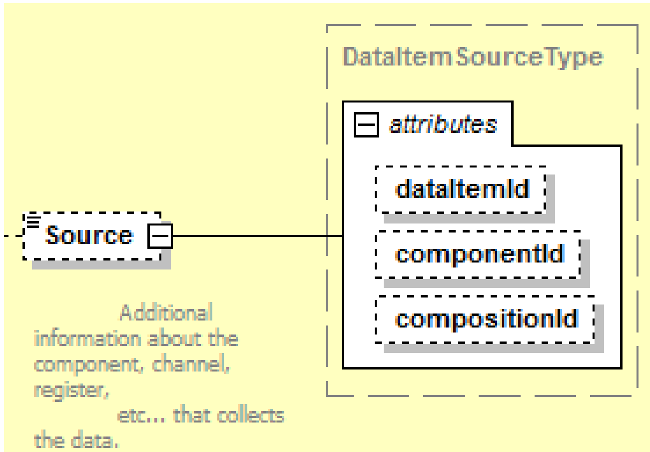
7.2.3.1 `Source` Element for `DataItem`

`Source` is an optional XML element that identifies the physical part of a piece of equipment where the data represented by `DataItem` originated.

As an example, data related to a servo motor on an `Axes` component may actually originate from a measurement made in the `Controller` element.

In the case where the real name associated with a `DataItem` element is either complex or does not meet the format requirements of a NMTOKEN XML type, the real name of the element may not be able to be expressed in the `name` attribute. When this occurs, a short name or nickname can be used for the `name` attribute and the real name can be provided as the `CDATA` for `Source`.

988 The following XML schema represents the structure of the `Source` XML element showing the
989 attributes defined for `Source`.
990

991 

992 **Figure 17: `Source` Schema Diagram**

993

994 7.2.3.1.1 Attributes for `Source`

995 The following table identifies the attributes available to identify `Source` for a measured value:

Attribute	Description	Occurrence
componentId	The identifier attribute of the Component element that represents the physical part of a piece of equipment where the data represented by the DataItem element originated. A valid data value reported for componentId **MUST** be the value of the Id attribute for the Component element identified. componentId is an optional attribute.	0..1*
dataItemId	The identifier attribute of the DataItem that represents the originally measured value of the data referenced by this data item. A valid data value reported for dataItemId **MUST** be the value of the Id attribute for the DataItem element identified. dataItemId is an optional attribute.	0..1*
compositionId	The identifier attribute of the Composition element that represents the physical part of a piece of equipment where the data represented by the DataItem element originated. A valid data value reported for compositionId **MUST** be the value of the Id attribute for the Composition element identified. compositionId is an optional attribute.	0..1*

Note: * One of componentId, componsitionId, or dataItemId **MUST** be provided.

7.2.3.2 Constraints Element for DataItem

For some types of DataItem elements, the expected value(s) for the data reported for the DataItem **MAY** be restricted to specific values or a range of values.

Constraints is an optional XML element that provides a way to define the expected value(s) or the upper and lower limits for the range of values that are expected to be reported in response to a Current or Sample request.

Constraints are used by a software application to evaluate the validity of the data reported.

The value associated with each Constraint element is reported in the CDATA for that element.

7.2.3.2.1 Schema for `Constraints`

The following XML schema represents the structure of the `Constraints` XML element and the elements defined for `Constraints`.

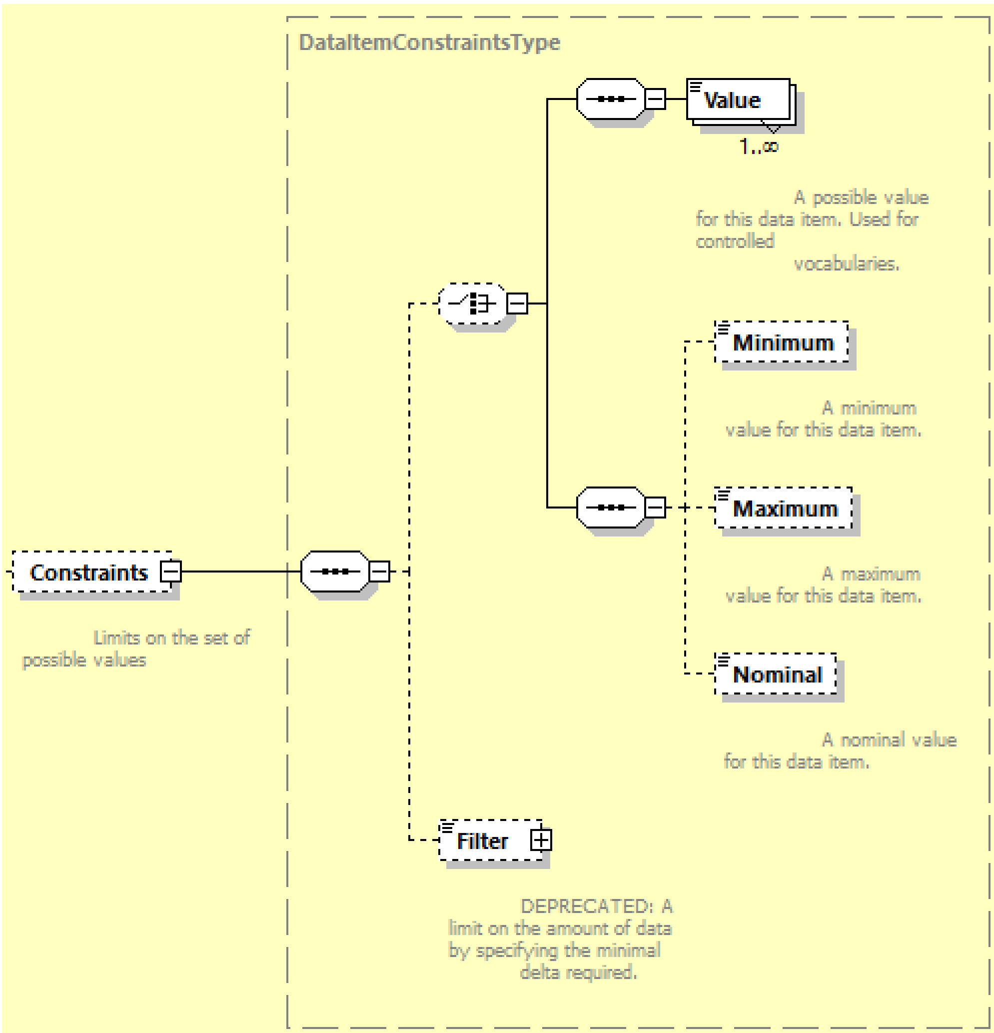


Figure 18: `Constraints` Schema Diagram

The following table identifies the elements available to identify `Constraints` for a measured value:

Element	Description	Occurrence
`Value`	`Value` represents a single data value that is expected to be reported for a `DataItem` element. The data value is provided in the CDATA for this element and may be any numeric or text content. When there are multiple data values that may be expected to be reported for a `DataItem` element, multiple `Value` elements may be defined. In the case where only one `Value` element is defined, the data returned in response to a `Current` or `Sample` request **MUST** be the data value defined for `Value` element. `Value` **MUST NOT** be used in conjunction with any other `Constraint` elements.	0..INF
`Maximum`	If the data reported for a data item is a range of numeric values, the expected value reported **MAY** be described with an upper limit defined by this constraint. The data value is provided in the `CDATA` for this element and **MUST** be an absolute value using the same units as the reported data.	0..1
`Minimum`	If the data reported for a data item is a range of numeric values, the expected value reported **MAY** be described with a lower limit defined by this constraint. The data value is provided in the `CDATA` for this element and **MUST** be an absolute value using the same units as the reported data.	0..1
`Nominal`	The target or expected value for this data item. The data value is provided in the `CDATA` for this element and **MUST** be an absolute value using the same units as the reported data.	0..1
~~`Filter`~~	**DEPRECATED** in *Version 1.4* – Moved to the `Filters` element of a `DataItem`. ~~If the data reported for a DataItem is a numeric value, a new value MUST NOT be reported if the change from the last reported value is less than the delta given as the CDATA of this element. Filter is an abstract type XML element. As such, Filter will never appear in the XML document, but will be replaced by a Filter type. The only currently supported Filter type is MINIMUM_DELTA. The CDATA MUST be an absolute value using the same Units as the reported data. Additional filter types MAY be supported in the future.~~	~~0..1*~~

Note: * Remains in schema for backwards compatibility.

7.2.3.3 **`Filters`** Element for **`DataItem`**

`Filters` is an optional XML container that organizes the `Filter` elements for `DataItem`.

`Filters` contains one or more `Filter` XML elements.

Element	Description	Occurrence
`Filters`	An XML container consisting of one or more types of `Filter` XML elements. Only one `Filters` container **MAY** appear for a `DataItem` element.	0..1

7.2.3.3.1 Filter

`Filter` provides a means to control when a *MTConnect Agent* records updated information for a data item. Currently, there are two types of `Filter` elements defined in the MTConnect Standard - `MINIMUM_DELTA` and `PERIOD`. More `Filter` types may be added in the future.

The value associated with each `Filter` element is reported in the `CDATA` for that element.

The following XML schema represents the structure for `Filter` XML element.

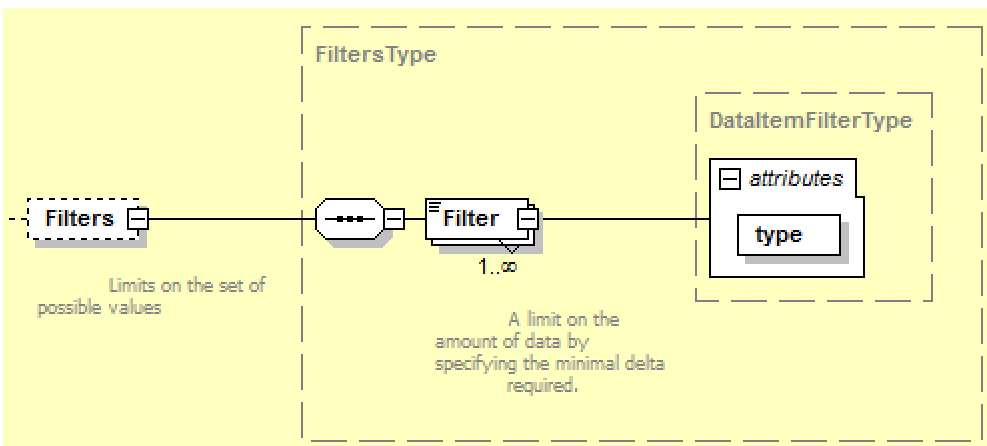


Figure 19: `Filter` Schema Diagram

The following table describes the types of `Filter` defined for a `DataItem` element and the expected behavior of a *MTConnect Agent* when a `Filter` is applied to `DataItem` element.

Type	Description	Occurrence
`MINIMUM_DELTA`	For a `MINIMUM_DELTA` type `Filter`, a new value **MUST NOT** be reported for a data item unless the measured value has changed from the last reported value by at least the delta given as the `CDATA` of this element. The `CDATA` **MUST** be an absolute value using the same units as the reported data.	0..1 *
`PERIOD`	For a `PERIOD` type `Filter`, the data reported for a data item is provided on a periodic basis. The `PERIOD` for reporting data is defined in the `CDATA` for the `Filter`. The `CDATA` **MUST** be an absolute value reported in seconds representing the time between reported samples of the value of the data item. If the `PERIOD` is smaller than one second, the number can be represented as a floating point number. For example, a `PERIOD` of 100 milliseconds would be 0.1	0..1 *

Note: * Either `MINIMUM_DELTA` or `PERIOD` can be defined, not both.

7.2.3.4 `InitialValue` Element for `DataItem`

`InitialValue` is an XML element that defines the value to be set for the data item after a reset event.

The value associated with the `InitialValue` element is reported in the `CDATA` for this element and **MUST** be an absolute value using the same units as the reported data.

7.2.3.5 `ResetTrigger` Element for `DataItem`

The value of some data types is periodically reset to the value of the `InitialValue` element. These reset events may be based upon a specific elapsed time or may be triggered by a physical or logical reset action that causes the reset to occur. `ResetTrigger` provides additional information regarding the meaning of the data – establishing an understanding of the time frame that the data represents so that the data may be correctly understood by a client software application.

Element	Description	Occurrence
`ResetTrigger`	`ResetTrigger` is an XML element that describes the reset action that causes a reset to occur. It is additional information regarding the meaning of the data that establishes an understanding of the time frame that the data represents so that the data may be correctly understood by a client software application.	0..1

The reset action that **MAY** cause a reset to occur is provided in the `CDATA` for this element.

The reset actions that may cause a reset to occur are described in the following table.

Reset Actions	Description
`ACTION_COMPLETE`	The value of the *Data Entity* that is measuring an action or operation is to be reset upon completion of that action or operation.
`ANNUAL`	The value of the *Data Entity* is to be reset at the end of a 12-month period.
`DAY`	The value of the *Data Entity* is to be reset at the end of a 24-hour period.
`LIFE`	The value of the data item is not reset and accumulates for the entire life of the piece of equipment.
`MAINTENANCE`	The value of the data item is to be reset upon completion of a maintenance event.
`MONTH`	The value of the *Data Entity* is to be reset at the end of a monthly period.
`POWER_ON`	The value of the *Data Entity* is to be reset when power was applied to the piece of equipment after a planned or unplanned interruption of power has occurred.
`SHIFT`	The value of the *Data Entity* is to be reset at the end of a work shift.
`WEEK`	The value of the *Data Entity* is to be reset at the end of a 7-day period.

8 Listing of Data Items

In the MTConnect Standard, DataItem elements are defined and organized based upon the category and type attributes.

The category attribute provides a high level grouping for DataItem elements based on the kind of information that is reported by the data item.

These categories are:

SAMPLE A SAMPLE reports a continuously variable or analog data value.

EVENT An EVENT reports information representing a functional state, with two or more discrete values, associated with a component or it contains a message. The data provided may be a numeric value or text.

CONDITION A CONDITION reports information about the health of a piece of equipment and its ability to function.

The type attribute specifies the specific kind of data that is reported. For some types of data items, a subType attribute may also be used to differentiate between multiple data items of the same type where the information reported by the data item has a different, but related, meaning.

Many types of data items provide two forms of data: a value (reported as either a SAMPLE or EVENT) and a health status (reported as a CONDITION). These DataItem types **MAY** be defined in more than one category based on the data that they report.

The following sections define the types and subtypes of DataItem elements that are defined for each of the above categories.

8.1 Data Items in category SAMPLE

The types of DataItem elements in the SAMPLE category report data representing a continuously changing or analog data value. This data can be measured at any point-in-time and will always produce a result. The data provided may be a scalar floating point number or integers that have an infinite number of possible values. The units attribute **MUST** be defined and reported for each DataItem in this category.

The table below defines the types and subtypes of `DataItem elements` defined for the `SAMPLE` category. The subtypes are indented below their associated types.

DataItem type/subType	Description	Units
ACCELERATION	Rate of change of velocity	MILLIMETER/SECOND^2
ACCUMULATED_TIME	The measurement of accumulated time for an activity or event. **DEPRECATION WARNING:** May be deprecated in the future. Recommend using PROCESS_TIMER and MACHINE_TIMER.	SECOND
ANGULAR_ACCELERATION	Rate of change of angular velocity.	DEGREE/SECOND^2
ANGULAR_VELOCITY	Rate of change of angular position.	DEGREE/SECOND
AMPERAGE	The measurement of electrical current	AMPERE
ALTERNATING	The measurement of alternating current. If not specified further in `statistic`, defaults to RMS current	AMPERE
DIRECT	The measurement of DC current	AMPERE
ACTUAL	The measured amperage being delivered from a power source.	AMPERE
TARGET	The desired or preset amperage to be delivered from a power source.	AMPERE
ANGLE	The measurement of angular position	DEGREE
ACTUAL	The actual angular position as read from the physical component.	DEGREE
COMMANDED	A calculated value for angular position computed by the `Controller` type component	DEGREE
AXIS_FEEDRATE	The feedrate of a linear axis.	MILLIMETER/SECOND
ACTUAL	The measured value of the feedrate of a linear axis.	MILLIMETER/SECOND

DataItem type/subType	Description	Units
COMMANDED	The feedrate of a linear axis as specified by the Controller type component. The COMMANDED feedrate is a calculated value that includes adjustments and overrides.	MILLIMETER/SECOND
JOG	The feedrate specified by a logic or motion program, by a pre-set value, or set by a switch as the feedrate for a linear axis when operating in a manual state or method (jogging).	MILLIMETER/SECOND
PROGRAMMED	The feedrate specified by a logic or motion program or set by a switch for a linear axis.	MILLIMETER/SECOND
RAPID	The feedrate specified by a logic or motion program, by a pre-set value, or set by a switch as the feedrate for a linear axis when operating in a rapid positioning mode.	MILLIMETER/SECOND
~~OVERRIDE~~	~~The operator's overridden value. Percent of commanded~~. **DEPRECATED** in *Version 1.3*. See EVENT category data items.	~~PERCENT~~
CLOCK_TIME	The value provided by a timing device at a specific point in time. CLOCK_TIME **MUST** be reported in W3C ISO 8601 format.	YYYY-MM-DDThh:mm:ss.ffff
CONCENTRATION	Percentage of one component within a mixture of components	PERCENT
CONDUCTIVITY	The ability of a material to conduct electricity	SIEMENS/METER
DISPLACEMENT	The change in position of an object	MILLIMETER
ELECTRICAL_ENERGY	The measurement of electrical energy consumption by a component	WATT_SECOND

DataItem type/subType	Description	Units
EQUIPMENT_TIMER	The measurement of the amount of time a piece of equipment or a sub-part of a piece of equipment has performed specific activities. Often used to determine when maintenance may be required for the equipment Multiple subTypes of EQUIPMENT_TIMER **MAY** be defined. A subType **MUST** always be specified.	SECOND
LOADED	Measurement of the time that the sub-parts of a piece of equipment are under load. Example: For traditional machine tools, this is a measurement of the time that the cutting tool is assumed to be engaged with the part.	SECOND
WORKING	Measurement of the time that a piece of equipment is performing any activity – the equipment is active and performing a function under load or not. Example: For traditional machine tools, this includes LOADED, plus rapid moves, tool changes, etc.	SECOND
OPERATING	Measurement of the time that the major sub-parts of a piece of equipment are powered or performing any activity whether producing a part or product or not. Example: For traditional machine tools, this includes WORKING, plus idle time.	SECOND
POWERED	The measurement of time that primary power is applied to the piece of equipment and, as a minimum, the controller or logic portion of the piece of equipment is powered and functioning or components that are required to remain on are powered. Example: Heaters for an extrusion machine that are required to be powered even when the equipment is turned off.	SECOND
DELAY	Measurement of the time that a piece of equipment is waiting for an event or an action to occur.	SECOND

DataItem type/subType	Description	Units
FILL_LEVEL	The measurement of the amount of a substance remaining compared to the planned maximum amount of that substance	PERCENT
FLOW	The rate of flow of a fluid	LITER/SECOND
FREQUENCY	The measurement of the number of occurrences of a repeating event per unit time	HERTZ
~~GLOBAL_POSITION~~	**DEPRECATED** in *Version 1.1*	
~~LEVEL~~	**DEPRECATED** in *Version 1.2*. See FILL_LEVEL	
LENGTH	The length of an object	MILLIMETER
STANDARD	The standard or original length of an object.	MILLIMETER
REMAINING	The remaining total length of an object.	MILLIMETER
USEABLE	The remaining useable length of an object.	MILLIMETER
LINEAR_FORCE	The measure of the push or pull introduced by an actuator or exerted on an object.	NEWTON
LOAD	The measurement of the actual versus the standard rating of a piece of equipment.	PERCENT
MASS	The measurement of the mass of an object(s) or an amount of material.	KILOGRAM
PATH_FEEDRATE	The feedrate for the axes, or a single axis, associated with a Path component– a vector.	MILLIMETER/SECOND
ACTUAL	The measured value of the feedrate of the axes, or a single axis, associated with a Path component.	MILLIMETER/SECOND
COMMANDED	The feedrate as specified by the Controller type component for the axes, or a single axis, associated with a Path component. The COMMANDED feedrate is a calculated value that includes adjustments and overrides.	MILLIMETER/SECOND

DataItem type/subType	Description	Units
JOG	The feedrate specified by a logic or motion program, by a pre-set value, or set by a switch as the feedrate for the axes, or a single axis, associated with a Path when operating in a manual state or method (jogging).	MILLIMETER/SECOND
PROGRAMMED	The feedrate specified by a logic or motion program or set by a switch as the feedrate for the axes, or a single axis, associated with a Path.	MILLIMETER/SECOND
RAPID	The feedrate specified by a logic or motion program, by a pre-set value, or set by a switch as the feedrate for the axes, or a single axis, associated with a Path when operating in a rapid positioning mode.	MILLIMETER/SECOND
~~OVERRIDE~~	~~The operator's overridden value. Percent of commanded.~~ **DEPRECATED** in *Version 1.3*. See EVENT category DataItems.	~~PERCENT~~
PATH_POSITION	A measured or calculated position of a control point associated with a CONTROLLER element, or PATH element if provided, of a piece of equipment. The control point **MUST** be reported as a set of space-delimited floating-point numbers representing a point in 3-D space. The position of the control point **MUST** be reported in units of MILLIMETER and listed in order of X, Y, and Z referenced to the coordinate system of the piece of equipment. Any control point representing a position in 1-D or 2-D space **MAY** be represented in terms of 3-D space by setting any undefined coordinate to zero (0). PATH_POSITION **SHOULD** be further defined with a coordinateSytem attribute. If a coordinateSystem attribute is not specified, the position of the control point **MUST** be reported in WORK coordinates.	MILLIMETER_3D
ACTUAL	The measured position of the current program control point as reported by the piece of equipment.	MILLIMETER_3D

DataItem type/subType	Description	Units
COMMANDED	The position computed by the Controller type component.	MILLIMETER_3D
TARGET	The desired end position for a movement or a series of movements. Multiple discrete movements may need to be completed to achieve the final TARGET position.	MILLIMETER_3D
PROBE	The position provided by a measurement probe.	MILLIMETER_3D
PH	The measure of the acidity or alkalinity.	PH
POSITION	A calculated or measured position related to a Component element. POSITION **SHOULD** be further defined with a coordinateSytem attribute. If a coordinateSystem attribute is not specified, the position of the control point **MUST** be reported in MACHINE coordinates.	MILLIMETER
ACTUAL	The physical measured position of the control point for a Component.	MILLIMETER
COMMANDED	A position calculated by the Controller type component for a discrete movement.	MILLIMETER
PROGRAMMED	The position of the control point for a Component specified by a logic or motion program	MILLIMETER
TARGET	The desired end position of the control point for a Component resulting from a movement or a series of movements. Multiple discrete movements may need to be completed to achieve the final TARGET position.	MILLIMETER
POWER_FACTOR	The measurement of the ratio of real power flowing to a load to the apparent power in that AC circuit.	PERCENT
PRESSURE	The force per unit area exerted by a gas or liquid	PASCAL

DataItem type/subType	Description	Units
PROCESS_TIMER	The measurement of the amount of time a piece of equipment has performed different types of activities associated with the process being performed at that piece of equipment. Multiple subtypes of PROCESS_TIMER may be defined. Typically, PROCESS_TIMER **SHOULD** be modeled as a data item for the Device element, but **MAY** be modeled for either a Controller or Path *Structural Element* in the XML document. A subType **MUST** always be specified.	SECOND
PROCESS	The measurement of the time from the beginning of production of a part or product on a piece of equipment until the time that production is complete for that part or product on that piece of equipment. This includes the time that the piece of equipment is running, producing parts or products, or in the process of producing parts.	SECOND
DELAY	Measurement of the time that a process is waiting and unable to perform its intended function.	SECOND
RESISTANCE	The degree to which a substance opposes the passage of an electric current.	OHM
ROTARY_VELOCITY	The rotational speed of a rotary axis.	REVOLUTION/MINUTE
ACTUAL	The measured value of rotational speed that the rotary axis is spinning.	REVOLUTION/MINUTE
COMMANDED	The rotational speed as specified by the Controller type component. The COMMANDED velocity is a calculated value that includes adjustments and overrides.	REVOLUTION/MINUTE
PROGRAMMED	The rotational velocity specified by a logic or motion program or set by a switch	REVOLUTION/MINUTE
~~OVERRIDE~~	~~The operator's overridden value. Percent of commanded.~~ **DEPRECATED** in *Version 1.3*. See EVENT category DataItems.	~~PERCENT~~

DataItem type/subType	Description	Units
SOUND_LEVEL	Measurement of a sound level or sound pressure level relative to atmospheric pressure	DECIBEL
NO_SCALE	No weighting factor on the frequency scale	DECIBEL
A_SCALE	A Scale weighting factor. This is the default weighting factor if no factor is specified	DECIBEL
B_SCALE	B Scale weighting factor	DECIBEL
C_SCALE	C Scale weighting factor	DECIBEL
D_SCALE	D Scale weighting factor	DECIBEL
~~SPINDLE_SPEED~~	**DEPRECATED** in *Version 1.2*. Replaced by ROTARY_VELOCITY	
~~ACTUAL~~	~~The rotational speed of a rotary axis. ROTARY_MODE **MUST** be SPINDLE.~~	~~REVOLUTION/MINUTE~~
~~COMMANDED~~	~~The rotational speed the as specified by the Controller type Component.~~	~~REVOLUTION/MINUTE~~
~~OVERRIDE~~	~~The operator's overridden value. Percent of commanded.~~	~~PERCENT~~
STRAIN	The amount of deformation per unit length of an object when a load is applied.	PERCENT
TEMPERATURE	The measurement of temperature	CELSIUS
TENSION	A measurement of a force that stretches or elongates an object	NEWTON
TILT	A measurement of angular displacement	MICRO_RADIAN
TORQUE	The turning force exerted on an object or by an object	NEWTON_METER
VOLT_AMPERE	The measure of the apparent power in an electrical circuit, equal to the product of root-mean-square (RMS) voltage and RMS current (commonly referred to as VA)	VOLT_AMPERE
VOLT_AMPERE_REACTIVE	The measurement of reactive power in an AC electrical circuit (commonly referred to as VAR)	VOLT_AMPERE_REACTIVE

DataItem type/subType	Description	Units
VELOCITY	The rate of change of position.	MILLIMETER/SECOND
VISCOSITY	A measurement of a fluid's resistance to flow	PASCAL_SECOND
VOLTAGE	The measurement of electrical potential between two points	VOLT
ALTERNATING	The measurement of alternating voltage. If not specified further in statistic, defaults to RMS voltage	VOLT
DIRECT	The measurement of DC voltage	VOLT
ACTUAL	The measured voltage being delivered from a power source.	VOLT
TARGET	The desired or preset voltage to be delivered from a power source.	VOLT
WATTAGE	The measurement of power flowing through or dissipated by an electrical circuit or piece of equipment.	WATT
ACTUAL	The measured wattage being delivered from a power source.	WATT
TARGET	The desired or preset wattage to be delivered from a power source.	WATT

8.2 Data Items in category EVENT

DataItem types in the EVENT category represent a discrete piece of information from a piece of equipment. EVENT does not have intermediate values that vary over time.

An EVENT is information that, when provided at any specific point in time, represents the current state of the piece of equipment.

There are two types of EVENT: those representing state, with two or more discrete values, and those representing messages that contain plain text data.

The table below defines the DataItem types and subtypes defined for the EVENT category. The subtypes are indented below their associated types.

DataItem type/subType	Description
ACTUATOR_STATE	Represents the operational state of an apparatus for moving or controlling a mechanism or system. The valid data value **MUST** be ACTIVE or INACTIVE.
~~ALARM~~	**DEPRECATED in *Version 1.1*.** Replaced with CONDITION category.
ACTIVE_AXES	The set of axes currently associated with a Path or Controller *Structural Element*. If this DataItem is not provided, it will be assumed that all axes are currently associated with the Controller *Structural Element* and with an individual Path. The valid data value for ACTIVE_AXES **SHOULD** be a space-delimited set of axes reported as the value of the name attribute for each axis. If name is not available, the piece of equipment **MUST** report the value of the nativeName attribute for each axis.
AVAILABILITY	Represents the *Agent's* ability to communicate with the data source. This **MUST** be provided for a Device Element and **MAY** be provided for any other *Structural Element*. The valid data value **MUST** be AVAILABLE or UNAVAILABLE.
AXIS_COUPLING	Describes the way the axes will be associated to each other. This is used in conjunction with COUPLED_AXES to indicate the way they are interacting. The valid data value **MUST** be TANDEM, SYNCHRONOUS, MASTER, and SLAVE. The coupling **MUST** be viewed from the perspective of a specific axis. Therefore, a MASTER coupling indicates that this axis is the master for the COUPLED_AXES.
AXIS_FEEDRATE_OVERRIDE	The value of a signal or calculation issued to adjust the feedrate of an individual linear type axis. The value provided for AXIS_FEEDRATE_OVERRIDE is expressed as a percentage of the designated feedrate for the axis. When AXIS_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the axis is limited to the value of the original feedrate multiplied by the value of the AXIS_FEEDRATE_OVERRIDE. There **MAY** be different subtypes of AXIS_FEEDRATE_OVERRIDE; each representing an override value for a designated subtype of feedrate depending on the state of operation of the axis. The subtypes of operation of an axis are currently defined as PROGRAMMED, JOG, and RAPID.

DataItem type/subType	Description
JOG	The value of a signal or calculation issued to adjust the feedrate of an individual linear type axis when that axis is being operated in a manual state or method (jogging). When the JOG subtype of AXIS_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the axis is limited to the value of the original JOG subtype of the AXIS_FEEDRATE multiplied by the value of the JOG subtype of AXIS_FEEDRATE_OVERRIDE.
PROGRAMMED	The value of a signal or calculation issued to adjust the feedrate of an individual linear type axis that has been specified by a logic or motion program or set by a switch. When the PROGRAMMED subtype of AXIS_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the axis is limited to the value of the original PROGRAMMED subtype of the AXIS_FEEDRATE multiplied by the value of the PROGRAMMED subtype of AXIS_FEEDRATE_OVERRIDE.
RAPID	The value of a signal or calculation issued to adjust the feedrate of an individual linear type axis that is operating in a rapid positioning mode. When the RAPID subtype of AXIS_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the axis is limited to the value of the original RAPID subtype of the AXIS_FEEDRATE multiplied by the value of the RAPID subtype of AXIS_FEEDRATE_OVERRIDE.
AXIS_INTERLOCK	An indicator of the state of the axis lockout function when power has been removed and the axis is allowed to move freely. The valid data value **MUST** be ACTIVE or INACTIVE.
AXIS_STATE	An indicator of the controlled state of a LINEAR or ROTARY component representing an axis. The valid data value **MUST** be HOME, TRAVEL, PARKED, or STOPPED.
BLOCK	The line of code or command being executed by a Controller *Structural Element*. The value reported for Block **MUST** include the entire expression for a line of program code, including all parameters.
BLOCK_COUNT	The total count of the number of blocks of program code that have been executed since execution started. BLOCK_COUNT counts blocks of program code executed regardless of program structure (e.g., looping or branching within the program). The starting value for BLOCK_COUNT **MAY** be established by an initial value provided in the Constraint element defined for the data item.

DataItem type/subType	Description
CHUCK_INTERLOCK	An indication of the state of an interlock function or control logic state intended to prevent the associated CHUCK component from being operated. The valid data value **MUST** be ACTIVE or INACTIVE.
MANUAL_UNCLAMP	An indication of the state of an operator controlled interlock that can inhibit the ability to initiate an unclamp action of an electronically controlled chuck. The valid data value **MUST** be ACTIVE or INACTIVE. When MANUAL_UNCLAMP is ACTIVE, it is expected that a chuck cannot be unclamped until MANUAL_UNCLAMP is set to INACTIVE.
CHUCK_STATE	An indication of the operating state of a mechanism that holds a part or stock material during a manufacturing process. It may also represent a mechanism that holds any other mechanism in place within a piece of equipment. The valid data value **MUST** be OPEN, CLOSED, or UNLATCHED.
~~CODE~~	**DEPRECATED** in *Version 1.1*.
COMPOSITION_STATE	An indication of the operating condition of a mechanism represented by a Composition type element. A subType **MUST** always be specified. A compositionId **MUST** always be specified.
ACTION	An indication of the operating state of a mechanism represented by a Composition type component. The operating state indicates whether the Composition element is activated or disabled. The valid data value **MUST** be ACTIVE or INACTIVE.
LATERAL	An indication of the position of a mechanism that may move in a lateral direction. The mechanism is represented by a Composition type component. The position information indicates whether the Composition element is positioned to the right, to the left, or is in transition. The valid data value **MUST** be RIGHT, LEFT, or TRANSITIONING.
MOTION	An indication of the open or closed state of a mechanism. The mechanism is represented by a Composition type component. The operating state indicates whether the state of the Composition element is open, closed, or unlatched. The valid data value **MUST** be OPEN, UNLATCHED, or CLOSED.

DataItem type/subType	Description
SWITCHED	An indication of the activation state of a mechanism represented by a Composition type component. The activation state indicates whether the Composition element is activated or not. The valid data value **MUST** be ON or OFF.
VERTICAL	An indication of the position of a mechanism that may move in a vertical direction. The mechanism is represented by a Composition type component. The position information indicates whether the Composition element is positioned to the top, to the bottom, or is in transition. The valid data value **MUST** be UP, DOWN, or TRANSITIONING.
CONTROLLER_MODE	The current mode of the Controller component. The valid data value **MUST** be AUTOMATIC, MANUAL, MANUAL_DATA_INPUT, SEMI_AUTOMATIC, or EDIT.
CONTROLLER_MODE_OVERRIDE	A setting or operator selection that changes the behavior of a piece of equipment. A subType **MUST** always be specified.
DRY_RUN	A setting or operator selection used to execute a test mode to confirm the execution of machine functions. The valid data value **MUST** be ON or OFF. When DRY_RUN is ON, the equipment performs all of its normal functions, except no part or product is produced. If the equipment has a spindle, spindle operation is suspended.
SINGLE_BLOCK	A setting or operator selection that changes the behavior of the controller on a piece of equipment. The valid data value **MUST** be ON or OFF. Program execution is paused after each BLOCK of code is executed when SINGLE_BLOCK is ON. When SINGLE_BLOCK is ON, EXECUTION **MUST** change to INTERRUPTED after completion of each BLOCK of code.
MACHINE_AXIS_LOCK	A setting or operator selection that changes the behavior of the controller on a piece of equipment. The valid data value **MUST** be ON or OFF. When MACHINE_AXIS_LOCK is ON, program execution continues normally, but no equipment motion occurs

DataItem type/subType	Description
OPTIONAL_STOP	A setting or operator selection that changes the behavior of the controller on a piece of equipment. The valid data value **MUST** be ON or OFF. The program execution is stopped after a specific program block is executed when OPTIONAL_STOP is ON. In the case of a G-Code program, a program BLOCK containing a M01 code designates the command for an OPTIONAL_STOP. EXECUTION **MUST** change to OPTIONAL_STOP after a program block specifying an optional stop is executed and the OPTIONAL_STOP selection is ON.
TOOL_CHANGE_STOP	A setting or operator selection that changes the behavior of the controller on a piece of equipment. The valid data value **MUST** be ON or OFF. Program execution is paused when a command is executed requesting a cutting tool to be changed. EXECUTION **MUST** change to INTERRUPTED after completion of the command requesting a cutting tool to be changed and TOOL_CHANGE_STOP is ON.
COUPLED_AXES	Refers to the set of associated axes. The valid data value for COUPLED_AXES **SHOULD** be a space-delimited set of axes reported as the value of the name attribute for each axis. If name is not available, the piece of equipment **MUST** report the value of the nativeName attribute for each axis.
DIRECTION	The direction of motion. A subType **MUST** always be specified.
ROTARY	The rotational direction of a rotary motion using the right hand rule convention. The valid data value **MUST** be CLOCKWISE or COUNTER_CLOCKWISE.
LINEAR	The direction of motion of a linear motion. The valid data value **MUST** be POSTIVE or NEGATIVE.
DOOR_STATE	The opened or closed state of the door. The valid data value **MUST** be OPEN, UNLATCHED, or CLOSED.
END_OF_BAR	An indication of whether the end of a piece of bar stock being feed by a bar feeder has been reached. The valid data value **MUST** be expressed as a Boolean expression of YES or NO.

DataItem type/subType	Description
PRIMARY	Specific applications **MAY** reference one or more locations on a piece of bar stock as the indication for the END_OF_BAR. The main or most important location **MUST** be designated as the PRIMARY indication for the END_OF_BAR. If no subType is specified, PRIMARY **MUST** be the default END_OF_BAR indication.
AUXILIARY	When multiple locations on a piece of bar stock are referenced as the indication for the END_OF_BAR, the additional location(s) **MUST** be designated as AUXILIARY indication(s) for the END_OF_BAR.
EMERGENCY_STOP	The current state of the emergency stop signal. The valid data value **MUST** be ARMED (the circuit is complete and the device is allowed to operate) or TRIGGERED (the circuit is open and the device must cease operation).
EQUIPMENT_MODE	An indication that a piece of equipment, or a sub-part of a piece of equipment, is performing specific types of activities. EQUIPMENT_MODE **MAY** have more than one subtype defined. A subType **MUST** always be specified.
LOADED	An indication that the sub-parts of a piece of equipment are under load. Example: For traditional machine tools, this is an indication that the cutting tool is assumed to be engaged with the part. The valid data value **MUST** be ON or OFF.
WORKING	An indication that a piece of equipment is performing any activity – the equipment is active and performing a function under load or not. Example: For traditional machine tools, this includes when the piece of equipment is LOADED, making rapid moves, executing a tool change, etc. The valid data value **MUST** be ON or OFF.
OPERATING	An indication that the major sub-parts of a piece of equipment are powered or performing any activity whether producing a part or product or not. Example: For traditional machine tools, this includes when the piece of equipment is WORKING or it is idle. The valid data value **MUST** be ON or OFF.

DataItem type/subType	Description
POWERED	An indication that primary power is applied to the piece of equipment and, as a minimum, the controller or logic portion of the piece of equipment is powered and functioning or components that are required to remain on are powered. Example: Heaters for an extrusion machine that required to be powered even when the equipment is turned off. The valid data value **MUST** be ON or OFF.
DELAY	An indication that a piece of equipment is waiting for an event or an action to occur.
EXECUTION	The execution status of the Controller. The valid data value **MUST** be READY, ACTIVE, INTERRUPTED, FEED_HOLD, STOPPED, OPTIONAL_STOP, PROGRAM_STOPPED, or PROGRAM_COMPLETED.
FUNCTIONAL_MODE	The current intended production status of the device or component. Typically, the FUNCTIONAL_MODE **SHOULD** be modeled as a data item for the Device element, but **MAY** be modeled for any *Structural Element* in the XML document. The valid data value **MUST** be PRODUCTION, SETUP, TEARDOWN, MAINTENANCE, or PROCESS_DEVELOPMENT.
HARDNESS	The measurement of the hardness of a material. The measurement does not provide a unit. A subType **MUST** always be specified to designate the hardness scale associated with the measurement.
ROCKWELL	A scale to measure the resistance to deformation of a surface.
VICKERS	A scale to measure the resistance to deformation of a surface.
SHORE	A scale to measure the resistance to deformation of a surface.
BRINELL	A scale to measure the resistance to deformation of a surface.
LEEB	A scale to measure the elasticity of a surface.
MOHS	A scale to measure the resistance to scratching of a surface.
INTERFACE_STATE	The current functional or operational state of an Interface type element indicating whether the interface is active or is not currently functioning. The valid data value **MUST** be ENABLED or DISABLED.

DataItem type/subType	Description
~~LINE~~	~~The current line of code being executed.~~ ~~The data will be an alpha numeric value representing the line number of the current line of code being executed.~~ **DEPRECATED** in *Version 1.4*
~~MAXIMUM~~	~~The maximum line number of the code being executed.~~
~~MINIMUM~~	~~The minimum line number of the code being executed.~~
LINE_LABEL	An optional identifier for a BLOCK of code in a PROGRAM.
LINE_NUMBER	A reference to the position of a block of program code within a control program. The line number **MAY** represent either an absolute position starting with the first line of the program or an incremental position relative to the occurrence of the last LINE_LABEL. LINE_NUMBER does not change subject to any looping or branching in a control program. A subType **MUST** be defined.
ABSOLUTE	The position of a block of program code relative to the beginning of the control program.
INCREMENTAL	The position of a block of program code relative to the occurrence of the last LINE_LABEL encountered in the control program.
MATERIAL	The identifier of a material used or consumed in the manufacturing process. The valid data value **MUST** be a text string.
MESSAGE	Any text string of information to be transferred from a piece of equipment to a client software application.
OPERATOR_ID	The identifier of the person currently responsible for operating the piece of equipment. **DEPRECATION WARNING:** May be deprecated in the future. See USER below.
PALLET_ID	The identifier for a pallet. The valid data value **MUST** be a text string.
PART_COUNT	The current count of parts produced as represented by the Controller. The valid data value **MUST** be an integer value.
ALL	The count of all the parts produced. If the subtype is not given, this is the default.

DataItem type/subType	Description
GOOD	Indicates the count of correct parts made.
BAD	Indicates the count of incorrect parts produced.
TARGET	Indicates the number of parts that are projected or planned to be produced.
REMAINING	The number of parts remaining in stock or to be produced.
PART_ID	An identifier of a part in a manufacturing operation. The valid data value **MUST** be a text string.
PART_NUMBER	An identifier of a part or product moving through the manufacturing process. The valid data value **MUST** be a text string.
PATH_FEEDRATE_OVERRIDE	The value of a signal or calculation issued to adjust the feedrate for the axes associated with a Path component that may represent a single axis or the coordinated movement of multiple axes. The value provided for PATH_FEEDRATE_OVERRIDE is expressed as a percentage of the designated feedrate for the path. When PATH_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the path is limited to the value of the original feedrate multiplied by the value of the PATH_FEEDRATE_OVERRIDE. There **MAY** be different subtypes of PATH_FEEDRATE_OVERRIDE; each representing an override value for a designated subtype of feedrate depending on the state of operation of the path. The states of operation of a path are currently defined as PROGRAMMED, JOG, and RAPID.
JOG	The value of a signal or calculation issued to adjust the feedrate of the axes associated with a Path component when the axes, or a single axis, are being operated in a manual mode or method (jogging). When the JOG subtype of PATH_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the axes, or a single axis, associated with the path are limited to the value of the original JOG subtype of the PATH_FEEDRATE multiplied by the value of the JOG subtype of PATH_FEEDRATE_OVERRIDE.

DataItem type/subType	Description
PROGRAMMED	The value of a signal or calculation issued to adjust the feedrate of the axes associated with a Path component when the axes, or a single axis, are operating as specified by a logic or motion program or set by a switch. When the PROGRAMMED subtype of PATH_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the axes, or a single axis, associated with the path are limited to the value of the original PROGRAMMED subtype of the PATH_FEEDRATE multiplied by the value of the PROGRAMMED subtype of PATH_FEEDRATE_OVERRIDE.
RAPID	The value of a signal or calculation issued to adjust the feedrate of the axes associated with a Path component when the axes, or a single axis, are being operated in a rapid positioning mode or method (rapid). When the RAPID subtype of PATH_FEEDRATE_OVERRIDE is applied, the resulting commanded feedrate for the axes, or a single axis, associated with the path are limited to the value of the original RAPID subtype of the PATH_FEEDRATE multiplied by the value of the RAPID subtype of PATH_FEEDRATE_OVERRIDE.
PATH_MODE	Describes the operational relationship between a PATH *Structural Element* and another PATH *Structural Element* for pieces of equipment comprised of multiple logical groupings of controlled axes or other logical operations. The valid data value **MUST** be INDEPENDENT, MASTER, SYNCHRONOUS, or MIRROR. The default value **MUST** be INDEPENDENT if PATH_MODE is not specified.
POWER_STATE	The indication of the status of the source of energy for a *Structural Element* to allow it to perform its intended function or the state of an enabling signal providing permission for the *Structural Element* to perform its functions. The valid data value **MUST** be ON or OFF. **DEPRECATION WARNING:** May be deprecated in the future.
LINE	The state of the power source for the *Structural Element*.
CONTROL	The state of the enabling signal or control logic that enables or disables the function or operation of the *Structural Element*.
~~POWER_STATUS~~	**DEPRECATED** in *Version 1.1*.
PROGRAM	The name of the logic or motion program being executed by the Controller component. The valid data value **MUST** be a text string.

DataItem type/subType	Description
PROGRAM_EDIT	An indication of the Controller component's program editing mode. On many controls, a program can be edited while another program is currently being executed. The valid data value **MUST** be: ACTIVE: The controller is in the program edit mode. READY: The controller is capable of entering the program edit mode and no function is inhibiting a change of mode. NOT_READY: A function is inhibiting the controller from entering the program edit mode.
PROGRAM_EDIT_NAME	The name of the program being edited. This is used in conjunction with PROGRAM_EDIT when in ACTIVE state. The valid data value **MUST** be a text string.
PROGRAM_COMMENT	A comment or non-executable statement in the control program. The valid data value **MUST** be a text string.
PROGRAM_HEADER	The non-executable header section of the control program. The valid data value **MUST** be a text string.
ROTARY_MODE	The mode for a Rotary type axis. The valid data value **MUST** be SPINDLE, INDEX, or CONTOUR.
ROTARY_VELOCITY_OVERRIDE	A command issued to adjust the programmed velocity for a Rotary type axis. This command represents a percentage change to the velocity calculated by a logic or motion program or set by a switch for a Rotary type axis. ROTARY_VELOCITY_OVERRIDE is expressed as a percentage of the programmed ROTARY_VELOCITY.
SERIAL_NUMBER	The serial number associated with a Component, Asset, or Device. The valid data value **MUST** be a text string.
SPINDLE_INTERLOCK	An indication of the status of the spindle for a piece of equipment when power has been removed and it is free to rotate. The valid data value **MUST** be: • ACTIVE if power has been removed and the spindle cannot be operated. • INACTIVE if power to the spindle has not been deactivated.
~~TOOL_ID~~	**DEPRECATED** in *Version 1.2*. See TOOL_ASSET_ID. ~~The identifier of the tool currently in use for a given Path~~

DataItem type/subType	Description
TOOL_ASSET_ID	The identifier of an individual tool asset. The valid data value **MUST** be a text string.
TOOL_NUMBER	The identifier of a tool provided by the piece of equipment controller. The valid data value **MUST** be a text string.
TOOL_OFFSET	A reference to the tool offset variables applied to the active cutting tool associated with a Path in a Controller type component. The valid data value **MUST** be a text string. The reported value returned for TOOL_OFFSET identifies the location in a table or list where the actual tool offset values are stored. A subType **MUST** always be specified.
RADIAL	A reference to a radial type tool offset variable.
LENGTH	A reference to a length type tool offset variable.
USER	The identifier of the person currently responsible for operating the piece of equipment. A subType **MUST** always be specified.
OPERATOR	The identifier of the person currently responsible for operating the piece of equipment.
MAINTENANCE	The identifier of the person currently responsible for performing maintenance on the piece of equipment.
SET_UP	The identifier of the person currently responsible for preparing a piece of equipment for production or restoring the piece of equipment to a neutral state after production.
WIRE	The identifier for the type of wire used as the cutting mechanism in Electrical Discharge Machining or similar processes. The valid data value **MUST** be a text string.
WORKHOLDING_ID	The identifier for the workholding currently in use. The valid data value **MUST** be a text string.
WORK_OFFSET	A reference to the offset variables for a work piece or part associated with a Path in a Controller type component. The valid data value **MUST** be a text string. The reported value returned for WORK_OFFSET identifies the location in a table or list where the actual tool offset values are stored.

8.3 Data Items in `category` CONDITION

CONDITION category data items report data representing a *Structural Element's* status regarding its ability to operate or it provides an indication whether the data reported for the *Structural Element* is within an expected range.

CONDITION is reported differently than SAMPLE or EVENT. CONDITION **MUST** be reported as NORMAL, WARNING, or FAULT.

All DataItem types in the SAMPLE category **MAY** have associated CONDITION states. CONDITION states indicate whether the value for the data is within an expected range and **MUST** be reported as NORMAL, or the value is unexpected or out of tolerance for the data and a WARNING or FAULT **MUST** be provided.

Some DataItem types in the EVENT category **MAY** have associated CONDITION states.

Additional CONDITION types are provided to represent the health and fault status of *Structural Elements*. The table below defines these additional DataItem types.

CONDITION type data items are unlike other data item types since they **MAY** have multiple concurrently active values at any point in time.

DataItem Type	Description
ACTUATOR	An indication of a fault associated with an actuator.
CHUCK_INTERLOCK	An indication of the operational condition of the interlock function for an electronically controller chuck.
COMMUNICATIONS	An indication that the piece of equipment has experienced a communications failure.
DATA_RANGE	An indication that the value of the data associated with a measured value or a calculation is outside of an expected range.
DIRECTION	An indication of a fault associated with the direction of motion of a *Structural Element.*
END_OF_BAR	An indication that the end of a piece of bar stock has been reached.
HARDWARE	An indication of a fault associated with the hardware subsystem of the *Structural Element*.
INTERFACE_STATE	An indication of the operation condition of an Interface component.
LOGIC_PROGRAM	An indication that an error occurred in the logic program or programmable logic controller (PLC) associated with a piece of equipment.

DataItem Type	Description
MOTION_PROGRAM	An indication that an error occurred in the motion program associated with a piece of equipment
SYSTEM	A general purpose indication of a fault associated with a piece of equipment that is classified elsewhere.

9 *Sensor*

Sensor is a unique type of a piece of equipment. A *Sensor* is typically comprised of two major components: a *sensor unit* that provides signal processing, conversion, and communications and the *sensing elements* that provides a signal or measured value.

In MTConnect, the *sensor unit* is modeled as a *Lower Level* `Component` called `Sensor`. The *sensing element* may be modeled as a `Composition` element of a `Sensor` element and the measured value would be modeled as a `DataItem` (See *Section 8* of this document for more information on `DataItem` elements). Each *sensor unit* may have multiple *sensing elements*; each representing the data for a variety of measured values.

Example: A pressure transducer could be modeled as a `Sensor` (`Component`) with a `name =` *Pressure Transducer B* and its measured value could be modeled as a `PRESSURE` type `DataItem`.

While a *Sensor* may be modeled in the XML document in different ways, it will always be modeled to associate the information measured by each *sensor element* with the *Structural Element* to which the measured value is most closely associated.

9.1 Sensor Data

The most basic implementation of a sensor occurs when the *sensing element* itself is not identified in the data model, but the data that is measured by the *sensing element* is provided as a data item associated with a `Component`. An example would be the measured value of the temperature of a spindle motor. This would be represented as a `DataItem` called `TEMPERATURE` that is associated with the `Rotary` type axis element called "C" as follows:

```
<Components>
  <Axes
    <Components>
      <Rotary id="c" name="C">
        <DataItems>
          <DataItem type="TEMPERATURE" id="ctemp" category="SAMPLE"
            name="Stemp" units="DEGREE"/>
        </DataItems>
      </Rotary>
    </Components>
  </Axes>
</Components>
```

A sensor may measure values associated with any `Component` or `Device` element. Some examples of how sensor data may be modeled are represented in *Figure 12* below:

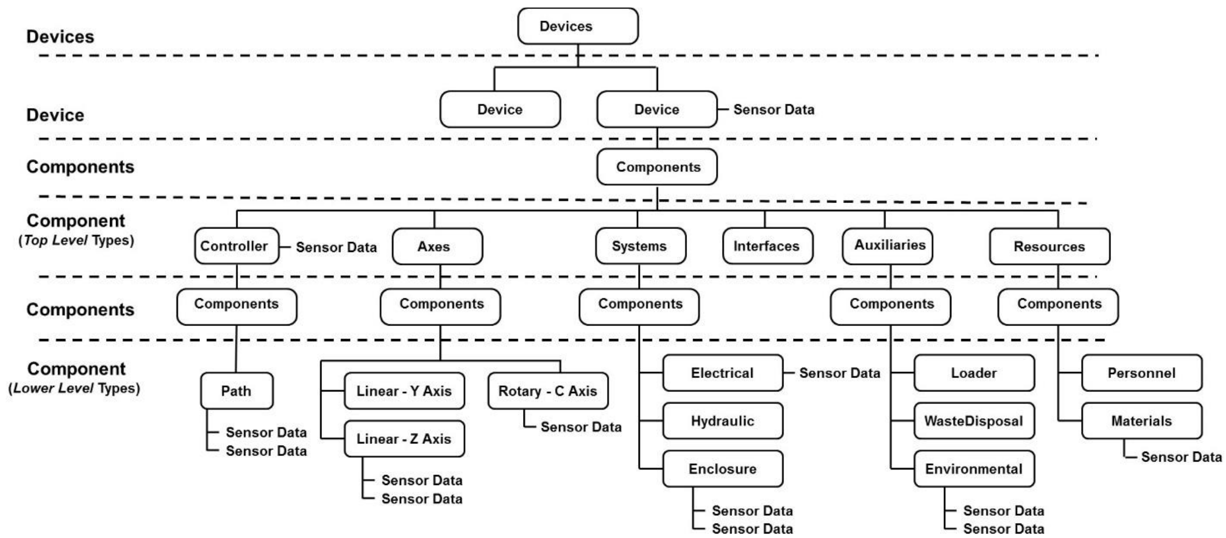


Figure 20: Sensor Data Associations

9.2 Sensor Unit

A *sensor unit* is an intelligent piece of equipment that manages the functions of one or more *sensing elements*.

Typical functions of the *sensor unit* include:

- convert low level signals from the *sensing elements* into data that can be used by other pieces of equipment. (Example: Convert a non-linear millivolt signal from a temperature sensor into a scaled temperature value that can be transmitted to another piece of equipment.)
- process *sensing element* data into calculated values. (Example: temperature sensor data is converted into calculated values of average temperature, maximum temperature, minimum temperature, etc.)
- provide calibration and configuration information associated with each *sensing element*
- monitor the health and integrity of the *sensing elements* and the *sensor unit*. (Example: The *sensor unit* may provide diagnostics on each *sensing element* (e.g., open wire detection) and itself (e.g., measure internal temperature of the *sensor unit*).

Depending on how the *sensor unit* is used, it may be considered as either an independent piece of equipment and modeled in the XML document as a `Device,` or it may be modeled as a *Lower Level* `Component` called `Sensor` if it is integral to a piece of equipment.

A `Sensor` **MAY** have its own `uuid` so it can be tracked throughout its lifetime.

The following examples demonstrate how a *Sensor* may be modeled in the XML document differently based on how the *Sensor* functions within the overall piece of equipment.

Example#1: If the *Sensor* provides vibration measurement data for the spindle on a piece of equipment, it could be modeled as a `Sensor` for rotary axis named C.

```
<Components>
  <Axes>
    <Components>
      <Rotary id="c" name="C">
        <Components>
          <Sensor id="spdlm" name="Spindlemonitor">
            <DataItems>
              <DataItem type="DISPLACEMENT" id="cvib"
                category="SAMPLE” name="Svib" units="MILLIMETER"/>
            </DataItems>
          </Sensor >
        <Components>
      </Rotary>
    </Components>
   </Axes>
 </Components>
```

Example#2: If a *Sensor* provides measurement data for multiple `Component` elements within a piece of equipment and is not associated with any particular `Component` element, it **MAY** be modeled in the XML document as an independent *Lower Level* `Component` and the data associated with measurements are associated with their associated `Component` elements.

This example represents a *sensor unit* with two *sensing elements*, one measures spindle vibration and the other measures the temperature for the X axis. The *sensor unit* also has a *sensing element* measuring the internal temperature of the *sensor unit*.

```
<Device id="d1" uuid="HM1" name="HMC_3Axis">
  <Description>3 Axis Mill</Description>
  <Components>
    <Axes>
      <Components>
        <Sensor id="sens1" name="Sensorunit">
          <DataItems>
            <DataItem type="TEMPERATURE" id="sentemp"
                category="SAMPLE" name="Sensortemp" units="DEGREE"/>
          </DataItems>
        </Sensor>
        <Rotary id="c" name="C">
          <DataItems>
            <DataItem type="DISPLACEMENT" id="cvib" category="SAMPLE"
                name="Svib" units="MILLIMETER">
              <Source componentid="sens1"/>
            <DataItem/>
          </DataItems>
        </Rotary>
        <Linear id="x" name="X">
          <DataItems>
            <DataItem type="TEMPERATURE" id="xt" category="SAMPLE"
                name="Xtemp" units="DEGREE">
              <Source componentid="sens1"/>
            <DataItem/>
          </DataItems>
        </Linear>
      </Components>
    </Axes>
  </Components>
</Device>
```

9.3 Sensor Configuration

When a *Sensor unit* is modeled in the XML document as a `Component` or as a separate piece of equipment, it may provide additional configuration information for the *sensor elements* and the *sensor unit* itself.

Configuration data provides information required for maintenance and support of the sensor.

Configuration data is *only* available when the *Sensor unit* is modeled as a `Component` or a separate piece of equipment. For details on the modeling of configuration data in the XML document, see *Section 4.4.3.2* `Configuration` *for* `Component`. Details specific to `SensorConfiguration` are provided below.

When Sensor represents the *sensor unit* for multiple *sensing element(s)*, each *sensing element* is represented by a Channel. The sensor unit itself and each Channel representing one *sensing element* **MAY** have its own configuration data.

SensorConfiguration can contain any descriptive content for a *sensor unit*. This element is defined to contain mixed content and additional XML elements (indicated by the any element in the schema below) **MAY** be added to extend the schema for SensorConfiguration.

The following XML schema represents the structure of the `SensorConfiguration` XML element showing the attributes defined for `SensorConfiguration`.

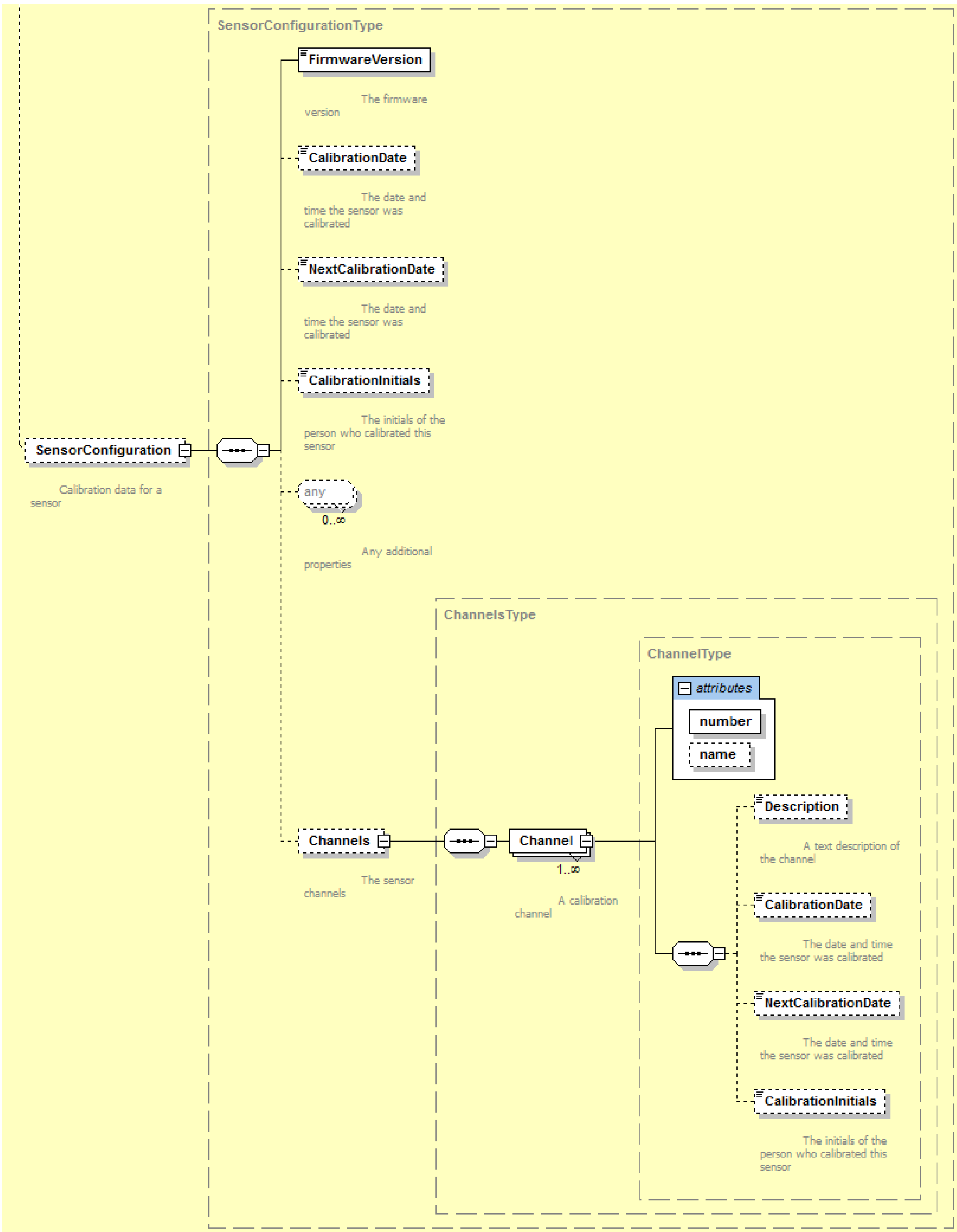


Figure 21: `SensorConfiguration` Schema Diagram

Element	Description	Occurrence
`SensorConfiguration`	An element that can contain descriptive content defining the configuration information for `Sensor`. For `Sensor`, the valid configuration is `SensorConfiguration` which provides data from a subset of items commonly found in a transducer electronic data sheet for sensors and actuators called TEDS. TEDS formats are defined in IEEE 1451.0 and 1451.4 transducer interface standards (ref 15 and 16, respectively). MTConnect does not support all of the data represented in the TEDS data, nor does it duplicate the function of the TEDS data sheets.	0..1

9.3.1 Elements for `SensorConfiguration`

The following table defines the configuration elements available for `SensorConfiguration`:

Element	Description	Occurrence
`FirmwareVersion`	Version number for the *sensor unit* as specified by the manufacturer. `FirmwareVersion` is a required element if `SensorConfiguration` is used. The data value for `FirmwareVersion` is provided in the CDATA for this element and **MAY** be any numeric or text content.	1
`CalibrationDate`	Date upon which the *sensor unit* was last calibrated. The data value for `CalibrationDate` is provided in the CDATA for this element and **MUST** be represented in the W3C ISO 8601 format.	0..1
`NextCalibrationDate`	Date upon which the *sensor unit* is next scheduled to be calibrated. The data value for NextCal`ibrationDate` is provided in the CDATA for this element and **MUST** be represented in the W3C ISO 8601 format.	0..1

Element	Description	Occurrence
`CalibrationInitials`	The initials of the person verifying the validity of the calibration data. The data value for `CalibrationInitials` is provided in the CDATA for this element and **MAY** be any numeric or text content.	0..1
`Channels`	When `Sensor` represents multiple *sensing elements*, each *sensing element* is represented by a `Channel` for the `Sensor`. `Channels` is an XML container used to organize information for the *sensing elements*.	0..1

9.3.1.1 Attributes for `Channel`

`Channel` represents each *sensing element* connected to a *sensor unit*. The table below defines the `attributes` for `Channel`:

Attribute	Description	Occurrence
`number`	A unique identifier that will only refer to a specific *sensing element*. `number` is a required attribute. For example, this can be the manufacturer code and the serial number. `number` **SHOULD** be alphanumeric and not exceeding 255 characters. An NMTOKEN XML type.	1
`name`	The `name` of the *sensing element*. `name` is an optional attribute. `name` **SHOULD** be unique within the *sensor unit* to allow for easier data integration. An NMTOKEN XML type.	0..1

9.3.1.2 Elements for `Channel`

The following table describes the elements provided for `Channel`.

Element	Description	Occurrence
`Description`	An XML element that can contain any descriptive content. The `CDATA` of `Description` **MAY** include any additional descriptive information the implementer chooses to include regarding a *sensor element*.	0..1
`CalibrationDate`	Date upon which the *sensor unit* was last calibrated to the *sensor element* The data value for `CalibrationDate` is provided in the CDATA for this element and **MUST** be represented in the W3C ISO 8601 format.	0..1
`NextCalibrationDate`	Date upon which the *sensor element* is next scheduled to be calibrated with the *sensor unit*. The data value for `NextCalibrationDate` is provided in the CDATA for this element and **MUST** be represented in the W3C ISO 8601 format.	0..1
`CalibrationInitials`	The initials of the person verifying the validity of the calibration data The data value for `CalibrationInitials` is provided in the CDATA for this element and **MAY** be any numeric or text content.	0..1

The following is an example of the configuration data for `Sensor` that is modeled as a `Component`. It has `Configuration` data for the *sensor unit*, one `Channel` named A/D:1, and two `DataItems` – `Voltage` (as a `SAMPLE`) and `Voltage` (as a `CONDITION` or alarm).

```
<Sensor id="sensor" name="sensor">
  <Configuration>
    <SensorConfiguration>
      <FirmwareVersion>2.02</FirmwareVersion>
      <CalibrationDate>2010-05-16</CalibrationDate>
      <NextCalibrationDate>2010-05-16</NextCalibrationDate>
      <CalibrationInitials>WS</CalibrationInitials>
      <Channels>
        <Channel number="1" name="A/D:1">
          <Description>A/D With Thermister</Description>
        </Channel>
      </Channels>
    </SensorConfiguration>
  </Configuration>
  <DataItems>
    <DataItem category="CONDITION" id="senvc" type="VOLTAGE" />
    <DataItem category="SAMPLE" id="senv" type="VOLTAGE" units="VOLT"
      subType="DIRECT" />
  </DataItems>
</Sensor>
```

Appendices

A. Bibliography

1. Engineering Industries Association. *EIA Standard - EIA-274-D*, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.

2. ISO TC 184/SC4/WG3 N1089. *ISO/DIS 10303-238*: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.

3. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.

4. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.

5. International Organization for Standardization. *ISO 6983/1* – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.

6. Electronic Industries Association. *ANSI/EIA-494-B-1992*, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.

7. National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.

8. International Organization for Standardization. *ISO 10303-11*: 1994, Industrial automation systems and integration Product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

9. International Organization for Standardization. *ISO 10303-21*: 1996, Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.

10. H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.

11. International Organization for Standardization. *ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature.* Geneva, Switzerland, 2001.

12. ASME B5.57: *Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers,* 1998

13. ASME/ANSI B5.54: *Methods for Performance Evaluation of Computer Numerically Controlled Machining Centers. 2005.*

14. OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.*

15. IEEE STD 1451.0-2007*, Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH99684, October 5, 2007.*

16. IEEE STD 1451.4-1994, Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH95225, December 15, 2004.

MTConnect® Standard
Part 3.0 – Streams Information Model

Version 1.4.0

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

MTConnect® Specification and Materials

Table of Contents

Table of Figures

1 Purpose of This Document

This document, *Part 3.0 - Streams Information Model* of the MTConnect® Standard, establishes the rules and terminology that describes the information returned by an *MTConnect Agent* from a piece of equipment. The *Streams Information Model* also defines, in *Section 3*, the structure for the XML documents that are returned from an *MTConnect Agent* in response to a `Sample` or `Current` request.

Part 3.0 - Streams Information Model is not a stand-alone document. This document is used in conjunction with *Part 1.0 – Overview and Functionality* which defines the fundamentals of the operation of the MTConnect Standard and *Part 2.0 – Devices Information Model* that defines the semantic model representing the information that may be returned from a piece of equipment.

Note: *Part 5 – Interfaces* provides details on extensions to the *Streams Information Model* required to describe the interactions between pieces of equipment.

In the MTConnect Standard, *equipment* represents any tangible property that is used in the operation of a manufacturing facility. Examples of *equipment* are machine tools, ovens, sensor units, workstations, software applications, and bar feeders.

2 Terminology

Refer to *Section 5* of *Part 1.0 – Overview and Functionality* for a dictionary of terms, reserved language, and document conventions used in the MTConnect® Standard.

3 *Streams Information Model*

The *Streams Information Model* provides a representation of the data reported by a piece of equipment used for a manufacturing process, or used for any other purpose. Additional descriptive information associated with the reported data is defined in the `MTConnectDevices` document, which is described in *Part 2.0 – Devices Information Model*.

Information defined in the *Streams Information Model* allows a software application to (1) determine the value for *Data Entities* returned from a piece of equipment and (2) interpret the data associated with those *Data Entities* with the same meaning, value, and context that it had at its original source. To do this, the software application issues one of two HTTP requests to an *MTConnect Agent* associated with a piece of equipment. They are:

- `sample`**:** Returns a designated number of time stamped *Data Entities* from an *MTConnect Agent* associated with a piece of equipment; subject to any HTTP filtering associated with the request. See *Section 8.3.3 of Part 1.0 – Overview and Functionality* of the MTConnect Standard for details on the `sample` HTTP request.

- `current`: Returns a snapshot of either the most recent values or the values at a given sequence number for all *Data Entities* associated with a piece of equipment from an *MTConnect Agent*; subject to any HTTP filtering associated with the request. See *Section 8.3.2* of *Part 1.0 – Overview and Functionality* of the MTConnect Standard for details on the `current` HTTP request.

An *MTConnect Agent* responds to either the `sample` or `current` HTTP request with an `MTConnectStreams` XML document. This document contains information describing *Data Entities* reported by an *MTConnect Agent* associated with a piece of equipment. A client software application may correlate the information provided in the `MTConnectStreams` XML document with the physical and logical structure for that piece of equipment defined in the `MTConnectDevices` document to form a clear and unambiguous understanding of the information provided. (See details on the structure for a piece of equipment described in *Part 2.0 – Devices Information Model*).

The `MTConnectStreams` XML document is comprised of two sections: `Header` and `Streams.`

The `Header` section contains protocol related information as defined in *Section 6.5* of *Part 1.0 – Overview and Functionality* of the MTConnect Standard.

The `Streams` section of the `MTConnectStreams` document contains a `DeviceStream` XML container for each piece of equipment represented in the document. Each `DeviceStream` container is comprised of two primary types of XML elements – *Structural Elements* and *Data Entities*. The contents of the `DeviceStream` container are described in detail in this document, *Part 3.0* of the MTConnect Standard.

Structural Elements are defined for both the `MTConnectDevices` and the `MTConnectStreams` XML documents. These *Structural Elements* are used to provide a logical organization of the information provided in each document. While used for a similar purpose, the *Structural Elements* in the `MTConnectStreams` document are specifically designed to be distinctly different from those in the `MTConnectDevices` document:

- `MTConnectDevices` document: *Structural Elements* organize information that represents the physical and logical parts and sub-parts of a piece of equipment. (See *Part 2.0 –Devices Information Model, Section 4* of the MTConnect Standard for more details on *Structural Elements* used in the `MTConnectDevices` document).

- `MTConnectStreams` document: *Structural Elements* provide the structure to organize the data returned from a piece of equipment and establishes the proper context for that data. The *Structural Elements* specifically defined for use in the `MTConnectStreams` document are `DeviceStream` (described in *Section 4.2* of this document) and `ComponentStream` (described in *Section 4.3* of this document).

  `DeviceStream` and `ComponentStream` elements have a direct correlation to each of the *Structural Elements* defined in the `MTConnectDevices` document.

Data Entities that describe data reported by a piece of equipment are also defined for both the `MTConnectDevices` and the `MTConnectStreams` XML documents. The *Data Entities* provided in both documents directly relate to each other. However, *Data Entities* are used for different purposes in each document:

- `MTConnectDevices` document: *Data Entity* elements define the data that may be returned from a piece of equipment. *Part 2.0 – Devices Information Model, Sections 7 and 8* lists the possible *Data Entity* XML elements that can be returned in a `MTConnectDevices` document.

- `MTConnectStreams` document: *Data Entity* elements provide the data reported by a piece of equipment. This data is organized in separate `ComponentStream` XML containers for each of the *Structural Elements* defined in the `MTConnectDevices` document associated with the data that is reported by a piece of equipment.

Within each `ComponentStream` XML container in the `MTConnectStreams` document, *Data Entities* are organized into three types of XML container elements - `Samples`, `Events`, and `Condition`. (Refer to *Sections 5 and 6* of this document for more information on these elements.)

4 *Structural Elements* for `MTConnectStreams`

Structural Elements are XML elements that form the logical structure for the `MTConnectStreams` XML document. These elements are used to organize the information and data that is reported by an *MTConnect Agent* for a piece of equipment. Refer to *Figure 1* below for an overview of the *Structural Elements* used in an `MTConnectStreams` document.

The first, or highest level, *Structural Element* in an `MTConnectStreams` XML document is `Streams`. `Streams` is a container type XML element used to group the data reported from one or more pieces of equipment into a single XML document. `Streams` **MUST** always appear in the `MTConnectStreams` document.

`DeviceStream` is the next *Structural Element* in the `MTConnectStreams` document. `DeviceStream` is also a XML container type element. A separate `DeviceStream` container is used to organize the information and data reported by each piece of equipment represented in the `MTConnectStreams` document. There **MUST** be at least one `DeviceStream` element in the `Streams` container.

A `DeviceStream` element provides the data reported by a piece of equipment. Each `DeviceStream` element **MUST** contain the attributes `name` and `uuid` to correlate the `DeviceStream` with a specific `Device` defined in the `MTConnectDevices` document. Once the `DeviceStream` element is associated with a specific piece of equipment based on this identity, all data reported by that piece of equipment is directly associated with that unique identity and that association does not need to be repeated for every piece of data reported. A client software application may then directly relate the information provided in the `MTConnectDevices` document with the data provided in the `MTConnectStreams` document based on this identity.

`ComponentStream` is the next level XML element in the `MTConnectStreams` document. `ComponentStream` is also a container type XML element. There **MUST** be a separate `ComponentStream` XML element for each of the *Structural Elements* (`Device` elements, *Top Level* `Component` elements, or *Lower Level* `Component` elements) defined for that piece of equipment in the associated `MTConnectDevices` XML document. A `ComponentStream` representing a *Structural Element* will only appear if there is data reported for that *Structural Element*. (Note: See *Part 2.0 – Devices Information Model* of the MTConnect Standard for a description of the *Structural Elements* for a piece of equipment).

There are three (3) *Structural Elements* – `Samples`, `Events`, and `Condition` at the next level of the `MTConnectStreams` document. Each one of these *Structural Elements* is a container type XML element. These *Structural Elements* group the data reported for each component of a piece of equipment according to the *Data Entity* categories defined in *Part 2.0 – Devices Information Model, Sections 7 and 8*. Therefore,

- `Samples` contains `SAMPLE` category *Data Entities* defined in the `MTConnectDevices` XML document (See *Part 2.0 – Devices Information Model, Section 8.1*)
- `Events` contains `EVENT` category *Data Entities* defined in the `MTConnectDevices` XML document (See *Part 2.0 – Devices Information Model, Section 8.2*)
- `Condition` contains `CONDITION` category *Data Entities* defined in the `MTConnectDevices` XML document (See *Part 2.0 – Devices Information Model, Section 8.3*)

There **MUST** be at least one of `Samples`, `Events`, or `Condition` elements in each `ComponentStream` container.

The following XML tree structure illustrates the various *Structural Elements* used to organize the data reported by a piece of equipment and the relationship between these elements.

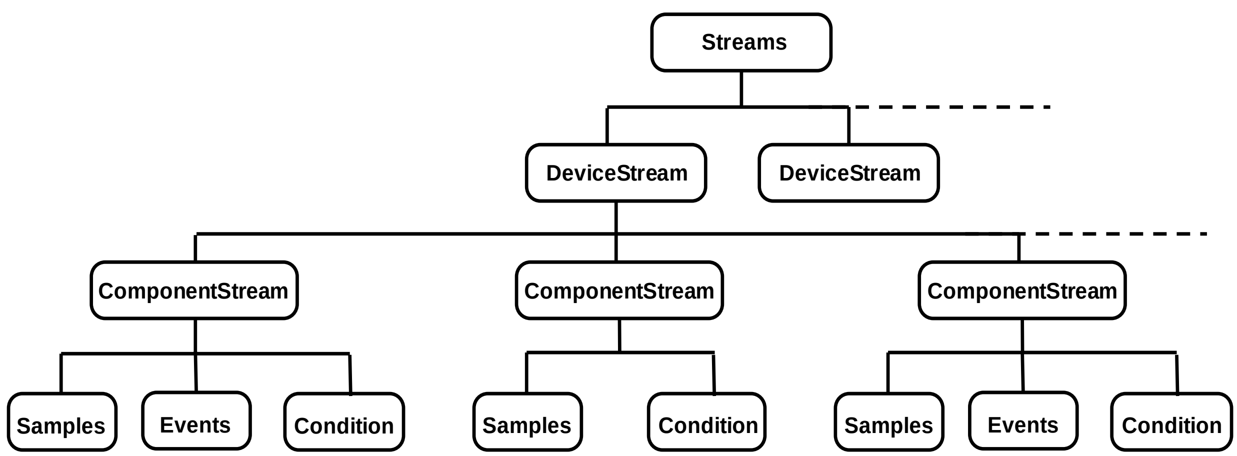


Figure 1: `Streams` Data Structure

Below is a sample from an `MTConnectStreams` XML document that contains the response from an *MTConnect Agent* representing two pieces of equipment, *mill-1* and *mill-2*. The data from each piece of equipment is reported in a separate `DeviceStream` container.

```
<MTConnectStreams …>
  <Header … />
  <Streams>
    <DeviceStream name="mill-1" uuid="1">
      <ComponentStream component="Device" name="mill-1"
        componentId="d1">
        <Events>
          <Availability dataItemId="avail1" name="avail" sequence="5"
            timestamp="2010-04-06T06:19:35.153141">
            AVAILABLE</Availability>
        </Events>
      </ComponentStream>
    </DeviceStream>
    <DeviceStream name="mill-2" uuid="2">
      <ComponentStream component="Device" name="mill-2"
        componentId="d2">
        <Events>
          <Availability dataItemId="avail2" name="avail" sequence="15"
              timestamp="2010-04-06T06:19:35.153141">
              AVAILABLE</Availability>
        </Events>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```

In the example above, it should be noted that the *sequence numbers* are unique across the two pieces of equipment. Client software applications **MUST NOT** assume that the `Events` and `Samples` sequence numbers are strictly in sequence. All sequence numbers **MAY NOT** be included. For instance, such a case would occur when HTTP filtering is applied to the request and the `SAMPLE`, `EVENT`, and `CONDITION` data types for other components are not returned. Another case would occur when an *MTConnect Agent* is supporting more than one piece of equipment and data from only one piece of equipment is requested. Refer to MTConnect Standard *Part 1.0 – Overview and Functionality*, *Section 5: MTConnect Fundamentals* for more information on *sequence numbers.*

4.1 Streams

Streams is a container type XML element that **MUST** contain only DeviceStream elements. Streams **MAY** contain any number of DeviceStream elements. If there is no data to be reported for a request for data, an MTConnectStreams document **MUST** be returned with an empty Streams container. *Data Entities* **MAY NOT** be directly associated with the Streams container.

The following XML schema represents the structure of the Streams XML element.

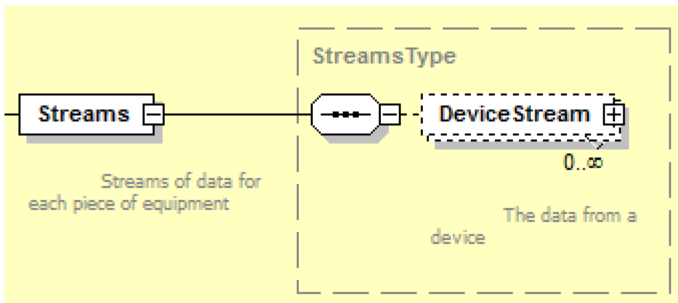


Figure 2: Streams Schema Diagram

Element	Description	Occurrence
Streams	The first, or highest, level XML container element in an *MTConnectStreams Response Document* provided by an *MTConnect Agent* in response to a sample or current *HTTP Request*. There **MAY** be only one Streams element in an *MTConnectStreams Response Document* for each piece of equipment represented in the document. An empty Streams container **MAY** be provided to indicate that no data is available for the given *Request*. The Streams element **MAY** contain any number of DeviceStream elements, one for each piece of equipment represented in the MTConnectStreams document.	1

4.2 DeviceStream

DeviceStream is a XML container that organizes data reported from a single piece of equipment. A DeviceStream element **MUST** be provided for *each* piece of equipment reporting data in an MTConnectStreams document.

A DeviceStream **MAY** contain any number of ComponentStream elements; limited to one for each component element represented in the MTConnectDevices document. If the response to the request for data from an *MTConnect Agent* does not contain any data for a specific piece of equipment, an empty DeviceStream element **MAY** be created to indicate that the piece of equipment exists, but there was no data available. In this case, there will be no ComponentStream elements provided.

Element	Description	Occurrence
DeviceStream	A XML container element provided in the Streams container in the MTConnectStreams document. There **MAY** be one or more DeviceStream elements in a Streams container; one for each piece of equipment represented in the MTConnectStreams document.	0..INF

4.2.1 XML Schema for DeviceStream

The following XML schema represents the structure of the DeviceStream XML element showing the attributes defined for DeviceStream and the elements that **MAY** be associated with DeviceStream.

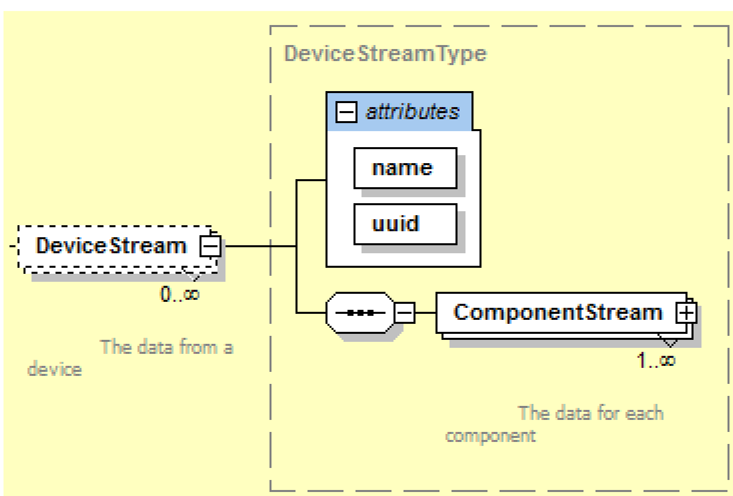


Figure 3: DeviceStream Schema Diagram

4.2.2 Attributes for `DeviceStream`

The following table defines the attributes that **MUST** be provided to uniquely identify each specific piece of equipment associated with the information provided in each `DeviceStream`.

Attribute	Description	Occurrence
`name`	The name associated with the piece of equipment reporting the data contained in this `DeviceStream` container. `name` is a required attribute. The value reported for `name` **MUST** be the same as the value defined for the `name` attribute of the same piece of equipment in the `MTConnectDevices` document. An NMTOKEN XML type. **WARNING**: `name` may become an optional attribute in future versions of the MTConnect Standard.	1
`uuid`	The `uuid` associated with the piece of equipment reporting the data contained in this `DeviceStream` container. `uuid` is a required attribute. The value reported for `uuid` **MUST** be the same as the value defined for the `uuid` attribute of the same piece of equipment in the `MTConnectDevices` document.	1

4.2.3 Elements for `DeviceStream`

The following table lists the XML element(s) that **MAY** be provided in the `DeviceStream` XML element.

Element	Description	Occurrence
`ComponentStream`	A XML container type element that organizes data returned from an *MTConnect Agent* in response to a `current` or `sample` HTTP request. Any number of `ComponentStream` elements **MAY** be provided in a `DeviceStream` container. There **MUST** be a separate `ComponentStream` XML element for each of the *Structural Elements* (`Device` elements, *Top Level* `Component` elements, or *Lower Level* `Component` elements) defined for that piece of equipment in the associated `MTConnectDevices` XML document. A `ComponentStream` representing a *Structural Element* will only appear if there is data reported for that *Structural Element*.	0..INF

4.3 ComponentStream

ComponentStream is a XML container that organizes the data associated with each *Structural Element* (Device element, *Top Level* Component, or *Lower Level* Component element) defined for that piece of equipment in the associated MTConnectDevices XML document. The data reported in each ComponentStream element **MUST** be grouped into individual XML containers based on the value of the category attribute (SAMPLE, EVENT, or CONDITION) defined for each *Data Entity* in the MTConnectDevices XML document. These containers are Samples, Events, and Condition.

4.3.1 XML Schema for ComponentStream

The following XML schema represents the structure of a ComponentStream XML element showing the attributes defined for ComponentStream and the elements that **MAY** be associated with ComponentStream.

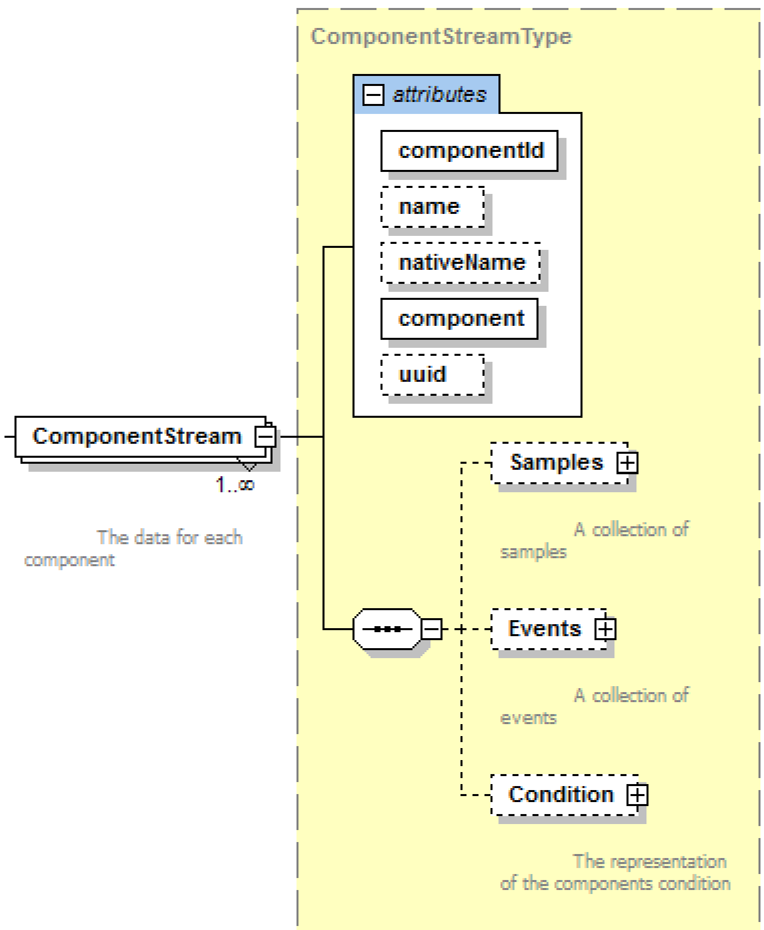


Figure 4: ComponentStream Schema Diagram

`ComponentStream` is similar to `DeviceStream` in that the attributes uniquely identify the *Structural Element* with which the data reported is directly associated. This information does not have to be repeated for each *Data Entity*. In the case of the `DeviceStream,` the attributes uniquely identify the piece of equipment associated with the data. In the case of the `ComponentStream`, the attributes identify the specific *Structural Element* within a piece of equipment associated with each *Data Entity*.

4.3.2 Attributes for `ComponentStream`

The following table defines the attributes used to uniquely identify the specific *Structural Element(s)* of a piece of equipment associated with the data reported in the `MTConnectStreams` document.

Attribute	Description	Occurrence
`componentId`	The identifier of the *Structural Element* (`Device` element, *Top Level* `Component` element, or *Lower Level* `Component` element) as defined by the `id` attribute of the corresponding *Structural Element* in the `MTConnectDevices` XML document. `componentId` is a required attribute. The identifier **MUST** be the same as that defined in the `MTConnectDevices` document to associate the data reported in the `ComponentStream` container with the *Structural Element* identified in the `MTConnectDevices` document.	1
`name`	The name of the `ComponentStream` element. `name` is an optional attribute. If `name` is not defined for a specific *Structural Element* in the `MTConnectDevices` document, it **MUST NOT** be provided for the corresponding `ComponentStream` element in the `MTConnectStreams` document. If `name` is defined for a specific *Structural Element* in the `MTConnectDevices` document, it **MAY** be provided for the corresponding `ComponentStream` element in the `MTConnectStreams` document. If provided, the value reported for `name` **MUST** be the same as the value defined for the `name` attribute of the corresponding *Structural Element* (`Device` element, *Top Level* `Component` element, or *Lower Level* `Component` element) defined in the `MTConnectDevices` XML document. An NMTOKEN XML type.	0..1

Attribute	Description	Occurrence
nativeName	nativeName identifies the common name normally associated with the ComponentStream element. nativeName is an optional attribute. If nativeName is not defined for a specific *Structural Element* in the MTConnectDevices document, it **MUST NOT** be provided for the corresponding ComponentStream element in the MTConnectStreams document. If nativeName is defined for a specific *Structural Element* in the MTConnectDevices document, it **MAY** be provided for the corresponding ComponentStream element in the MTConnectStreams document. If provided, the value reported for nativeName **MUST** be the same as the value defined for the naiveName attribute of the corresponding *Structural Element* (Device element, *Top Level* Component element, or *Lower Level* Component element) defined in the MTConnectDevices XML document.	0..1
component	component identifies the *Structural Element* (Device, *Top Level* Component, or *Lower Level* Component) associated with the ComponentStream element. component is a required attribute. The value reported for component **MUST** be the same as the value defined for the *Element Name* of the XML container representing the corresponding *Structural Element* (Device element, *Top Level* Component element, or *Lower Level* Component element) defined in the MTConnectDevices XML document. Examples of component are Device, Axes, Controller, Linear, Electrical, User, and Loader.	1

Attribute	Description	Occurrence
`uuid`	`uuid` of the `ComponentStream` element. `uuid` is an optional attribute. If `uuid` is not defined for a specific *Structural Element* in the `MTConnectDevices` document, it **MUST NOT** be provided for the corresponding `ComponentStream` element in the `MTConnectStreams` document. If `uuid` is defined for a specific *Structural Element* in the `MTConnectDevices` document, it **MAY** be provided for the corresponding `ComponentStream` element in the `MTConnectStreams` document, but it is not required. If provided, the value reported for `uuid` **MUST** be the same as the value defined for the `uuid` attribute of the corresponding *Structural Element* (`Device` element, *Top Level* `Component` element, or *Lower Level* `Component` element) defined in the `MTConnectDevices` XML document.	0..1

4.3.3 Elements for `ComponentStream`

In the `ComponentStream` container, an *MTConnect Agent* **MUST** organize the data reported in each `ComponentStream` into individual `Samples`, `Events`, or `Condition` XML containers based on the value of the category attribute (i.e., `SAMPLE`, `EVENT`, or `CONDITION`) defined for each *Data Entity* defined in the `MTConnectDevices` XML document.

Each `ComponentStream` element **MUST** include at least one `Events`, `Samples`, or `Condition` XML container element. *Data Entities* returned in each of the `ComponentStream` container elements are defined in the table below.

Element	Description	Occurrence
`Samples`	A XML container type element. `Samples` organizes the `SAMPLE` type *Data Entities* defined in the `MTConnectDevices` document that are reported in each `ComponentStream` XML element.	0..1 *
`Events`	A XML container type element. `Events` organizes the `EVENT` type *Data Entities* defined in the `MTConnectDevices` document that are reported in each `ComponentStream` XML element.	0..1 *

Element	Description	Occurrence
`Condition`	A XML container type element. `Condition` organizes the `CONDITION` type *Data Entities* defined in the `MTConnectDevices` document that are reported in each `ComponentStream` XML element.	0..1 *

Note: * The `ComponentStream` element **MUST** contain at least one of these element types.

5 *Data Entities*

When a piece of equipment reports values associated with `DataItem` elements defined in the `MTConnectDevices` document, that information is organized as *Data Entities* in the `MTConnectStreams` document. These *Data Entities* are organized in containers within each `ComponentStream` element based on the `category` attribute defined for the corresponding `DataItem` in the `MTConnectDevices` document:

- `DataItem` elements defined with a `category` attribute of `SAMPLE` in the `MTConnectDevices` document are mapped to the `Samples` XML container in the associated `ComponentStream` element.

- `DataItem` elements defined with a `category` attribute of `EVENT` in the `MTConnectDevices` document are mapped to the `Events` XML container in the associated `ComponentStream` element.

- `DataItem` elements defined with a `category` attribute of `CONDITION` in the `MTConnectDevices` document are mapped to the `Condition` XML container in the associated `ComponentStream` element.

The XML tree below demonstrates how *Data Entities* are organized in these containers.

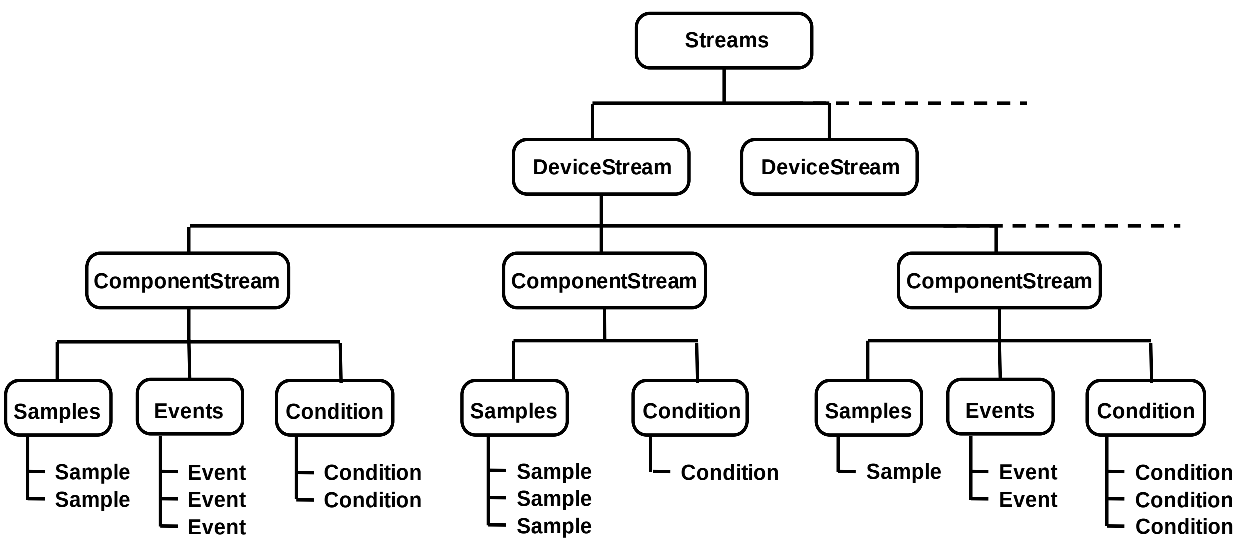


Figure 5: `ComponentStream` XML Tree Diagram

The following is an illustration of the structure of an XML document demonstrating how *Data Entities* are reported in a `MTConnectStreams` document:

```
<MTConnectStreams>
 <Header/>
 <Streams>
   <DeviceStream>
      <ComponentStream>
        <Samples>
          <Sample>
          <Sample>
          <Sample>
        </Samples>
        <Events>
          <Event>
          <Event>
        </Events>
        </Condition>
          <Condition>
          <Condition>
        </Condition>
      </ComponentStream>
      <ComponentStream>
        <Samples>
          <Sample>
          <Sample>
        </Samples>
        <Events>
          <Event>
          <Event>
          <Event>
        </Events>
        <Condition>
          <Condition>
        </Condition>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```

Note: There are no specific requirements defining the sequence in which the `ComponentStream` XML elements are organized in the `MTConnectStreams` document. They **MAY** be organized in any sequence based on the implementation of an *MTConnect Agent*. The sequence in which the `ComponentStream` XML elements appear does not impact the ability for a client software application to interpret the information that it receives in the document.

When an *MTConnect Agent* responds to a `current` HTTP request, the information returned in the `MTConnectStreams` document **MUST** include the most current value for every *Data Entity* defined in the `MTConnectDevices` document subject to any filtering included within the request.

When an *MTConnect Agent* responds to a `sample` HTTP request, the information returned in the `MTConnectStreams` document **MUST** include the occurrences for each *Data Entity* that are available to an *MTConnect Agent* subject to filtering and the count parameter included within the request (see *Part 1 - Overview and Functionality* for a full definition of the protocol).

5.1 *Element Names* for *Data Entities*

In the `MTConnectDevices` document, *Data Entities* are grouped as `DataItem` XML elements within each `Device`, *Top Level* `Component`, and *Lower Level* `Component` *Structural Element*. The *Data Entities* reported in the `MTConnectStreams` document associated with each of these *Structural Elements* are represented with an *Element Name* based on the `category` and `type` defined for each of the `DataItem` elements in the `MTConnectDevices` document.

5.1.1 *Element Names* when `MTConnectDevices` category is SAMPLE or EVENT

The *Data Entities* reported in the `MTConnectStreams` document associated with each `DataItem` element defined in the `MTConnectDevices` document with a `category` attribute of `SAMPLE` or `EVENT` **MUST** be identified in the `MTConnectStreams` document with an *Element Name* derived from the `type` attribute defined for that `DataItem` element in the `MTConnectDevices` document.

The example below describes the most common method used to derive the *Element Name* for a *Data Entity* reported in the `MTConnectStreams` document from the information describing that `DataItem` element in the `MTConnectDevices` document:

`DataItem` Represented in the `MTConnectDevices` Document

```
<DataItem type="AXIS_FEEDRATE" id="xf" name="Xfrt"
  category="SAMPLE" units="MILLIMETER/SECOND"
  nativeUnits="MILLIMETER/SECOND"/>
```

- `DataItem`: The XML *Element Name* for this *Data Entity*.

  Note: *Element Name* must not be confused with the `name` attribute for the data item element.

- `type`, `category`, `units`, and `nativeUnits`: Attributes that provide additional information regarding each data item in the `MTConnectDevices` document.

Response Format reported in the `MTConnectStreams` Document

```
<AxisFeedrate name="Xfrt" sequence="61315517" timestamp="2016-07-
  28T02:06:01.364428Z" dataItemId="xf">10.83333</AxisFeedrate>
```

- `AxisFeedrate`: The *Element Name* provided in the `MTConnectStreams` response format for the data item. The *Element Name* for a data item is defined by the `type` attribute of `AXIS_FEEDRATE` in the `MTConnectDevices` document. The *Element Name* **MUST** be provided in Pascal case format (first letter of each word is capitalized).

5.1.2 Changes to *Element Names* when `representation` attribute is used

The *Element Name* for a *Data Entity* reported in the `MTConnectStreams` document is extended when the `representation` attribute is used to further describe that `DataItem` element in the `MTConnectDevices` document.

When a `DataItem` element is defined in the `MTConnectDevices` document with a `representation` attribute of `TIME_SERIES` or `DISCRETE`, the XML *Element Name* for the associated *Data Entity* reported in the `MTConnectStreams` document **MUST** be extended by adding the value of the `representation` attribute to the *Element Name*.

For example, the `DataItem` element `ANGULAR_VELOCITY` with a `representation` attribute defined as `TIME_SERIES` **MUST** be transformed to the *Element Name* `AngularVelocityTimeSeries`.

Similarly, the `DataItem` element `PART_COUNT` with a `representation` attribute defined as `DISCRETE` **MUST** be transformed to the *Element Name* `PartCountDiscrete.`

5.1.3 *Element Names* when `MTConnectDevices` category is CONDITION

Data Entities defined in the `MTConnectDevices` document with a `category` attribute of `CONDITION` are reported with an *Element Name* that is defined differently from other *Data Entity* types. The *Element Name* for these *Data Entities* are defined based on the *Fault State* (`Normal`, `Warning`, or `Fault`) associated with each *Data Entity* at the time that a value for that *Data Entity* is reported. See *Sections 5.7.1 and 5.8* for details on how these *Data Entities* are reported in the `MTConnectStreams` document.

5.2 `Samples` Container

`Samples` is a XML container type element. `Samples` organizes the *Data Entities* returned in the `MTConnectStreams` XML document for those `DataItem` elements defined with a `category` attribute of `SAMPLE` in the `MTConnectDevices` document.

A separate `Samples` container will be provided for the data returned for the `DataItem` elements associated with each *Structural Element* of a piece of equipment defined in the `MTConnectDevices` document.

Element	Description	Occurrence
`Samples`	A XML container type element that organizes the data reported in the `MTConnectStreams` document for `DataItem` elements defined in the `MTConnectDevices` document with a `category` attribute of `SAMPLE`. A separate `Samples` container **MUST** be provided for each `ComponentStream` element for which data is returned for a `DataItem` element defined in the `MTConnectDevices` document with a `category` attribute of `SAMPLE`. If provided in the document, a `Samples` XML container **MUST** contain at least one `Sample` element.	0..1

5.3 `Sample` *Data Entities*

A `Sample` XML element provides the information and data reported from a piece of equipment for those `DataItem` elements defined with a `category` attribute of `SAMPLE` in the `MTConnectDevices` document.

`Sample` is an abstract type XML element and will never appear directly in the `MTConnectStreams` XML document. As an abstract type XML element, `Sample` will be replaced in the XML document by a specific type of `Sample` specified by the *Element Name* for that *Data Entity*. The different types of `Sample` elements are defined in *Section 6.1*. Examples of XML elements representing `Sample` include `PathPosition`, `Temperature`, and `AxisVelocity`.

Element	Description	Occurrence
`Sample`	A XML element that provides the information and data reported from a piece of equipment for those `DataItem` elements defined with a `category` attribute of `SAMPLE` in the `MTConnectDevices` document. `Sample` is an abstract type XML element. It is replaced in the `MTConnectStreams` document by a specific type of `Sample` element. There **MAY** be multiple types of `Sample` elements in a `Samples` container.	1..INF

5.3.1 XML Schema Structure for `Sample`

The following XML schema represents the structure of a `Sample` XML element showing the attributes defined for `Sample` elements.

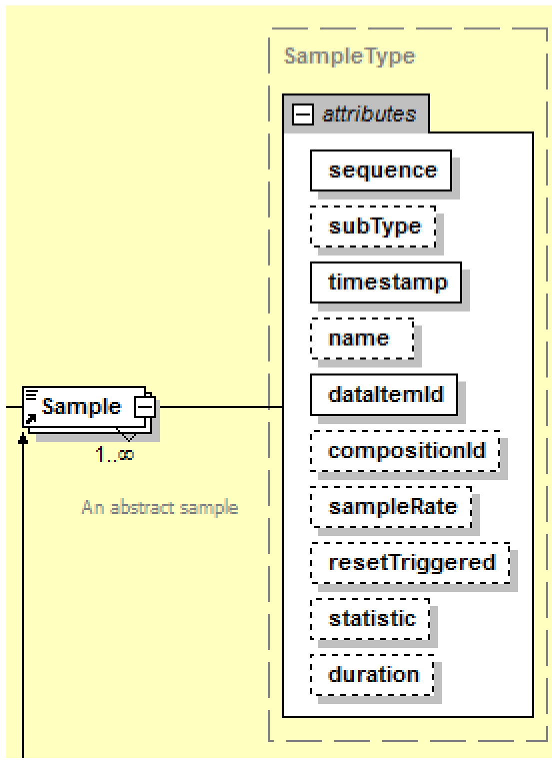


Figure 6: `Sample` Schema Diagram

5.3.2 Attributes for `Sample`

The following table defines the attributes used to provide additional information for a `Sample` XML element.

Attribute	Description	Occurrence
`sequence`	A number representing the sequential position of an occurrence of the `Sample` in the data buffer of an *MTConnect Agent*. `sequence` is a required attribute. `sequence` **MUST** have a value represented as an unsigned 64-bit value from 1 to 2^64-1.	1
`subType`	The `subtype` of the *Data Entity*. `subType` is an optional attribute. `subType` **MUST** match the `subType` attribute of the `DataItem` element as defined in the `MTConnectDevices` document that the `Sample` element represents.	0..1
`timestamp`	The most accurate time available to a piece of equipment that represents the point in time that the data reported for the `Sample` was measured. When the `Sample` element represents a `DataItem` element defined in the `MTConnectDevices` document with a `representation` or `statistic` attribute, `timestamp` **MUST** represent the time that the data collection was completed. `timestamp` is a required attribute.	1
`name`	The name of the `Sample` element. `name` is an optional attribute. `name` **MUST** match the `name` attribute of the `DataItem` element defined in the `MTConnectDevices` document that the `Sample` element represents. An NMTOKEN XML type.	0..1
`dataItemId`	The unique identifier for the `Sample` element. `dataItemId` is a required attribute. `dataItemId` **MUST** match the `id` attribute of the `DataItem` element defined in the `MTConnectDevices` document that the `Sample` element represents.	1

Attribute	Description	Occurrence
sampleRate	The rate at which successive samples of the value of a data item are recorded. sampleRate is expressed in terms of samples per second. sampleRate is an optional attribute. If the sampleRate is smaller than one, the number can be represented as a decimal type floating-point number. For example, a rate of 1 per 10 seconds would be 0.1 sampleRate **MUST** be provided when the representation attribute of the DataItem element defined in the MTConnectDevices document that this Sample element represents is TIME_SERIES. For DataItem elements where the representation attribute defined in the MTConnectDevices document that this Sample element represents is not TIME_SERIES, it **MUST** be assumed that the data reported is represented by a single value and sampleRate **MUST NOT** be reported in the MTConnectStreams document.	0..1
statistic	The type of statistical calculation defined by the statistic attribute of the DataItem element defined in the MTConnectDevices document that this Sample element represents. statistic is an optional attribute.	0..1
duration	The time-period over which the data was collected. duration is an optional attribute. duration **MUST** be provided when the statistic attribute of the DataItem element is defined in the MTConnectDevices document that this Sample element represents.	0..1
resetTriggered	For those DataItem elements that report data that may be periodically reset to an initial value, resetTriggered identifies when a reported value has been reset and what has caused that reset to occur. resetTriggered is an optional attribute. resetTriggered **MUST** only be provided for the specific occurrence of a *Data Entity* reported in the MTConnectStreams document when the reset occurred and **MUST NOT** be provided for any other occurrence of the *Data Entity* reported in a MTConnectStreams document.	0..1
compositionId	The identifier of the Composition element defined in the MTConnectDevices document associated with the data reported for the Sample element. compositionId is an optional attribute.	0..1

5.3.2.1 `duration` Attribute for `Sample`

`Sample` elements that represent the result of a computed value of a statistic **MUST** contain a `duration` attribute. For these *Data Entities*, the `timestamp` associated with the `Sample` **MUST** reference the time the data collection was completed. `timestamp` **MUST NOT** represent any other time associated with the data collection or the calculation of the statistic. The actual time the interval began can be computed by subtracting the `duration` from the `timestamp`.

Two `Sample` elements **MAY** have overlapping time periods when statistics are computed at different frequencies. For example, there may be two *Data Entities* reporting a statistic representing the average value for the readings of the same measured signal calculated over one and five minute intervals. These *Data Entities* can both have the same start time for their calculations (e.g., 05:10:00), but the `timestamp` and `duration` will be 05:11:00 and 60 seconds, respectively, for the *Data Entity* reporting the one-minute average and 05:15:00 and 300 seconds, respectively, for the *Data Entity* reporting the five-minute average. This allows for varying statistical methods to be applied with different interval lengths each having different values for the `timestamp` and `duration` attributes.

5.3.2.2 `resetTriggered` Attribute for `Sample`

Some *Data Entities* **MAY** have their reported value reset to an initial value. These reset actions may be based upon a specific elapsed time or may be triggered by a physical or logical reset action that causes the reset to occur. Examples of *Data Entities* that **MAY** have their reported value reset to an initial value are *Data Entities* representing a counter, a timer, or a statistic.

`resetTriggered` defines the type of reset action that caused the value of the reported data to be reset. The value reported for `resetTriggered` **MAY** be defined by the `ResetTrigger` element for the *Data Entity* in the `MTConnectDevices` document that this `Sample` element represents. If the `ResetTrigger` element is not defined in the `MTConnectDevices` document, a `resetTriggered` attribute **SHOULD** be reported in the `MTConnectStreams` document if the type of reset action can be determined and reported by the piece of equipment.

`resetTriggered` **MUST** only be reported for the first occurrence of a *Data Entity* after a reset action has occurred and **MUST NOT** be provided for any other occurrence of the *Data Entity* reported in a `MTConnectStreams` document. When a reset occurs, the piece of equipment **MUST** report an occurrence of the *Data Entity* that was reset even if that occurrence of the *Data Entity* would normally be suppressed based on the filtering criteria established in the `MTConnectDevices` document that this `Sample` element represents.

The following table provides the values that **MAY** be reported for `resetTriggered`:

Value for resetTriggered	Description
`ACTION_COMPLETE`	The value of the *Data Entity* that is measuring an action or operation was reset upon completion of that action or operation.
`ANNUAL`	The value of the *Data Entity* was reset at the end of a 12-month period.
`DAY`	The value of the *Data Entity* was reset at the end of a 24-hour period.
`MAINTENANCE`	The value of the *Data Entity* was reset upon completion of a maintenance event.
`MANUAL`	The value of the *Data Entity* was reset based on a physical reset action.
`MONTH`	The value of the *Data Entity* was reset at the end of a monthly period.
`POWER_ON`	The value of the *Data Entity* was reset when power was applied to the piece of equipment after a planned or unplanned interruption of power has occurred.
`SHIFT`	The value of the *Data Entity* was reset at the end of a work shift.
`WEEK`	The value of the *Data Entity* was reset at the end of a 7-day period.

5.3.3 Response for SAMPLE category DataItem Elements with a representation attribute of TIME_SERIES

SAMPLE category DataItem elements defined in the MTConnectDevices document with a representation attribute of TIME_SERIES **MUST** be represented in the MTConnectSteams document as Sample elements that report data that includes multiple values representing a series of readings of a measured value taken at a specific sample rate. Such a DataItem element can be defined for collecting high frequency readings of a measured value and then providing the entire series of values to a client software application as the data reported for a single *Data Entity*. In this case, the sampleCount and sampleRate attributes **MUST** be provided.

Note: sampleCount is an attribute **MUST** only be provided for Sample elements that represent SAMPLE category DataItem elements defined in the MTConnectDevices document with a representation attribute of TIME_SERIES.

The CDATA provided for the *Data Entity* **MUST** be a series of space delimited floating-point numbers. The number of values **MUST** match the sampleCount.

5.3.3.1 XML Schema Structure for `Sample` when reporting Time Series data

The following XML schema represents the extended structure of a `Sample` XML element that represents a `SAMPLE` category `DataItem` element defined in the `MTConnectDevices` document with a `representation` attribute of `TIME_SERIES`.

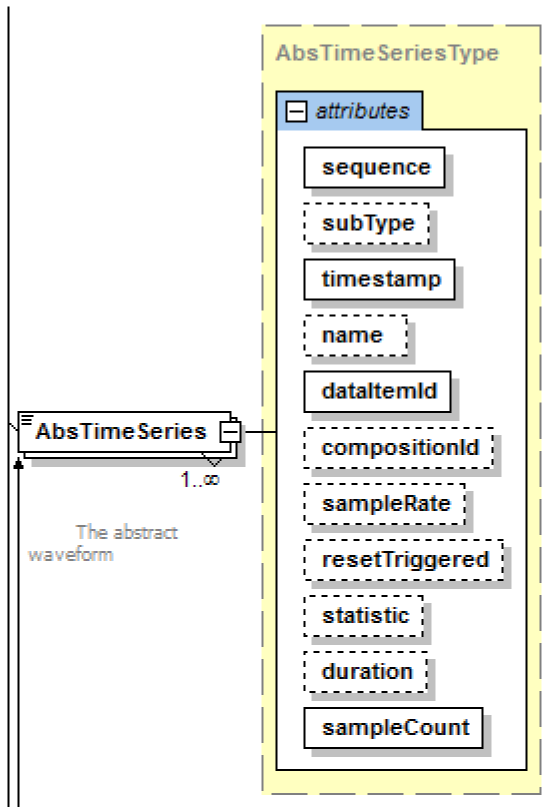


Figure 7: `AbsTimeSeries` Schema Diagram

Note: The `AbsTimeSeries` element shown in the XML schema is an abstract type element and will be replaced in the `MTConnectStreams` document by the element name derived from the `type` attribute defined for the associated `DataItem` element defined in the `MTConnectDevices` document.

5.3.3.2 Attributes for a **`Sample`** when reporting Time Series data

The following table defines the additional attribute provided for a `Sample` XML element that represents a `SAMPLE` category `DataItem` element defined in the `MTConnectDevices` document with a `representation` attribute of `TIME_SERIES`.

Attribute	Description	Occurrence
`sampleCount`	The number of readings reported in the data returned for the `DataItem` element defined in the `MTConnectDevices` document that this `Sample` element represents. `sampleCount` is an optional attribute. `sampleCount` **MUST** be provided when the `representation` attribute of the `DataItem` element is `TIME_SERIES`. `sampleCount` **MUST NOT** be provided when the `representation` attribute is defined as `DISCRETE` or `VALUE`, or when it is not defined.	0..1

5.3.4 Valid Data Values for **`Sample`**

All `Sample` elements reported in an `MTConnectStreams` XML document **MUST** provide a value in the CDATA of the *Data Entity*.

The value returned in the CDATA **MUST** be reported as either a *Valid Data Value* representing the information reported from a piece of equipment or `UNAVAILABLE` when a *Valid Data Value* cannot be determined.

The *Valid Data Value* reported for a `Sample` represents the reading of the value of a continuously variable or analog data source.

The `representation` attribute for a `SAMPLE` category `DataItem` element defined in the `MTConnectDevices` document specifies how an *MTConnect Agent* **MUST** record instances of the data associated with that data item and how often that data **MUST** be reported as a `Sample` element in the `MTConnectStreams` document.

The data reported for a `Sample` element associated with a `SAMPLE` category `DataItem` element with a `representation` of `VALUE` can be measured at any point-in-time and **MUST** always produce a result with a single data value.

> Note: If a `representation` attribute is not specified in the `MTConnectDevices` document for a `DataItem` element, it **MUST** be assumed that the data reported in the `MTConnectStreams` document for the *Data Entity* has a `representation` type of `VALUE`.

In the case of a `Sample` element associated with a `SAMPLE` category `DataItem` element with a `representation` attribute of `TIME_SERIES`, the data provided **MUST** be a series of data values representing multiple sequential samples of the measured value that will be provided only at the end of the completion of a sampling period. (See *Section 5.3.3* of this document for more information on `TIME_SERIES` type data).

Data values provided for a `Sample` **MUST** always be a floating-point number. In the MTConnect Standard, floating-point numbers are defined as XML xs:float type numbers as defined by W3C. Any of the following number formats are valid XML floating type numbers: 1267.43233E12, -1E4, 12.78e-2, 12, 137.2847, 0, and INF.

> Note: For some `Sample` elements, the *Valid Data Value* **MAY** be restricted to specific formats. See *Section 6.1* of this document for a description of any restrictions of the acceptable format for *Valid Data Values*.

For `Sample` elements, a client software application can determine the appropriate accuracy of the value reported for the *Data Entity* by applying the `significantDigits` attribute defined for the corresponding `DataItem` element defined in the `MTConnectDevices` document.

The *Valid Data Value* reported as CDATA for a `Sample` element **MUST** be formatted as part of the content between the element tags in the XML element representing that *Data Entity*. As an example, a `Position` is formatted as follows in the XML document:

```
<Position sequence="112" timestamp="2007-08-09T12:32:45.1232"
   name="Xabs" dataItemId="10">123.3333</Position>
```

> Note: The **BOLDED** item is identified for emphasis only.

In this example, the 123.3333 is the CDATA for `Position`. All CDATA in a `Sample` element is *typed*, which means that the value reported for the *Data Entity* **MUST** be formatted as defined in *Section 6.1* for each *Data Entity* so that it can be validated.

5.3.5 Unavailability of *Valid Data Values* for `Sample`

If an *MTConnect Agent* cannot determine a *Valid Data Value* for a `Sample` element, the value returned for the CDATA for the *Data Entity* **MUST** be reported as `UNAVAILABLE`.

The example below demonstrates how an *MTConnect Agent* reports the value for a `Sample` in the CDATA when it is unable to determine a *Valid Data Value*:

```
<Samples>
  <PathPosition dataItemId="p2" timestamp="2009-03-04T19:45:50.458305"
    subType="ACTUAL" name="Zact"
    sequence="15065113">UNAVAILABLE</PathPosition>
  <Temperature dataItemId="t6"
      timestamp="2009-03-04T19:45:50.458305"
      name="temp" sequence="150651134">UNAVAILABLE</Temperature>
</Samples>
```

Note: The **BOLDED** items are identified for emphasis only.

5.4 `Events` Container

`Events` is a XML container type element. `Events` organizes the *Data Entities* returned in the `MTConnectStreams` XML document for those `DataItem` elements defined with a `category` attribute of `EVENT` in the `MTConnectDevices` document.

A separate `Events` container will be provided for the data returned for the `DataItem` elements associated with each *Structural Element* of a piece of equipment defined in the `MTConnectDevices` document.

Element	Description	Occurrence
`Events`	A XML container type element that organizes the data reported in the `MTConnectStreams` document for `DataItem` elements defined in the `MTConnectDevices` document with a `category` attribute of `EVENT`. A separate `Events` container **MUST** be provided for each `ComponentStream` element for which data is returned for a `DataItem` element defined in the `MTConnectDevices` document with a `category` attribute of `EVENT`. If provided in the document, an `Events` XML container **MUST** contain at least one `Event` element.	0..1

5.5 `Event` *Data Entities*

An `Event` XML element provides the information and data provided from a piece of equipment for those `DataItem` elements defined with a `category` attribute of `EVENT` in the `MTConnectDevices` document.

`Event` is an abstract type XML element and will never appear directly in the `MTConnectStreams` XML document. As an abstract type XML element, `Event` will be replaced in the XML document by a specific type of `Event` specified by the *Element Name* for that *Data Entity*. The different types of `Event` elements are defined in *Section 6.2*. Examples of XML elements representing `Event` include `Block`, `Execution`, and `Line`.

`Event` is similar to `Sample`, but its value can change with unpredictable frequency. `Events` do not report intermediate values. As an example, when `Availability` transitions from `UNAVAILABLE` to `AVAILABLE`, there is no intermediate state that can be inferred.

`Event` elements **MAY** report data values defined by a controlled vocabulary as specified in *Section 6.2* of this document, by numeric values, or by a character string representing text or a message provided by the piece of equipment.

Element	Description	Occurrence
`Event`	A XML element which provides the information and data reported from a piece of equipment for those `DataItem` elements defined with a `category` attribute of `EVENT` in the `MTConnectDevices` document. `Event` is an abstract type XML element. It is replaced in the `MTConnectStreams` document by a specific type of `Event` element. There **MAY** be multiple types of `Event` elements in an `Events` container.	1..INF

5.5.1 XML Schema Structure for `Event`

The following XML schema represents the structure of an `Event` XML element showing the attributes defined for `Event` elements.

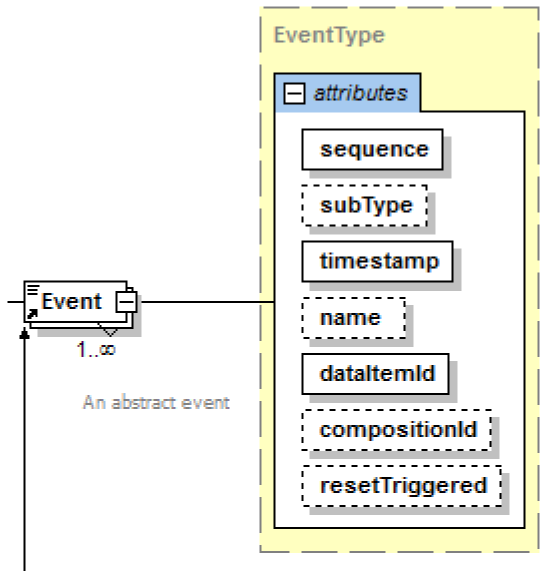


Figure 8: `Event` Schema Diagram

5.5.2 Attributes for `Event`

The following table defines the attributes that **MAY** be used to provide additional information for an `Event` XML element.

Attribute	Description	Occurrence
`sequence`	A number representing the sequential position of an occurrence of the `Event` in the data buffer of an *MTConnect Agent*. `sequence` is a required attribute. `sequence` **MUST** have a value represented as an unsigned 64-bit value from 1 to 2^64-1.	1
`subType`	The `subtype` of the *Data Entity*. `subType` is an optional attribute. `subType` **MUST** match the `subType` attribute of the `DataItem` element as defined in the `MTConnectDevices` document that the `Event` element represents.	0..1
`timestamp`	The most accurate time available to a piece of equipment that represents the point in time that the data reported for the `Event` was measured. `timestamp` is a required attribute.	1

Attribute	Description	Occurrence
`name`	The name of the `Event` element. `name` is an optional attribute. `name` **MUST** match the `name` attribute of the `DataItem` element as defined in the `MTConnectDevices` document that the `Event` element represents. An NMTOKEN XML type.	0..1
`dataItemId`	The unique identifier for the `Event` element. `dataItemId` is a required attribute. `dataItemId` **MUST** match the `id` attribute of the `DataItem` element as defined in the `MTConnectDevices` document that the `Event` element represents.	1
`resetTriggered`	For those `DataItem` elements that report data that **MAY** be periodically reset to an initial value, `resetTriggered` identifies when a reported value has been reset and what that has caused that reset to occur. `resetTriggered` is an optional attribute. `resetTriggered` **MUST** only be provided for the specific occurrence of a *Data Entity* reported in the `MTConnectStreams` document when the reset occurred and **MUST NOT** be provided for any other occurrence of the *Data Entity* reported in a `MTConnectStreams` document.	0..1
`compositionId`	The identifier of the `Composition` element defined in the `MTConnectDevices` document that the data reported for the `Event` element is associated. `compositionId` is an optional attribute.	0..1

5.5.3 Response for `EVENT` category Data Items with a `representation` attribute of `DISCRETE`

`EVENT` category `DataItem` elements defined in an `MTConnectDevices` document with a `representation` attribute of `DISCRETE` indicate that the value of successive occurrences of the data reported in the associated `Event` type *Data Entity* in an `MTConnectStreams` document **MAY** be identical. Duplicate values **MUST NOT** be suppressed by an *MTConnect Agent* since each occurrence of the data item represents a different and unique `Event`.

An example of an `EVENT` category `DataItem` element with a `representation` attribute of `DISCRETE` would be a parts counter that reports the completion of each part produced, versus reporting the accumulation of parts produced over time. In this case, the associated `Event` element would be represented by a *Data Entity* with an *Element Name* of `PartCountDiscrete`. Each occurrence of this *Data Entity* in an `MTConnectStreams` document would indicate the completion of a fixed number of parts (typically 1).

5.5.4 Response for `EVENT` category Data Items with a `type` attribute of `MESSAGE`

`EVENT` category `DataItem` elements defined in the `MTConnectDevices` document with a `type` attribute of `MESSAGE` **MAY NOT** report a state change between successive occurrences of the associated *Data Entity* being reported by a piece of equipment in the `MTConnectStreams` document. If the *Data Entity* representing a message does not have a reset state, it **SHOULD** be defined with a `representation` attribute of `DISCRETE` in the `MTConnectDevices` document. In this case, each occurrence of this *Data Entity* in an `MTConnectStreams` document represents a different and unique `Event`. The *Element Name* for this `Event` element **MUST** be `MessageDiscrete` and each occurrence of this *Data Entity* in an `MTConnectStreams` document would indicate a unique occurrence of the message.

5.5.5 Valid Data Values for `Event`

`Event` elements reported in an `MTConnectStreams` XML document **MUST** provide a value in the CDATA of the *Data Entity*.

The value reported in the CDATA **MUST** be reported as either a *Valid Data Value* representing the information reported from a piece of equipment or `UNAVAILABLE` when a *Valid Data Value* cannot be determined.

The *Valid Data Value* reported for an `Event` represents a distinct piece of information provided from a piece of equipment. Unlike `Sample`, `Event` does not report intermediate values that vary over time. `Event` reports information that, when provided at any specific point in time, represents the current state of the piece of equipment.

The `representation` attribute for an `EVENT` category data item defined in the `MTConnectDevices` document specifies how an *MTConnect Agent* **MUST** record instances of data associated with that data item and how that data **MUST** be reported as an `Event` element in the `MTConnectStreams` document.

The data reported for an `Event` element associated with an `EVENT` category data item with a `representation` attribute of `VALUE` **MUST** be either an integer, a floating-point number, a descriptive value (text string) representing one of two or more state values defined for that data item, or a text string representing a message.

If a `representation` attribute is not specified for a data item in an `MTConnectDevices` document, the designation for the `representation` attribute **MUST** be interpreted as `VALUE`.

The data reported for an `Event` element associated with an `EVENT` category data item with a `representation` attribute of `DISCRETE` **MUST** be a numeric value representing a repetitive occurrence of a single data value or a message. An `EVENT` with a `representation` attribute of `DISCRETE` is the only case where an *MTConnect Agent* **MAY** provide successive occurrences of a data item with identical data values since each occurrence of the `Event` element represents a different and unique occurrence of the *Data Entity*.

The *Valid Data Value* reported as CDATA for an `Event` element **MUST** be formatted as part of the content between the element tags in the XML element representing that *Data Entity*. As an example, `Event` elements are formatted as follows in the XML document:

```
<PartCount dataItemId="pc4" timestamp="2009-02-26T02:02:36.48303"
  name="pcount" sequence="185">238</PartCount>
<ControllerMode dataItemId="p3" timestamp="2009-02-26T02:02:35.716224"
  name="mode" sequence="192">AUTOMATIC</ControllerMode>
    <Block dataItemId="cn2" name="block" sequence="206"
      timestamp="2009-02-26T02:02:37.394055">G0Z1</Block>
```

Note: The **BOLDED** items are identified for emphasis only.

In these examples, `238` is the CDATA for `PartCount` and is a numeric value; `AUTOMATIC` is the CDATA for the `ControllerMode` and is a descriptive value representing a state for the *Data Entity*; and `G0Z1` is a text string representing a message describing the program code associated with the `Block` *Data Entity*.

5.5.6 Unavailability of *Valid Data Values* for `Event`

If an *MTConnect Agent* cannot determine a *Valid Data Value* for an `Event` element, the value returned for the CDATA for the *Data Entity* **MUST** be reported as `UNAVAILABLE`.

The example below demonstrates how an *MTConnect Agent* reports the value for an `Event` in the CDATA when it is unable to determine a *Valid Data Value*:

```
<Events>
  <ControllerMode dataItemId="p3" timestamp="2009-02-26T02:02:35.716224"
    name="mode" sequence="182">UNAVAILABLE</ControllerMode>
</Events>
```

Note: The **BOLDED** items are identified for emphasis only.

5.6 `Condition` Container

`Condition` is a XML container type element. `Condition` organizes the *Data Entities* returned in the `MTConnectStreams` XML document for those `DataItem` elements defined with a `category` attribute of `CONDITION` in the `MTConnectDevices` document.

A separate Condition container will be provided for the data returned for the DataItem elements associated with each *Structural Element* of a piece of equipment defined in the MTConnectDevices document.

Element	Description	Occurrence
Condition	A XML container type element that organizes the data reported in the MTConnectStreams document for DataItem elements defined in the MTConnectDevices document with a category attribute of CONDITION. A separate Condition container **MUST** be provided for each ComponentStream element for which data is returned for a DataItem element defined in the MTConnectDevices document with a category attribute of CONDITION. If provided in the document, a Condition XML container **MUST** contain at least one Condition data element.	0..1

5.7 Condition *Data Entities*

A Condition XML element provides the information and data provided from a piece of equipment for those DataItem elements defined with a category attribute of CONDITION in the MTConnectDevices document.

Condition provides information reported by a piece of equipment describing its health and ability to function.

Condition is an abstract type XML element and will never appear directly in the MTConnectStreams XML document. As an abstract type XML element, Condition will be replaced in the XML document by a *Data Entity* representing the CONDITION category DataItem element defined in the MTConnectDevices document that this Condition element represents.

The *Data Entities* represented by `Condition` are structured differently than the *Data Entities* representing `Sample` and `Event`. The *Element Name* for each `Condition` element reported in the `MTConnectStreams` document defines the *Fault State* of the *Data Entity*. A `Condition` element is identified by the *Structural Element* to which it is associated, along with the `type` and `dataItemId` defined for the element. *Section 6.3* provides details on the different types of `Condition` elements.

Element	Description	Occurrence
`Condition`	A XML element that provides the information and data reported from a piece of equipment for those `DataItem` elements defined with a `category` attribute of `CONDITION` in the `MTConnectDevices` document. `Condition` is an abstract type XML element. It is replaced in the `MTConnectStreams` document by a specific type of `Condition` element. There **MAY** be multiple types of `Condition` elements in a `Condition` container.	1..INF

`CONDITION` type `DataItem` elements defined in the `MTConnectDevices` document **MAY** report multiple simultaneous *Fault States* in the `MTConnectStreams` document. This is unlike a `SAMPLE` or `EVENT` `DataItem` element that can only report a single occurrence of a `Sample` or `Event` element in the `MTConnectStreams` document at any one point in time.

For example, a controller on a piece of equipment may detect and report multiple format errors in a motion program. Each error represents a separate *Fault State* from the controller. Each *Fault State* is represented as a separate `Condition` element in the `MTConnectStreams` document since each *Fault State* **MUST** be identified and tracked individually in the document.

5.7.1 *Element Names* for `Condition`

`Condition` elements are reported differently from other *Data Entity* types. The *Element Name* reported for a `Condition` element represents the *Fault State* (`Normal`, `Warning`, or `Fault`) associated with each `Condition`.

Examples of XML elements representing `Condition` elements for each of the possible *Fault States* are:

```
<Normal type="MOTION_PROGRAM" dataItemId="cc2" sequence="25"
  timestamp="2010-04-06T06:19:35.153141"</Normal>
<Fault type="COMMUNICATIONS" dataItemId="cc1" sequence="26"
  nativeCode="IO1231" timestamp="2010-04-
  06T06:19:35.153141">Communications error</Fault>
<Warning type="LOGIC_PROGRAM" dataItemId="pm6" sequence="32"
  timestamp="2010-04-06T06:19:35.153141"<Warning/>
```

Note: The **BOLDED** item is identified for emphasis only.

5.7.2 XML Schema Structure for `Condition`

The following XML schema represents the structure of a `Condition` XML element showing the attributes defined for `Condition` elements.

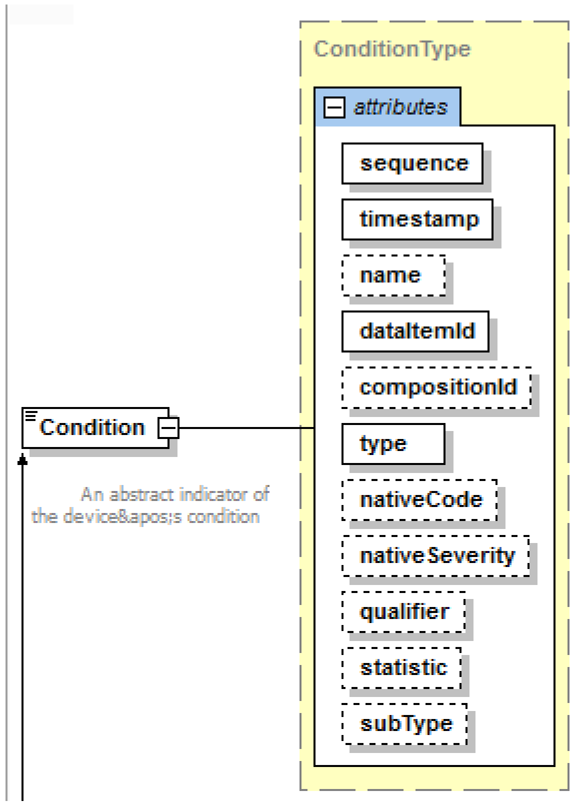


Figure 9: `Condition` Schema Diagram

5.7.3 Attributes for `Condition`

The following table defines the attributes used to provide additional information for a `Condition` XML element.

Attribute	Description	Occurrence
`sequence`	A number representing the sequential position of an occurrence of the `Condition` in the data buffer of an *MTConnect Agent*. `sequence` is a required attribute. `sequence` **MUST** have a value represented as an unsigned 64-bit value from 1 to 2^64-1.	1

Attribute	Description	Occurrence
timestamp	The most accurate time available to a piece of equipment that represents the point in time that the data reported for the Condition was measured or detected. timestamp is a required attribute.	1
name	The name of the Condition element. name is an optional attribute. name **MUST** match the name attribute of the DataItem element as defined in the MTConnectDevices document that this Condition element represents. An NMTOKEN XML type.	0..1
dataItemId	The unique identifier for the Condition element. dataItemId is a required attribute. dataItemId **MUST** match the id attribute of the DataItem element defined in the MTConnectDevices document that this Condition element represents.	1
type	An identifier of the type of fault represented by the Condition element. type is a required attribute. type **MUST** match the type attribute of the DataItem element defined in the MTConnectDevices document that this Condition element represents.	1
nativeCode	The native code (usually an alpha-numeric value) generated by the controller of a piece of equipment providing a reference identifier for a Condition. nativeCode is an optional attribute. This is the same information an operator or maintenance personnel may see as a reference code designating a specific fault code provided by the piece of equipment.	0..1
nativeSeverity	If the piece of equipment designates a severity level to a fault, nativeSeverity reports that severity information to a client software application. nativeSeverity is an optional attribute.	0..1

Attribute	Description	Occurrence
`qualifier`	`qualifier` provides additional information regarding a *Fault State* associated with the measured value of a process variable. `qualifier` is an optional attribute. `qualifier` defines whether the *Fault State* represented by the `Condition` indicates a measured value that is above or below an expected value of a process variable. If the *Fault State* represents a measured value that is greater than the expected value for the process variable, `qualifier` **MUST** report a value of `HIGH`. If the *Fault State* represents a measured value that is less than the expected value for the process variable, `qualifier` **MUST** report a value of `LOW`.	0..1
`statistic`	`statistic` provides additional information describing the meaning of the `Condition` element. `statistic` is an optional attribute. `statistic` **MUST** match the `statistic` attribute of the `DataItem` element defined in the `MTConnectDevices` document that this `Condition` element represents.	0..1
`subType`	`subType` provides additional information describing the meaning of the `Condition` element. `subType` is an optional attribute. `subType` **MUST** match the `subType` attribute of the `DataItem` element defined in the `MTConnectDevices` document that this `Condition` element represents.	0..1
`compositionId`	The identifier of the `Composition` element defined in the `MTConnectDevices` document that the data reported for this `Condition` element represents. `compositionId` is an optional attribute.	0..1
`xs:lang`	An optional attribute that specifies the language of the CDATA returned for the `Condition`. Refer to IETF RFC 4646 (http://www.ietf.org/rfc/rfc4646.txt) or successor for a full definition of the values for this attribute. `xs:lang` does not appear in the schema diagram.	0..1

5.7.3.1 `qualifier` Attribute for `Condition`

Many `Condition` elements report the *Fault State* associated with the measured value of a process variable.

`qualifier` provides an indication whether the measured value is above or below an expected value of a process variable

As an example, a `Condition` element with a `type` attribute of `AMPERAGE` may differentiate between a higher than expected amperage and a lower than expected amperage by using the `qualifier` attribute.

When a `qualifier` of either HIGH or LOW is used with `Fault` and `Warning`, the *Fault States* can be differentiated as follows:

```
Fault,LOW
Warning,LOW
Normal
Warning,HIGH
Fault,HIGH
```

The following is an example of an XML element representing `Condition` using `qualifier`:

```
<Warning type="FILL_LEVEL" dataItemId="pm6" qualifier=”HIGH”
  sequence="32" timestamp="2009-11-13T08:32:18">…</Warning>
```

Note: The `qualifier` attribute of “high” is **BOLDED** for emphasis only.

5.7.4 Valid Data Values for `Condition`

`Condition` elements reported in an `MTConnectStreams` XML document **MAY** provide a value in the CDATA of the *Data Entity* when additional information regarding the *Fault State* is available.

A *Valid Data Value* for the CDATA included in a `Condition` element **MAY** be any text string. A *Valid Data Value* is not required to be reported for a `Condition` category *Data Entity*. The *Fault State* and the attributes provided in a `Condition` element **MAY** be sufficient to fully describe the *Data Entity*.

The *Valid Data Value* reported as CDATA for a `Condition` element **MUST** be formatted as part of the content between the element tags in the XML element representing that *Data Entity*. As an example, `Condition` elements are formatted as follows in the XML document:

```
<Warning type="FILL_LEVEL" dataItemId="pm6" qualifier="HIGH"
  sequence="32" timestamp="2009-11-13T08:32:18">Fill Level on Tank
  #12 is reaching a high level</Warning>
```

Note: The **BOLDED** items are identified for emphasis only.

In this example, the “Fill Level on Tank #12 is reaching a high level” is the CDATA for the *Data Entity*.

5.8 Unavailability of *Fault State* for Condition

When an *MTConnect Agent* cannot determine a valid *Fault State* for a `Condition` element, it **MUST** report the *Element Name* for the *Data Entity* as `Unavailable`.

The example below demonstrates how an *MTConnect Agent* reports a `Condition` category *Data Entity* when it is unable to determine a valid *Fault State*:

```
<Unavailable type="MOTION_PROGRAM" dataItemId="cc2" sequence="25"
  timestamp="2009-11-13T08:32:18">…</Unavailable>
<Unavailable type="COMMUNICATIONS" dataItemId="cc1" sequence="26"
  timestamp="2009-11-13T08:32:18">…</Unavailable>
<Unavailable type="LOGIC_PROGRAM" dataItemId="cc3" sequence="28"
  timestamp="2009-11-13T08:32:18">…</Unavailable>
<Unavailable type="LOGIC_PROGRAM" dataItemId="pm6" sequence="32"
  timestamp="2009-11-13T08:32:18">…</Unavailable>
```

Note: The **BOLDED** items are identified for emphasis only.

6 Listing of *Data Entities*

Data Entities that report data in MTConnectStreams documents are represented by Sample, Event, or Condition elements based upon the category and type attributes defined for the corresponding DataItem XML element in the MTConnectDevices document.

Each *Data Entity* in the MTConnectStreams document has an *Element Name*, as defined in the following sections, based upon the corresponding category attribute defined for that DataItem element in the MTConnectDevices document.

6.1 Sample *Element Names*

The following is a list of the XML elements that can be placed in the Samples container of the ComponentStream element.

The table shows both the type attribute for each SAMPLE category DataItem element as defined in the MTConnectDevices document and the corresponding *Element Name* for the *Data Entity* that **MUST** be reported as a Sample element in the MTConnectStreams document.

SAMPLE Data Item Type	Sample *Element Name*	Description
ACCELERATION	Acceleration	The measurement of the rate of change of velocity. Acceleration **MUST** be reported in units of MILLIMETER/SECOND^2.
ACCUMULATED_TIME	AccumulatedTime	The measurement of accumulated time for an activity or event. AccumulatedTime **MUST** be reported in units of SECOND. **DEPRECATION WARNING:** May be deprecated in the future. Recommend using ProcessTimer and MachineTimer.
ANGULAR_ACCELERATION	AngularAcceleration	The measurement of the rate of change of angular velocity. AngularAcceleration **MUST** be reported in units of DEGREE/SECOND^2 .
ANGULAR_VELOCITY	AngularVelocity	The measurement of the rate of change of angular position. AngularVelocity **MUST** be reported in units of DEGREE/SECOND.

SAMPLE Data Item Type	Sample *Element Name*	Description
AMPERAGE	Amperage	The measurement of electrical current. Subtypes of Amperage are ALTERNATING, DIRECT, ACTUAL, and TARGET. If a subType is not specified, the reported value for the data **MUST** default to the subtype of ACTUAL. Amperage **MUST** be reported in units of AMPERE.
ANGLE	Angle	The measurement of angular position. Subtypes of Angle are ACTUAL and COMMANDED. If a subType is not specified, the reported value for the data **MUST** default to the subtype of ACTUAL. Angle **MUST** be reported in units of DEGREE.
AXIS_FEEDRATE	AxisFeedrate	The measurement of the feedrate of a linear axis. Subtypes of AxisFeedrate are ACTUAL, COMMANDED, JOG, PROGRAMMED, and RAPID. If a subType is not specified, the reported value for the data **MUST** default to the subtype of PROGRAMMED. AxisFeedrate **MUST** be reported in units of MILLIMETER/SECOND.
CLOCK_TIME	ClockTime	The value provided by a timing device at a specific point in time. ClockTime **MUST** be reported in W3C ISO 8601 format of YYYY-MM-DDThh:mm:ss.ffff.
CONCENTRATION	Concentration	The measurement of the percentage of one component within a mixture of components. Concentration **MUST** be reported in units of PERCENT.
CONDUCTIVITY	Conductivity	The measurement of the ability of a material to conduct electricity. Conductivity **MUST** be reported in units of SIEMENS/METER.

SAMPLE Data Item Type	Sample *Element Name*	Description
DISPLACEMENT	Displacement	The measurement of the change in position of an object. Displacement **MUST** be reported in units of MILLIMETER.
ELECTRICAL_ENERGY	ElectricalEnergy	The measurement of electrical energy consumption by a component. ElectricalEnergy **MUST** be reported in units of WATT_SECOND.
EQUIPMENT_TIMER	EquipmentTimer	The measurement of the amount of time a piece of equipment or a sub-part of a piece of equipment has performed specific activities. Subtypes of EquipmentTimer are LOADED, WORKING, OPERATING, POWERED, and DELAY. A subType **MUST** always be specified. EquipmentTimer **MUST** be reported in units of SECOND.
FILL_LEVEL	FillLevel	The measurement of the amount of a substance remaining compared to the planned maximum amount of that substance. FillLevel **MUST** be reported in units of PERCENT.
FLOW	Flow	The measurement of the rate of flow of a fluid. Flow **MUST** be reported in units of LITER/SECOND.
FREQUENCY	Frequency	The measurement of the number of occurrences of a repeating event per unit time. Frequency **MUST** be reported in units of HERTZ.
~~GLOBAL_POSITION~~	~~GlobalPosition~~	**DEPRECATED** in *Version 1.1.0.*
~~LEVEL~~	~~Level~~	**DEPRECATED** in *Version 1.2.0.* See FILL_LEVEL

SAMPLE Data Item Type	Sample *Element Name*	Description
LENGTH	Length	The measurement of the length of an object. Subtypes of Length are STANDARD, REMAINING, and USEABLE. If a subType is not specified, the reported value for the data **MUST** default to the subtype of REMAINING. Length **MUST** be reported in units of MILLIMETER.
LINEAR_FORCE	LinearForce	The measurement of the push or pull introduced by an actuator or exerted on an object. LinearForce **MUST** be reported in units of NEWTON.
LOAD	Load	The measurement of the actual versus the standard rating of a piece of equipment. Load **MUST** be reported in units of PERCENT.
MASS	Mass	The measurement of the mass of an object(s) or an amount of material. Mass **MUST** be reported in units of KILOGRAM.
PATH_FEEDRATE	PathFeedrate	The measurement of the feedrate for the axes, or a single axis, associated with a Path component– a vector. Subtypes of PathFeedrate are ACTUAL, COMMANDED, JOG, PROGRAMMED, and RAPID. If a subType is not specified, the reported value for the data **MUST** default to the subtype of PROGRAMMED. PathFeedrate **MUST** be reported in units of MILLIMETER/SECOND.

SAMPLE Data Item Type	Sample *Element Name*	Description
PATH_POSITION	PathPosition	A measured or calculated position of a control point reported by the CONTROLLER element of a piece of equipment expressed in WORK coordinates. The coordinate system will revert to MACHINE coordinates if WORK coordinates are not available. Subtypes of PathPosition are ACTUAL, PROGRAMMED, COMMANDED, TARGET, and PROBE. If a subType is not specified, the reported value for the data **MUST** default to the subtype of ACTUAL. PathPosition **MUST** be reported as a set of space-delimited floating-point numbers representing a point in 3-D space. The position of the control point **MUST** be reported in units of MILLIMETER and listed in order of X, Y, and Z referenced to the coordinate system of the piece of equipment. An example of the value reported for PathPosition would be: <PathPosition …>10.123 55.232 100.981 </PathPosition> Where X = 10.123, Y = 55.232, and Z=100.981.
PH	Ph	The measurement of acidity or alkalinity. PH **MUST** be reported in units of PH.

SAMPLE Data Item Type	Sample *Element Name*	Description
POSITION	Position	A measured or calculated position of a component element as reported by a piece of equipment. Subtypes of Position are ACTUAL, COMMANDED, PROGRAMMED, and TARGET. If a subType is not specified, the reported value for the data **MUST** default to the subtype of ACTUAL. When Position is provided representing a measured value for the physical axes of the piece of equipment, the data **MUST** be provided in MACHINE coordinates. When Position is provided representing a logical or calculated position, the data **MUST** be provided in WORK coordinates and is associated with a Path element of the equipment controller. Position **MUST** be reported in units of MILLIMETER.
POWER_FACTOR	PowerFactor	The measurement of the ratio of real power flowing to a load to the apparent power in that AC circuit. PowerFactor **MUST** be reported in units of PERCENT.
PRESSURE	Pressure	The measurement of the force per unit area exerted by a gas or liquid. Pressure **MUST** be reported in units of PASCAL.
PROCESS_TIMER	ProcessTimer	The measurement of the amount of time a piece of equipment has performed different types of activities associated with the process being performed at that piece of equipment. Subtypes of ProcessTimer are PROCESS and DELAY. A subType **MUST** always be specified. ProcessTimer **MUST** be reported in units of SECOND.
RESISTANCE	Resistance	The measurement of the degree to which a substance opposes the passage of an electric current. Resistance **MUST** be reported in units of OHM.

SAMPLE Data Item Type	Sample *Element Name*	Description
ROTARY_VELOCITY	RotaryVelocity	The measurement of the rotational speed of a rotary axis. Subtypes of RotaryVelocity are ACTUAL, COMMANDED, and PROGRAMMED. If a subType is not specified, the reported value for the data **MUST** default to the subtype of ACTUAL. RotaryVelocity **MUST** be reported in units of REVOLUTION/MINUTE.
SOUND_LEVEL	SoundLevel	The measurement of a sound level or sound pressure level relative to atmospheric pressure. Subtypes of SoundLevel are NO_SCALE, A_SCALE, B_SCALE, C_SCALE, and D_SCALE. If a subType is not specified, the reported value for the data **MUST** default to the subtype of NO_SCALE. SoundLevel **MUST** be provided in DECIBEL.
~~SPINDLE_SPEED~~	~~SpindleSpeed~~	**DEPRECATED** in *Version 1.2.0.* Replaced by ROTARY_VELOCITY
STRAIN	Strain	The measurement of the amount of deformation per unit length of an object when a load is applied. Strain **MUST** be reported in units of PERCENT.
TEMPERATURE	Temperature	The measurement of temperature. Temperature **MUST** be reported in units of degrees CELSIUS.
TENSION	Tension	The measurement of a force that stretches or elongates an object. Tension **MUST** be reported in units of NEWTON.
TILT	Tilt	A measurement of angular displacement. Tilt **MUST** be reported in units of MICRO_RADIAN.
TORQUE	Torque	The measurement of the turning force exerted on an object or by an object. Torque **MUST** be reported in units of NEWTON_METER.

SAMPLE Data Item Type	Sample *Element Name*	Description
VOLT_AMPERE	VoltAmpere	The measurement of the apparent power in an electrical circuit, equal to the product of root-mean-square (RMS) voltage and RMS current (commonly referred to as VA). VoltAmpere **MUST** be reported in units of VOLT_AMPERE.
VOLT_AMPERE_REACTIVE	VoltAmpereReactive	The measurement of reactive power in an AC electrical circuit (commonly referred to as VAR). VoltAmpereReactive **MUST** be reported in units of VOLT_AMPERE_REACTIVE.
VELOCITY	Velocity	The measurement of the rate of change of position of a component. When provided as the Velocity of the Axes component, it represents the value of the velocity vector for all given axes, similar to PathFeedrate. When provided as the Velocity of an individual axis component, it represents the value of the velocity for that specific axis with no influence of the relative velocity of any other axes. Velocity **MUST** be reported in units of MILLIMETER/SECOND.
VISCOSITY	Viscosity	A measurement of a fluid's resistance to flow. Viscosity **MUST** be reported in units of PASCAL_SECOND.
VOLTAGE	Voltage	The measurement of electrical potential between two points. Subtypes of Voltage are ALTERNATING, DIRECT, ACTUAL, and TARGET. If a subType is not specified, the reported value for the data **MUST** default to the subtype of ACTUAL. Voltage **MUST** be reported in units of VOLT.

SAMPLE Data Item Type	Sample *Element Name*	Description
WATTAGE	Wattage	The measurement of power flowing through or dissipated by an electrical circuit or piece of equipment. Subtypes of Wattage are ACTUAL and TARGET. If a subType is not specified, the reported value for the data **MUST** default to the subtype of ACTUAL. Wattage **MUST** be reported in units of WATT.

Note: The Sample response format **MUST** be extended when the representation attribute for the data item is TIME_SERIES. See *Section 5.3.3* of this document for details on extending the response format.

6.2 Event *Element Names*

The following is a list of the XML elements that can be placed in the Events container of the ComponentStream element.

The table shows both the type for each EVENT category DataItem element defined in the MTConnectDevices document and the corresponding *Element Name* for the *Data Entity* that **MUST** be reported as an Event element in the MTConnectStreams document.

The table also defines the *Valid Data Values* for those Event type data items where the reported values are restricted to a *Controlled Vocabulary*.

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
ACTUATOR_STATE	ActuatorState	ActuatorState represents the operational state of an apparatus for moving or controlling a mechanism or system. *Valid Data Values*: - ACTIVE: The actuator is operating - INACTIVE: The actuator is not operating

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
~~ALARM~~	~~Alarm~~	**DEPRECATED**: Replaced with CONDITION category data items in *Version 1.1.0*.
ACTIVE_AXES	ActiveAxes	The set of axes currently associated with a Path or Controller *Structural Element*. The *Valid Data Value* reported **SHOULD** be a space-delimited set of axes names. The names returned **SHOULD** match the name attribute of the Linear or Rotary *Structural Elements* defined in the MTConnectDevices document that this Event element represents. If name is not available, nativeName **MUST** be returned to identify the Linear or Rotary *Structural Elements*. For example: <ActiveAxes …>X Y Z W S</ActiveAxes> where X, Y, Z, W, and S are the nativeName attributes of the *Structural Elements*. If it is not specified elsewhere in the MTConnectDevices document, it **MUST** be assumed that all of the axes are associated with the Path component.
AVAILABILITY	Availability	Represents an *MTConnect Agent's* ability to communicate with the data source. Availability **MUST** be provided for each Device *Structural Element* and **MAY** be provided for any other *Structural Element*. *Valid Data Values*: - AVAILABLE: The *Structural Element* is active and capable of providing data. - UNAVAILABLE: The *Structural Element* is either inactive or not capable of providing data.

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
AXIS_COUPLING	AxisCoupling	Describes the way axes are associated to each other. This is used in conjunction with COUPLED_AXES to indicate the interaction between axes. The coupling of the axes **MUST** be viewed from the perspective of a specified axis. Therefore, a MASTER coupling indicates that this axis is the master for the COUPLED_AXES. AxisCoupling **MUST** be provided for each axis element associated with a set of axes defined by the COUPLED_AXES data item element defined in the MTConnectDevices document. *Valid Data Values*: - TANDEM: The axes are physically connected to each other and operate as a single unit. - SYNCHRONOUS: The axes are not physically connected to each other but are operating together in lockstep. - MASTER: The axis is the master of the CoupledAxes - SLAVE: The axis is a slave to the CoupledAxes

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
AXIS_FEEDRATE_ OVERRIDE	AxisFeedrateOverride	The value of a signal or calculation issued to adjust the feedrate of an individual linear type axis. The value provided for AxisFeedrateOverride is expressed as a percentage of the designated feedrate for the axis. Subtypes of AxisFeedrateOverride are JOG, PROGRAMMED, and RAPID. If a subType is not specified, the reported value for the data **MUST** default to the subtype of PROGRAMMED. The *Valid Data Value* **MUST** be a floating-point number.
AXIS_INTERLOCK	AxisInterlock	An indicator of the state of the axis lockout function when power has been removed and the axis is allowed to move freely. *Valid Data Values*: - ACTIVE: The axis lockout function is activated, power has been removed from the axis, and the axis is allowed to move freely. - INACTIVE: The axis lockout function has not been activated, the axis may be powered, and the axis is capable of being controlled by another component.
AXIS_STATE	AxisState	An indicator of the controlled state of a LINEAR or ROTARY component representing an axis. *Valid Data Values*: - HOME: The axis is in its home position. - TRAVEL: The axis is in motion - PARKED: The axis has been moved to a fixed position and is being maintained in that position either electrically or mechanically. Action is required to release the axis from this position. - STOPPED: The axis is stopped

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
BLOCK	Block	The line of code or command being executed by a Controller *Structural Element*. Block **MUST** include the entire expression for a line of program code, including all parameters The *Valid Data Value* **MUST** be any text string.
BLOCK_COUNT	BlockCount	The total count of the number of blocks of program code that have been executed since execution started. The *Valid Data Value* **MUST** be an integer.
CHUCK_INTERLOCK	ChuckInterlock	An indication of the state of an interlock function or control logic state intended to prevent the associated CHUCK component or composition element from being operated. A CHUCK component or composition element may be controlled by more than one type of ChuckInterlock function. When the ChuckInterlock function is provided by an operator controlled interlock that can inhibit the ability to initiate an unclamp action of an electronically controlled chuck, this ChuckInterlock function **SHOULD** be further characterized by specifying a subType of MANUAL_UNCLAMP. *Valid Data Values*: - ACTIVE: The chuck cannot be unclamped - INACTIVE: The chuck can be unclamped.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
CHUCK_STATE	ChuckState	An indication of the operating state of a mechanism that holds a part or stock material during a manufacturing process. It may also represent a mechanism that holds any other item in place within a piece of equipment. *Valid Data Values*: - OPEN: The CHUCK component or composition element is open to the point of a positive confirmation - CLOSED: The CHUCK component or composition element is closed to the point of a positive confirmation - UNLATCHED: The CHUCK component or composition element is not closed to the point of a positive confirmation and not open to the point of a positive confirmation. It is in an intermediate position.
~~CODE~~	~~Code~~	**DEPRECATED** in *Version 1.1.0*.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
COMPOSITION_STATE	CompositionState	An indication of the operating condition of a mechanism represented by a Composition type element. Subtypes of CompositionState are ACTION, LATERAL, MOTION, SWITCHED, and VERTICAL. A subType **MUST** be provided. *Valid Data Values* for subtype ACTION are: - ACTIVE: The Composition element is operating - INACTIVE: The Composition element is not operating *Valid Data Values* for subtype LATERAL are: - RIGHT: The position of the Composition element is oriented to the right to the point of a positive confirmation - LEFT: The position of the Composition element is oriented to the left to the point of a positive confirmation - TRANSITIONING: The position of the Composition element is not oriented to the right to the point of a positive confirmation and is not oriented to the left to the point of a positive confirmation. It is in an intermediate position. *Valid Data Values* for subtype MOTION are: - OPEN: The position of the Composition element is open to the point of a positive confirmation - CLOSED: The position of the Composition element is closed to the point of a positive confirmation - UNLATCHED: The position of the Composition element is not open to the point of a positive confirmation and is not closed to the point of a positive confirmation. It is in an intermediate position.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
COMPOSITION_STATE (Continued)	CompositionState (Continued)	*Valid Data Values* for subtype SWITCHED are: - ON: The activation state of the Composition element is in an ON condition, it is operating, or it is powered. - OFF: The activation state of the Composition element is in an OFF condition, it is not operating, or it is not powered. *Valid Data Values* for subtype VERTICAL are: - UP: The position of the Composition element is oriented in an upward direction to the point of a positive confirmation - DOWN: The position of the Composition element is oriented in a downward direction to the point of a positive confirmation - TRANSITIONING: The position of the Composition element is not oriented in an upward direction to the point of a positive confirmation and is not oriented in a downward direction to the point of a positive confirmation. It is in an intermediate position.

EVENT Data Item Type	**Event** *Element Name*	**Description and** *Valid Data Values*
CONTROLLER_MODE	ControllerMode	The current operating mode of the Controller component. *Valid Data Values*: - AUTOMATIC: The controller is configured to automatically execute a program. - MANUAL: The controller is not executing an active program. It is capable of receiving instructions from an external source – typically an operator. The controller executes operations based on the instructions received from the external source. - MANUAL_DATA_INPUT: The operator can enter a series of operations for the controller to perform. The controller will execute this specific series of operations and then stop. - SEMI_AUTOMATIC: The controller is operating in a single cycle mode. It executes a single set of instructions from an active program and then stops until given a command to execute the next set of instructions. - EDIT: The controller is currently functioning as a programming device and is not capable of executing an active program.
CONTROLLER_MODE_ OVERRIDE	ControllerModeOverride	A setting or operator selection that changes the behavior of a piece of equipment. Subtypes of CompositionState are DRY_RUN, SINGLE_BLOCK, MACHINE_AXIS_LOCK, OPTIONAL_STOP, and TOOL_CHANGE_STOP. A subType **MUST** always be specified. *Valid Data Values*: - ON: The indicator of the ControllerModeOverride is in the ON state and the mode override is active. - OFF: The indicator of the ControllerModeOverride is in the OFF state and the mode override is inactive

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
COUPLED_AXES	CoupledAxes	Refers to a set of associated axes. Used in conjunction with AxisCoupling to describe how the CoupledAxes relate to each other. The *Valid Data Value* reported **SHOULD** be a space-delimited set of axes names. The names returned **SHOULD** match the name attribute of the Linear or Rotary *Structural Elements* defined in the MTConnectDevices document that this Event element represents. If name is not available, nativeName **MUST** be returned to identify the Linear or Rotary *Structural Elements*. Example: <CoupledAxes …>Y1 Y2</CoupledAxes>
DIRECTION	Direction	The direction of motion. Subtypes of Direction are ROTARY and LINEAR. A subType **MUST** always be specified. *Valid Data Values* for subtype ROTARY are: - CLOCKWISE: A ROTARY type component is rotating in a clockwise fashion using the right-hand rule. - COUNTER_CLOCKWISE: A ROTARY type component is rotating in a counter clockwise fashion using the right-hand rule. *Valid Data Values* for subtype LINEAR are: - POSITIVE: A LINEAR type component is moving in the direction of increasing position value - NEGATIVE: A LINEAR type component is moving in the direction of decreasing position value

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
DOOR_STATE	DoorState	The operational state of a DOOR type component or composition element. *Valid Data Values*: - OPEN: The Door is open to the point of a positive confirmation - CLOSED: The Door is closed to the point of a positive confirmation - UNLATCHED: The DOOR is not closed to the point of a positive confirmation and is not open to the point of a positive confirmation. It is in an intermediate position.
END_OF_BAR	EndOfBar	An indication of whether the end of a piece of bar stock being fed by a bar feeder has been reached. Subtypes of EndOfBar are PRIMARY and AUXILIARY. If a subType is not specified, the reported value for the data **MUST** default to the subtype of PRIMARY. *Valid Data Values*: - YES: The EndOfBar has been reached. - NO: The EndOfBar has not been reached.
EMERGENCY_STOP	EmergencyStop	The current state of the emergency stop signal for a piece of equipment, controller path, or any other component or subsystem of a piece of equipment. *Valid Data Values*: - ARMED: The emergency stop circuit is complete and the piece of equipment, component, or composition element is allowed to operate. - TRIGGERED: The emergency stop circuit is open and the operation of the piece of equipment, component, or composition element is inhibited.

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
EQUIPMENT_MODE	EquipmentMode	An indication that a piece of equipment, or a sub-part of a piece of equipment, is performing specific types of activities. Subtypes of EquipmentMode are LOADED, WORKING, OPERATING, and POWERED. A subType **MUST** always be specified. *Valid Data Values*: - ON: The equipment is functioning in the mode designated by the subType. - OFF: The equipment is not functioning in the mode designated by the subType.

EVENT Data Item Type	**Event** *Element Name*	**Description and** *Valid Data Values*
EXECUTION	Execution	The execution status of the Controller component. *Valid Data Values*: - READY: The controller is ready to execute instructions. It is currently idle. - ACTIVE: The controller is actively executing an instruction. - INTERRUPTED: The execution of the controller's program has been suspended due to an external signal. Action is required to resume execution. - FEED_HOLD: Motion of the device has been commanded to stop at its current position. The controller remains able to execute instructions but cannot complete the current set of instructions until after motion resumes. The command to stop the motion must be removed before execution can resume. - STOPPED: The execution of the controller's program has been stopped in an unplanned manner and execution of the program cannot be resumed without intervention by an operator or external signal. - OPTIONAL_STOP: The controller's program has been intentionally stopped using an M01 or similar command. The program may be stopped at the designated location based upon the state of a secondary indication provided to the controller indicating whether the program execution must be stopped at this location or program execution should continue. - PROGRAM_STOPPED: The execution of the controller's program has been stopped by a command from within the program. Action is required to resume execution. - PROGRAM_COMPLETED: The program has completed execution.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
FUNCTIONAL_MODE	FunctionalMode	The current intended production status or intended use of a piece of equipment or component. Typically, the FunctionalMode **SHOULD** be associated with the Device *Structural Element*, but it **MAY** be associated with any *Structural Element* in the XML document. *Valid Data Values*: - PRODUCTION: The Device element or another *Structural Element* is currently producing product, ready to produce product, or its current intended use is to be producing product. - SETUP: The Device element or another *Structural Element* is not currently producing product. It is being prepared or modified to begin production of product. - TEARDOWN: The Device element or another *Structural Element* is not currently producing product. Typically, it has completed the production of a product and is being modified or returned to a neutral state such that it may then be prepared to begin production of a different product. - MAINTENANCE: The Device element or another *Structural Element* is not currently producing product. It is currently being repaired, waiting to be repaired, or has not yet been returned to a normal production status after maintenance has been performed. - PROCESS_DEVELOPMENT: The Device element or another *Structural Element* is being used to prove-out a new process, testing of equipment or processes, or any other active use that does not result in the production of product.
HARDNESS	Hardness	The measurement of the hardness of a material. Subtypes of Hardness are ROCKWELL, VICKERS, SHORE, BRINELL, LEEB, and MOHS. A subType **MUST** always be specified. The *Valid Data Value* **MUST** be a floating-point number.

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
~~LINE~~	~~Line~~	**DEPRECATED** in *Version 1.4.0.*
LINE_LABEL	LineLabel	An optional identifier for a BLOCK of code in a PROGRAM. The *Valid Data Value* **MUST** be any text string.
LINE_NUMBER	LineNumber	A reference to the position of a block of program code within a control program. Subtypes of LineNumber are ABSOLUTE and INCREMENTAL. A subType **MUST** always be specified. The *Valid Data Value* **MUST** be an integer.
MATERIAL	Material	The identifier of a material used or consumed in the manufacturing process. The *Valid Data Value* **MUST** be any text string.
MESSAGE	Message	Any text string of information to be transferred from a piece of equipment to a client software application. The *Valid Data Value* **MUST** be any text string.
OPERATOR_ID	OperatorId	The identifier of the person currently responsible for operating the piece of equipment. The *Valid Data Value* **MAY** be any text string. **DEPRECATION WARNING:** May be deprecated in the future. See USER below.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
PALLET_ID	PalletId	The identifier for a pallet. The *Valid Data Value* **MAY** be any text string.
PART_COUNT	PartCount	The current count of parts produced as represented by the Controller component. Subtypes of PartCount are ALL, GOOD, BAD, TARGET, and REMAINING. PartCount will not be accumulated by an *MTConnect Agent* and **MUST** only be supplied if the Controller provides the count. PartCount **MAY** have a representation of DISCRETE. In this case, each occurrence of PartCount in an MTConnectStreams document represents a unique count of parts or product produced – it is not an accumulated count of parts or product produced. The *Valid Data Value* **MUST** be a floating-point number, usually an integer.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
PART_ID	PartId	An identifier of a part in a manufacturing operation. The *Valid Data Value* **MAY** be any text string.
PATH_FEEDRATE_ OVERRIDE	PathFeedrateOverride	The value of a signal or calculation issued to adjust the feedrate for the axes associated with a Path component that may represent a single axis or the coordinated movement of multiple axes. The value provided for PathFeedrateOverride is expressed as a percentage of the designated feedrate for the path. Sub-types of PathFeedrateOverride are JOG, PROGRAMMED, and RAPID. If a subType is not specified, the reported value for the data **MUST** default to the subtype of PROGRAMMED. The *Valid Data Value* **MUST** be a floating-point number.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
PATH_MODE	PathMode	Describes the operational relationship between a PATH *Structural Element* and another PATH *Structural Element* for pieces of equipment comprised of multiple logical groupings of controlled axes or other logical operations. *Valid Data Values*: - INDEPENDENT: The path is operating independently and without the influence of another path. - MASTER: The path provides the reference motion for a SYNCHRONOUS or MIRROR type path to follow. For non-motion type paths, the MASTER provides information or state values that influences the operation of other paths - SYNCHRONOUS: The axes associated with the path are following the motion of the MASTER type path. - MIRROR: The axes associated with the path are mirroring the motion of the MASTER path. When PathMode is not specified, the operational mode of the path **MUST** be interpreted as INDEPENDENT.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
POWER_STATE	PowerState	The indication of the status of the source of energy for a *Structural Element* to allow it to perform its intended function or the state of an enabling signal providing permission for the *Structural Element* to perform its functions. Subtypes of PowerState are LINE and CONTROL. When the subType is LINE, PowerState represents the primary source of energy for a *Structural Element*. When the subType is CONTROL, PowerState represents an enabling signal providing permission for the *Structural Element* to perform its function(s). If a subType is not specified, the reported value for the data **MUST** default to the subtype of LINE. *Valid Data Values*: - ON: The source of energy for a *Structural Element* or the enabling signal providing permission for the *Structural Element* to perform its function(s) is present and active. - OFF: The source of energy for a *Structural Element* or the enabling signal providing permission for the *Structural Element* to perform its function(s) is not present or is disconnected. **DEPRECATION WARNING**: PowerState may be deprecated in the future.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
~~POWER_STATUS~~	~~PowerStatus~~	**DEPRECATED** in *Version 1.1.0.*
PROGRAM	Program	The name of the logic or motion program being executed by the Controller component. This is usually the name of the file containing the program instructions. The *Valid Data Value* **MUST** be any text string.

EVENT Data Item Type	Event *Element Name*	Description and *Valid Data Values*
PROGRAM_EDIT	ProgramEdit	An indication of the status of the Controller component's program editing mode. On many controls, a program can be edited while another program is currently being executed. ProgramEdit provides an indication of whether the controller is being used to edit programs in either case. *Valid Data Values*: - ACTIVE: The controller is in the program edit mode. - READY: The controller is capable of entering the program edit mode and no function is inhibiting a change to that mode. - NOT_READY: A function is inhibiting the controller from entering the program edit mode.
PROGRAM_EDIT_NAME	ProgramEditName	The name of the program being edited. This is used in conjunction with PROGRAM_EDIT when it is in an ACTIVE state. The *Valid Data Value* **MUST** be any text string.
PROGRAM_COMMENT	ProgramComment	A comment or non-executable statement in the control program. The *Valid Data Value* **MUST** be any text string.
PROGRAM_HEADER	ProgramHeader	The non-executable header section of the control program. The content **SHOULD** be limited to 512 bytes. The *Valid Data Value* **MUST** be any text string.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
ROTARY_MODE	RotaryMode	The current operating mode for a Rotary type axis. *Valid Data Values*: - SPINDLE: The axis is functioning as a spindle. Generally, it is configured to rotate at a defined speed. - INDEX: The axis is configured to index to a set of fixed positions or to incrementally index by a fixed amount. - CONTOUR: The position of the axis is being interpolated as part of the PathPosition defined by the Controller *Structural Element*.
ROTARY_VELOCITY_ OVERRIDE	RotaryVelocity Override	The value of a command issued to adjust the programmed velocity for a Rotary type axis. This command represents a percentage change to the velocity calculated by a logic or motion program or set by a switch for a Rotary type axis. RotaryVelocityOverride is expressed as a percentage of the programmed RotaryVelocity. The *Valid Data Value* **MUST** be a floating-point number.
SERIAL_NUMBER	SerialNumber	The serial number associated with a Component, Asset, or Device. The *Valid Data Value* **MUST** be any text string.
SPINDLE_INTERLOCK	SpindleInterlock	An indication of the status of the spindle for a piece of equipment when power has been removed and it is free to rotate. *Valid Data Values*: - ACTIVE: Power has been removed and the spindle cannot be operated. - INACTIVE: Spindle has not been deactivated.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
~~TOOL_ID~~	~~ToolID~~	**DEPRECATED** in *Version 1.2.0*. See Tool_ASSET_ID. ~~The identifier of the tool currently in use for a given Path~~
TOOL_ASSET_ID	ToolAssetId	The unique identifier of an individual tool asset. The *Valid Data Value* **MUST** be any text string.
TOOL_NUMBER	ToolNumber	The identifier assigned by the Controller component to a cutting tool when in use by a piece of equipment. The *Valid Data Value* **MUST** be any text string.
TOOL_OFFSET	ToolOffset	A reference to the tool offset variables applied to the active cutting tool associated with a Path in a Controller type component. Subtypes of ToolOffset are RADIAL and LENGTH. A subType **MUST** always be specified. The *Valid Data Value* **MUST** be a floating-point number.
USER	User	The identifier of the person currently responsible for operating the piece of equipment. Subtypes of User are OPERATOR, MAINTENANCE, and SET_UP. A subType **MUST** always be specified. The Valid Data Value **MUST** be any text string.
WIRE	Wire	The identifier for the type of wire used as the cutting mechanism in Electrical Discharge Machining or similar processes. The Valid Data Value **MUST** be any text string.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** ***Valid Data Values***
WORKHOLDING_ID	WorkholdingId	The identifier for the current workholding or part clamp in use by a piece of equipment. The *Valid Data Value* **MUST** be any text string.
WORK_OFFSET	WorkOffset	A reference to the offset variables for a work piece or part associated with a Path in a Controller type component. The Valid Data Value **MUST** be a floating-point number.

Note: The Event response format **MUST** be extended to represent those data items where the representation attribute is DISCRETE. See *Section 5.5.3* of this document for details on extending the response format.

6.3 Types of Condition Elements

As described above in Section 5.7, `Condition` *Data Entities* are reported differently from other data item types. They are reported based on the *Fault State* for each `Condition`. Unlike `Sample` and `Event` data items that are identified by their *Element Name*, `Condition` data items are defined by the `type` and `subType` (where applicable) attributes defined for each `Condition`.

The `type` and `subType` (where applicable) attributes for a `Condition` element **MAY** be any of the `type` and `subType` attributes defined for `SAMPLE` category or `EVENT` category data item listed in the *Device Information Model*.

The following table lists additional `Condition` *Data Entities* that have been defined to represent the health and fault status of *Structural Elements*. The table defines the `type` attribute for each of these additional `Condition` category elements that **MAY** be reported in the `MTConnectStreams` document.

CONDITION Data Item Type	Description
ACTUATOR	An indication of a fault associated with an actuator.
CHUCK_INTERLOCK	An indication of the operational condition of the interlock function for an electronically controller chuck.
COMMUNICATIONS	An indication that the piece of equipment has experienced a communications failure.
DATA_RANGE	An indication that the value of the data associated with a measured value or a calculation is outside of an expected range.
DIRECTION	An indication of a fault associated with the direction of motion of a *Structural Element.*
END_OF_BAR	An indication that the end of a piece of bar stock has been reached.
HARDWARE	An indication of a fault associated with the hardware subsystem of the *Structural Element*.
INTERFACE_STATE	An indication of the operational condition of an `Interface` component.
LOGIC_PROGRAM	An indication that an error occurred in the logic program or programmable logic controller (PLC) associated with a piece of equipment.
MOTION_PROGRAM	An indication that an error occurred in the motion program associated with a piece of equipment

CONDITION Data Item Type	Description
SYSTEM	A general purpose indication associated with an electronic component of a piece of equipment or a controller that represents a fault that is not associated with the operator, program, or hardware.

Appendices

A. Bibliography

1. Engineering Industries Association. *EIA Standard* - *EIA-274-D*, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.

2. ISO TC 184/SC4/WG3 N1089. *ISO/DIS 10303-238*: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.

3. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.

4. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.

5. International Organization for Standardization. *ISO 6983/1* – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.

6. Electronic Industries Association. *ANSI/EIA-494-B-1992*, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.

7. National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.

8. International Organization for Standardization. *ISO 10303-11*: 1994, Industrial automation systems and integration product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

9. International Organization for Standardization. *ISO 10303-21*: 1996, Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.

10. H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.

11. International Organization for Standardization. *ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature.* Geneva, Switzerland, 2001.

12. ASME B5.57: *Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers,* 1998

13. ASME/ANSI B5.54: *Methods for Performance Evaluation of Computer Numerically Controlled Machining Centers. 2005.*

14. OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.*

15. IEEE STD 1451.0-2007*, Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH99684, October 5, 2007.*

16. IEEE STD 1451.4-1994*, Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH95225, December 15, 2004.*

MTConnect® Standard
Part 4.0 – Assets Information Model

Version 1.4.0

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

MTConnect® Specification and Materials

Table of Content

Table of Figures

1 Purpose of This Document

This document, *Part 4.0 – Assets Information Model* of the MTConnect Standard, details information that is common to all types of *MTConnect Assets*. *Part 4.0* and its sub-parts of the MTConnect Standard provide semantic models for entities that are used in the manufacturing process, but are not considered to be a piece of equipment. These entities are defined as *MTConnect Assets*. These *Assets* may be removed from a piece of equipment without detriment to the function of the equipment and can be associated with other pieces of equipment during their lifecycle. The data associated with these *Assets* may be retrieved from multiple sources that are each responsible for providing their knowledge of the *Asset*.

2 Terminology and Conventions

Please refer to *Part 1.0 - Overview and Fundamentals*, *Section 2* for a dictionary of terms, reserved language, and document conventions used in the MTConnect Standard.

3 MTConnect Assets

3.1 Overview

The MTConnect Standard supports a simple distributed storage mechanism that allows applications and equipment to share and exchange complex information models in a similar way to a distributed data store. The *Asset Information Model* associates each electronic *MTConnectAssets* document with a unique identifier and allows for some predefined mechanisms to find, create, request, updated, and delete these electronic documents in a way that provides for consistency across multiple pieces of equipment.

The protocol provides a limited mechanism of accessing *MTConnect Assets* using the following properties: `assetId`, *Asset* type (element name of *Asset* root), and the piece of equipment associated with the *Asset*. These access strategies will provide the following services and answer the following questions: What *Assets* are from a particular piece of equipment? What are the *Assets* of a particular type? What *Assets* is stored for a given `assetId`?

Although these mechanisms are provided, an *MTConnect Agent* should not be considered a data store or a system of reference. The *Agent* is providing an ephemeral storage capability that will temporarily manage the data for applications wishing to communicate and manage data as needed by the various processes. An application cannot rely on an *Agent* for long term persistence or durability since the *Agent* is only required to temporarily store the *Asset* data and may require another system to provide the source data upon initialization. An *MTConnect Agent* is always providing the best-known equipment centric view of the data given the limitations of that piece of equipment.

Note: Currently only cutting tools have been addressed by the MTConnect Standard and other *MTConnect Assets* will be defined in later versions of the Standard.

3.2 MTConnectAssets

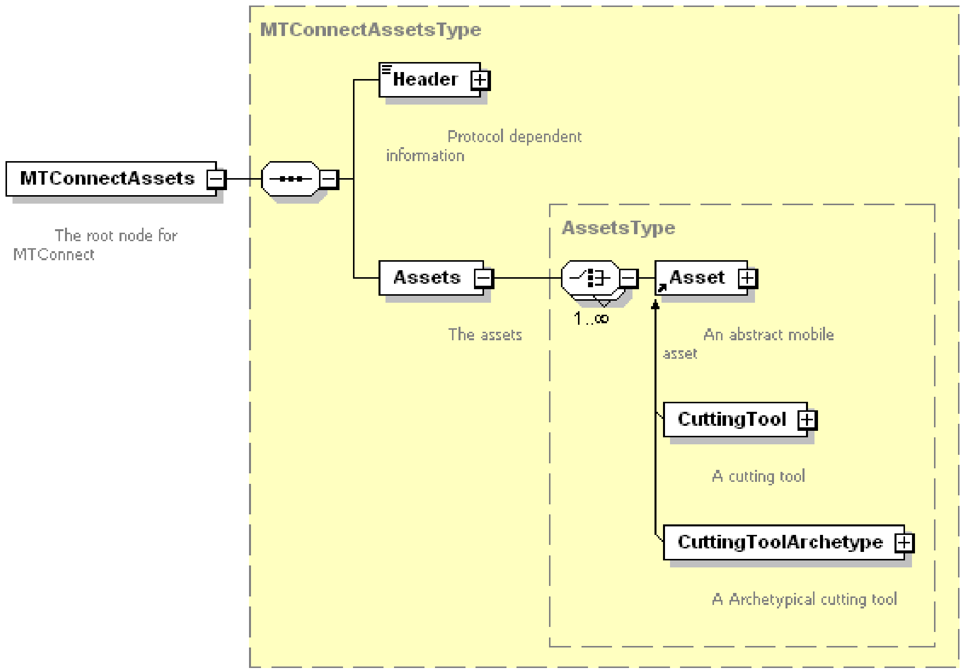


Figure 1: `MTConnectAssets` Schema

At the top level of the `MTConnectAssets` document is a standard header, as stated in *Part 1.0 - Overview and Fundamentals*, and one or more *MTConnect Assets*. Each *Asset* is required to have an `assetId` that serves as a unique identifier of that *Asset*. `assetId` allows an application to request the *Asset* data from an *MTConnect Agent*.

In the remaining *Part 4.x* sub-part documents of *MTConnect Assets*, various types of *Assets* will be introduced such as cutting tools and other *Asset* types. Currently only cutting tools have been defined in *Part 4.1 – Cutting Tools*.

3.2.1 MTConnectAssets Header

The MTConnectAssets header is where the protocol sequence information **MUST** be provided. The following XML schema represents the structure of the MTConnectAssets header showing the attributes defined for MTConnectAssets.

Refer to *Part 1.0 – Overview and Fundamentals* for more information on headers.

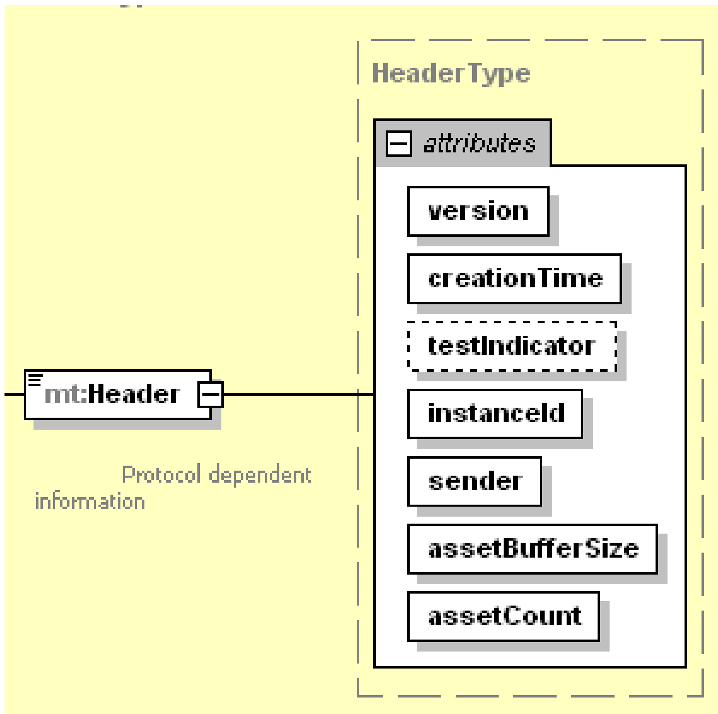


Figure 2: Header Schema Diagram for MTConnectAssets

3.2.1.1 Header Attributes

The following table defines the attributes used to provide information for an MTConnectAssets header.

Attribute	Description	Occurrence
version	The protocol version number. This is the *major* and *minor* version number of the MTConnect Standard being used. For example, if the version number of the Standard used is 10.21.33, the version will be 10.21. version is a required attribute.	1

Attribute	Description	Occurrence
`creationTime`	The time the response was created. `creationTime` is a required attribute.	1
`testIndicator`	Optional flag that indicates the system is operating in test mode. This data is only for testing and indicates that the data is simulated. `testIndicator` is an optional attribute.	0..1
`instanceId`	A number indicating which invocation of the *MTConnect Agent*. This is used to differentiate between separate instances of the *Agent*. This value **MUST** have a maximum value of 2^64-1 and **MUST** be stored in an unsigned 64-bit integer. `instanceId` is a required attribute.	1
`sender`	The *MTConnect Agent* identification information. `sender` is a required attribute.	1
`assetBufferSize`	The maximum number of *MTConnect Assets* that will be retained by the *MTConnect Agent*. The `assetBufferSize` **MUST** be an unsigned positive integer value with a maximum value of 2^32-1. `assetBufferSize` is a required attribute.	1
`assetCount`	The total number of *MTConnect Assets* in an *MTConnect Agent*. This **MUST** be an unsigned positive integer value with a maximum value of 2^32-1. This value **MUST NOT** be greater than `assetBufferSize` `assetCount` is a required attribute.	1

Example:

```
<Header creationTime="2010-03-13T07:59:11+00:00" sender="localhost"
  instanceId="1268463594" assetBufferSize="1024" version="1.1"
  assetCount="12" />
```

3.2.2 Assets

`Assets` is an XML container used to group information about various *MTConnect Asset* types. `Assets` contains one or more `Asset` XML elements.

Element	Description	Occurrence
`Assets`	XML container that consists of one or more types of `Asset` XML elements.	0..1

3.2.3 Asset

An `Asset` XML element is a container type XML element used to organize information describing an entity that is not a piece of equipment. `Asset` is an abstract type XML element and will never appear directly in the MTConnect XML document. As an abstract type XML element, `Asset` will be replaced in the XML document by specific *MTConnect Asset* type.

Element	Description	Occurrence
`Asset`	An abstract XML element. Replaced in the XML document by types of `Asset` elements representing entities that are not pieces of equipment. There can be multiple types of `Asset` XML elements in the document.	1..INF

There are various types of entities or *Asset* types. Each type of *Asset* is described in sub-parts of *Part 4.0 – Assets Information Model*. These sub-parts are designated by a *Part 4.x* document number. Currently only the *MTConnect Asset* type of cutting tools has been defined in *Part 4.1 – Cutting Tools*.

For all *MTConnect Asset* types there are some common attributes and elements that apply to all of them. The following defines these common attributes and elements.

3.2.3.1 Common `Asset` Attributes

The following XML schema represents the structure of `Asset` showing the attributes defined for `Asset`.

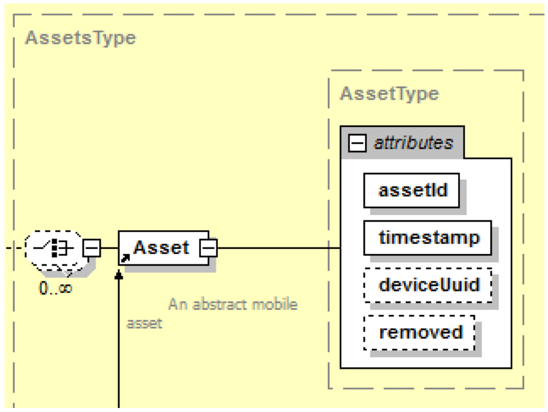


Figure 3: `Asset` Schema

The following table defines the attributes that are used to provide information for the `Asset` element.

Attribute	Description	Occurrence
`assetId`	The unique identifier for the *MTConnect Asset*. The identifier **MUST** be unique with respect to all other *Assets* in an MTConnect installation. The identifier **SHOULD** be globally unique with respect to all other *Assets*. `assetId` is a required attribute.	1
`timestamp`	The time this *MTConnect Asset* was last modified. Always given in UTC. The timestamp **MUST** be provided in UTC (Universal Time Coordinate, also known as GMT). This is the time the *Asset* data was last modified. `timestamp` is a required attribute.	1
`deviceUuid`	The piece of equipment's UUID that supplied this data. This is an optional element references to the `uuid` attribute given in the `Device` element. This can be any series of numbers and letters as defined by the XML type NMTOKEN.	0..1

Attribute	Description	Occurrence
`removed`	This is an optional attribute that is an indicator that the *MTConnect Asset* has been removed from the piece of equipment. If the *Asset* is marked as removed, it will not be visible to the client application unless the `includeRemoved=true` parameter is provided in the URL. If this attribute is not present it **MUST** be assumed to be false. The value is an `xsi:boolean` type and **MUST** be `true` or `false`.	0..1

All *MTConnect Assets* **MUST** have an `assetId` that differs from all the other *Assets* in a facility and preferably globally unique, such as a RFC 4122 UUID. There **MUST** never be more than one *Asset* provided by an *Agent* with the same `assetId` in the same shop.

The following attributes **MUST** be provided and are common to all *MTConnect Asset* types: the `assetId` attribute providing the unique identifier for the *Asset*, and the `timestamp` providing the time the *Asset* was inserted or updated. A removed flag that if `true` indicates the *Asset* has been removed (deleted) from the equipment is optional, however the *Asset* will still be available if requested directly or a request is made that includes removed *Assets*.

An `MTConnectAssets` document contains information pertaining to something that is not a direct component of the piece of equipment and can be relocated to another piece of equipment or location during its lifecycle. The `Asset` will contain data that will be changed as a unit, meaning that at any given point in time the latest version of the complete state for this *Asset* will be provided.

Each piece of equipment or location may have a different view of this *Asset* and it is the responsibility of an application to collect and determine the aggregate information and keep a historical record if required. An *MTConnect Agent* will allow any application or other equipment to request this information. The piece of equipment **MUST** supply the latest and most accurate information regarding a given *Asset*.

3.2.3.2 Common `Asset` Elements

The element `Description` is the only element common to all *Asset* types.

The following XML schema represents the structure of `Description`.

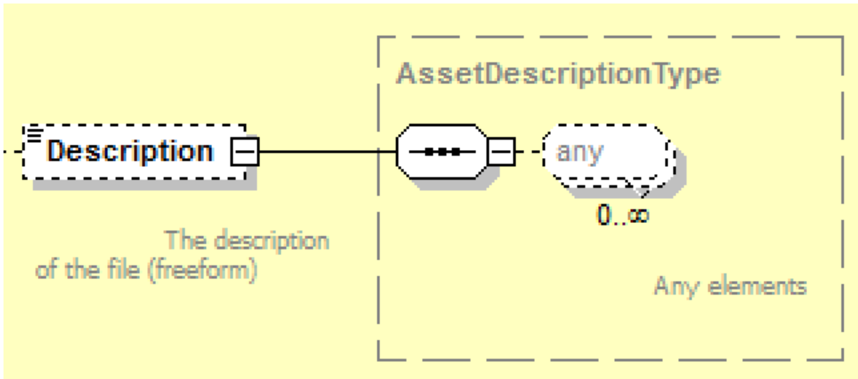


Figure 4: `Description` Schema

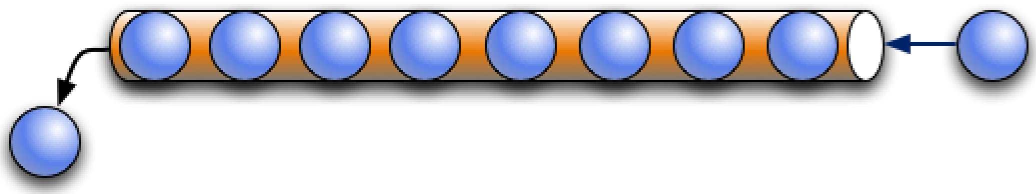
The following table defines the elements that are used to provide information for `Asset`.

Element	Description	Occurrence
`Description`	An optional element that can contain any descriptive content. This can contain configuration information and manufacturer specific details. This element is defined to contain mixed content and XML elements can be added to extend the descriptive semantics of MTConnect Standard.	0..1

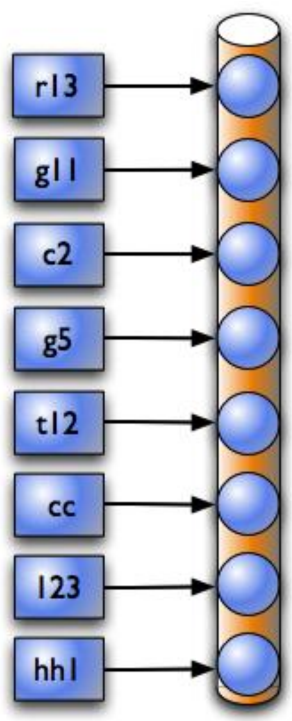
4 MTConnect Assets Architecture

4.1 MTConnect Agent Asset Storage

The *MTConnect Agent* stores *MTConnect Assets* in a similar fashion as the *Agent* data storage described in *Part 1.0 – Overview and Fundamentals*. The storage of information is contained in the *asset buffer*. The *MTConnect Agent* provides a limited number of *Assets* that can be stored at one time and uses the same method of pushing out the oldest *Asset* when the *asset buffer* is full. The *asset buffer* size for the *Asset* storage is maintained separately from the `Sample`, `Event`, and `Condition` storage.



MTConnect Assets also behave like a key/value in memory database. In the case of the *Asset,* the key is the `assetId` and the value is the XML document describing the *Asset*. The key can be any string of letters, punctuation or digits and represent the domain specific coding scheme for their assets. Each *Asset* type will have a recommended way to construct a unique `assetId`, for example, a cutting tool **SHOULD** be identified by the tool ID and serial number as a composed synthetic identifier.



As in this example above, each of the *Assets* is referred to by their key. The key is independent of the order in the *asset buffer* storage.

4.2 *Asset* Protocol

MTConnect Standard provides methods to retrieve an *MTConnect Asset* or a set of *Assets* given various criteria. These criteria are as follows: The `assetId`, the *Asset* `type` as defined by the name of the *Asset's* topmost element, and the originating piece of equipment.

The URL format is similar to the `Probe` and `Sample` structure. For example, to request an *MTConnect Asset* by `assetId`, reference each `assetId` directly as follows:

4.2.1 *Asset* by `assetId`

```
url: http://example.com/asset/e39d23ba-ef2d-11e6-b12c-
  28cfe91a82ef
```

Returns the `MTConnectAssets` document for *Asset* e39d23ba-ef2d-11e6-b12c-28cfe91a82ef

Request multiple *Assets* by each `assetId`:

```
url: http://example.com/asset/e39d23ba-ef2d-11e6-b12c-
  ;8cfe91a82ef;e46d5256-ef2d-11e6-96aa-28cfe91a82ef
```

Returns the `MTConnectAssets` document for *Assets* e39d23ba-ef2d-11e6-b12c-28cfe91a82ef and e46d5256-ef2d-11e6-96aa-28cfe91a82ef.

Request for all the *Assets* in the *MTConnect Agent*:

```
url: http://example.com/assets
```

Returns all available *MTConnect Assets* in the *MTConnect Agent*. The *Agent* **MAY** return a limited set if there are too many *Asset* records. The *Assets* **MUST** be added to the beginning with the most recently modified *Asset*.

4.2.2 *Asset* for a Given Type

```
url: http://example.com/assets?type="CuttingTool"
```

Returns all available `CuttingTool` *Assets* from the *MTConnect Agent* of the type `CuttingTool`. The *Agent* **MAY** return a limited set if there are too many *Asset* records. The *Assets* **MUST** be added to the beginning with the most recently modified assets.

Request for all *Assets* of a given type in the *MTConnect Agent* up to a maximum count:

```
url: http://example.com/assets?type=CuttingTool&count=1000
```

Returns all available `CuttingTool` *Assets* from the *MTConnect Agent*. The *Agent* **MUST** return up to 1000 *Assets* beginning with the most recently modified *Assets* if they exist.

4.2.3 *Assets* Including `Removed` *Assets*

```
1.  url: http://example.com/assets?type=CuttingTool&removed=true
```

Returns all available `CuttingTool` *Assets* from the *MTConnect Agent*. With the removed flag, *Assets* that have been removed but are included in the result set.

4.2.4 *Assets* for a Piece of Equipment

If no `assetId` is provided with a general *Assets* request, it would be as follows:

```
1.  url: http://example.com/Mill123/assets
```

All *MTConnect Assets* will be provided for that piece of equipment (`Device`) up to the *MTConnect Agent's* maximum count or as specified with the `count` parameter. These *Assets* will be returned starting from the newest to oldest list.

Any of the previous constraints can also be applied to the request, for example, to get all the *CuttingTool* instances for a given piece of equipment:

```
1.  url: http://example.com/Mill123/asset/?type=CuttingTool&count=100
```

The previous request will get the newest 100 Cutting Tool Instance *Assets* from the *MTConnect Agent* for `Mill123`. Similarly:

```
1.  url: http://example.com/Mill123/asset/?type=CuttingToolArchetype
```

Will provide all Cutting Tool Archetype *Assets* with the `deviceUuid` of `Mill123`.

5 Extensions to *Part 2.0 – Devices Information Model*

This document will add the following data item types to support change notification when an *MTConnect Asset* is added or updated. The data item **MUST** be placed in the `DataItems` container associated with `Device`. The `Device` **MUST** be the piece of equipment that is supplying the asset data.

5.1 Data Item Types added for `EVENT` Category

Data Item type/subtype	Description
`ASSET_CHANGED`	The value of the **CDATA** for the event **MUST** be the `assetId` of the asset that has been added or changed. There will not be a separate message for new assets.
`ASSET_REMOVED`	The value of the **CDATA** for the event **MUST** be the `assetId` of the asset that has been removed. The asset will still be visible if requested with the `includeRemoved` parameter as described in the protocol section. When assets are removed they are not moved to the beginning of the most recently modified list.

5.1.1 `ASSET_CHANGED` Data Item Type

When an *MTConnect Asset* is added or modified, an `AssetChanged` event **MUST** be published to inform an application that new asset data is available. The application can request the new asset data from the piece of equipment at that time. Every time the asset data is modified an `AssetChanged` event will be published. Since the asset data is a complete electronic document, the system will publish a single `AssetChanged` event for the entire set of changes.

The asset data **MUST** remain constant until the `AssetChanged` event is published. Once it is published the data **MUST** change to reflect the new content at that instant. The timestamp of the asset will reflect the time the last change was made to the asset data.

5.1.2 `ASSET_REMOVED` Data Item Type

When an *MTConnect Asset* has been removed from an *MTConnect Agent*, or marked as removed, an `AssetRemoved` event **MUST** be generated in a similar way to the `AssetChanged` event. The CDATA of the `AssetRemoved` event **MUST** contain the `assetId` that was just removed.

Every time an *MTConnect Asset* is modified or added it will be moved to the beginning of the *asset buffer* and become the newest *Asset*. As the *asset buffer* fills up, the oldest *Asset* will be pushed out and its information will be removed. The MTConnect Standard does not specify the maximum size of the *asset buffer*, and if the implementation desires, permanent storage **MAY** be used to store the *Assets*. A value of `4,294,967,296` or 2^{32} can be given to indicate unlimited storage.

There is no requirement for persistent *Asset* storage. If the *MTConnect Agent* fails, all existing *MTConnect Assets* **MAY** be lost. It is the responsibility of the implementation to restore the lost *Asset* data and it is the responsibility of the application to persist the *Asset* data. The *MTConnect Agent* **MAY** make no guarantees about availability of *Asset* data after the *Agent* stops.

6 Extensions to *Part 3.0 – Streams Information Model*

The associated modifications **MUST** be added to *Part 3.0 – Streams Information Model* to add the following event to the `Events` in the streams.

6.1 `AssetChanged` Extension to `Events`

The `AssetChanged` element extends the base `Event` type XML data element defined in *Part 3.0 – Streams Information Model* and adds the `assetType` attribute to the base `Event`. This new `Event` will signal whenever a new *MTConnect Asset* is added or the existing definition of an *Asset* is updated. The `assetId` is provided as the CDATA value and can be used to request the *Asset* data from the *MTConnect Agent*.

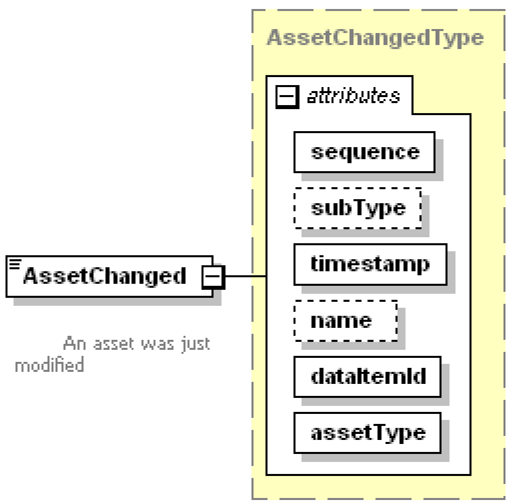


Figure 5: `AssetChanged` Schema

`AssetChanged` An *MTConnect Asset* has been added or modified. The **CDATA** for the `AssetChanged` element **MUST** be the `assetId` of the *Asset* that has been modified.

6.1.1 `AssetChanged` Attributes:

Attribute	Description	Occurrence
`assetType`	The type of asset that changed. `assetType` is a required attribute. Valid Data Values: -Cutting Tool	1

6.2 `AssetRemoved` Extension to `Events`

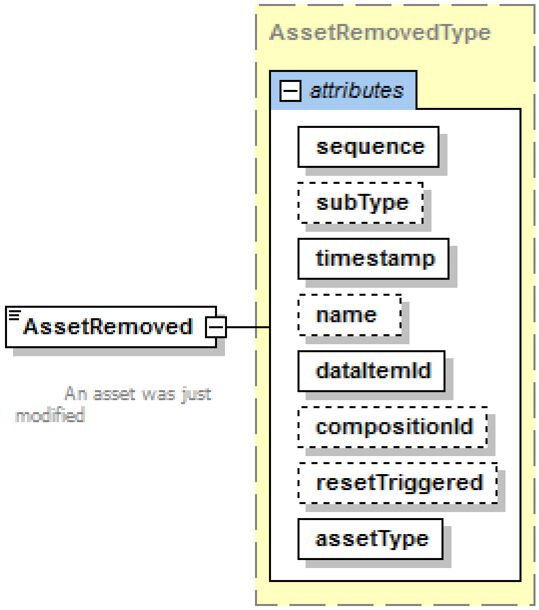


Figure 6: `AssetRemoved` Schema

`AssetRemoved` An *MTConnect Asset* has been removed. The **CDATA** for the `AssetRemoved` element **MUST** be the `assetId` of the *Asset* that has been removed.

6.2.1 `AssetRemoved` Attributes:

Attribute	Description	Occurrence
`assetType`	The type of asset that was removed. `assetType` is a required attribute. Valid Data Values: -Cutting Tool	1

The *MTConnect Asset* will still be available if requested if the removed=true argument is supplied. The `assetId` is provide as the CDATA value and can be used to request the *Asset* data from the *MTConnect Agent*.

Appendices

A. Bibliography

Engineering Industries Association. *EIA Standard - EIA-274-D*, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.

ISO TC 184/SC4/WG3 N1089. *ISO/DIS 10303-238*: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.

International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.

International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.

International Organization for Standardization. *ISO 6983/1* – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.

Electronic Industries Association. *ANSI/EIA-494-B-1992*, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.

National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.

International Organization for Standardization. *ISO 10303-11*: 1994, Industrial automation systems and integration Product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

International Organization for Standardization. *ISO 10303-21*: 1996, Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.

H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.

International Organization for Standardization. *ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature.* Geneva, Switzerland, 2001.

ASME B5.59-2 Version 9c: Data Specification for Properties of Machine Tools for Milling and Turning. 2005.

ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers. 2005.

OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.*

International Organization for Standardization. *ISO 13399*: *Cutting tool data representation and exchange*. Geneva, Switzerland, 2000.

MTConnect® Standard
Part 4.1 – Cutting Tools

Version 1.4.0

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

MTConnect® Specification and Materials

Table of Contents

Table of Figures

1 Purpose of This Document

This document, *Part 4.1 – Cutting Tools* of the MTConnect® Standard, establishes the rules and terminology to be used by designers to describe the function and operation of Cutting Tools used within manufacturing and to define the data that is provided by an *MTConnect Agent* from a piece of equipment. This part of the Standard also defines the structure for the XML document that is returned from an *MTConnect Agent* in response to a `Probe` request.

The data associated with these Cutting Tools will be retrieved from multiple sources that are responsible for providing their knowledge of an *MTConnect Asset*.

2 Terminology and Conventions

Refer to *Section 2* of *Part 1 - Overview and Functionality* for a dictionary of terms, reserved language, and document conventions used in the MTConnect Standard.

3 Cutting Tool and Cutting Tool Archetype

There are two *Information Models* used to represent a Cutting Tool, a `CuttingToolArchetype` and a `CuttingTool`. The `CuttingToolArchetype` represent the static Cutting Tool geometries and nominal values as one would expect from a tool catalog and the `CuttingTool` represents the use or application of the tool on the shop floor with actual measured values and process data. In Version 1.3.0 of the MTConnect Standard it was decided to separate out these two concerns since not all pieces of equipment will have access to both pieces of information. In this way, a generic definition of the Cutting Tool can coexist with a specific assembly information model with minimal redundancy of data.

3.1 XML Schema Structure for `CuttingTool` and `CuttingToolArchetype`

The following figure shows the XML schema that applies to both the `CuttingTool` *Information Model* and the `CuttingToolArchetype` *Information Model*.

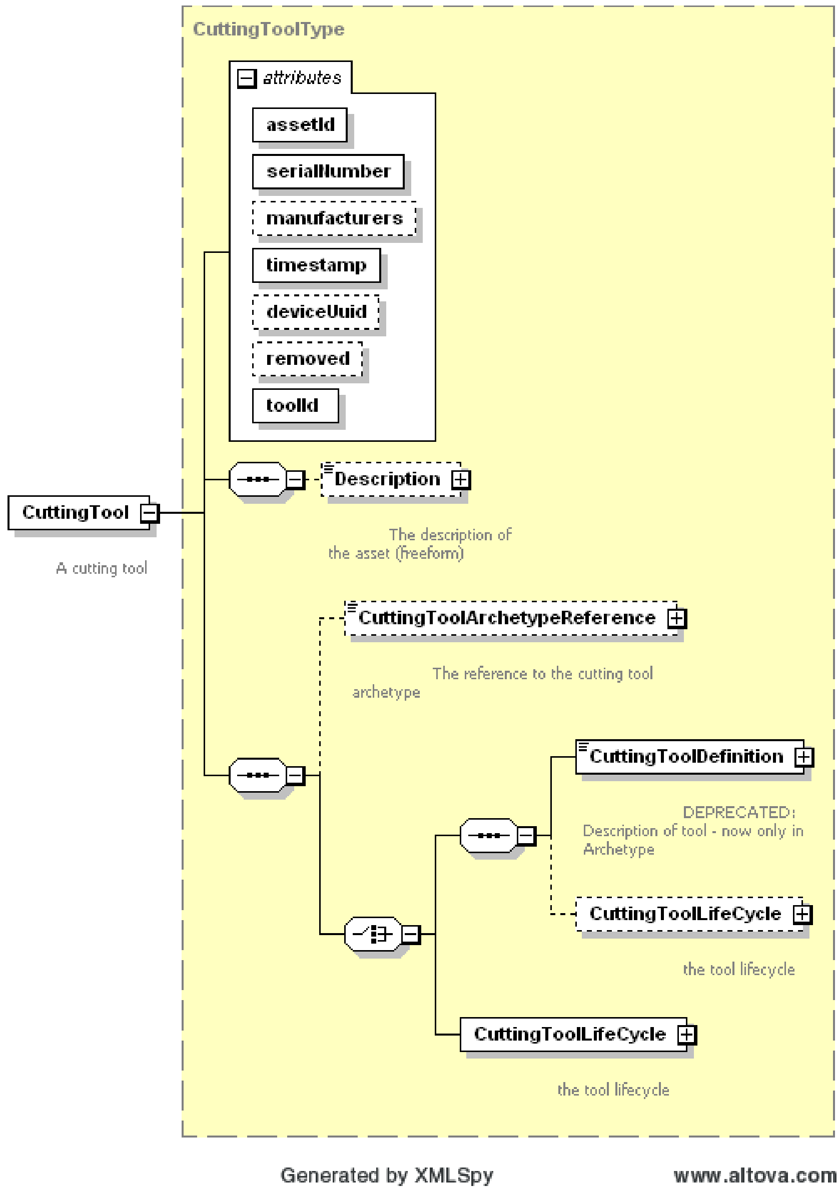


Figure 1: `CuttingTool` Schema

Note: The use of the XML element `CuttingToolDefinition` has been **DEPRECATED** in the `CuttingTool` schema, but remains in the `CuttingToolArchetype` schema.

The following sections contain the definitions of `CuttingTool` and `CuttingToolArchetype` and describe their unique components. The following are the common entities for both elements.

3.2 Common Attributes for `CuttingTool` and `CuttingToolArchetype`

Attribute	Description	Occurrence
`timestamp`	The time this *MTConnect Asset* was last modified. Always given in UTC. The timestamp **MUST** be provided in UTC (Universal Time Coordinate, also known as GMT). This is the time the *Asset* data was last modified. `timestamp` is a required attribute.	1
`assetId`	The unique identifier of the instance of this tool. This will be the same as the `toolId` and `serialNumber` in most cases. The `assetId` **SHOULD** be the combination of the `toolId` and `serialNumber` as in `toolId.serialNumber` or an equivalent implementation dependent identification scheme. `assetId` is a required attribute. `assetId` is a permanent identifier that will be associated with an *MTConnect Asset* for its entire life.	1
`serialNumber`	The unique identifier for this assembly. This is defined as an XML string type and is implementation dependent. `serialNumber` is a required attribute.	1
`toolId`	The identifier for a class of Cutting Tools. This is defined as an XML string type and is implementation dependent. `toolId` is a required attribute.	1
`deviceUuid`	The piece of equipment UUID that supplied this data. This optional element references to the UUID attribute given in the `Device` element. This can be any series of numbers and letters as defined by the XML type NMTOKEN.	1
`manufacturers`	An optional attribute referring to the manufacturer(s) of this Cutting Tool, for this element, this will reference the Tool Item and Adaptive Items specifically. The Cutting Items manufacturers' will be an attribute of the `CuttingItem` elements. The representation will be a comma (`,`) delimited list of manufacturer names. This can be any series of numbers and letters as defined by the XML type `string`.	0..1

Attribute	Description	Occurrence
`removed`	This is an indicator that the Cutting Tool has been removed from the piece of equipment. `removed` is an optional attribute. If the *MTConnect Asset* is marked as removed, it will not be visible to the client application unless the `includeRemoved=true` parameter is provided in the URL. If this attribute is not present it **MUST** be assumed to be false. The value is an `xsi:boolean` type and **MUST** be `true` or `false`.	0..1

3.3 Common Elements for `CuttingTool` and `CuttingToolArchetype`

Element	Description	Occurrence
`Description`	An element that can contain any descriptive content. This can contain configuration information and manufacturer specific details. This element is defined to contain mixed content and XML elements can be added to extend the descriptive semantics of the MTConnect Standard.	0..1

3.3.1 `Description` Element for `CuttingTool` and `CuttingToolArchetype`

`Description` **MAY** contain mixed content, meaning that an additional XML element or plain text may be provided as part of the content of the description tag. Currently `Description` contains no attributes.

4 `CuttingToolArchetype` *Information Model*

The `CuttingToolArchetype` *Information Model* will have the identical structure as the `CuttingTool` *Information Model* illustrated in *Figure 1*, except for a few entities. The `CuttingTool` will no longer carry the `CuttingToolDefinition`, this **MUST** only appear in the `CuttingToolArchetype`. The `CuttingToolArchetype` **MUST NOT** have measured values and **MUST NOT** have any of the following items: `CutterStatus`, `ToolLife` values, `Location`, or a `ReconditionCount`.

MTConnect Standard will adopt the ISO 13399 structure when formulating the vocabulary for Cutting Tool geometries and structure to be represented in the `CuttingToolArchetype`. The nominal values provided in the `CuttingToolLifeCycle` section are only concerned with two aspects of the Cutting Tool, the Cutting Tool and the Cutting Item. The Tool Item, Adaptive Item, and Assembly Item will only be covered in the `CuttingToolDefinition` section of this document since this section contains the full ISO 13399 information about a Cutting Tool.

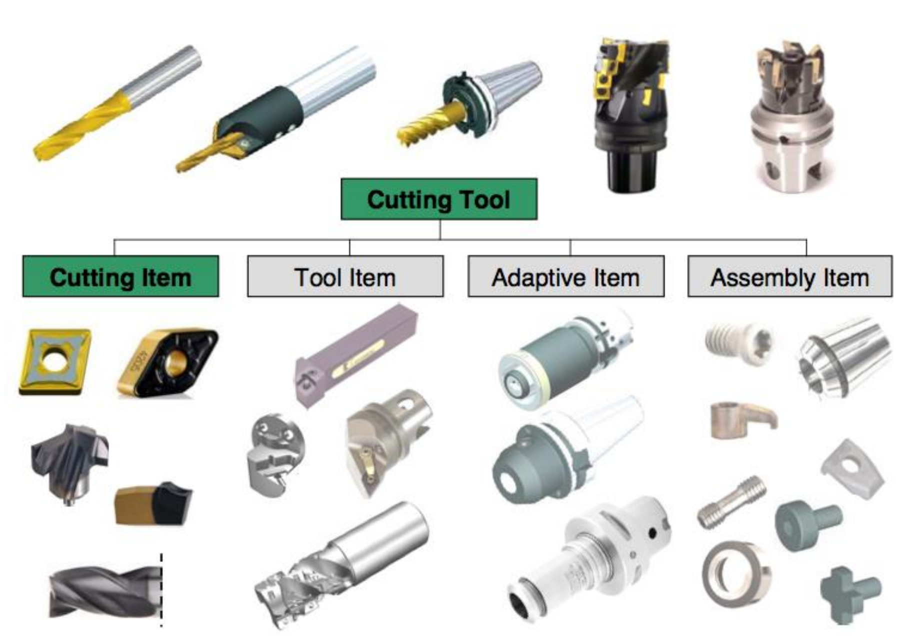


Figure 2: Cutting Tool Parts

The previous diagram illustrates the parts of a Cutting Tool. The Cutting Tool is the aggregate of all the components and the Cutting Item is the part of the tool that removes the material from the workpiece. These are the primary focus of the MTConnect Standard.

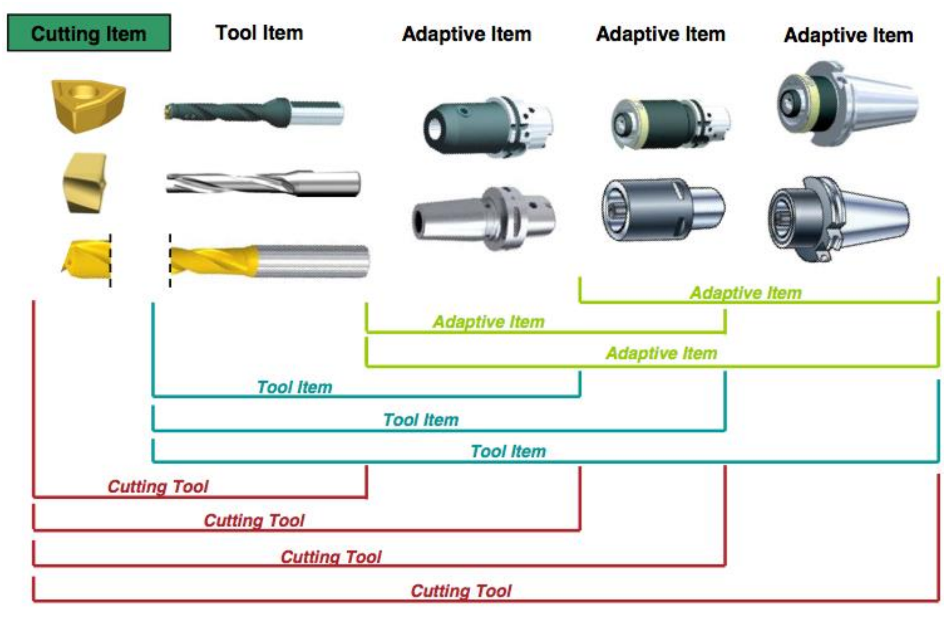


Figure 3: Cutting Tool Composition

Figure 3 provides another view of the composition of a Cutting Tool. The Adaptive Items and Tool Items will be used for measurements, but will not be modeled as separate entities. When we are referencing the Cutting Tool we are referring to the entirety of the assembly and when we provide data regarding the Cutting Item we are referencing each individual item as illustrated on the left of the previous diagram.

Figures 4 and *5* further illustrates the components of the Cutting Tool. As we compose the Tool Item, Cutting Item, Adaptive Item, we get a Cutting Tool. The Tool Item, Adaptive Item, and Assembly Item will only be in the `CuttingToolDefinition` section that will contain the full ISO 13399 information.

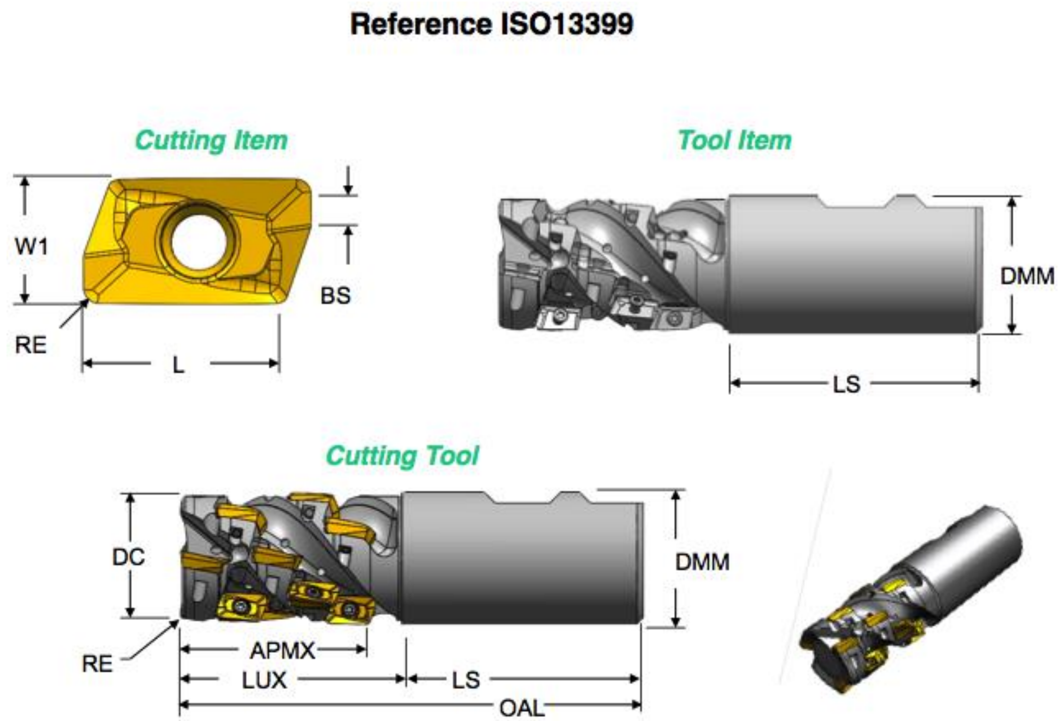


Figure 4: Cutting Tool, Tool Item and Cutting Item

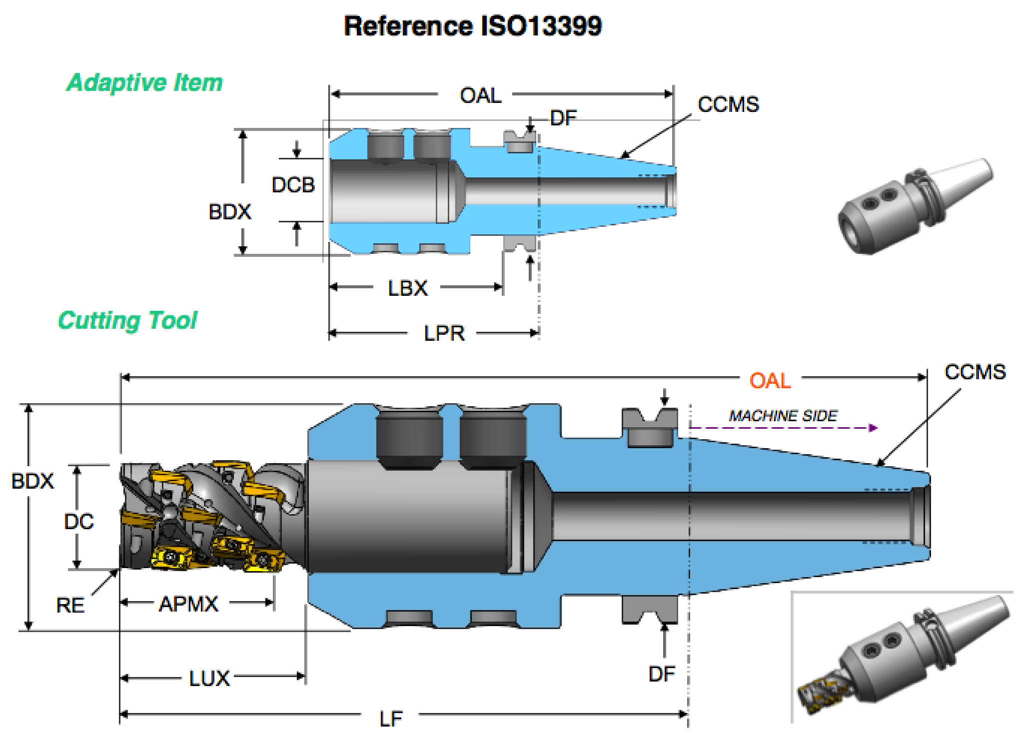


Figure 5: Cutting Tool, Tool Item and Cutting Item

The above diagrams use the ISO 13399 codes for each of the measurements. These codes will be translated into the MTConnect Standard vocabulary as illustrated below. The measurements will have a maximum, minimum, and nominal value representing the tolerance of allowable values for this dimension. See below for a full discussion.

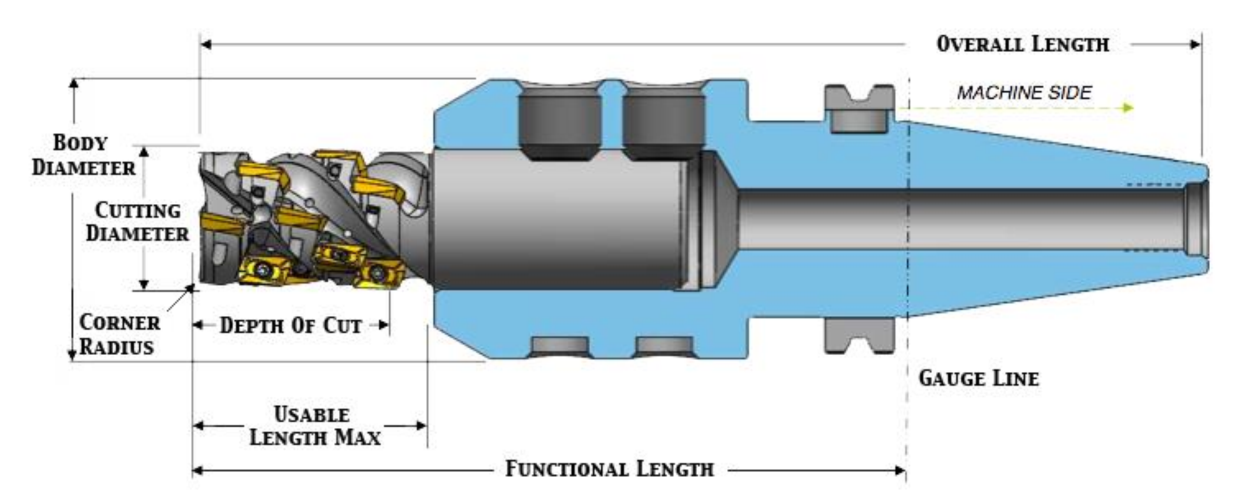


Figure 6: Cutting Tool Measurements

The MTConnect Standard will not define the entire geometry of the Cutting Tool, but will provide the information necessary to use the tool in the manufacturing process. Additional information can be added to the definition of the Cutting Tool by means of schema extensions.

Additional diagrams will reference these dimensions by their codes that will be defined in the measurement tables. The codes are consistent with the codes used in ISO 13399 and have been standardized. MTConnect Standard will use the full text name for clarity in the XML document.

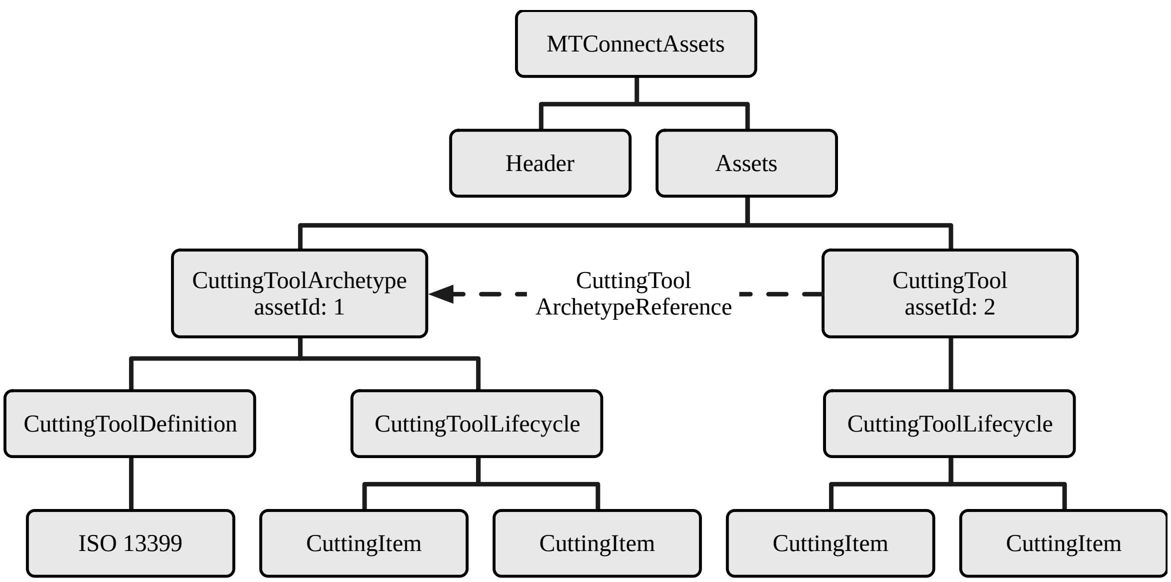


Figure 7: Cutting Tool Asset Structure

The structure of the MTConnectAssets header is defined in *Part 1 - Overview and Fundamentals* of the Standard. A finite number of *MTConnect Assets* will be stored in the M*TConnect Agent*. This finite number is implementation specific and will depend on memory and storage constraints. The standard will not prescribe the number or capacity requirements for an implementation.

4.1 Attributes for CuttingToolArchetype

Refer to *Section 3.2* for a full description of the attributes for CuttingToolArchetype *Information Model*.

4.2 Elements for CuttingToolArchetype

The elements associated with CuttingToolArchetype are given below. Each element will be described in more detail below and any possible values will be presented with full definitions. The elements **MUST** be provided in the following order as prescribed by XML. At least one of CuttingToolDefinition or CuttingToolLifeCycle **MUST** be supplied.

Element	Description	Occurrence
Description	An element that can contain any descriptive content. This can contain configuration information and manufacturer specific details. This element is defined to contain mixed content and XML elements can be added to extend the descriptive semantics of MTConnect Standard.	0..1
CuttingToolDefinition	Reference to an ISO 13399.	0..1
CuttingToolLifeCycle	Data regarding the use of this tool. The archetype will only contain nominal values.	0..1

4.2.1 `CuttingToolDefinition` Element for `CuttingToolArchetype`

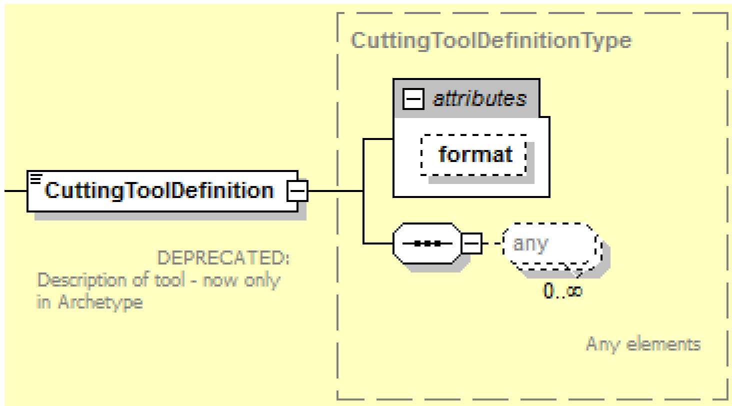


Figure 8: `CuttingToolDefinition` Schema

The `CuttingToolDefinition` contains the detailed structure of the Cutting Tool. The information contained in this element will be static during its lifecycle. Currently we are referring to the external ISO 13399 standard to provide the complete definition and composition of the Cutting Tool as defined in *Section 6.1* of this document.

4.2.1.1 Attributes for `CuttingToolDefinition`

Attribute	Description	Occurrence
`format`	Identifies the expected representation of the enclosed data. `format` is an optional attribute. Valid values of `format` are – EXPRESS, XML, TEXT, or UNDEFINED. If `format` is not specified, the assumed format is XML.	0..1

4.2.1.1.1 `format` Attribute for `CuttingToolDefinition`

The `format` attribute describes the expected representation of the enclosed data. If no value is given, the assumed format will be `XML`.

Value	Description
`XML`	The default value for the definition. The content will be an XML document.
`EXPRESS`	The document will confirm to the ISO 10303 Part 21 standard.
`TEXT`	The document will be a text representation of the tool data.
`UNDEFINED`	The document will be provided in an undefined format.

4.2.1.2 Elements for `CuttingToolDefinition`

The only acceptable Cutting Tool definition at present is defined by the ISO 13399 standard. Additional formats **MAY** be considered in the future.

4.2.1.3 ISO 13399 Standard

The ISO 13399 data **MUST** be presented in either XML (ISO 10303-28) or EXPRESS format (ISO 10303-21). An XML schema will be preferred as this will allow for easier integration with the MTConnect Standard XML tools. EXPRESS will also be supported, but software tools will need to be provided or made available for handling this data representation.

There will be the root element of the ISO13399 document when XML is used. When EXPRESS is used the XML element will be replaced by the text representation.

4.2.2 `CuttingToolLifeCycle` Element for `CuttingToolArchetype`

Refer to *Section 6 – Common Entity `CuttingToolLifeCycle`* for a complete description of `CuttingToolLifeCycle` element.

5 CuttingTool *Information Model*

The CuttingTool *Information Model* illustrated in *Figure 1* has the identical structure as the CuttingToolArchetype *Information Model* except for the XML element CuttingToolDefinition that has been **DEPRECATED** in the CuttingTool schema.

5.1 Attributes for CuttingTool

Refer to *Section 3.2* for a full description of the attributes for CuttingTool *Information Model*.

5.2 Elements for CuttingTool

The elements associated with CuttingTool are given below. The elements **MUST** be provided in the following order as prescribed by XML.

Element	Description	Occurrence
Description	An element that can contain any descriptive content. This can contain configuration information and manufacturer specific details. This element is defined to contain mixed content and XML elements can be added to extend the descriptive semantics of MTConnect Standard.	0..1
~~CuttingToolDefinition~~	**DEPRECATED** for CuttingTool in Version 1.3.0. ~~Reference to an ISO 13399.~~	~~0..1~~
CuttingToolLifeCycle	Data regarding the use of this tool.	0..1
CuttingToolArchetypeReference	The content of this XML element is the assetId of the CuttingToolArchetype document. It **MAY** also contain a source attribute that gives the URL of the archetype data as well.	0..1

5.2.1 CuttingToolLifeCycle Elements for CuttingTool Only

The following CuttingToolLifeCycle elements are used only in the CuttingTool *Information Model* and are not part of the CuttingToolArchetype *Information Model*. Refer to *Section 6* for a complete description of the remaining elements for CuttingToolLifeCycle that are common in both *Information Models*. Refer also to the CuttingToolLifeCycle schema illustrated in *Figure 12*.

5.2.1.1 `CutterStatus` Element for `CuttingToolLifeCycle`

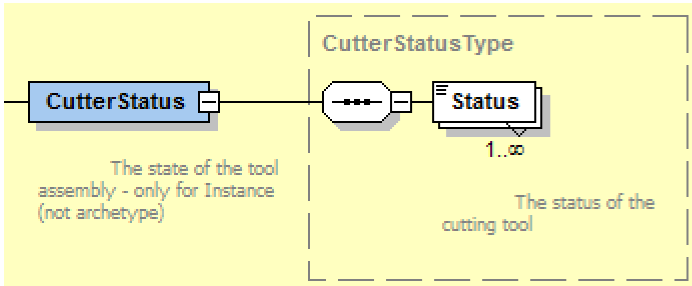


Figure 9: `CutterStatus` Schema

The elements of the `CutterStatus` element can be a combined set of `Status` elements. The MTConnect Standard allows any set of statuses to be combined, but only certain combinations make sense. A Cutting Tool **SHOULD** not be both `NEW` and `USED` at the same time. There are no rules in the schema to enforce this, but this is left to the implementer. The following combinations **MUST NOT** occur:

- `NEW` **MUST NOT** be used with `USED`, `RECONDITIONED`, or `EXPIRED`.
- `UNKNOWN` **MUST NOT** be used with any other status.
- `ALLOCATED` and `UNALLOCATED` **MUST NOT** be used together.
- `AVAILABLE` and `UNAVAILABLE` **MUST NOT** be used together.
- If the tool is `EXPIRED`, `BROKEN`, or `NOT_REGISTERED` it **MUST NOT** be `AVAILABLE`.
- All other combinations are allowed.

Element	Description	Occurrence
`Status`	The status of the Cutting Tool. There can be multiple `Status` elements.	1..INF

5.2.1.1.1 `Status` Element for `CutterStatus`

One of the values for the status of the Cutting Tool.

Value	Description
`NEW`	A new tool that has not been used or first use. Marks the start of the tool history.
`AVAILABLE`	Indicates the tool is available for use. If this is not present, the tool is currently not ready to be used.

Value	Description
`UNAVAILABLE`	Indicates the tool is unavailable for use in metal removal. If this is not present, the tool is currently not ready to be used.
`ALLOCATED`	Indicates if this tool is has been committed to a piece of equipment for use and is not available for use in any other piece of equipment. If this is not present, this tool has not been allocated for this piece of equipment and can be used by another piece of equipment.
`UNALLOCATED`	Indicates this Cutting Tool has not been committed to a process and can be allocated.
`MEASURED`	The tool has been measured.
`RECONDITIONED`	The Cutting Tool has been reconditioned. See `ReconditionCount` for the number of times this cutter has been reconditioned.
`USED`	The Cutting Tool is in process and has remaining tool life.
`EXPIRED`	The Cutting Tool has reached the end of its useful life.
`BROKEN`	Premature tool failure.
`NOT_REGISTERED`	This Cutting Tool cannot be used until it is entered into the system.
`UNKNOWN`	The Cutting Tool is an indeterminate state. This is the default value.

5.2.1.2 `ToolLife` Element for `CuttingToolLifeCycle`

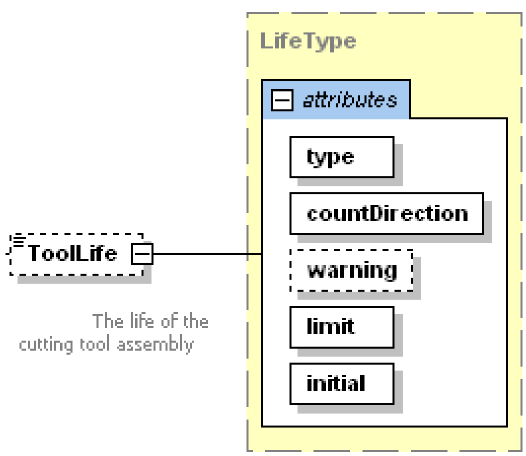


Figure 10: `ToolLife` Schema

The value is the current value for the tool life. The value **MUST** be a number. `ToolLife` is an option element which can have three types, either minutes for time based, part count for parts based, or wear based using a distance measure. One tool life element can appear for each type, but there cannot be two entries of the same type. Additional types can be added in the future.

5.2.1.2.1 Attributes for `ToolLife`

`ToolLife` has the following attributes that can be used to indicate the behavior of the tool life management mechanism.

Attribute	Description	Occurrence
`type`	The type of tool life being accumulated. `MINUTES`, `PART_COUNT`, or `WEAR`. `type` is a required attribute.	1
`countDirection`	Indicates if the tool life counts from zero to maximum or maximum to zero. The value **MUST** be one of `UP` or `DOWN`. `countDirection` is a required attribute.	1
`warning`	The point at which a tool life warning will be raised. `warning` is an optional attribute.	0..1
`limit`	The end of life limit for this tool. If the `countDirection` is `DOWN`, the point at which this tool should be expired, usually zero. If the `countDirection` is `UP`, this is the upper limit for which this tool should be expired. `limit` is a required attribute.	0..1
`initial`	The initial life of the tool when it is new. `initial` is a required attribute.	0..1

5.2.1.2.2 `type` Attribute for `ToolLife`

The value of `type` must be one of the following:

Value	Description
`MINUTES`	The tool life measured in minutes. All units for minimum, maximum, and nominal **MUST** be provided in minutes.
`PART_COUNT`	The tool life measured in parts. All units for minimum, maximum, and nominal **MUST** be provided as the number of parts.

Value	Description
WEAR	The tool life measured in tool wear. Wear **MUST** be provided in millimeters as an offset to nominal. All units for minimum, maximum, and nominal **MUST** be given as millimeter offsets as well. The standard will only consider dimensional wear at this time.

5.2.1.2.3 countDirection Attribute for ToolLife

The value of type must be one of the following:

Value	Description
DOWN	The tool life counts down from the maximum to zero.
UP	The tool life counts up from zero to the maximum.

5.2.1.3 Location Element for CuttingToolLifeCycle

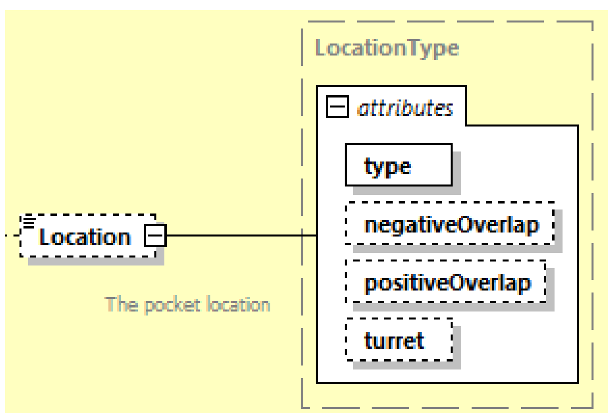


Figure 11: Location Schema

Location element identifies the specific location where a tool resides in a piece of equipment tool storage or in a tool crib. This can be any series of numbers and letters as defined by the XML type NMTOKEN. When a POT or STATION type is used, the value **MUST** be a numeric value. If a negativeOverlap or the positiveOverlap is provided, the tool reserves additional locations on either side, otherwise if they are not given, no additional locations are required for this tool. If the pot occupies the first or last location, a rollover to the beginning or the end of the index-able values may occur. For example, if there are 64 pots and the tool is in pot 64 with a positiveOverlap of 1, the first pot **MAY** be occupied as well.

5.2.1.3.1 Attributes for `Location`

Attribute	Description	Occurrence
`type`	The type of location being identified. `type` **MUST** be one of `POT`, `STATION`, or `CRIB`. `type` is a required attribute.	1
`positiveOverlap`	The number of locations at higher index value from this location. `positiveOverlap` is an optional attribute.	0..1
`negativeOverlap`	The number of location at lower index values from this location. `negativeOverlap` is an optional attribute.	0..1

5.2.1.3.2 `Type` Attribute for `Location`

The type of location being identified.

Value	Description
`POT`	The number of the pot in the tool handling system.
`STATION`	The tool location in a horizontal turning machine.
`CRIB`	The location with regard to a tool crib.

5.2.1.3.3 `positiveOverlap` Attribute for `Location`

The number of locations at higher index values that the Cutting Tool occupies due to interference. The value **MUST** be an integer. If not provided it is assumed to be 0.

5.2.1.3.4 `negativeOverlap` Attribute for `Location`

The number of locations at lower index values that the Cutting Tool occupies due to interference. The value **MUST** be an integer. If not provided it is not assumed to be 0.

The tool number assigned in the part program and is used for cross referencing this tool information with the process parameters. The value **MUST** be an integer.

5.2.1.4 `ReconditionCount` Element for `CuttingToolLifeCycle`

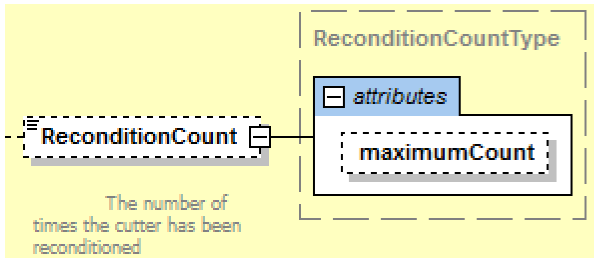


Figure 12: `ReconditionCount` Schema

This element **MUST** contain an integer value as the CDATA that represents the number of times the cutter has been reconditioned.

5.2.1.4.1 Attributes for `ReconditionCount`

Attribute	Description	Occurrence
`maximumCount`	The maximum number of times this tool may be reconditioned. `maximumCount` is an optional attribute.	0..1

5.2.2 `CuttingToolArchetypeReference Element for CuttingTool`

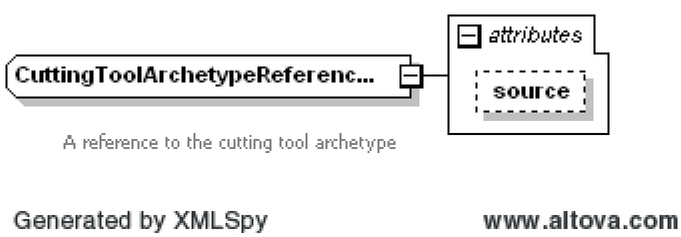


Figure 13: `CuttingToolArchetypeReference` Schema

This optional element references another *MTConnect Asset* document providing the static geometries and nominal values for all the measurements. This reduces the amount of data duplication as well as providing a mechanism for asset definitions to be provided before complete measurement has occurred.

5.2.2.1 `Source` Attribute for `CuttingToolArchetypeReference`

Attribute	Description	Occurrence
`Source`	The URL of the `CuttingToolArchetype` *Information Model*. This **MUST** be a fully qualified URL as in http://example.com/asset/A213155	0..1

6 Common Entity `CuttingToolLifeCycle`

6.1 `CuttingToolLifeCycle`

The life cycle refers to the data pertaining to the application or the use of the tool. This data is provided by various pieces of equipment (i.e. machine tool, presetter) and statistical process control applications. Life cycle data will not remain static, but will change periodically when a tool is used or measured. The life cycle has three conceptual parts; tool and Cutting Item identity, properties, and measurements. A measurement is defined as a constrained value that is reported in defined units and as a W3C floating point format.

The `CuttingToolLifeCycle` contains data for the entire tool assembly. The specific Cutting Items that are part of the `CuttingToolLifeCycle` are contained in the `CuttingItems` element. Each Cutting Item has similar properties as the assembly; identity, properties, and measurements.

The units for all measurements have been predefined in the MTConnect Standard and will be consistent with *Part 2 – Devices Information Model* and *Part 3 – Streams Information Model* of the Standard. This means that all lengths and distances will be given in millimeters and all angular measures will be given in degrees. Quantities like `ProcessSpindleSpeed` will be given in RPM, the same as the `RotaryVelocity` in *Part 3 – Streams Information Model*.

6.1.1 XML Schema Structure for `CuttingToolLifeCycle`

The `CuttingToolLifeCycle` schema shown in *Figure 12* is used in both the `CuttingToolArchetype` and `CuttingTool` *Information Models*. The only difference is that the elements `CutterStatus`, `ToolLife`, `Location`, and `ReconditionCount` are used only in the `CuttingTool` *Information Model*.

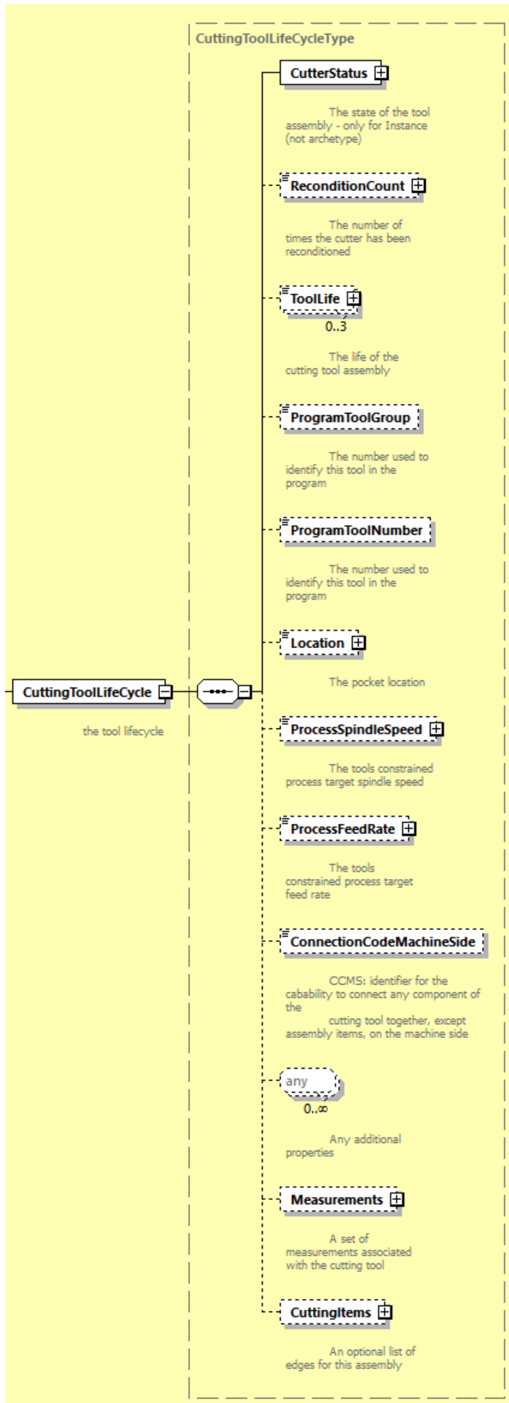


Figure 14: `CuttingToolLifeCycle` Schema

6.2 Elements for `CuttingToolLifeCycle`

The elements associated with this Cutting Tool are given below. Each element will be described in more detail below and any possible values will be presented with full definitions. The elements **MUST** be provided in the following order as prescribed by XML.

Element	Description	Occurrence
`CutterStatus`	The status of this assembly. `CutterStatus` can be one of the following values: `NEW, AVAILABLE, UNAVAILABLE, ALLOCATED, UNALLOCATED, MEASURED, RECONDITIONED, NOT_REGISTERED, USED, EXPIRED, BROKEN, or UNKNOWN.` **MUST** only be used in the `CuttingTool` *Information Model*.	1
`ReconditionCount`	The number of times this cutter has been reconditioned. **MUST** only be used in the `CuttingTool` *Information Model.*	0..1
`ToolLife`	The Cutting Tool life as related to this assembly. **MUST** only be used in the `CuttingTool` *Information Model.*	0..1
`Location`	The Pot or Spindle this tool currently resides in. **MUST** only be used in the `CuttingTool` *Information Model.*	0..1
`ProgramToolGroup`	The tool group this tool is assigned in the part program.	0..1
`ProgramToolNumber`	The number of the tool as referenced in the part program.	0..1
`ProcessSpindleSpeed`	The constrained process spindle speed for this tool.	0..1
`ProcessFeedRate`	The constrained process feed rate for this tool in mm/s.	0..1
`ConnectionCodeMachineSide`	Identifier for the capability to connect any component of the Cutting Tool together, except Assembly Items, on the machine side. Code: CCMS	0..1
`Measurements`	A collection of measurements for the tool assembly.	0..1
`CuttingItems`	An optional set of individual Cutting Items.	0..1
`xs:any`	Any additional properties not in the current document model. **MUST** be in separate XML namespace.	0..n

6.2.1 `ProgramToolGroup` Element for `CuttingToolLifeCycle`

The optional identifier for the group of Cutting Tools when multiple tools can be used interchangeably. This is defined as an XML string type and is implementation dependent.

6.2.2 `ProgramToolNumber` Element for `CuttingToolLifeCycle`

The tool number assigned in the part program and is used for cross referencing this tool information with the process parameters. The value **MUST** be an integer.

6.2.3 `ProcessSpindleSpeed` Element for `CuttingToolLifeCycle`

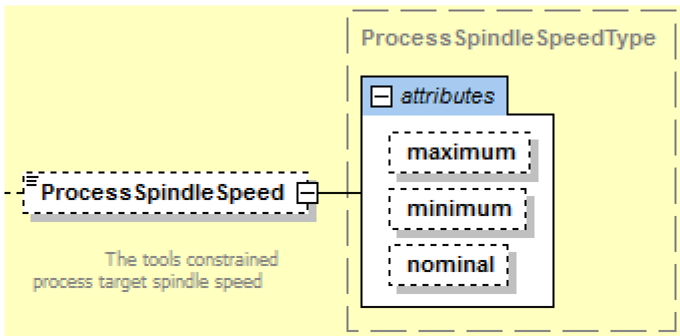


Figure 15: `ProcessSpindleSpeed` Schema

The `ProcessSpindleSpeed` **MUST** be specified in revolutions/minute (RPM). The CDATA **MAY** contain the nominal process target spindle speed if available. The maximum and minimum speeds **MAY** be provided as attributes. If `ProcessSpindleSpeed` is provided, at least one value of `maximum`, `nominal`, or `minimum` **MUST** be specified.

6.2.3.1 Attributes for `ProcessSpindleSpeed`

Attribute	Description	Occurrence
`maximum`	The upper bound for the tool's target spindle speed. `maximum` is an optional attribute.	0..1
`minimum`	The lower bound for the tools spindle speed. `minimum` is an optional attribute.	0..1
`nominal`	The nominal speed the tool is designed to operate at. `nominal` is an optional attribute.	0..1

6.2.4 `ProcessFeedRate` Element for `CuttingToolLifeCycle`

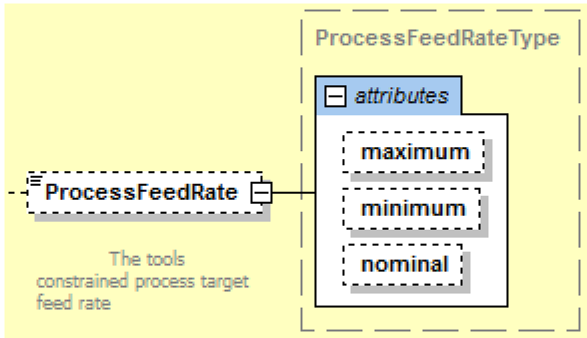


Figure 16: `ProcessFeedRate` Schema

The `ProcessFeedRate` **MUST** be specified in millimeters/second (mm/s). The CDATA **MAY** contain the nominal process target feed rate if available. The maximum and minimum rates **MAY** be provided as attributes. If `ProcessFeedRate` is provided, at least one value of `maximum`, `nominal`, or `minimum` **MUST** be specified.

6.2.4.1 Attributes for `ProcessFeedRate`

Attribute	Description	Occurrence
`maximum`	The upper bound for the tool's process target feedrate. `maximum` is an optional attribute.	0..1
`minimum`	The lower bound for the tools feedrate. `minimum` is an optional attribute.	0..1
`nominal`	The nominal feedrate the tool is designed to operate at. `nominal` is and optional attribute.	0..1

6.2.5 `ConnectionCodeMachineSide` Element for `CuttingToolLifeCycle`

This is an optional identifier for implementation specific connection component of the Cutting Tool on the machine side. Code: `CCMS`. The CDATA **MAY** be any valid string according to the referenced connection code standards.

6.2.6 `xs:any` Element for `CuttingToolLifeCycle`

Utilizing the new capability in XMLSchema 1.1, we are now able to add extension points where an additional element can be added to the document without being part of a substitution group. The new elements have the restriction that they **MUST NOT** be part of the MTConnect namespace and **MUST NOT** be one of the predefined elements mentioned above.

This will allow users to add additional properties to the Cutting Tool without having to change the definition of the Cutting Tool or modify the standard. We will begin making use of this capability in Version 1.3 of MTConnect Standard which will necessitate upgrading to Version 1.1 of XMLSchema.

6.2.7 `Measurements` Element for `CuttingToolLifeCycle`

The `Measurements` element is a collection of one or more constrained scalar values associated with this Cutting Tool. The contents **MUST** be a subtype of `CommonMeasurement` or `AssemblyMeasurement`. The following section will define the abstract `Measurement` type used in both `CuttingToolLifeCycle` and `CuttingItem`. This section will then describe the `AssemblyMeasurement` types. The `CuttingItemMeasurement` types will be described at the end of the `CuttingItem` section.

A measurement is specific to a process and a machine tool at a particular shop. The tool zero reference point or gauge line will be different depending on the particular implementation and will be assumed to be consistent within the shop. MTConnect Standard does not standardize the manufacturing process or the definition of the zero point.

6.2.8 `Measurement`

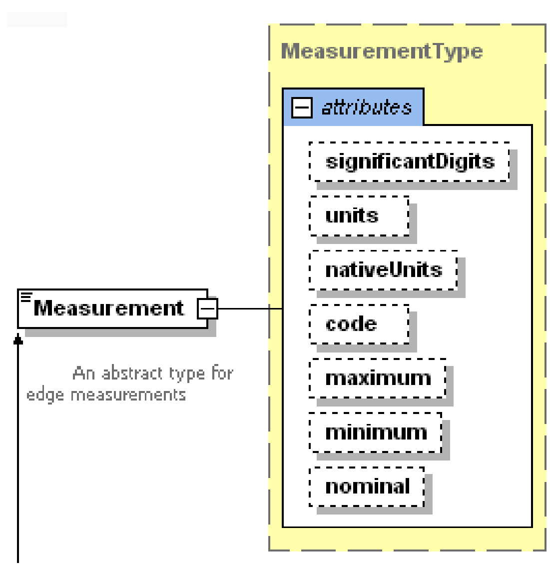


Figure 17: `Measurement` Schema

A measurement **MUST** be a scalar floating-point value that **MAY** be constrained to a maximum and minimum value. Since the `CuttingToolLifeCycle`'s main responsibility is to track aspects of the tool that change over its use in the shop, MTConnect represents the current value of the measurement **MUST** be in the `CDATA` (text between the start and end element) as the most current valid value.

The `minimum` and `maximum` **MAY** be supplied if they are known or relevant to the measurement. A `nominal` value **MAY** be provided to show the reference value for this measurement.

There are three subtypes of `Measurement`: `CommonMeasurement`, `AssemblyMeasurement`, and `CuttingItemMeasurement`. These abstract types **MUST NOT** appear in an `MTConnectAssets` document, but are used in the schema as a way to separate which measurements **MAY** appear in the different sections of the document. Only subtypes that have extended these types **MAY** appear in the `MTConnectAssets` XML.

Measurements in the `CuttingToolLifeCycle` section **MUST** refer to the entire assembly and not to an individual Cutting Item. Cutting Item measurements **MUST** be located in the measurements associated with the individual Cutting Item.

Measurements **MAY** provide an optional `units` attribute to reinforce the given units. The units **MUST** always be given in the predefined MTConnect units. If `units` are provided, they are only for documentation purposes. `nativeUnits` **MAY** optionally be provided to indicate the original units provided for the measurements.

6.2.8.1 Attributes for `Measurement`

Attribute	Description	Occurrence
`code`	A shop specific code for this measurement. ISO 13399 codes **MAY** be used for these codes as well. `code` is an optional attribute.	0..1
`maximum`	The maximum value for this measurement. Exceeding this value would indicate the tool is not usable. `maximum` is an optional attribute.	0..1
`minimum`	The minimum value for this measurement. Exceeding this value would indicate the tool is not usable. `minimum` is an optional attribute.	0..1
`nominal`	The as advertised value for this measurement. `nominal` is an optional attribute.	0..1

Attribute	Description	Occurrence
`significantDigits`	The number of significant digits in the reported value. This is used by applications to determine accuracy of values. This **MAY** be specified for all numeric values. `significantDigits` is an optional attribute.	0..1
`units`	The units for the measurements. MTConnect Standard defines all the units for each measurement, so this is mainly for documentation sake. See *MTConnect Part 2 – Devices Information Model Section 7.2.2.5* for the full list of units. `units` is an optional attribute.	0..1
`nativeUnits`	The units the measurement was originally recorded in. This is only necessary if they differ from units. See *MTConnect Part 2 – Devices Information Model Section 7.2.2.6* for the full list of units. `nativeUnits` is an optional attribute.	0..1

6.2.8.2 `Measurement` Subtypes for `CuttingToolLifeCycle`

These measurements for `CuttingTool` are specific to the entire assembly and **MUST NOT** be used for the measurement pertaining to a `CuttingItem`. The following diagram will be used to reference the assembly specific measurements.

The Code in the following table will refer to the acronyms in the diagrams. We will be referring to many diagrams to disambiguate all measurements of the `CuttingTool` and `CuttingItem`.

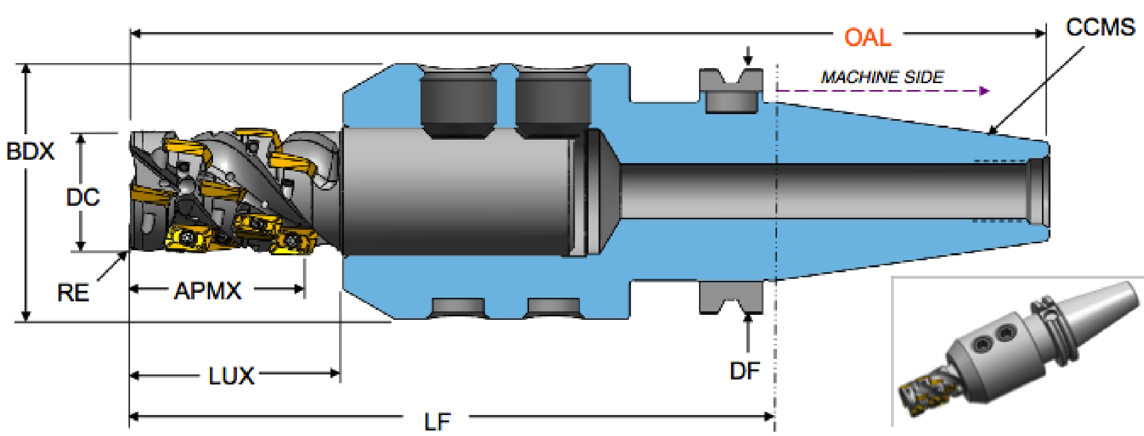


Figure 18: Cutting Tool Measurement Diagram 1
(Cutting Item, Tool Item, and Adaptive Item – ISO 13399)

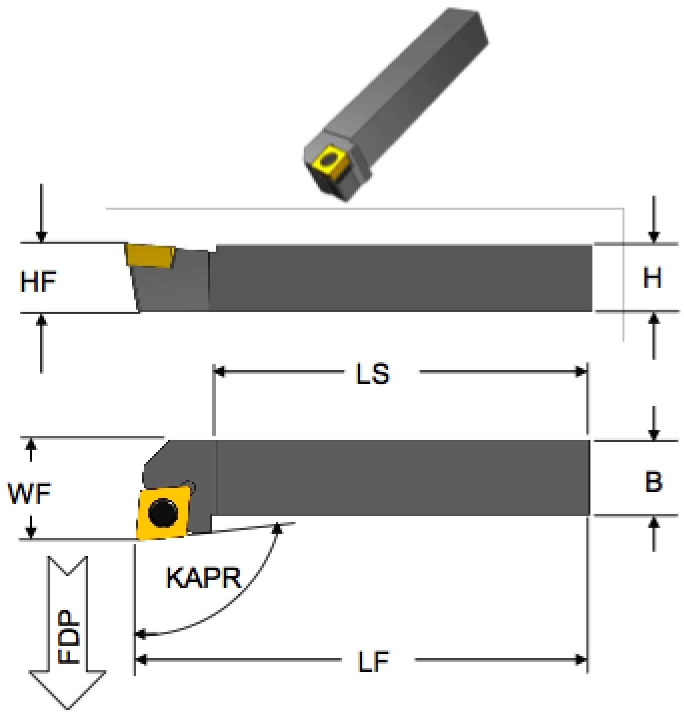


Figure 19: Cutting Tool Measurement Diagram 2
(Cutting Item, Tool Item, and Adaptive Item – ISO 13399)

Measurement	Code	Description	Units
`BodyDiameterMax`	`BDX`	The largest diameter of the body of a Tool Item.	mm
`BodyLengthMax`	`LBX`	The distance measured along the X axis from that point of the item closest to the workpiece, including the Cutting Item for a Tool Item but excluding a protruding locking mechanism for an Adaptive Item, to either the front of the flange on a flanged body or the beginning of the connection interface feature on the machine side for cylindrical or prismatic shanks.	mm
`DepthOfCutMax`	`APMX`	The maximum engagement of the cutting edge or edges with the workpiece measured perpendicular to the feed motion.	mm
`CuttingDiameterMax`	`DC`	The maximum diameter of a circle on which the defined point Pk of each of the master inserts is located on a Tool Item. The normal of the machined peripheral surface points towards the axis of the Cutting Tool.	mm
`FlangeDiameterMax`	`DF`	The dimension between two parallel tangents on the outside edge of a flange.	mm

Measurement	Code	Description	Units
`OverallToolLength`	`OAL`	The largest length dimension of the Cutting Tool including the master insert where applicable.	mm
`ShankDiameter`	`DMM`	The dimension of the diameter of a cylindrical portion of a Tool Item or an Adaptive Item that can participate in a connection.	mm
`ShankHeight`	`H`	The dimension of the height of the shank.	mm
`ShankLength`	`LS`	The dimension of the length of the shank.	mm
`UsableLengthMax`	`LUX`	maximum length of a Cutting Tool that can be used in a particular cutting operation including the non-cutting portions of the tool.	mm
`ProtrudingLength`	`LPR`	The dimension from the yz-plane to the furthest point of the Tool Item or Adaptive Item measured in the -X direction.	mm
`Weight`	`WT`	The total weight of the Cutting Tool in grams. The force exerted by the mass of the Cutting Tool.	grams
`FunctionalLength`	`LF`	The distance from the gauge plane or from the end of the shank to the furthest point on the tool, if a gauge plane does not exist, to the cutting reference point determined by the main function of the tool. The `CuttingTool` functional length will be the length of the entire tool, not a single Cutting Item. Each `CuttingItem` can have an independent `FunctionalLength` represented in its measurements.	mm

6.2.9 `CuttingItems` Element for `CuttingToolLifeCycle`

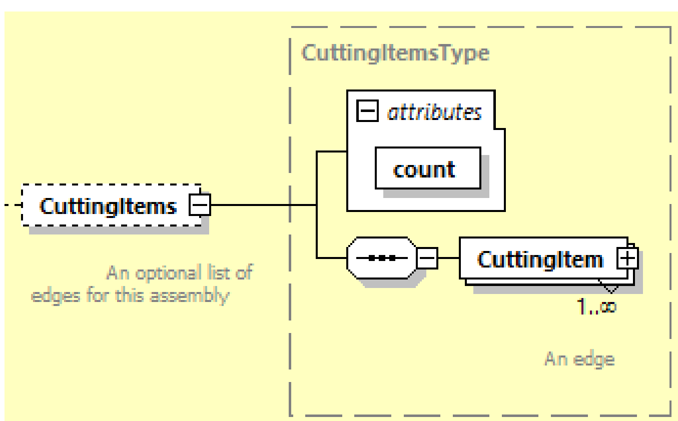


Figure 20: `CuttingItems` Schema

An optional collection of Cutting Items that **SHOULD** be provided for each independent edge or insert. If the `CuttingItems` are not present; it indicates there is no specific information with respect to each of the Cutting Items. This does not imply there are no Cutting Items – there **MUST** be at least one Cutting Item – but there is no specific information.

6.2.9.1 Attributes for `CuttingItems`

Attribute	Description	Occurrence
`count`	The number of Cutting Items. `count` is a required attribute.	1

6.2.10 `CuttingItem`

A Cutting Item is the portion of the tool that physically removes the material from the workpiece by shear deformation. The Cutting Item can be either a single piece of material attached to the Tool Item or it can be one or more separate pieces of material attached to the Tool Item using a permanent or removable attachment. A Cutting Item can be comprised of one or more cutting edges. Cutting Items include: replaceable inserts, brazed tips and the cutting portions of solid Cutting Tools.

MTConnect Standard considers Cutting Items as part of the Cutting Tool. A Cutting Item **MUST NOT** exist in MTConnect unless it is attached to a Cutting Tool. Some of the measurements, such as `FunctionalLength`, **MUST** be made with reference to the entire Cutting Tool to be meaningful.

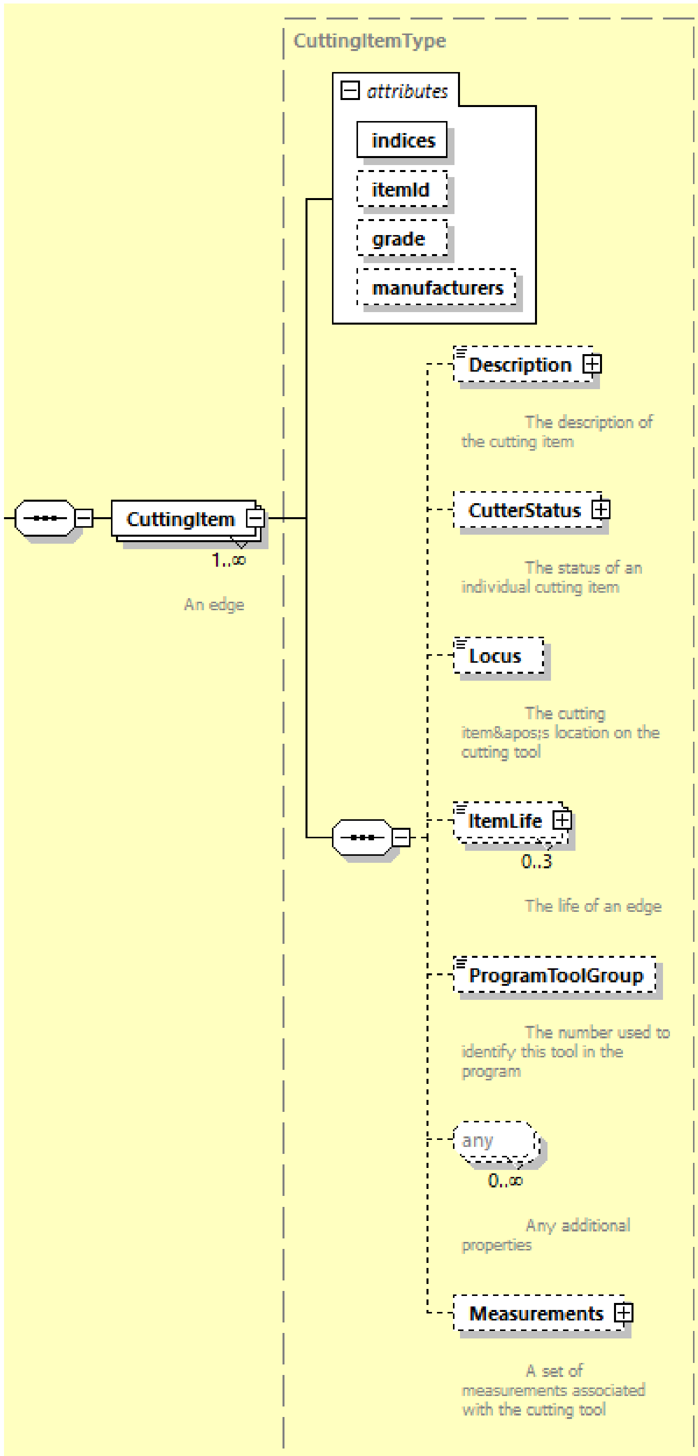


Figure 21: `CuttingItem` Schema

6.2.10.1 Attributes for `CuttingItem`

Attribute	Description	Occurrence
`indices`	The number or numbers representing the individual Cutting Item or items on the tool. `indices` is a required attribute	1
`itemId`	The manufacturer identifier of this Cutting Item. `itemId` is an optional attribute.	0..1
`manufacturers`	The manufacturers of the Cutting Item. `manufacturers` is an optional attribute.	0..1
`grade`	The material composition for this Cutting Item. `grade` is an optional attribute.	0..1

6.2.10.1.1 `Indices` Attribute for `CuttingItem`

An identifier that indicates the Cutting Item or items these data are associated with. The value **MUST** be a single number ("1") or a comma separated set of individual elements ("1,2,3,4"), or as a inclusive range of values as in ("1-10") or any combination of ranges and numbers as in "1-4,6-10,22". There **MUST NOT** be spaces or non-integer values in the text representation.

Indices **SHOULD** start numbering with the inserts or Cutting Item furthest from the gauge line and increasing in value as the items get closer to the gauge line. Items at the same distance **MAY** be arbitrarily numbered.

6.2.10.1.2 `itemId` Attribute for `CuttingItem`

The manufactures' identifier for this Cutting Item that **MAY** be its catalog or reference number. The value **MUST** be an XML NMTOKEN value of numbers and letters.

6.2.10.1.3 `manufacturers` Attribute for `CuttingItem`

This optional element references the manufacturers of this tool. At this level the manufacturers will reference the Cutting Item specifically. The representation will be a comma (`,`) delimited list of manufacturer names. This can be any series of numbers and letters as defined by the XML type `string`.

6.2.10.1.4 `grade` Attribute for `CuttingItem`

This provides an implementation specific designation for the material composition of this Cutting Item.

6.2.10.2 Elements for `CuttingItem`

Element	Description	Occurrence
`Description`	A free-form description of the Cutting Item.	0..1
`Locus`	A free form description of the location on the Cutting Tool.	0..1
`ItemLife`	The life of this Cutting Item.	0..3
`Measurements`	A collection of measurements relating to this Cutting Item.	0..1

6.2.10.2.1 `Description` Element for `CuttingItem`

An optional free form text description of this Cutting Item.

6.2.10.2.2 `Locus` Element for `CuttingItem`

Locus represents the location of the Cutting Item with respect to the Cutting Tool. For clarity, the words FLUTE, INSERT, and CARTRIDGE **SHOULD** be used to assist in noting the location of a Cutting Item. The Locus **MAY** be any free form text, but **SHOULD** adhere to the following rules:

1. The location numbering **SHOULD** start at the furthest Cutting Item (#1) and work it's way back to the Cutting Item closest to the gauge line.
2. Flutes **SHOULD** be identified as such using the word `FLUTE:`. For example: `FLUTE: 1, INSERT: 2` - would indicate the first flute and the second furthest insert from the end of the tool on that flute.
3. Other designations such as `CARTRIDGE` **MAY** be included, but should be identified using upper case and followed by a colon (`:`).

6.2.10.2.3 `ItemLife` Element for `CuttingItem`

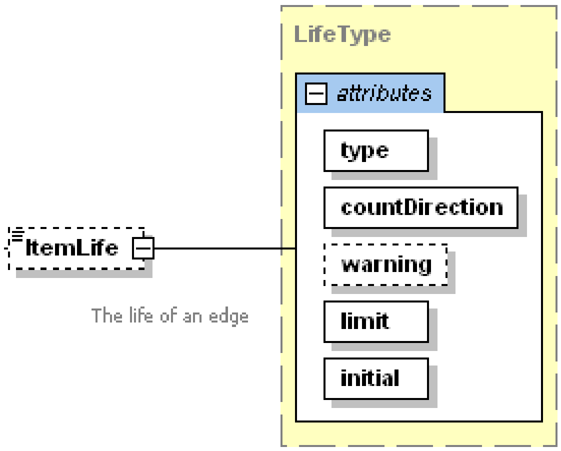


Figure 22: Item Life

The value is the current value for the tool life. The value **MUST** be a number. Tool life is an option element which can have three types, either minutes for time based, part count for parts based, or wear based using a distance measure. One tool life can appear for each type, but there cannot be two entries of the same type. Additional types can be added in the future.

6.2.10.2.4 Attributes for `ItemLife`

These is an optional attribute that can be used to further classify the operation type.

Attribute	Description	Occurrence
`type`	The type of tool life being accumulated. *Valid Data Values*: `MINUTES`, `PART_COUNT`, or `WEAR`. `type` is a required attribute.	1
`countDirection`	Indicates if the tool life counts from zero to maximum or maximum to zero. The values **MUST** be one of `UP` or `DOWN`. `countDirection` is a required attribute.	1
`warning`	The point at which a tool life warning will be raised. `warning` is an optional attribute.	0..1

Attribute	Description	Occurrence
`limit`	The end of life limit for this tool. If the `countDirection` is `DOWN`, the point at which this tool should be expired, usually zero. If the `countDirection` is `UP`, this is the upper limit for which this tool should be expired. `limit` is an optional attribute.	0..1
`initial`	The initial life of the tool when it is new. `initial` is an optional attribute.	0..1

6.2.10.2.5 `type` Attribute for `ItemLife`

The value of type must be one of the following:

Value	Description
`MINUTES`	The tool life measured in minutes. All units for minimum, maximum, and nominal **MUST** be provided in minutes.
`PART_COUNT`	The tool life measured in parts. All units for minimum, maximum, and nominal **MUST** be provided supplied as the number of parts.
`WEAR`	The tool life measured in tool wear. Wear **MUST** be provided in millimeters as an offset to nominal. All units for minimum, maximum, and nominal **MUST** be given as millimeter offsets as well.

6.2.10.2.6 `countDirection` Attribute for `ItemLife`

The value of type must be one of the following:

Value	Description
`DOWN`	The tool life counts down from the maximum to zero.
`UP`	The tool life counts up from zero to the maximum.

6.2.10.3 `Measurement` Subtypes for `CuttingItem`

These measurements for `CuttingItem` are specific to an individual Cutting Item and **MUST NOT** be used for the measurement pertaining to an assembly. The following diagram will be used to for reference for the Cutting Item specific measurements.

The Code in the following table will refer to the acronym in the diagram. We will be referring to many diagrams to disambiguate all measurements of the Cutting Tools and Items. We will present a few here; please refer to *Appendix B* for additional reference material.

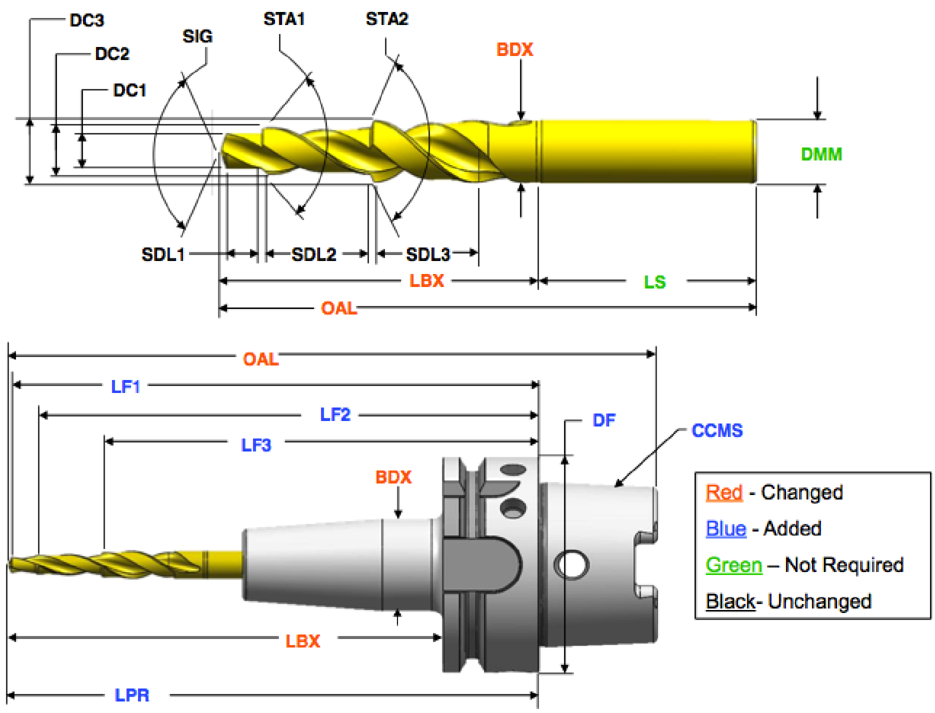


Figure 23: Cutting Tool

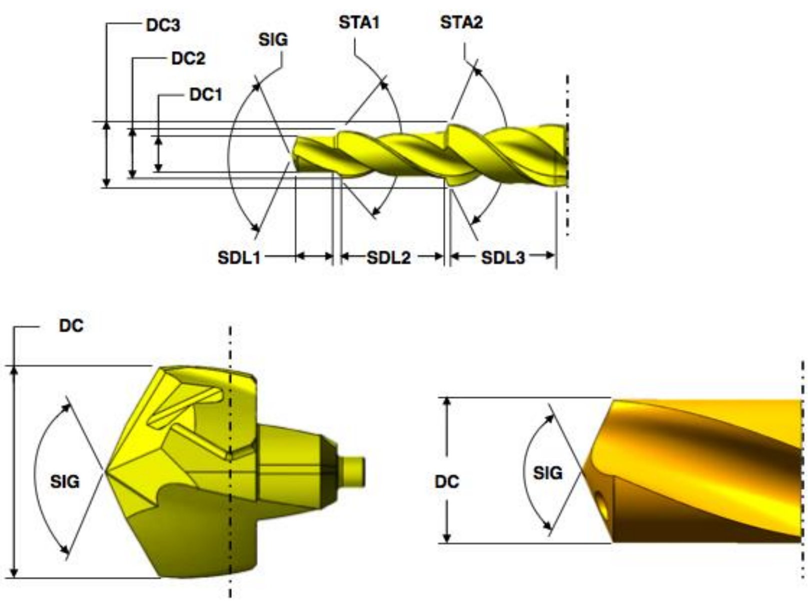


Figure 24: Cutting Item

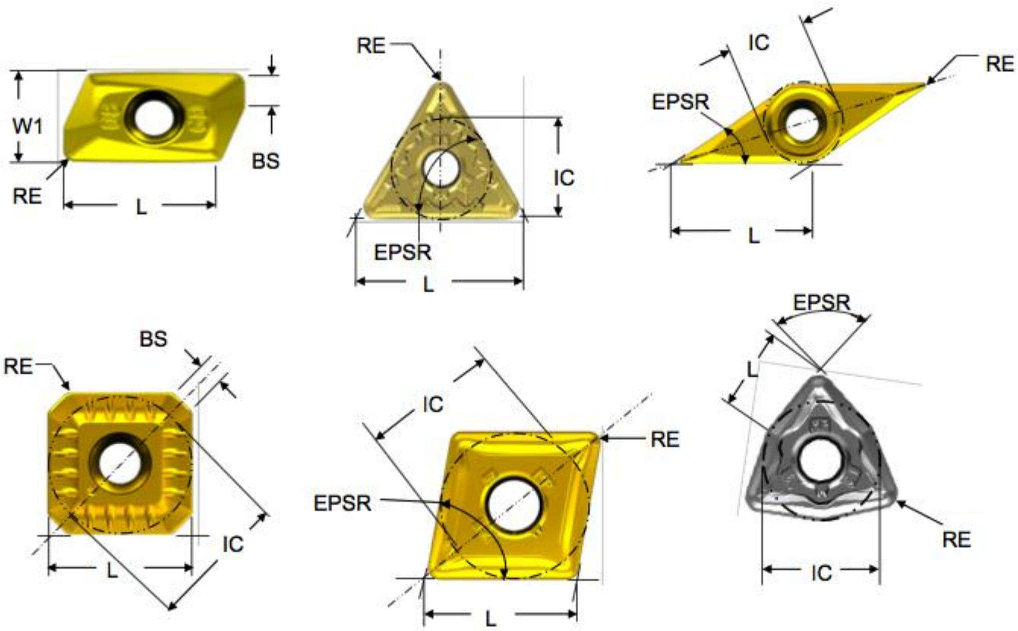


Figure 25: Cutting Item Measurement Diagram 3
(Cutting Item – ISO 13399)

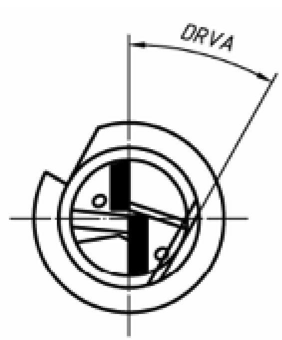


Figure 26: Cutting Item Drive Angle
(Cutting Item – ISO 13399)

The following `CuttingItem` `Measurements` will refer the diagram above.

Measurement Subtype	Code	Description	Units
`CuttingReferencePoint`	`CRP`	The theoretical sharp point of the Cutting Tool from which the major functional dimensions are taken.	mm

Measurement Subtype	Code	Description	Units
`CuttingEdgeLength`	`L`	The theoretical length of the cutting edge of a Cutting Item over sharp corners.	mm
`DriveAngle`	`DRVA`	Angle between the driving mechanism locator on a Tool Item and the main cutting edge	degree
`FlangeDiameter`	`DF`	The dimension between two parallel tangents on the outside edge of a flange.	mm
`FunctionalWidth`	`WF`	The distance between the cutting reference point and the rear backing surface of a turning tool or the axis of a boring bar.	mm
`IncribedCircleDiameter`	`IC`	The diameter of a circle to which all edges of a equilateral and round regular insert are tangential.	mm
`PointAngle`	`SIG`	The angle between the major cutting edge and the same cutting edge rotated by 180 degrees about the tool axis.	degree
`ToolCuttingEdgeAngle`	`KAPR`	The angle between the tool cutting edge plane and the tool feed plane measured in a plane parallel the xy-plane.	degree
`ToolLeadAngle`	`PSIR`	The angle between the tool cutting edge plane and a plane perpendicular to the tool feed plane measured in a plane parallel the xy-plane.	degree
`ToolOrientation`	`N/A`	The angle of the tool with respect to the workpiece for a given process. The value is application specific.	degree
`WiperEdgeLength`	`BS`	The measure of the length of a wiper edge of a Cutting Item.	mm
`StepDiameterLength`	`SDLx`	The length of a portion of a stepped tool that is related to a corresponding cutting diameter measured from the cutting reference point of that cutting diameter to the point on the next cutting edge at which the diameter starts to change.	mm
`StepIncludedAngle`	`STAx`	The angle between a major edge on a step of a stepped tool and the same cutting edge rotated 180 degrees about its tool axis.	degree
`CuttingDiameter`	`DCx`	The diameter of a circle on which the defined point Pk located on this Cutting Tool. The normal of the machined peripheral surface points towards the axis of the Cutting Tool.	mm

Measurement Subtype	Code	Description	Units
`CuttingHeight`	`HF`	The distance from the basal plane of the Tool Item to the cutting point.	mm
`CornerRadius`	`RE`	The nominal radius of a rounded corner measured in the X Y-plane.	mm
`Weight`	`WT`	The total weight of the Cutting Tool in grams. The force exerted by the mass of the Cutting Tool.	grams
`FunctionalLength`	`LFx`	The distance from the gauge plane or from the end of the shank of the Cutting Tool, if a gauge plane does not exist, to the cutting reference point determined by the main function of the tool. This measurement will be with reference to the Cutting Tool and **MUST NOT** exist without a Cutting Tool.	mm
`ChamferFlatLength`	`BCH`	The flat length of a chamfer.	mm
`ChamferWidth`	`CHW`	The width of the chamfer	mm
`InsertWidth`	`W1`	W1 is used for the insert width when an inscribed circle diameter is not practical.	mm

Appendices

A. Bibliography

1. Engineering Industries Association. *EIA Standard - EIA-274-D*, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.

2. ISO TC 184/SC4/WG3 N1089. *ISO/DIS 10303-238*: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.

3. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.

4. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.

5. International Organization for Standardization. *ISO 6983/1* – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.

6. Electronic Industries Association. *ANSI/EIA-494-B-1992*, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.

7. National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.

8. International Organization for Standardization. *ISO 10303-11*: 1994, Industrial automation systems and integration Product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

9. International Organization for Standardization. *ISO 10303-21*: 1996, Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.

10. H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.

11. International Organization for Standardization. *ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature.* Geneva, Switzerland, 2001.

12. *ASME B5.59-2 Version 9c: Data Specification for Properties of Machine Tools for Milling and Turning. 2005.*

13. *ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers. 2005.*

14. OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.*

15. International Organization for Standardization. *ISO 13399*: *Cutting Tool data representation and exchange*. Geneva, Switzerland, 2000.

B. Additional Illustrations

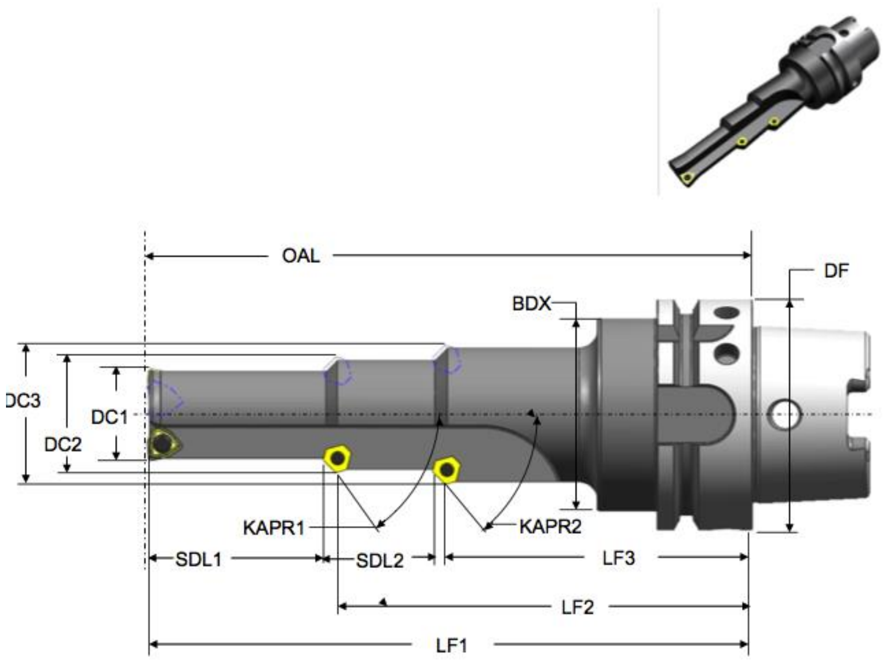


Figure 27: Cutting Tool Measurement Diagram 1
(Cutting Tool, Cutting Item, and Assembly Item – ISO 13399)

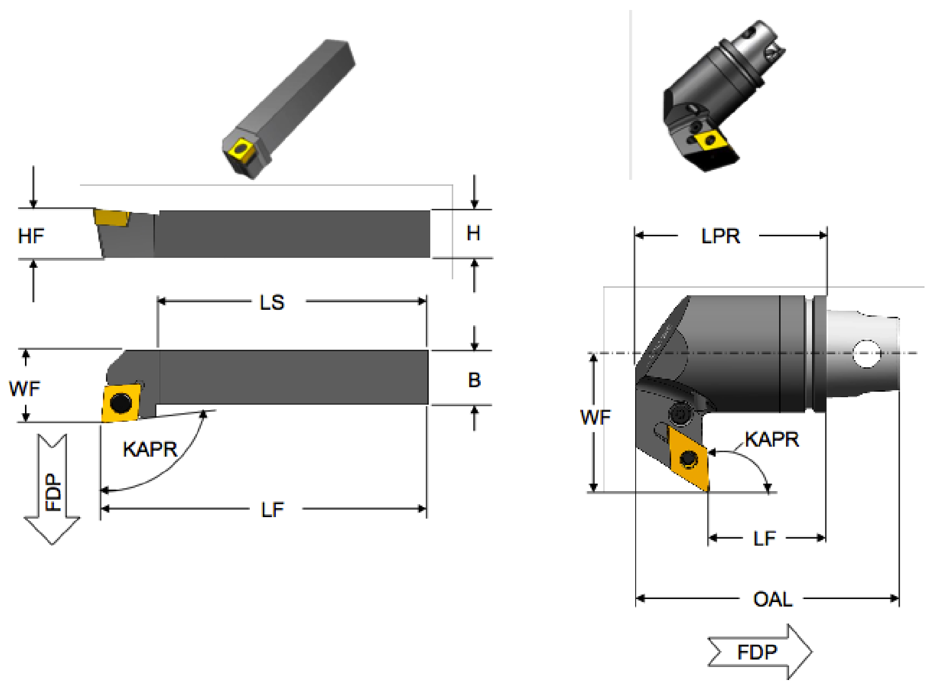


Figure 28: Cutting Tool Measurement Diagram 2
(Cutting Tool, Cutting Item, and Assembly Item – ISO 13399)

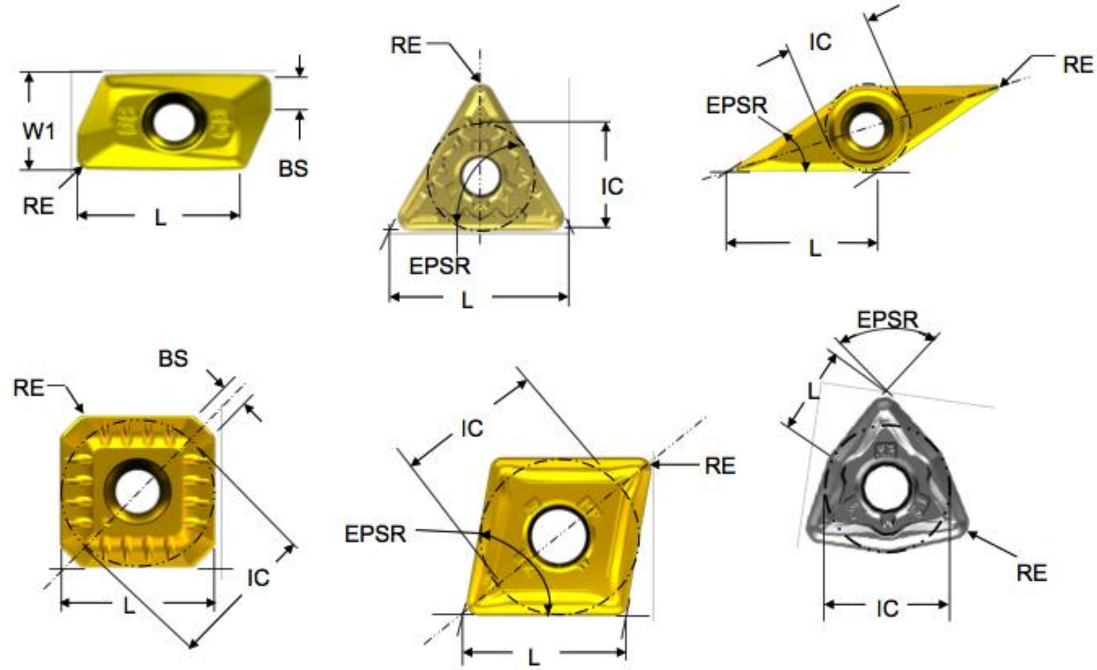


Figure 29: Cutting Item Measurement Diagram 3
(Cutting Item – ISO 13399)

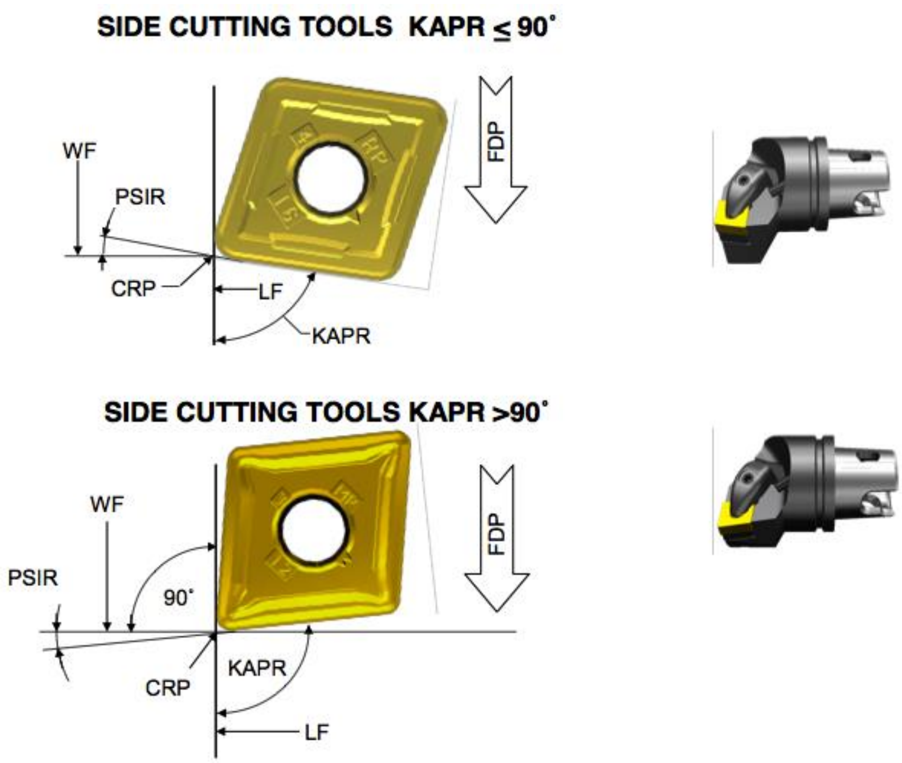


Figure 30: Cutting Item Measurement Diagram 4
(Cutting Item – ISO 13399)

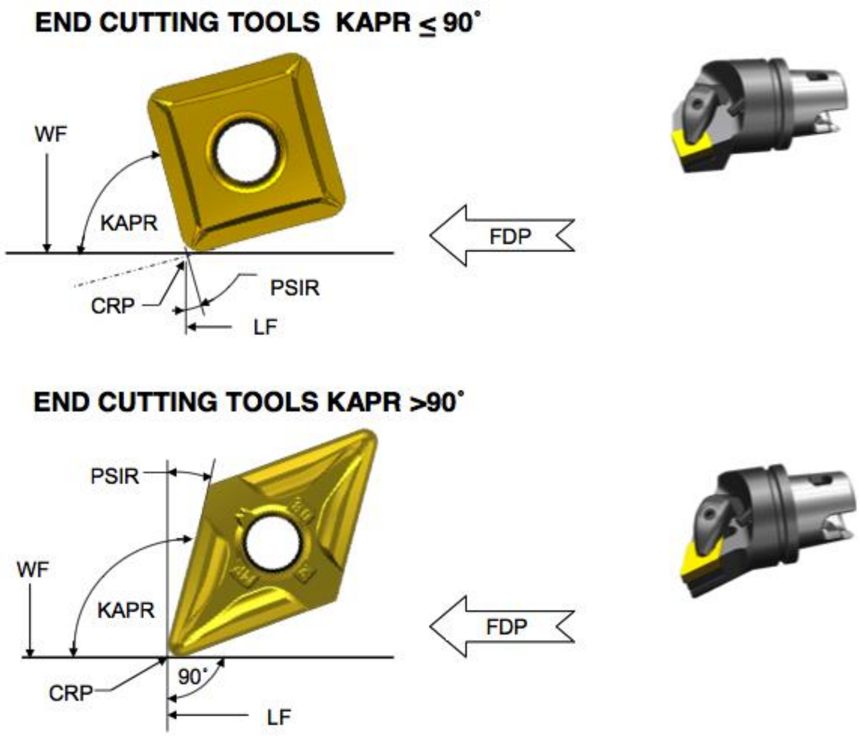


Figure 31: Cutting Item Measurement Diagram 5
(Cutting Item – ISO 13399)

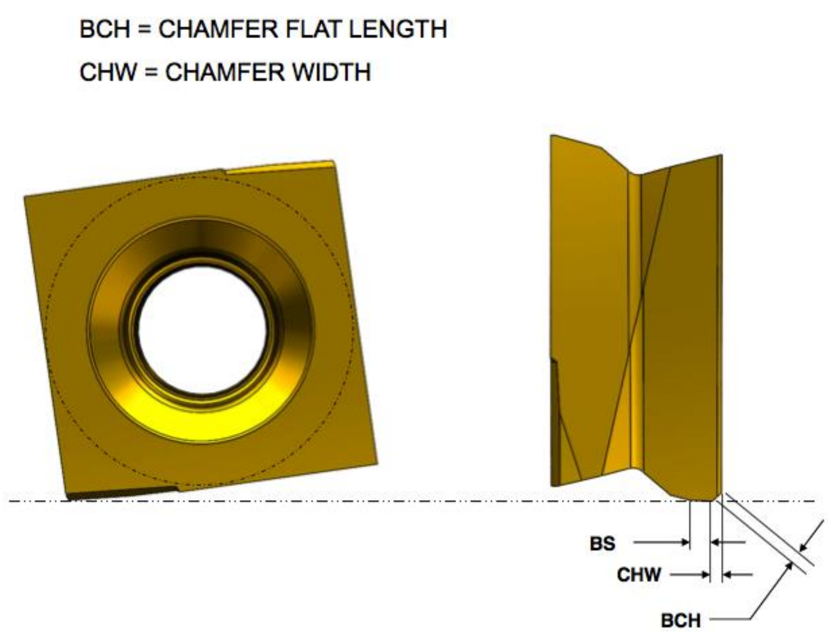


Figure 32: Cutting Item Measurement Diagram 6
(Cutting Item – ISO 13399)

C. Cutting Tool Example

C.1 Shell Mill

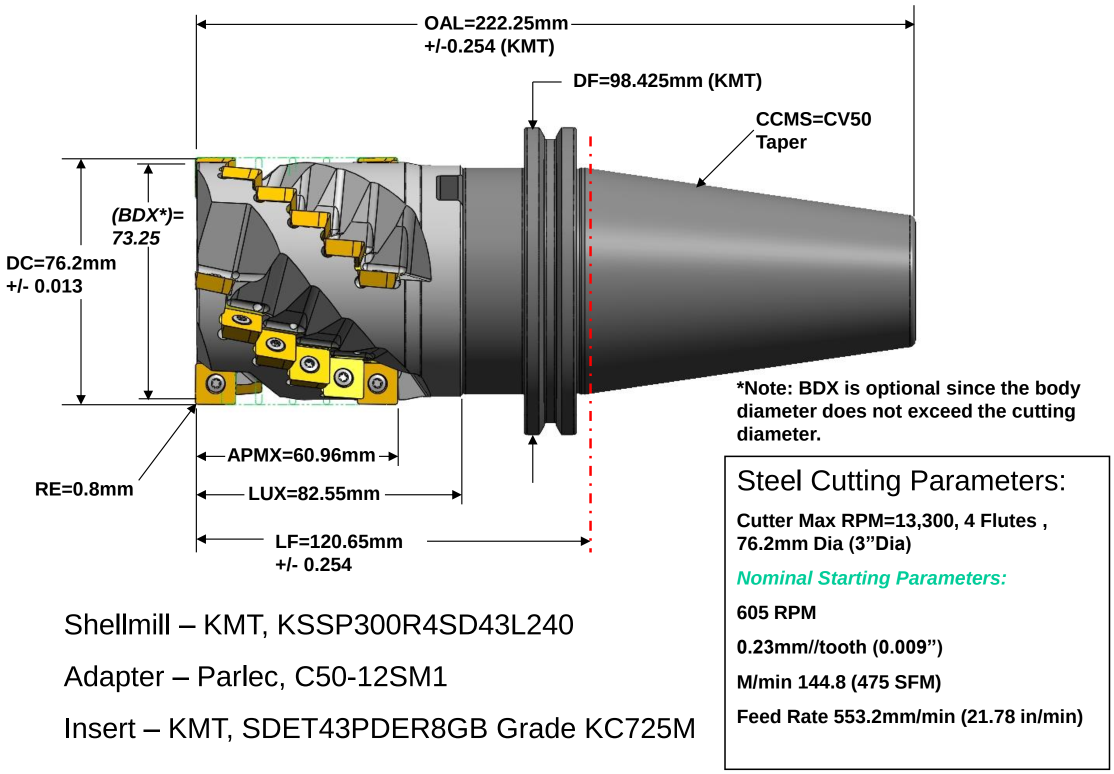


Figure 33: Shell Mill Side View

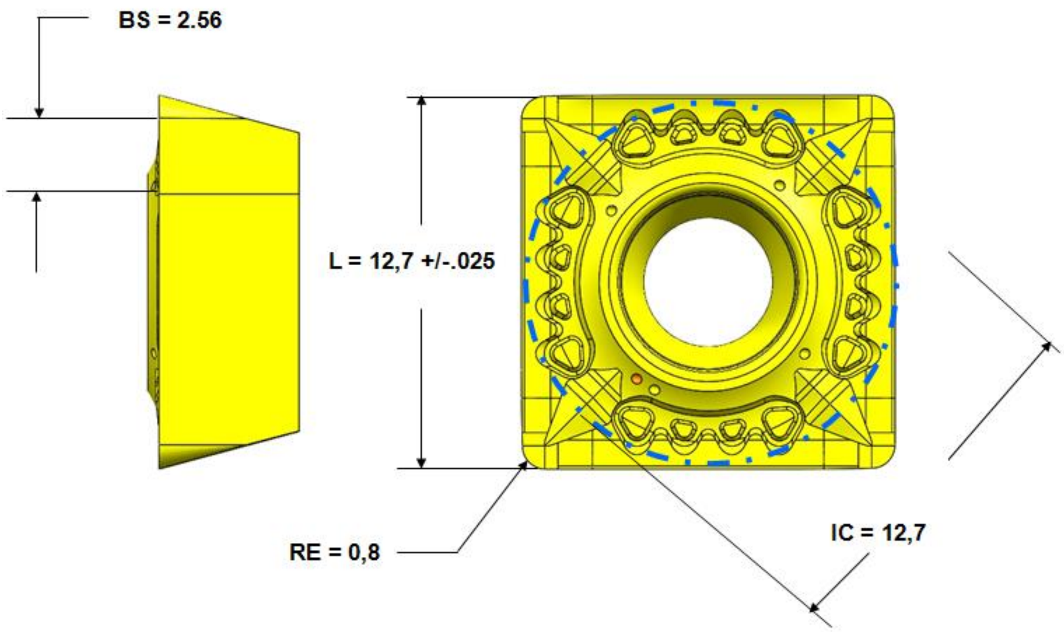


Figure 34: Indexable Insert Measurements

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectAssets xmlns:m="urn:mtconnect.org:MTConnectAssets:1.2"
   xmlns="urn:mtconnect.org:MTConnectAssets:1.2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:mtconnect.org:MTConnectAssets:1.2
   http://mtconnect.org/schemas/MTConnectAssets_1.2.xsd">
   <Header creationTime="2011-05-11T13:55:22" assetBufferSize="1024"
   sender="localhost" assetCount="2" version="1.2" instanceId="1234"/>
   <Assets>
    <CuttingTool serialNumber="1" toolId="KSSP300R4SD43L240" timestamp="2011-
   05-11T13:55:22" assetId="KSSP300R4SD43L240.1" manufacturers="KMT,Parlec">
     <CuttingToolLifeCycle>
      <CutterStatus><Status>NEW</Status></CutterStatus>
      <ProcessSpindleSpeed maximum="13300"
   nominal="605">10000</ProcessSpindleSpeed>
      <ProcessFeedRate nominal="9.22">9.22</ProcessSpindleSpeed>
      <ConnectionCodeMachineSide>CV50</ConnectionCodeMachineSide>
      <Measurements>
       <BodyDiameterMax code="BDX">73.25</BodyDiameterMax>
       <OverallToolLength nominal="222.25" minimum="221.996"
   maximum="222.504" code="OAL">222.25</OverallToolLength>
       <UsableLengthMax code="LUX" nominal="82.55">82.55</UsableLengthMax>
       <CuttingDiameterMax code="DC" nominal="76.2" maximum="76.213"
   minimum="76.187">76.2</CuttingDiameterMax>
```

```xml
          <BodyLengthMax code="LF" nominal="120.65" maximum="120.904"
minimum="120.404">120.65</BodyLengthMax>
          <DepthOfCutMax code="APMX" nominal="60.96">60.95</DepthOfCutMax>
          <FlangeDiameterMax code="DF"
nominal="98.425">98.425</FlangeDiameterMax>
        </Measurements>
        <CuttingItems count="24">
          <CuttingItem indices="1-24" itemId="SDET43PDER8GB" manufacturers="KMT"
grade="KC725M">
            <Measurements>
              <CuttingEdgeLength code="L" nominal="12.7" minimum="12.675"
maximum="12.725">12.7</CuttingEdgeLength>
              <WiperEdgeLength code="BS" nominal="2.56">2.56</WiperEdgeLength>
              <IncribedCircleDiameter code="IC"
nominal="12.7">12.7</IncribedCircleDiameter>
              <CornerRadius code="RE" nominal="0.8">0.8</CornerRadius>
            </Measurements>
          </CuttingItem>
        </CuttingItems>
      </CuttingToolLifeCycle>
    </CuttingTool>
  </Assets>
</MTConnectAssets>
```

C.2 Step Drill

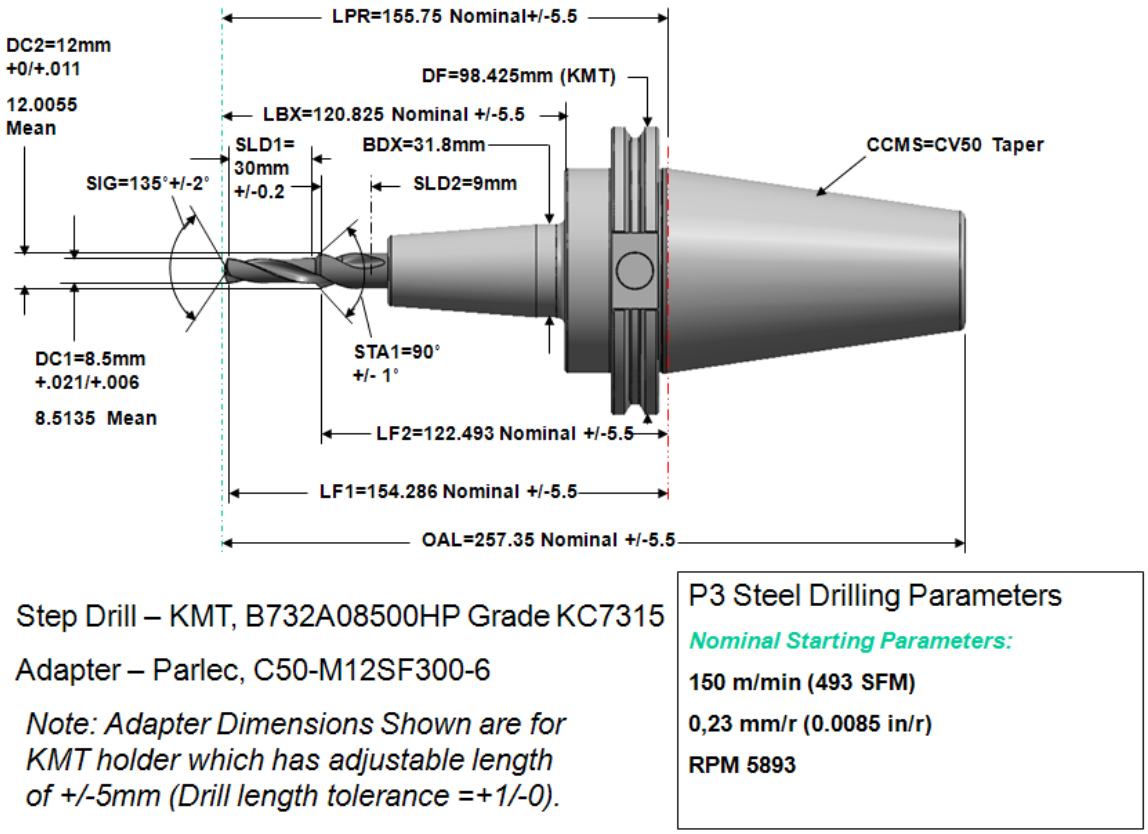
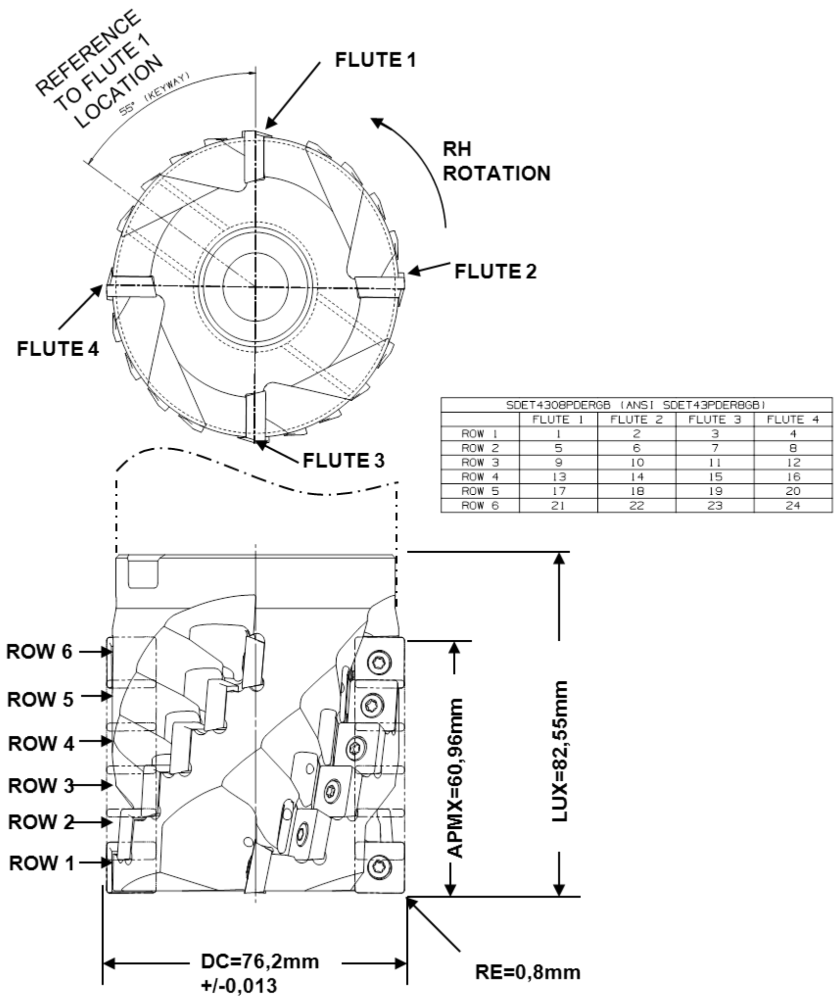


Figure 35: Step Drill Side View

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectAssets xmlns:m="urn:mtconnect.org:MTConnectAssets:1.2"
   xmlns="urn:mtconnect.org:MTConnectAssets:1.2"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:mtconnect.org:MTConnectAssets:1.2
   http://mtconnect.org/schemas/MTConnectAssets_1.2.xsd">
   <Header creationTime="2011-05-11T13:55:22" assetBufferSize="1024"
   sender="localhost" assetCount="2" version="1.2" instanceId="1234"/>
   <Assets>
    <CuttingTool serialNumber="1 " toolId="B732A08500HP" timestamp="2011-05-
   11T13:55:22" assetId="B732A08500HP " manufacturers="KMT,Parlec">
     <Description>
      Step Drill - KMT, B732A08500HP Grade KC7315
      Adapter - Parlec, C50-M12SF300-6
     </Description>
     <CuttingToolLifeCycle>
      <CutterStatus><Status>NEW</Status></CutterStatus>
```

```xml
        <ProcessSpindleSpeed nominal="5893">5893</ProcessSpindleSpeed>
        <ProcessFeedRate nominal="2.5">2.5</ProcessFeedRate>
        <ConnectionCodeMachineSide>CV50 Taper</ConnectionCodeMachineSide>
        <Measurements>
         <BodyDiameterMax code="BDX">31.8</BodyDiameterMax>
         <BodyLengthMax code="LBX" nominal="120.825" maximum="126.325"
  minimum="115.325">120.825</BodyLengthMax>
         <ProtrudingLength code="LPR" nominal="155.75" maximum="161.25"
  minimum="150.26">155.75</ProtrudingLength>
         <FlangeDiameterMax code="DF"
  nominal="98.425">98.425</FlangeDiameterMax>
         <OverallToolLength nominal="257.35" minimum="251.85" maximum="262.85"
  code="OAL">257.35</OverallToolLength>
        </Measurements>
        <CuttingItems count="2">
         <CuttingItem indices="1" manufacturers="KMT" grade="KC7315">>
          <Measurements>
           <CuttingDiameter code="DC1" nominal="8.5" maximum="8.521"
  minimum="8.506">8.5135</CuttingDiameter>
           <StepIncludedAngle code="STA1" nominal="90" maximum="91"
  minimum="89">90</StepIncludedAngle>
           <FunctionalLength code="LF1" nominal="154.286" minimum="148.786"
  maximum="159.786">154.286</FunctionalLength>
           <StepDiameterLength code="SDL1" nominal="9">9</StepDiameterLength>
           <PointAngle code="SIG" nominal="135" minimum="133"
  maximum="137">135</PointAngle>
          </Measurements>
         </CuttingItem>
         <CuttingItem indices="2" manufacturers="KMT" grade="KC7315">>
          <Measurements>
           <CuttingDiameter code="DC2" nominal="12" maximum="12.011"
  minimum="12">12</CuttingDiameter>
           <FunctionalLength code="LF2" nominal="122.493" maximum="127.993"
  minimum="116.993">122.493</FunctionalLength>
           <StepDiameterLength code="SDL2" nominal="9">9</StepDiameterLength>
          </Measurements>
         </CuttingItem>
        </CuttingItems>
      </CuttingToolLifeCycle>
     </CuttingTool>
    </Assets>
</MTConnectAssets>
```

C.3 Shell Mill with Individual Loci



SDET4308PDERGB (ANSI SDET43PDER8GB)				
	FLUTE 1	FLUTE 2	FLUTE 3	FLUTE 4
ROW 1	1	2	3	4
ROW 2	5	6	7	8
ROW 3	9	10	11	12
ROW 4	13	14	15	16
ROW 5	17	18	19	20
ROW 6	21	22	23	24

Figure 36: Shell Mill with Explicate Loci


```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectAssets xmlns:m="urn:mtconnect.org:MTConnectAssets:1.2"
xmlns="urn:mtconnect.org:MTConnectAssets:1.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mtconnect.org:MTConnectAssets:1.2
http://mtconnect.org/schemas/MTConnectAssets_1.2.xsd">
  <Header creationTime="2011-05-11T13:55:22" assetBufferSize="1024" sender="localhost"
assetCount="2" version="1.2" instanceId="1234"/>
  <Assets>
```

```xml
    <CuttingTool serialNumber="1" toolId="KSSP300R4SD43L240" timestamp="2011-05-11T13:55:22" assetId="KSSP300R4SD43L240.1" manufacturers="KMT,Parlec">
     <Description>Keyway: 55 degrees</Description>
      <CuttingToolLifeCycle>
        <CutterStatus><Status>NEW</Status></CutterStatus>
        <Measurements>
          <UsableLengthMax code="LUX" nominal="82.55">82.55</UsableLengthMax>
          <CuttingDiameterMax code="DC" nominal="76.2" maximum="76.213" minimum="76.187">76.2</CuttingDiameterMax>
          <DepthOfCutMax code="APMX" nominal="60.96">60.95</DepthOfCutMax>
        </Measurements>
        <CuttingItems count="24">
          <CuttingItem indices="1" itemId="SDET43PDER8GB" manufacturers="KMT">
            <Locus>FLUTE: 1, ROW: 1</Locus>
        <Measurements>
             <DriveAngle code="DRVA" nominal="55">55</DriveAngle>
         </Measurements>
          </CuttingItem>
          <CuttingItem indices="2-24" itemId="SDET43PDER8GB" manufacturers="KMT">
            <Locus>FLUTE: 2-4, ROW: 1; FLUTE: 1-4, ROW 2-6</Locus>
          </CuttingItem>
        </CuttingItems>
      </CuttingToolLifeCycle>
    </CuttingTool>
  </Assets>
</MTConnectAssets>
```

C.4 Drill with Individual Loci

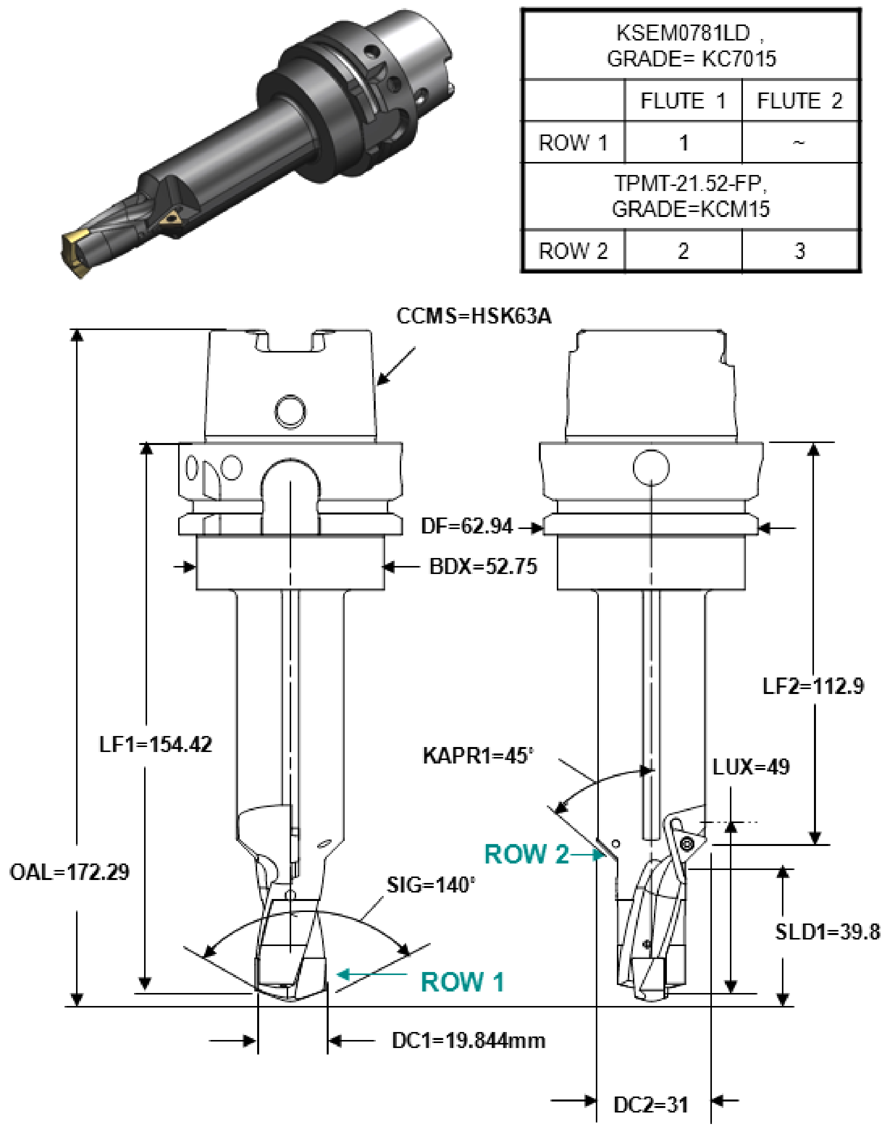


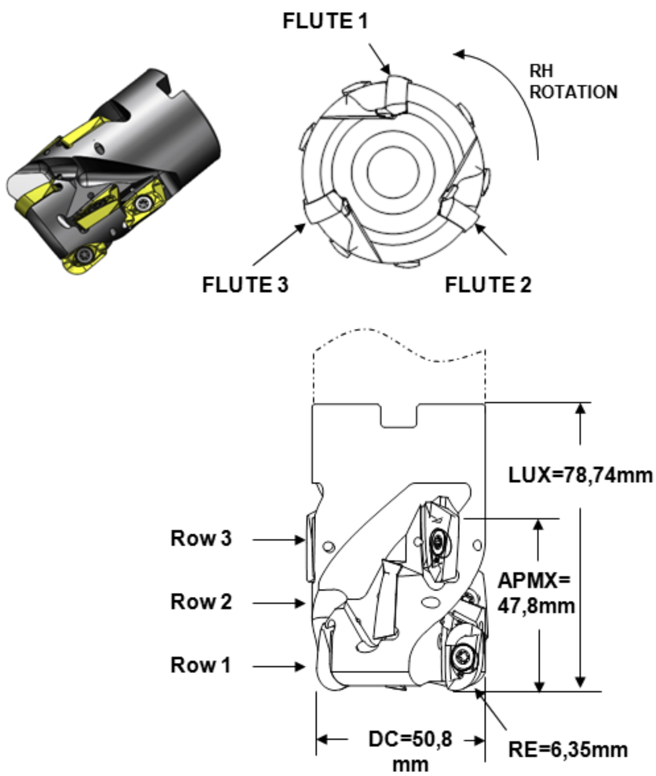
Figure 37: Step Drill with Explicate Loci

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectAssets xmlns:m="urn:mtconnect.org:MTConnectAssets:1.2"
xmlns="urn:mtconnect.org:MTConnectAssets:1.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mtconnect.org:MTConnectAssets:1.2
http://mtconnect.org/schemas/MTConnectAssets_1.2.xsd">
  <Header creationTime="2011-05-11T13:55:22" assetBufferSize="1024" sender="localhost"
assetCount="2" version="1.2" instanceId="1234"/>
```

```xml
<Assets>
  <CuttingTool serialNumber="1" toolId="KSEM0781LD" timestamp="2011-05-11T13:55:22" assetId="KSEM0781LD.1" manufacturers="KMT">
    <CuttingToolLifeCycle>
      <CutterStatus><Status>NEW</Status></CutterStatus>
      <ConnectionCodeMachineSide>HSK63A</ConnectionCodeMachineSide>
      <Measurements>
        <BodyDiameterMax code="BDX">52.75</BodyDiameterMax>
        <OverallToolLength nominal="172.29" code="OAL">172.29</OverallToolLength>
        <UsableLengthMax code="LUX" nominal="49">49</UsableLengthMax>
        <FlangeDiameterMax code="DF" nominal="62.94">62.94</FlangeDiameterMax>
      </Measurements>
      <CuttingItems count="3">
        <CuttingItem indices="1" itemId="KSEM0781LD" manufacturers="KMT" grade="KC7015">
          <Locus>FLUTE: 1, ROW: 1</Locus>
          <Measurements>
    <FunctionalLength code="LF1" nominal="154.42">154.42</FunctionalLength>
    <CuttingDiameter code="DC1" nominal="19.844">19.844</CuttingDiameter>
    <PointAngle code="SIG" nominal="140">140</PointAngle>
    <ToolCuttingEdgeAngle code="KAPR1" nominal="45">45</ToolCuttingEdgeAngle>
    <StepDiameterLength code="SLD1" nominal="39.8">39.8</StepDiameterLength>
          </Measurements>
        </CuttingItem>
        <CuttingItem indices="2-3" itemId="TPMT-21.52-FP" manufacturers="KMT" grade="KCM15">
          <Locus>FLUTE: 1-2, ROW: 2</Locus>
          <Measurements>
    <FunctionalLength code="LF2" nominal="112.9">119.2</FunctionalLength>
    <CuttingDiameter code="DC2" nominal="31">31</CuttingDiameter>
```

```
            </Measurements>
          </CuttingItem>
        </CuttingItems>
      </CuttingToolLifeCycle>
    </CuttingTool>
  </Assets>
</MTConnectAssets>
```

C.5 Shell Mill with Different Inserts on First Row



EDPT180564PDER-LD			
	FLUTE 1	FLUTE 2	FLUTE 3
ROW 1	1	2	3
EDPT180508PDER-LD			
ROW 2	4	5	6
ROW 3	7	8	9

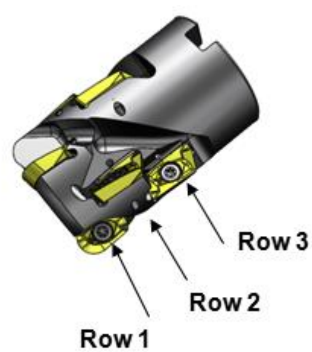


Figure 38: Shell Mill with Different Inserts on First Row

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectAssets xmlns:m="urn:mtconnect.org:MTConnectAssets:1.2"
xmlns="urn:mtconnect.org:MTConnectAssets:1.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mtconnect.org:MTConnectAssets:1.2
http://mtconnect.org/schemas/MTConnectAssets_1.2.xsd">
  <Header creationTime="2011-05-11T13:55:22" assetBufferSize="1024" sender="localhost"
assetCount="2" version="1.2" instanceId="1234"/>
  <Assets>
    <CuttingTool serialNumber="1" toolId="XXX" timestamp="2011-05-11T13:55:22"
assetId="XXX.1" manufacturers="KMT">
      <CuttingToolLifeCycle>
        <CutterStatus><Status>NEW</Status></CutterStatus>
        <Measurements>
          <DepthOfCutMax code="APMX" nominal="47.8">47.8</DepthOfCutMax>
          <CuttingDiameterMax code="DC" nominal="50.8">50.8</CuttingDiameterMax>
          <UsableLengthMax code="LUX" nominal="78.74">78.74</UsableLengthMax>
        </Measurements>
        <CuttingItems count="9">
          <CuttingItem indices="1-3" itemId="EDPT180564PDER-LD" manufacturers="KMT">
            <Locus>FLUTE: 1-3, ROW: 1</Locus>
            <Measurements>
              <CornerRadius code="RE" nominal="6.25">6.35</CornerRadius>
            </Measurements>
          </CuttingItem>
          <CuttingItem indices="4-9" itemId="EDPT180508PDER-LD" manufacturers="KMT">
            <Locus>FLANGE: 1-4, ROW: 2-3</Locus>
          </CuttingItem>
        </CuttingItems>
      </CuttingToolLifeCycle>
    </CuttingTool>
```

```
  </Assets>
</MTConnectAssets>
```

MTConnect® Standard
Part 5.0 – Interfaces

Version 1.4.0

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

MTConnect® Specification and Materials

Table of Content

Table of Figures

1 Purpose of This Document

This document, *Part 5.0 – Interfaces* of the MTConnect® Standard, defines a structured data model used to organize information required to coordinate inter-operations between pieces of equipment.

This data model is based on an *Interaction Model* that defines the exchange of information between pieces of equipment and is organized in the MTConnect Standard as the XML element `Interfaces`.

Interfaces is modeled as an extension to the `MTConnectDevices` and `MTConnectStreams` XML documents. `Interfaces` leverages similar rules and terminology as those used to describe a component in the `MTConnectDevices` XML document. `Interfaces` also uses similar methods for reporting data to those used in the `MTConnectStreams` XML document.

As defined in *Part 2.0 – Devices Information Model,* `Interfaces` is modeled as a *Top Level* component in the `MTConnectDevices` document (see *Figure 3* below). Each individual `Interface` XML element is modeled as a *Lower Level* component of `Interfaces`. The data associated with each *Interface* is modeled within each *Lower Level* component.

Note: See *Part 2.0 – Device Information Model* and *Part 3.0 - Streams Information Model* of the MTConnect Standard for information on how `Interfaces` is structured in the XML documents which are returned from an *MTConnect Agent* in response to a `Probe`, `Sample`, or `Current` request.

2 Terminology and Conventions

Refer to *Section 5* of *Part 1.0 - Overview and Functionality* for a dictionary of terms, reserved language, and document conventions used in the MTConnect® Standard.

3 Interfaces Overview

In many manufacturing processes, multiple pieces of equipment must work together to perform a task. The traditional method for coordinating the activities between individual pieces of equipment is to connect them together using a series of signal wires to communicate equipment states and demands for action. These interactions are usually accomplished by using simple binary ON/OFF signals.

In the MTConnect® Standard, *Interfaces* provides a means to replace this traditional method for interconnecting pieces of equipment with a structured *Interaction Model* that provides a rich set of information used to coordinate the actions between pieces of equipment. Implementers may utilize the information provided by this data model to (1) realize the interaction between pieces of equipment and (2) to extend the functionality of the equipment to improve the overall performance of the manufacturing process.

The *Interaction Model* used to implement *Interfaces* provides a lightweight and efficient protocol, simplifies failure recovery scenarios, and defines a structure for implementing a Plug-And-Play relationship between pieces of equipment. By standardizing the information exchange using this higher level semantic information model, an implementer may more readily replace a piece of equipment in a manufacturing system with any other piece of equipment capable of providing similar *Interaction Model* functions.

Two primary functions are required to implement the *Interaction Model* for *Interfaces* and manage the flow of information between pieces of equipment. Each piece of equipment needs to have:

- An *MTConnect Agent* which provides:
  - The data required to implement the *Interaction Model*.
  - Any other data from a piece of equipment needed to implement the *Interface* – operating states of the equipment, position information, execution modes, process information, etc.
- A client software application that enables the piece of equipment to acquire and interpret information from another piece of equipment.

3.1 *Interfaces* Architecture

MTConnect Standard is based on a communications method that provides no direct way for one piece of equipment to change the state of, or cause an action to occur by, another piece of equipment. The *Interaction Model* used to implement *Interfaces* is based on a *Publish/Subscribe* type of communications as described in *Part 1 – Overview and Functionality* and utilizes a *Request* and *Response* information exchange mechanism. For *Interfaces*, pieces of equipment must perform both the publish (*MTConnect Agent*) and subscribe (client) functions.

Note: The current definition of *Interfaces* addresses the interaction between two pieces of equipment. Future releases of the MTConnect Standard may address the interaction between multiple (more than two) pieces of equipment.

The diagram below provides a high-level overview of a typical system architecture used to implement *Interfaces*.

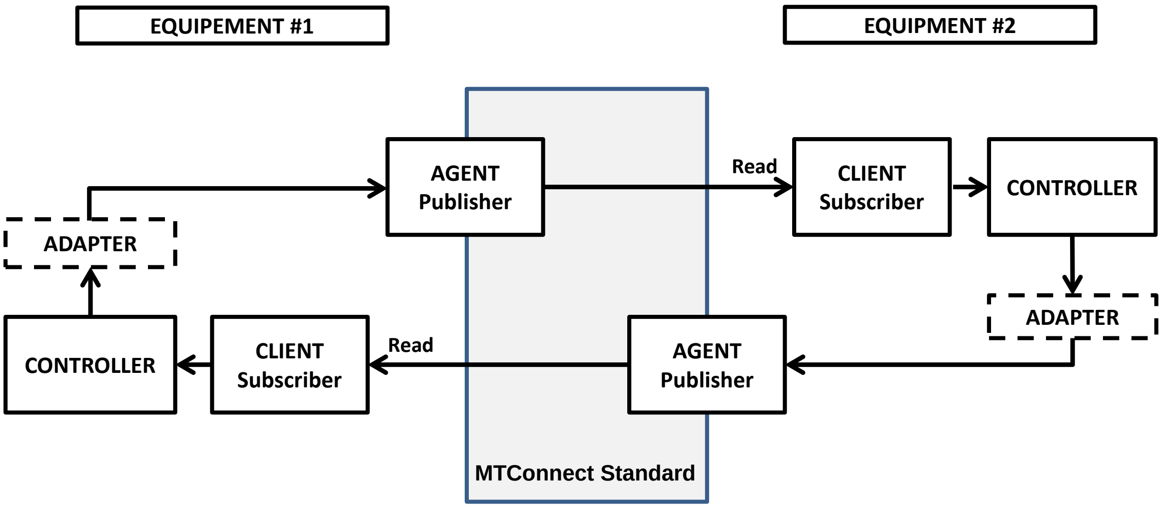


Figure 1: Data Flow Architecture for *Interfaces*

> Note: The data flow architecture illustrated in *Figure 1* above was historically referred to in the MTConnect Standard as a read-read concept.

In the implementation of the *Interaction Model* for *Interfaces*, two pieces of equipment can exchange information in the following manner. One piece of equipment indicates a *Request* for service by publishing a type of *Request* using a data item provided through an *MTConnect Agent* as defined in *Section 4* below. The client associated with a second piece of equipment, which is subscribing to data from the first machine, detects and interprets that *Request*. If the second machine chooses to take an action to fulfill this *Request*, it can indicate its acceptance by publishing a *Response* using a data item provided through its *MTConnect Agent*. The client on the first piece of equipment will continue to monitor information from the second piece of equipment until it detects an indication that the *Response* to the *Request* has been completed or has failed.

An example of this type of interaction between pieces of equipment can be represented by a machine tool that wants material to be loaded by a robot. In this example, the machine tool is the *Requester* and the robot is the *Responder*. On the other hand, if the robot wants the machine tool to open a door, the robot becomes the *Requester* and the machine tool the *Responder*.

3.2 *Request* and *Response* Information Exchange

The concept of a *Request* and *Response* information exchange is not unique to MTConnect *Interfaces*. This style of communication is used in many different types of environments and technologies.

An early version of a *Request* and *Response* information exchange was used by early sailors. When it was necessary to communicate between two ships before radio communications were available, or when secrecy was required, a sailor on each ship could communicate with the other using flags as a signaling device to request information or actions. The responding ship could acknowledge those requests for action and identify when the requested actions were completed.

The same basic *Request* and *Response* concept is implemented by MTConnect *Interfaces* using the `EVENT` data items defined in *Section 4*.

The `DataItem` elements defined by the *Interaction Model* each have a `Request` and `Response` subtype. These subtypes identify if the data item represents a *Request* or a *Response*. Using these data items, a piece of equipment changes the state of its *Request* or *Response* to indicate information that can be read by the other piece of equipment. To aid in understanding how the *Interaction Model* functions, one can view this *Interaction Model* as a simple state machine.

The interaction between two pieces of equipment can be described as follows. When the *Requester* wants an activity to be performed, it transitions its *Request* state from a `READY` state to an `ACTIVE` state. In turn, when the client on the *Responder* reads this information and interprets the *Request*, the *Responder* announces that it is performing the requested task by changing its response state to `ACTIVE`. When the action is finished, the *Responder* changes its response state to `COMPLETE`. This pattern of *Request* and *Response* provides the basis for the coordination of actions between pieces of equipment. These actions are implemented using `EVENT` category data items. (See *Section 4* for details on the `Event` type data items defined for *Interfaces*.)

Note: The implementation details of how the *Responder* piece of equipment reacts to the *Request* and then completes the requested task are up to the implementer.

The diagram below provides an example of the *Request* and *Response* state machine:

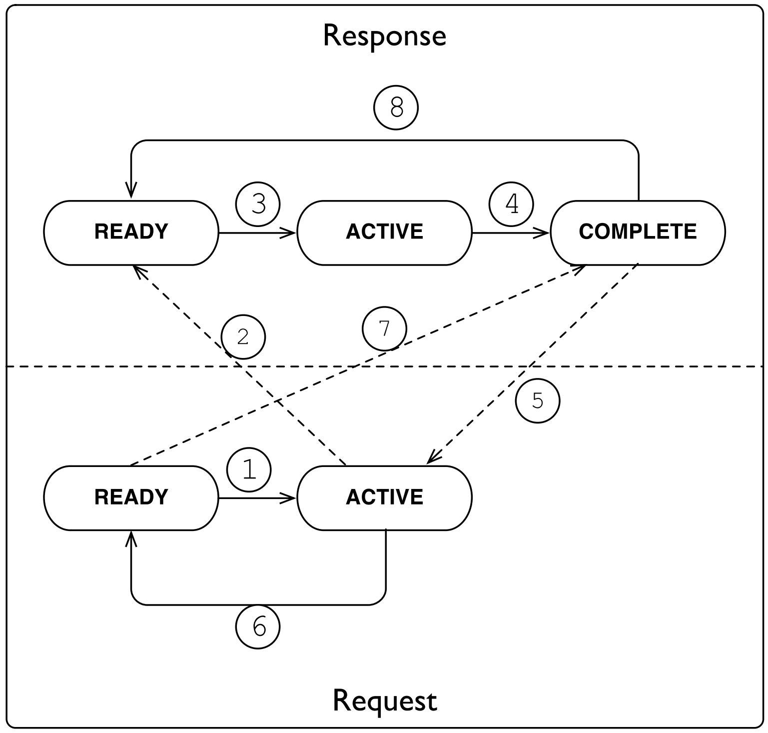


Figure 2: *Request* and *Response* Overview

The initial condition of both the *Request* and *Response* states on both pieces of equipment is `READY`. The dotted lines indicate the on-going communications that occur to monitor the progress of the interactions between the pieces of equipment.

The interaction between the pieces of equipment as illustrated in *Figure 2* progresses through the following sequence:

Step	Description
1	The *Request* transitions from `READY` to `ACTIVE` signaling that a service is needed.
2	The *Response* detects the transition of the *Request*.
3	The *Response* transitions from `READY` to `ACTIVE` indicating that it is performing the action.
4	Once the action has been performed, the *Response* transitions to `COMPLETE`.
5	The *Request* detects the action is `COMPLETE`.
6	The *Request* transitions back to `READY` acknowledging that the service has been performed.
7	The *Response* detects the *Request* has returned to `READY`.
8	In recognition of this acknowledgement, the *Response* transitions back to `READY`.

After the final action has been completed, both pieces of equipment are back in the `READY` state indicating that they are able to perform another action.

4 *Interfaces* for *Devices* and *Streams Information Models*

The *Interaction Model* for implementing *Interfaces* is defined in the MTConnect® Standard as an extension to the `MTConnectDevices` and `MTConnectStreams` XML documents.

A piece of equipment **MAY** support multiple different *Interfaces*. Each piece of equipment supporting *Interfaces* **MUST** organize the information associated with each *Interface* in a *Top Level* component called `Interfaces`. Each individual *Interface* is modeled as a *Lower Level* component called `Interface`. `Interface` is an abstract type XML element and will be replaced in the XML documents by specific `Interface` types defined below. The data associated with each *Interface* is modeled as data items within each of these *Lower Level* `Interface` components.

The following XML tree illustrates where `Interfaces` is modeled in the *Device Information Model* for a piece of equipment.

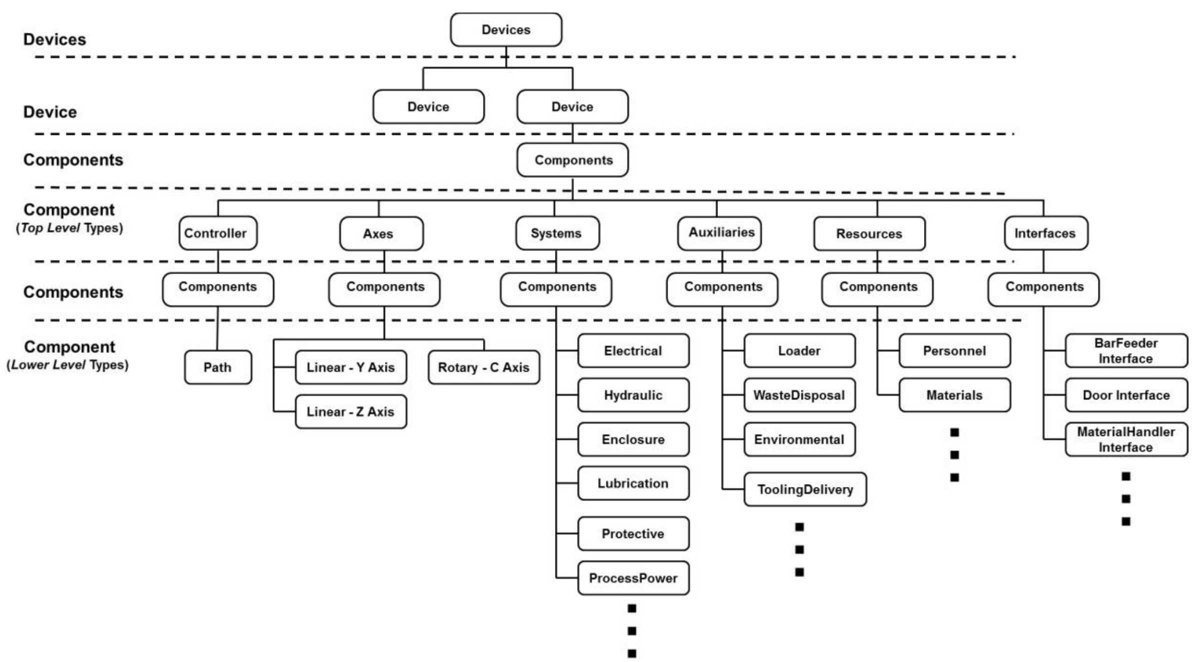


Figure 3: `Interfaces` as a *Structural Element*

4.1 Interfaces

`Interfaces` is an XML *Structural Element* in the `MTConnectDevices` XML document. `Interfaces` is a container type XML element. `Interfaces` is used to group information describing *Lower Level* `Interface` XML elements, which each provide information for an individual *Interface*.

If the `Interfaces` container appears in the XML document, it **MUST** contain one or more `Interface` type XML elements.

4.2 Interface

`Interface` is the next level of *Structural Element* in the `MTConnectDevices` XML document. As an abstract type XML element, `Interface` will be replaced in the XML documents by specific `Interface` types defined below.

Each `Interface` is also a container type element. As a container, the `Interface` XML element is used to organize information required to implement the *Interaction Model* for an *Interface*. It also provides structure for describing the *Lower Level Structural Elements* associated with the `Interface`. Each `Interface` contains *Data Entities* available from the piece of equipment that may be needed to coordinate activities with associated pieces of equipment.

The information provided by a piece of equipment for each *Interface* is returned in a `ComponentStream` container of an `MTConnectStreams` document in the same manner as all other types of components.

4.2.1 XML Schema Structure for `Interface`

The following XML schema represents the structure of an `Interface` XML element.

The schema for an `Interface` element is the same as defined for `Component` elements described in *Section 4.4* in *Part 2.0 – Devices Information Model* of the MTConnect Standard. The figure below shows the attributes defined for `Interface` and the elements that may be associated with `Interface`.

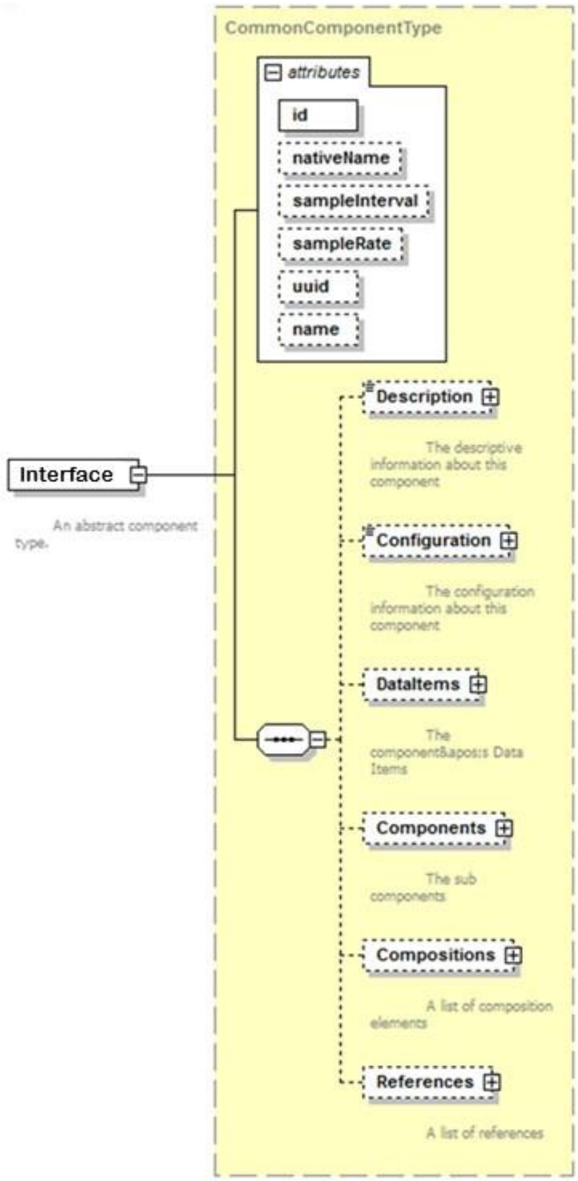


Figure 4: `Interface` Schema

Refer to *Part 2.0 – Devices Information Model*, *Section 4.4* for complete descriptions of the attributes and elements that are illustrated above for `Interface`.

4.2.2 `Interface` Types

As an abstract type XML element, `Interface` is replaced in the `MTConnectDevices` document with a XML element representing a specific type of *Interface*. An initial list of `Interface` types is defined below.

Interface	Description
`BarFeederInterface`	`BarFeederInterface` provides the set of information used to coordinate the operations between a Bar Feeder and another piece of equipment. Bar Feeder is a piece of equipment that pushes bar stock (i.e., long pieces of material of various shapes) into an associated piece of equipment – most typically a lathe or turning center.
`MaterialHandlerInterface`	`MaterialHandlerInterface` provides the set of information used to coordinate the operations between a piece of equipment and another associated piece of equipment used to automatically handle various types of materials or services associated with the original piece of equipment. A material handler is a piece of equipment capable of providing any one, or more, of a variety of support services for another piece of equipment or a process: Loading/unloading material or tooling Part inspection Testing Cleaning Etc. A robot is a common example of a material handler.
`DoorInterface`	`DoorInterface` provides the set of information used to coordinate the operations between two pieces of equipment, one of which controls the operation of a door. The piece of equipment that is controlling the door **MUST** provide the data item `Door_State` as part of the set of information provided.
`ChuckInterface`	`ChuckInterface` provides the set of information used to coordinate the operations between two pieces of equipment, one of which controls the operation of a chuck. The piece of equipment that is controlling the chuck **MUST** provide the data item `Chuck_State` as part of the set of information provided.

Note: Additional `Interface` types may be defined in future releases of the MTConnect Standard.

In order to implement the *Interaction Model* for *Interfaces*, each piece of equipment associated with an *Interface* **MUST** provide an `Interface` XML element for that type of *Interface*. A piece of equipment **MAY** support any number of unique *Interfaces*.

4.2.3 Data for `Interface`

Each *Interface* **MUST** provide (1) the data associated with the specific *Interface* to implement the *Interaction Model* and (2) any additional data that may be needed by another piece of equipment to understand the operating states and conditions of the first piece of equipment as it applies to the *Interface.*

Details on data items specific to the *Interaction Model* for each type of *Interface* are provided in *Section 4.2.4*.

An implementer may choose any other data available from a piece of equipment to describe the operating states and other information needed to support an *Interface.*

4.2.3.1 `References` for `Interface`

Some of the data items needed to support a specific *Interface* may already be defined elsewhere in the XML document for a piece of equipment. However, the implementer may not be able to directly associate this data with the *Interface* since the MTConnect Standard does not permit multiple occurrences of a piece of data to be configured in a XML document. `References` provides a mechanism for associating information defined elsewhere in the *Information Model* for a piece of equipment with a specific *Interface*.

`References` is an XML container that organizes pointers to information defined elsewhere in the XML document for a piece of equipment. `References` **MAY** contain one or more `Reference` XML elements.

`Reference` is an XML element that provides an individual pointer to information that is associated with another *Structural Element* or *Data Entity* defined elsewhere in the XML document that is also required for an *Interface*.

`References` is an economical syntax for providing interface specific information without directly duplicating the occurrence of the data. It provides an efficient, near-time, information flow between pieces of equipment.

For more information on the definition for `References` and `Reference`, see *Section 4.7* and *4.8* of *Part 2.0 - Devices Information Model*.

4.2.4 Data Items for `Interface`

Each `Interface` XML element contains data items which are used to communicate information required to execute the *Interface*. When these data items are read by another piece of equipment, that piece of equipment can then determine the actions that it may take based upon that data.

Some data items **MAY** be directly associated with the `Interface` element and others will be organized in a *Lower Level* `References` XML element.

It is up to an implementer to determine which additional data items are required for a particular *Interface*.

The data items that have been specifically defined to support the implementation of an *Interface* are provided below.

4.2.4.1 **INTERFACE_STATE for Interface**

INTERFACE_STATE is a data item specifically defined for *Interfaces*. It defines the operational state of the *Interface*. This is an indicator identifying whether the *Interface* is functioning or not.

An INTERFACE_STATE data item **MUST** be defined for every Interface XML element.

INTERFACE_STATE is reported in the MTConnectStreams XML document as InterfaceState. InterfaceState reports one of two states – ENABLED or DISABLED, which are provided in the CDATA for InterfaceState.

The table below shows both the INTERFACE_STATE data item as defined in the MTConnectDevices document and the corresponding *Element Name* that **MUST** be reported in the MTConnectStreams document.

EVENT **Data Item Type**	**Event** ***Element Name***	**Description and** **Valid Data Values**
INTERFACE_STATE	InterfaceState	The current functional or operational state of an Interface type element indicating whether the *Interface* is active or not currently functioning. Valid Data Values: - ENABLED: The *Interface* is currently operational and performing as expected. - DISABLED: The *Interface* is currently not operational. When the INTERFACE_STATE is DISABLED, the state of all data items that are specific for the *Interaction Model* associated with that *Interface* **MUST** be set to NOT_READY.

4.2.4.2 Specific Data Items for the *Interaction Model* for `Interface`

A special set of data items have been defined to be used in conjunction with `Interface` type elements. When modeled in the `MTConnectDevices` document, these data items are all *Data Entities* in the `EVENT` category (See *Part 3.0 – Streams Information Model* for details on how the corresponding data items are reported in the `MTConnectStreams` document). They provide information from a piece of equipment to *Request* a service to be performed by another associated piece of equipment; and for the associated piece of equipment to indicate its progress in performing its *Response* to the *Request* for service.

Many of the data items describing the services associated with an *Interface* are paired to describe two distinct actions – one to *Request* an action to be performed and a second to reverse the action or to return to an original state. For example, a `DoorInterface` will have two actions `OPEN_DOOR` and `CLOSE_DOOR`. An example of an implementation of this would be a robot that indicates to a machine that it would like to have a door opened so that the robot could extract a part from the machine and then asks the machine to close that door once the part has been removed.

When these data items are used to describe a service associated with an *Interface*, they **MUST** have one of the following two `subType` elements: `REQUEST` or `RESPONSE`. These `subType` elements **MUST** be specified to define whether the piece of equipment is functioning as the *Requester* or *Responder* for the service to be performed. The *Requester* **MUST** specify the `REQUEST` `subType` for the data item and the *Responder* **MUST** specify a corresponding `RESPONSE` `subType` for the data item to enable the coordination between the two pieces of equipment.

These data items and their associated `subType` provide the basic structure for implementing the *Interaction Model* for an *Interface*.

The table below provides a list of the data items that have been defined to identify the services to be performed for or by a piece of equipment associated with an *Interface*.

The table also provides the corresponding transformed *Element Name* for each data item that **MAY** be returned by an *MTConnect Agent* as an `Event type` XML *Data Entity* in the `MTConnectStreams` XML document. The *Controlled Vocabulary* for each of these data items are defined below in *Section 4.2.4.3*.

EVENT Data Item Type	**Event *Element Name***	**Description**
`MATERIAL_FEED`	`MaterialFeed`	Service to advance material or feed product to a piece of equipment from a continuous or bulk source.
`MATERIAL_CHANGE`	`MaterialChange`	Service to change the type of material or product being loaded or fed to a piece of equipment.
`MATERIAL_RETRACT`	`MaterialRetract`	Service to remove or retract material or product.
`PART_CHANGE`	`PartChange`	Service to change the part or product associated with a piece of equipment to a different part or product.
`MATERIAL_LOAD`	`MaterialLoad`	Service to load a piece of material or product.
`MATERIAL_UNLOAD`	`MaterialUnload`	Service to unload a piece of material or product.
`OPEN_DOOR`	`OpenDoor`	Service to open a door.
`CLOSE_DOOR`	`CloseDoor`	Service to close a door.
`OPEN_CHUCK`	`OpenChuck`	Service to open a chuck.
`CLOSE_CHUCK`	`CloseChuck`	Service to close a chuck

4.2.4.3 `Event` States for *Interfaces*

For each of the data items above, the *Valid Data Values* for the `CDATA` that is returned for these data items in the `MTConnectStreams` document is defined by a *Controlled Vocabulary*. This *Controlled Vocabulary* represents the state information to be communicated by a piece of equipment for the data items defined in the table above.

The *Request* portion of the *Interaction Model* for *Interfaces* has four states as defined in the table below:

Request State	Description
`NOT_READY`	The *Requester* is not ready to make a *Request*.
`READY`	The *Requester* is prepared to make a *Request*, but no *Request* for service is required. The *Requester* will transition to `ACTIVE` when it needs a service to be performed.
`ACTIVE`	The *Requester* has initiated a *Request* for a service and the service has not yet been completed by the *Responder*.
`FAIL`	CONDITION 1: When the *Requester* has detected a failure condition, it indicates to the *Responder* to either not initiate an action or stop its action before it completes by changing its state to `FAIL`. CONDITION 2: If the *Responder* changes its state to `FAIL`, the *Requester* **MUST** change its state to `FAIL`. ACTIONS: After detecting a failure, the *Requester* **SHOULD NOT** change its state to any other value until the *Responder* has acknowledged the `FAIL` state by changing its state to `FAIL`. Once the `FAIL` state has been acknowledged by the *Responder*, the *Requester* may attempt to clear its `FAIL` state. As part of the attempt to clear the `FAIL` state, the *Requester* **MUST** reset any partial actions that were initiated and attempt to return to a condition where it is again ready to perform a service. If the recovery is successful, the *Requester* changes its *Request* state from `FAIL` to `READY`. If for some reason the *Requester* is not again prepared to perform a service, it transitions its state from `FAIL` to `NOT_READY`.

The following diagram shows a graphical representation of the possible state transitions for a *Request*:

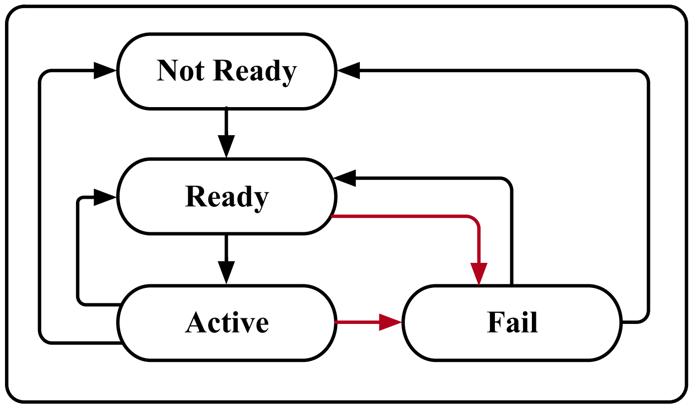


Figure 5: *Request* State Diagram

The *Response* portion of the *Interaction Model* for *Interfaces* has five states as defined in the table below:

Response State	Description
`NOT_READY`	The *Responder* is not ready to perform a service.
`READY`	The *Responder* is prepared to react to a *Request*, but no *Request* for service has been detected. The *Responder* **MUST** transition to **`ACTIVE`** to inform the *Requester* that it has detected and accepted the *Request* and is in the process of performing the requested service. If the *Responder* is not ready to perform a *Request*, it **MUST** transition to a `NOT_READY` state.
`ACTIVE`	The *Responder* has detected and accepted a *Request* for a service and is in the process of performing the service, but the service has not yet been completed. In normal operation, the *Responder* **MUST NOT** change its state to `ACTIVE` unless the *Requester* state is `ACTIVE`.
`FAIL`	CONDITION 1: The *Responder* has failed while executing the actions required to perform a service and the service has not yet been completed or the *Responder* has detected that the *Requestor* has unexpectedly changed state. CONDITION 2: If the *Requester* changes its state to `FAIL`, the *Responder* **MUST** change its state to `FAIL`. ACTIONS: After entering a `FAIL` state, the *Responder* **SHOULD NOT** change its state to any other value until the *Requester* has acknowledged the `FAIL` state by changing its state to `FAIL`. Once the `FAIL` state has been acknowledged by the *Requester*, the *Responder* may attempt to clear its `FAIL` state. As part of the attempt to clear the `FAIL` state, the *Responder* **MUST** reset any partial actions that were initiated and attempt to return to a condition where it is again ready to perform a service. If the recovery is successful, the *Responder* changes its *Response* state from `FAIL` to `READY`. If for some reason the *Responder* is not again prepared to perform a service, it transitions its state from `FAIL` to `NOT_READY`.
`COMPLETE`	The *Responder* has completed the actions required to perform the service. The *Responder* **MUST** remain in the `COMPLETE` state until the *Requester* acknowledges that the service is complete by changing its state to `READY`. At that point, the *Responder* **MUST** change its state to either `READY` if it is again prepared to perform a service or `NOT_READY` if it is not prepared to perform a service.

The state values described in the above tables **MUST** be provided in the CDATA for each of the *Interface* specific data items provided in the `MTConnectStreams` document.

The following diagram shows a graphical representation of the possible state transitions for a *Response*:

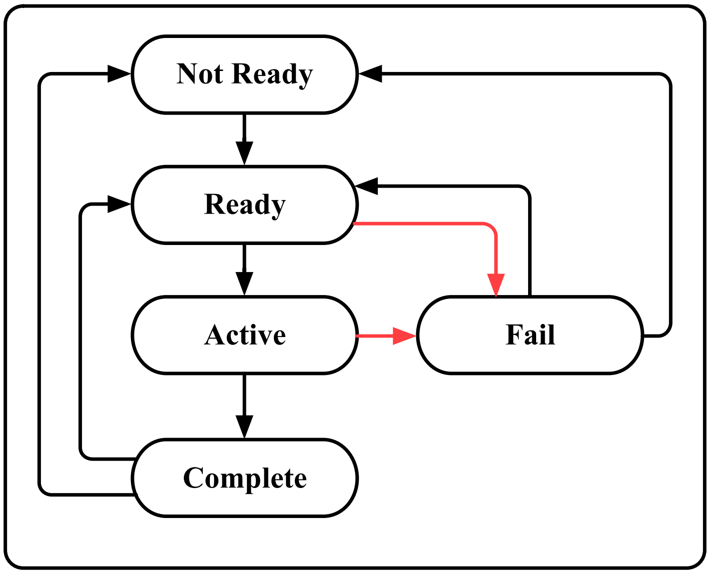


Figure 6: *Response* State Diagram

5 Operation and Error Recovery

The *Request/Response* state model implemented for *Interfaces* may also be represented by a graphical model. The following scenario demonstrates the state transitions that occur during a successful *Request* for service and the resulting *Response* to fulfill that service *Request*.

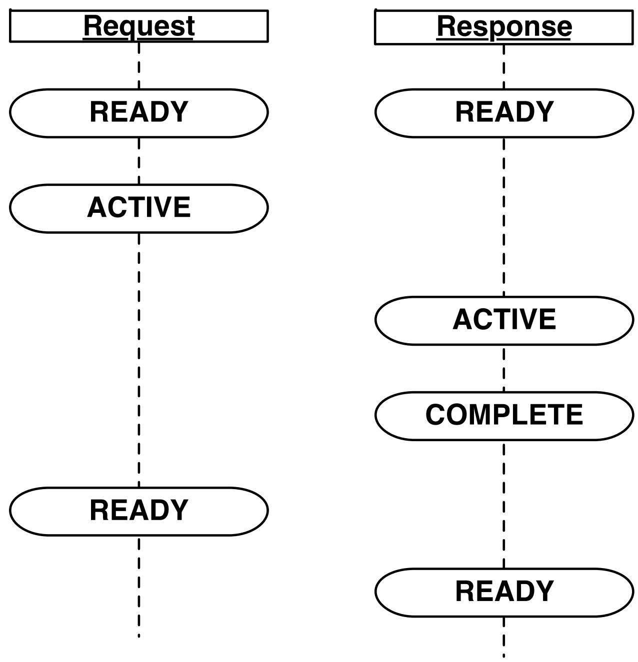


Figure 7: Success Scenario

5.1 *Request/Response* Failure Handling and Recovery

A significant feature of the *Request/Response Interaction Model* is the ability for either piece of equipment to detect a failure associated with either the *Request* or *Response* actions. When either a failure or unexpected action occurs, the *Request* and the *Response* portion of the *Interaction Model* can announce a `FAIL` state upon detecting a problem. The following are graphical models describing multiple scenarios where either the *Request*er or *Responder* detects and reacts to a failure. In these examples, either the *Requester* or *Responder* announces the detection of a failure by setting either the *Request* or the *Response* state to `FAIL`.

Once a failure is detected, the *Interaction Model* provides information from each piece of equipment as they attempt to recover from a failure, reset all of their functions associated with the *Interface* to their original state, and return to normal operation.

The following are scenarios that describe how pieces of equipment may react to different types of failures and how they indicate when they are again ready to request a service or respond to a request for service after recovering from those failures:

<u>Scenario #1 – *Responder* Fails Immediately</u>

In this scenario, a failure is detected by the *Responder* immediately after a *Request* for service has been initiated by the *Requester*.

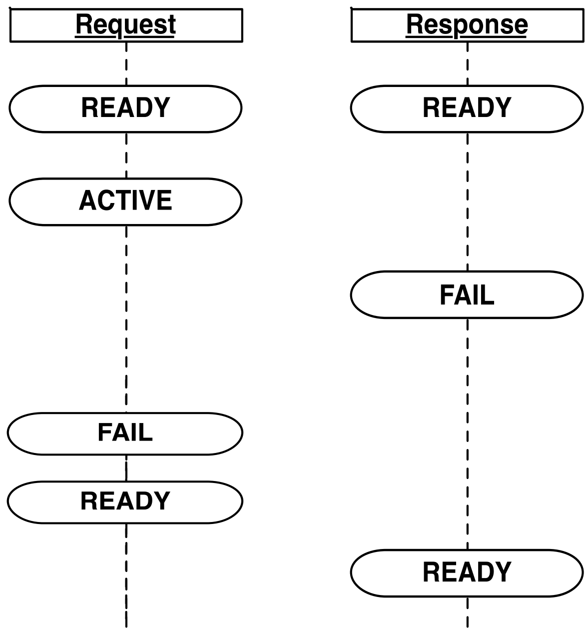


Figure 8: *Responder* – Immediate Failure

In this case, the *Request* transitions to `ACTIVE` and the *Responder* immediately detects a failure before it can transition the *Response* state to `ACTIVE`. When this occurs, the *Responder* transitions the *Response* state to `FAIL`.

After detecting that the *Responder* has transitioned its state to `FAIL`, the *Requester* **MUST** change its state to `FAIL`.

The *Requester*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the *Requester* changes its state from `FAIL` to `READY`. If for some reason the *Requester* cannot return to a condition where it is again ready to request a service, it transitions its state from `FAIL` to `NOT_READY`.

The *Responder*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *Responder* changes its *Response* state from `FAIL` to `READY`. If for some reason the *Responder* is not again prepared to perform a service, it transitions its state from `FAIL` to `NOT_READY`.

Scenario #2 – *Responder* Fails While Providing a Service

This is the most common failure scenario. In this case, the *Responder* will begin the actions required to provide a service. During these actions, the *Responder* detects a failure and transitions its *Response* state to `FAIL`.

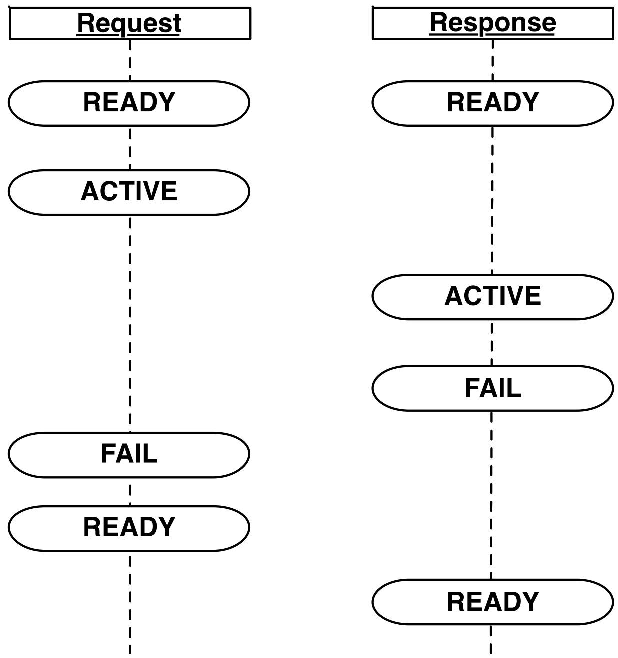


Figure 9: *Responder* Fails While Providing a Service

When a *Requester* detects a failure of a *Responder*, it transitions it state from `ACTIVE` to `FAIL`.

The *Requester* resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the *Requester* changes its state from `FAIL` to `READY` if the failure has been cleared and it is again prepared to request another service. If for some reason the *Requester* cannot return to a condition where it is again ready to request a service, it transitions its state from `FAIL` to `NOT_READY`.

The *Responder*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *Responder* changes its *Response* state from `FAIL` to `READY` if it is again prepared to perform a service. If for some reason the *Responder* is not again prepared to perform a service, it transitions its state from `FAIL` to `NOT_READY`.

<u>Scenario #3 – *Requester* Failure During a Service *Request*</u>

In this scenario, the *Responder* will begin the actions required to provide a service. During these actions, the *Requester* detects a failure and transitions its *Request* state to `FAIL`.

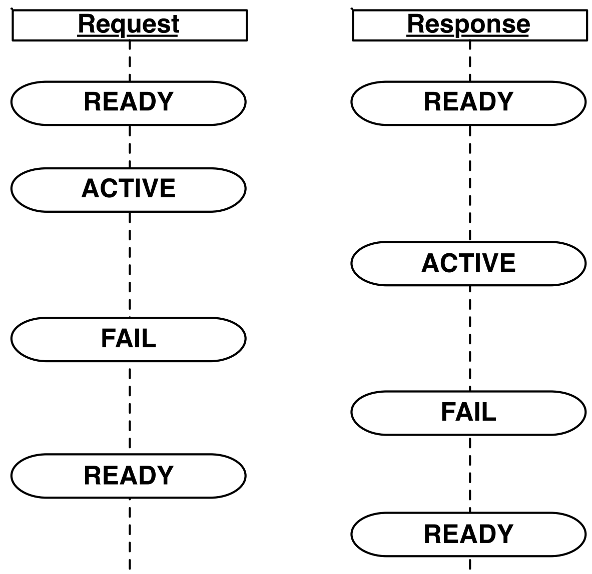


Figure 10: Requester Fails During a Service Request

When the *Responder* detects that the *Requester* has transitioned its *Request* state to `FAIL`, the *Responder* also transitions its *Response* state to `FAIL`.

The *Requester*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the *Requester* changes its state from `FAIL` to `READY`. If for some reason the *Requester* cannot return to a condition where it is again ready to request a service, it transitions its state from `FAIL` to `NOT_READY`.

The *Responder*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *Responder* changes its *Response* state from `FAIL` to `READY`. If for some reason the *Responder* is not again prepared to perform a service, it transitions its state from `FAIL` to `NOT_READY`.

Scenario #4 – *Requester* Changes to an Unexpected State While *Responder* is Providing a Service

In some cases, a *Requester* may transition to an unexpected state after it has initiated a *Request* for service.

As demonstrated below, the *Requester* has initiated a *Request* for service and its *Request* state has been changed to `ACTIVE`. The *Responder* begins the actions required to provide the service. During these actions, the *Requester* transitions its *Request* state back to `READY` before the *Responder* can complete its actions. This **SHOULD** be regarded as a failure of the *Requester*.

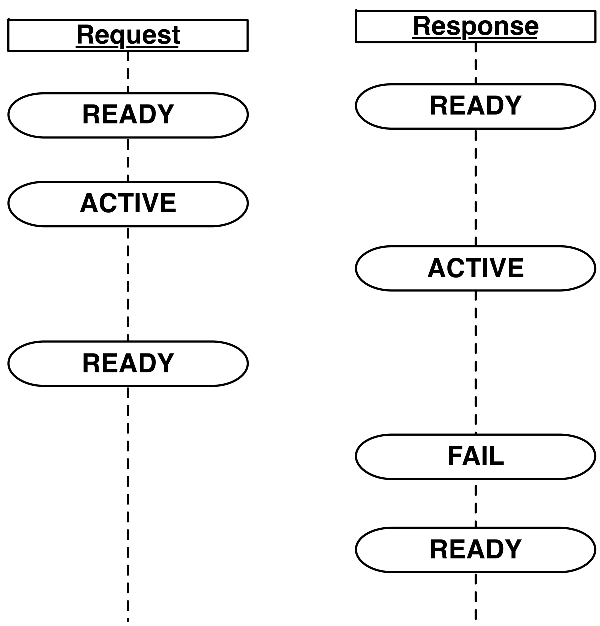


Figure 11: *Requester* Makes Unexpected State Change

In this case, the *Responder* reacts to this change of state of the *Requester* in the same way as though the *Requester* had transitioned its *Request* state to `FAIL` (i.e., the same as in Scenario #3 above).

At this point, the *Responder* then transitions its *Response* state to `FAIL`.

The *Responder* resets any partial actions that were initiated and attempts to return to its original condition where it is again ready to perform a service. If the recovery is successful, the *Responder* changes its *Response* state from `FAIL` to `READY`. If for some reason the *Responder* is not again prepared to perform a service, it transitions its state from `FAIL` to `NOT_READY`.

Note: The same scenario exists if the *Requester* transitions its *Request* state to `NOT_READY`. However, in this case, the *Requester* then transitions its *Request* state to `READY` after it resets all of its functions back to a condition where it is again prepared to make a *Request* for service.

Scenario #5 – *Responder* Changes to an Unexpected State While Providing a Service

Similar to Scenario #5, a *Responder* may transition to an unexpected state while providing a service.

As demonstrated below, the *Responder* is performing the actions to provide a service and the *Response* state is `ACTIVE`. During these actions, the *Responder* transitions its state to `NOT_READY` before completing its actions. This should be regarded as a failure of the *Responder*.

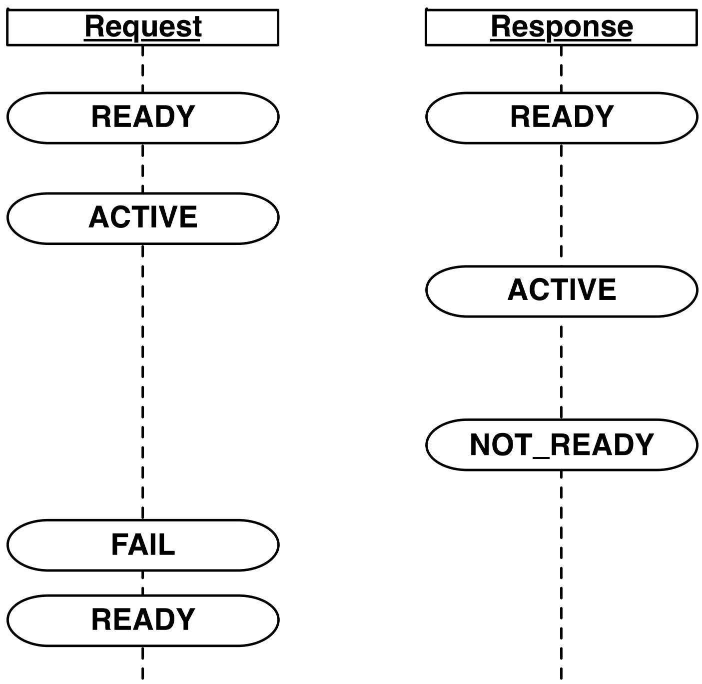


Figure 12: *Responder* Makes Unexpected State Change

Upon detecting an unexpected state change of the *Responder*, the *Requester* transitions its state to `FAIL`.

The *Requester* resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the *Requester* changes its state from `FAIL` to `READY`. If for some reason the *Requester* cannot return to a condition where it is again ready to request a service, it transitions its state from `FAIL` to `NOT_READY`.

Since the *Responder* has failed to an invalid state, the condition of the *Responder* is unknown. Where possible, the *Responder* should try to reset to an initial state.

The *Responder*, as part of clearing the cause for the change to the unexpected state, should attempt to reset any partial actions that were initiated and then return to a condition where it is again ready to perform a service. If the recovery is successful, the *Responder* changes its *Response* state from the unexpected state to `READY`. If for some reason the *Responder* is not again prepared to perform a service, it maintains its state as `NOT_READY`.

Scenario #6 – *Responder* or *Requester* Become `UNAVAILABLE` or Experience a Loss of Communications

In this scenario, a failure occurs in the communications connection between the *Responder* and *Requester*. This failure may result from the `InterfaceState` from either piece of equipment returning a value of `UNAVAILABLE` or one of the pieces of equipment does not provide a heartbeat within the desired amount of time (See *Part 1.0 - Overview and Functionality* for details on heartbeat).

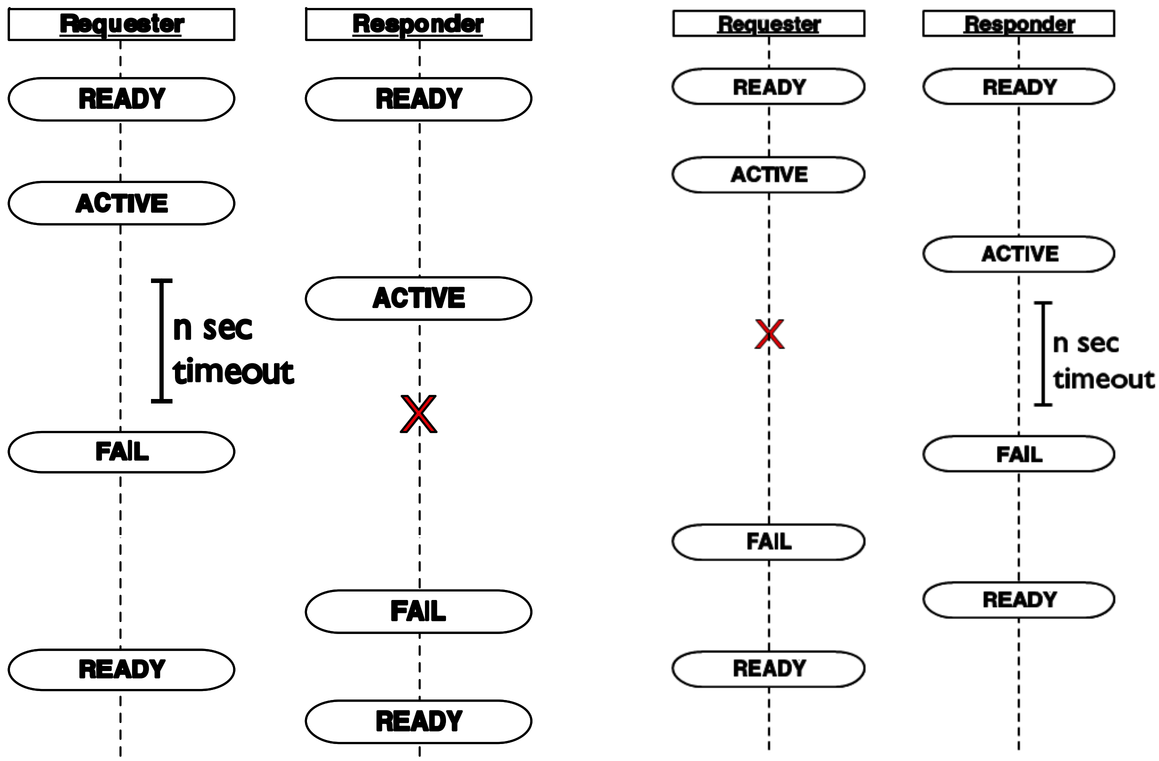


Figure 13: *Requester/Responder* Communication Failures

When one of these situations occurs, each piece of equipment assumes that there has been a failure of the other piece of equipment.

When normal communications are re-established, neither piece of equipment should assume that the *Request/Response* state of the other piece of equipment remains valid. Both pieces of equipment should set their state to `FAIL`.

The *Requester*, as part of clearing its `FAIL` state, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the *Requester* changes its state from `FAIL` to `READY`. If for some reason the *Requester* cannot return to a condition where it is again ready to request a service, it transitions its state from `FAIL` to `NOT_READY`.

The *Responder*, as part of clearing its `FAIL` state, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *Responder* changes its *Response* state from `FAIL` to `READY`. If for some reason the *Responder* is not again prepared to perform a service, it transitions its state from `FAIL` to `NOT_READY`.

Appendices

A. Bibliography

1. Engineering Industries Association. *EIA Standard - EIA-274-D*, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.

2. ISO TC 184/SC4/WG3 N1089. *ISO/DIS 10303-238*: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.

3. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.

4. International Organization for Standardization. *ISO 14649*: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.

5. International Organization for Standardization. *ISO 6983/1* – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.

6. Electronic Industries Association. *ANSI/EIA-494-B-1992*, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.

7. National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.

8. International Organization for Standardization. *ISO 10303-11*: 1994, Industrial automation systems and integration Product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

9. International Organization for Standardization. *ISO 10303-21*: 1996, Industrial automation systems and integration -- Product data representation and exchange -- Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.

10. H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.

11. International Organization for Standardization. *ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature.* Geneva, Switzerland, 2001.

12. ASME B5.57: *Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers,* 1998

13. ASME/ANSI B5.54: *Methods for Performance Evaluation of Computer Numerically Controlled Machining Centers. 2005.*

14. OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.*

15. IEEE STD 1451.0-2007*, Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH99684, October 5, 2007.*

16. IEEE STD 1451.4-1994*, Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH95225, December 15, 2004.*