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:

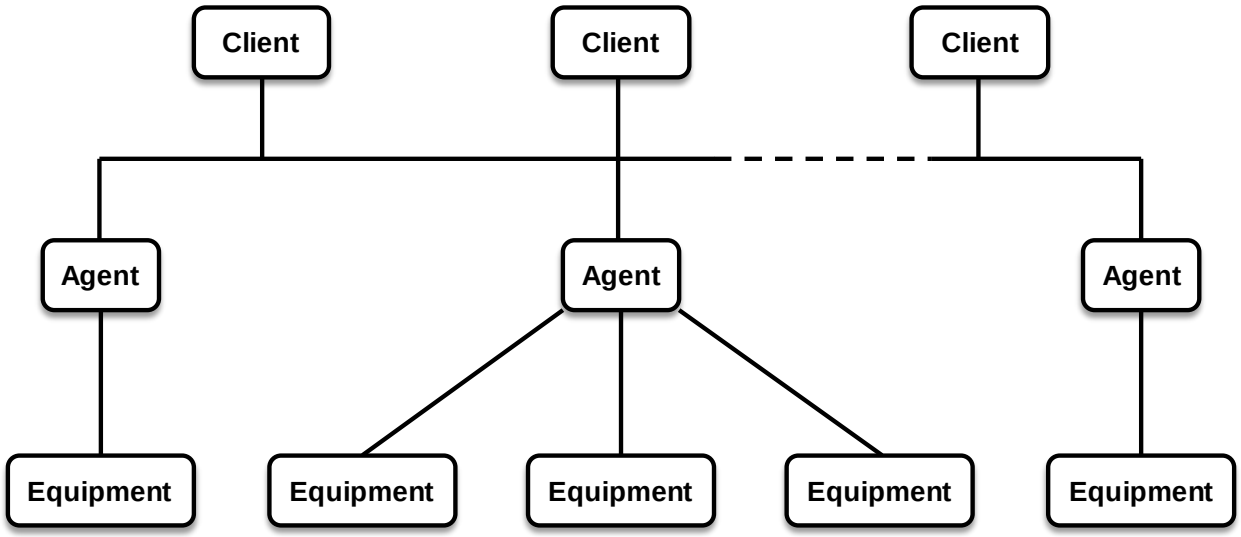


Figure 1: Basic MTConnect Implementation Structure

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.)	**Used when referring to an *MTConnect Asset*:** 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*. **Used as an XML container or element:** • 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`. **Used to describe information stored in an *MTConnect Agent*:** 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.* **Used as an XML representation of an *MTConnect Response document*:** 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`. **Used as an *MTConnect Request*:** 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*. **Used as part of an *HTTP Request*:** 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*.

Term	Definition as Used in the MTConnect Standard
Client	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.
Component	**General meaning:** 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*. **Used in *Information Models*:** 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* `Component` elements. Appears in the documents in the following form: `Components`. • When used as an abstract XML element. `Component` is replaced in a data model by a type of *Component* element. `Component` is also an XML container used to organize *Lower Level* `Component` elements, *Data Entities*, or both. Appears in the documents in the following form: `Component`.

Term	Definition as Used in the MTConnect Standard
Composition	**General meaning:** Data modeling elements that describe the lowest level basic structural or functional building blocks contained within a `Component` element. Appears in the documents in the following form: *Composition Element.* **Used in *Information Models*:** • When used as an XML container to organize `Composition` elements. Appears in the documents in the following form: `Compositions`. • When used as an abstract XML element. `Composition` is replaced in a data model by a type of *Composition Element*. Appears in the documents in the following form: `Composition`.

Term	Definition as Used in the MTConnect Standard
Condition	**General meaning:** An indicator of the health of a piece of equipment or a *Component* and its ability to function. **Used as a modeling element:** 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)*. **Used in *Information Models*:** An XML element used to represent *Condition Elements*. • When used as an XML container to organize *Lower Level* `Condition` elements. Appears in the documents in the following form: `Condition`. • When used as a *Lower Level* element, the form `Condition` is an abstract type XML element. This *Lower Level* element is a *Data Entity*. `Condition` is replaced in a data model by type of *Condition* element. Appears in the documents in the following form: `Condition`. Note: The form `Condition` is used to represent both above uses.
Controlled Vocabulary	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*.

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*.

Term	Definition as Used in the MTConnect Standard
Device	A part of an information model representing a piece of equipment. **Used in an XML representation of a *Response Document*:** • When used as an XML container to organize `Device` elements. Appears in the documents in the following form: `Devices`. • 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: `Device`.
Document	**General meaning:** A piece of written, printed, or electronic matter that provides information. **Used to represent an MTConnect Document:** 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. **Used to represent a specific representation of an MTConnect Document:** 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*. **Used to describe types of information stored in an *MTConnect Agent*:** 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*. **Used to describe information published by an MTConnect Agent:** 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*.

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.

Term	Definition as Used in the MTConnect Standard
Error Information Model	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*.
Event	**General meaning:** The occurrence of something that happens or takes place. Appears in the documents in the following form: event. **Used as a type of *Data Entity*:** 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. **Used as a `category` attribute for a *Data Entity*:** Used as a value for the `category` attribute for an XML `dataItem` element. Appears in the documents in the following form: `EVENT`. **Used as an XML container or element:** • When used as an XML container that consists of one or more types of `Event` XML elements. Appears in the documents in the following form: `Events`. • When used as an abstract XML element. It is replaced in the XML document by types of `Event` elements. Appears in the documents in the following form: `Event`.
Extensible	The ability for an implementer to extend *MTConnect Information Models* by adding content not currently addressed in the MTConnect Standard.
Fault State	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*.

Term	Definition as Used in the MTConnect Standard
Heartbeat	**General meaning:** 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*. **When used as part of an *HTTP Request*:** 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	**General meaning:** An identifier used to distinguish a piece of information. Appears in the documents in the following form: id. **Used as an XML attribute:** 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*.

Term	Definition as Used in the MTConnect Standard
Instance	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*.
Interaction Model	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*.
Interface	**General meaning:** The exchange of information between pieces of equipment and/or software systems. Appears in the documents in the following form: interface. **Used as an *Interaction Model*:** An *Interaction Model* that describes a method for inter-operations between pieces of equipment. Appears in the documents in the following form: *Interface.* **Used as an XML container or element:** • When used as an XML container that consists of one or more types of `Interface` XML elements. Appears in the documents in the following form: `Interfaces`. • When used as an abstract XML element. It is replaced in the XML document by types of `Interface` elements. Appears in the documents in the following form: `Interface`.

Term	Definition as Used in the MTConnect Standard
Message	**General meaning:** The content of a communication process. Appears in the documents in the following form: message. **Used relative to an *MTConnect Agent*:** 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*. **Used as a type *of Data Entity*:** 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`. **Used as an *Element Name*:** 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*.

Term	Definition as Used in the MTConnect Standard
Sample	**General meaning:** The collection of one or more pieces of information. **Used when referring to the collection of information:** When referring to the collection of a piece of information from a data source. Appears in the documents in the following form: sample. **Used as an *MTConnect Request*:** 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.* **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 *Sample Request* to an *MTConnect Agent* to publish an `MTConnectStreams` document. Appears in the documents in the following form: `sample`. **Used to describe a *Data Entity*:** 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*. **Used as an XML container or element:** • When used as an XML container that consists of one or more types of `Sample` XML elements. Appears in the documents in the following form: `Samples`. • When used as an abstract XML element. It is replaced in the XML document by types of `Sample` elements representing individual *Sample* type of *Data Entity*. Appears in the documents in the following form: `Sample`.

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	**General meaning:** 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. **Used as an attribute for an XML element:** 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.

141

4 MTConnect Standard

The MTConnect® Standard is organized in a series of documents (also referred to as MTConnect Documents) 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:

```
1.  <MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:1.1"
2.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.    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.

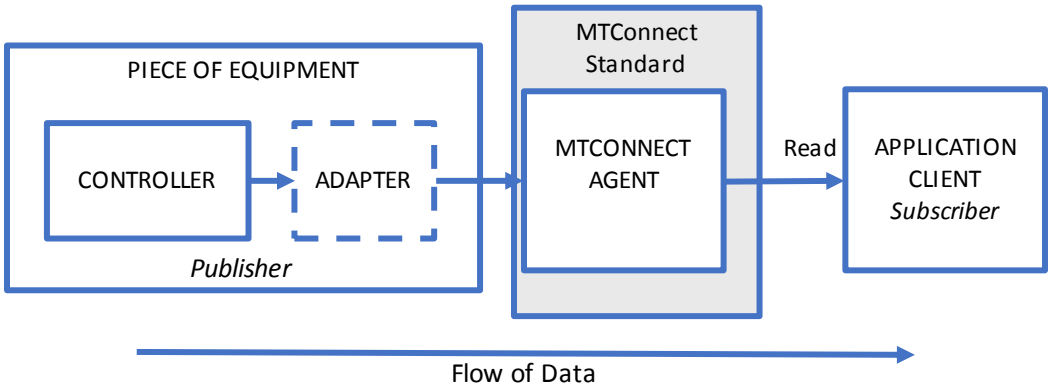


Figure 2: *MTConnect Architecture* Model

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 s*chemas* **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*

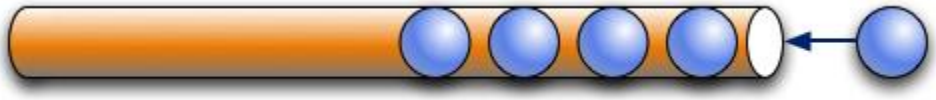
Streaming Data that is published from a piece(s) of equipment to an *MTConnect Agent* is stored by the *Agent* based upon the sequence upon which each piece of data is received. As described below, the order in which data is stored by the *Agent* is one of the factors that determines the data that may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

5.1.3.1 Management of *Streaming Data* Storage

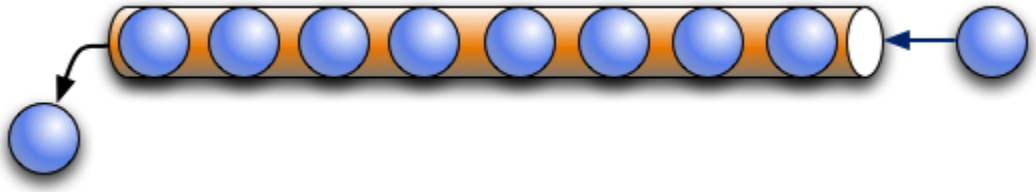
An *MTConnect Agent* stores a fixed amount of data. The amount of data stored by an *Agent* is dependent upon the implementation of a specific *MTConnect Agent*. The examples below demonstrate how discrete pieces of data received from pieces of equipment are stored.

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):

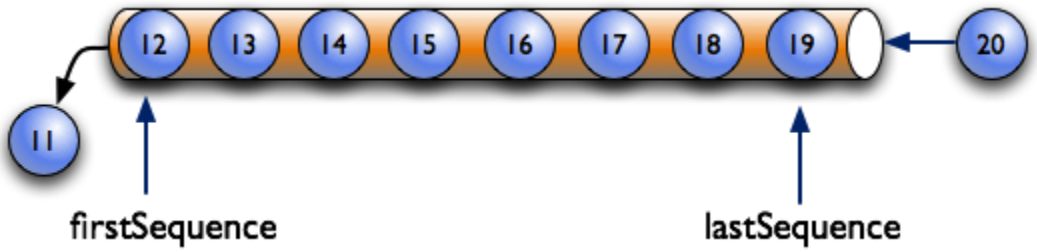
instanceId	**sequence**
234556	**234**
	235
	236
	237
	238
	Agent Stops and Restarts
234557	**1**
	2
	3
	4
	5

Figure 3: `instanceId` and `sequence`

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

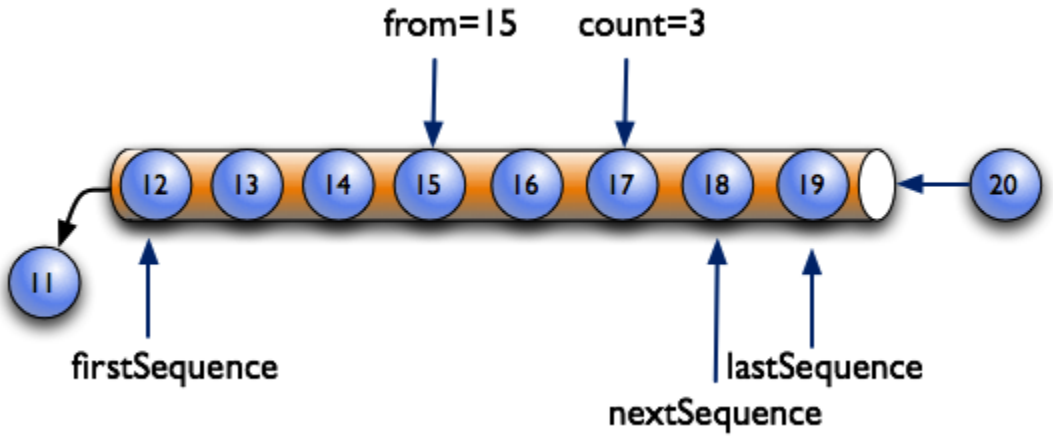
- `firstSequence` – the oldest piece of data contained in the *buffer*; i.e., the next piece of data to be moved out of the *buffer*
- `lastSequence` – the newest data added to the *buffer*

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



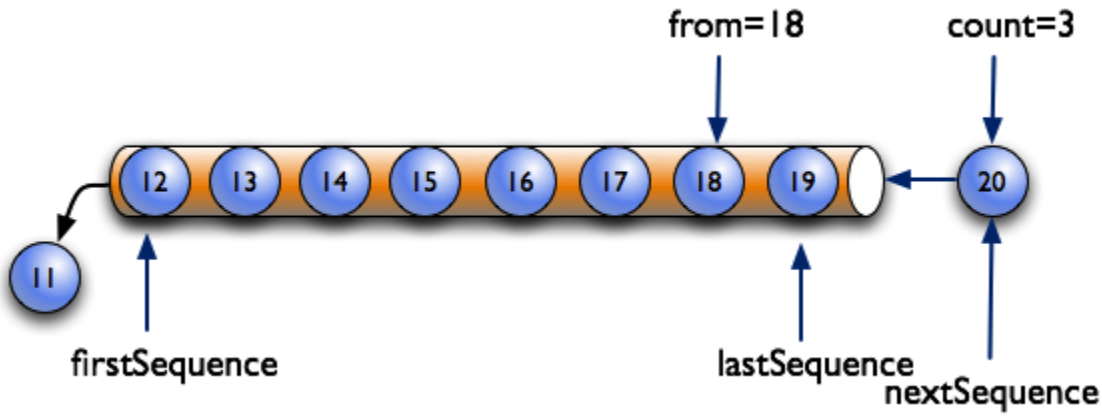
When a client software application requests data from an *MTConnect Agent*, it can specify both the *sequence number* of the first piece of data (`from`) that **MUST** be included in the Response Document and the total number (`count`) of pieces of data that **SHOULD** be included in that document.

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*.

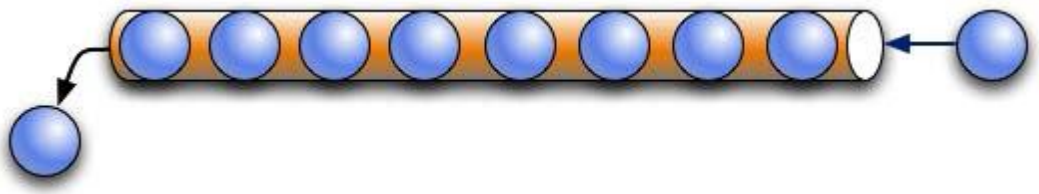
When a piece of equipment publishes a document that represents information associated with an *MTConnect Asset*, an *MTConnect Agent* stores that document in a *buffer*. This *buffer* is called the *assets buffer*. The document is called an *Asset Document*.

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.

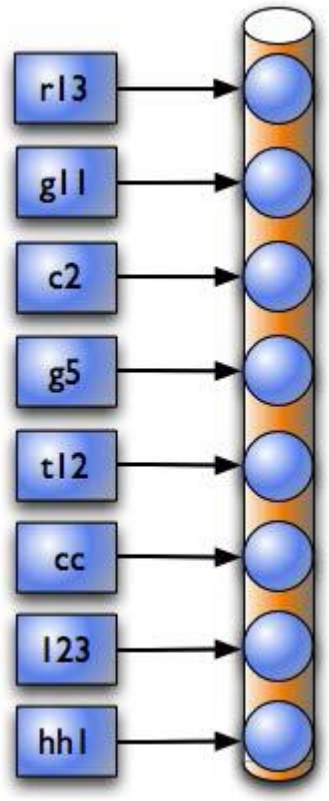
An *MTConnect Agent* will only retain a limited number of *Asset Documents* in the *assets buffer*. The *assets buffer* functions similar to the *buffer* for *Streaming Data*; i.e., when the *assets buffer* is full, the oldest *Assets Document* is pushed from the *buffer*.

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`.

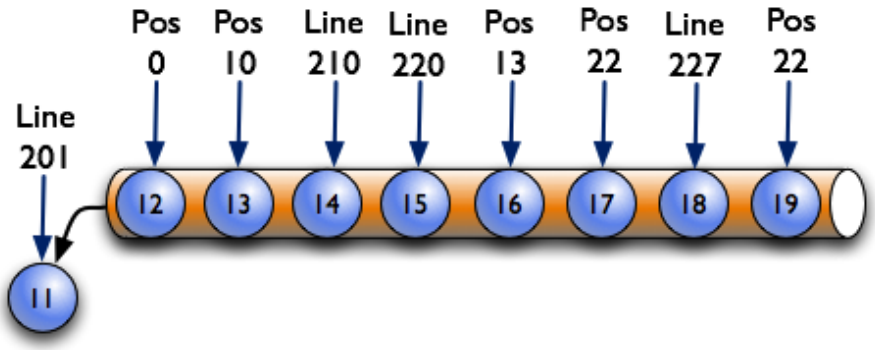


Figure 5: Example *Buffer*

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:

```
2.  <?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*.

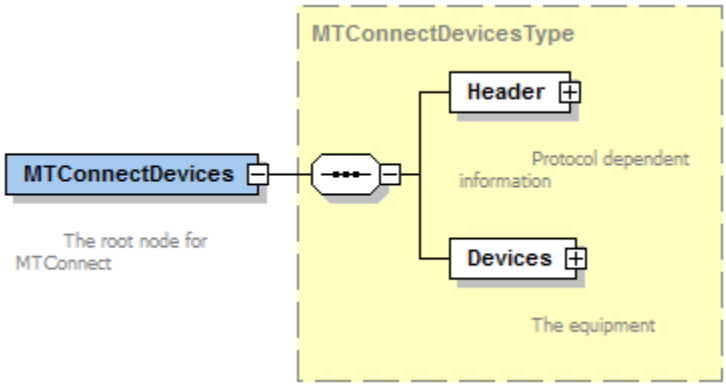


Figure 6: `MTConnectDevices` Structure

`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*.

1005 MTConnectDevices also has a number of attributes. These attributes are defined in *Section*
1006 *6.4, Schema and Namespace Declaration*.

1007 #### 6.3.1.1 MTConnectDevices Elements

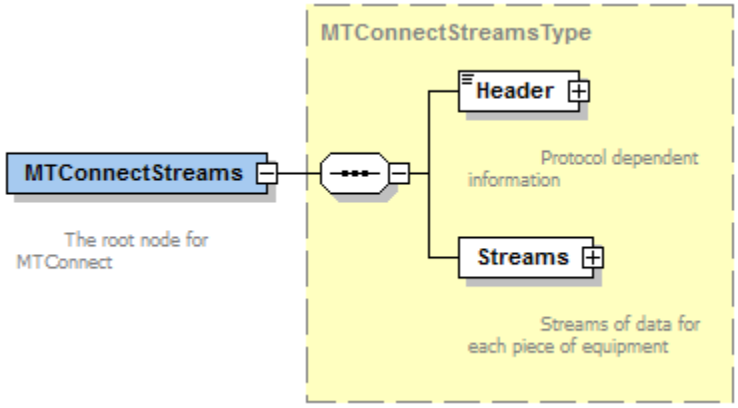
1008 An MTConnectDevices element **MUST** contain a Header and a Devices element.

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

1009

1010 ### 6.3.2 MTConnectStreams *Root Element*

1011 MTConnectStreams is the *Root Element* for the *MTConnectStreams Response Document*.



1012

1013 **Figure 7: MTConnectStreams Structure**

1014

1015 MTConnectStreams **MUST** contain two *Child Elements* - Header and Streams.

1016 Details for Header are defined in *Section 6.5, Document Header*.

1017 Streams is an XML container that represents the *Document Body* for a *MTConnectStreams*
1018 *Response Document* – see *Section 6.6*. Details for the *semantic data model* describing the
1019 contents for Streams are defined in *Part 3.0 - Streams Information Model*.

1020 MTConnectStreams also has a number of attributes. These attributes are defined in *Section*
1021 *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*.

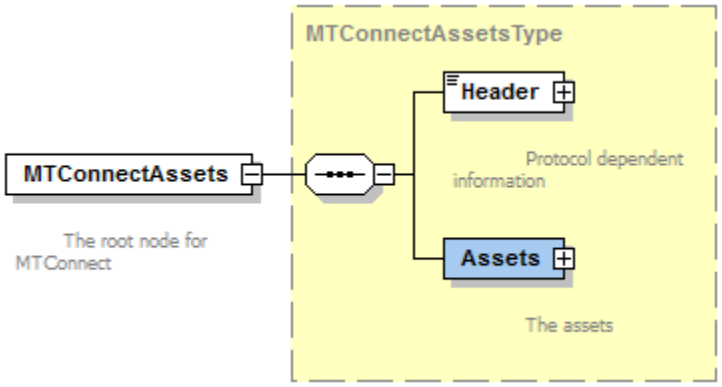


Figure 8: `MTConnectAssets` Structure

`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*.

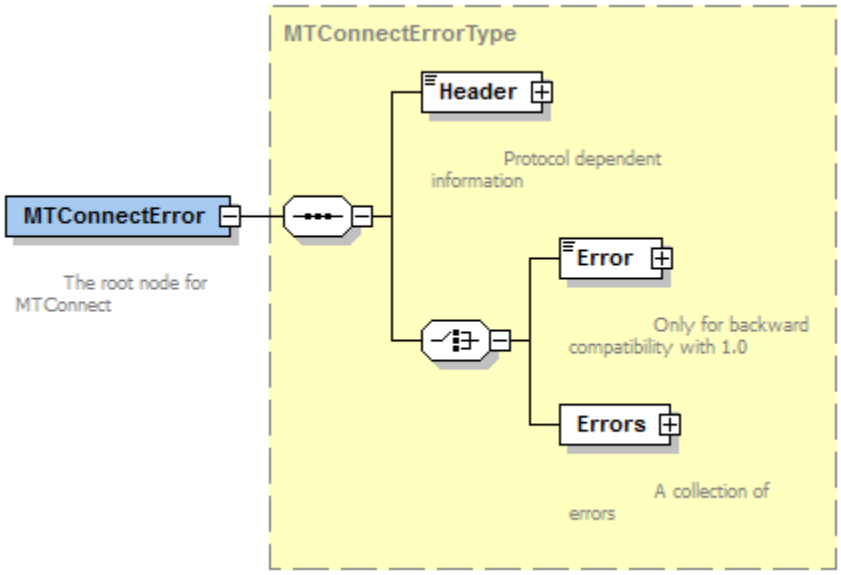


Figure 9: `MTConnectError` Structure

`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*.

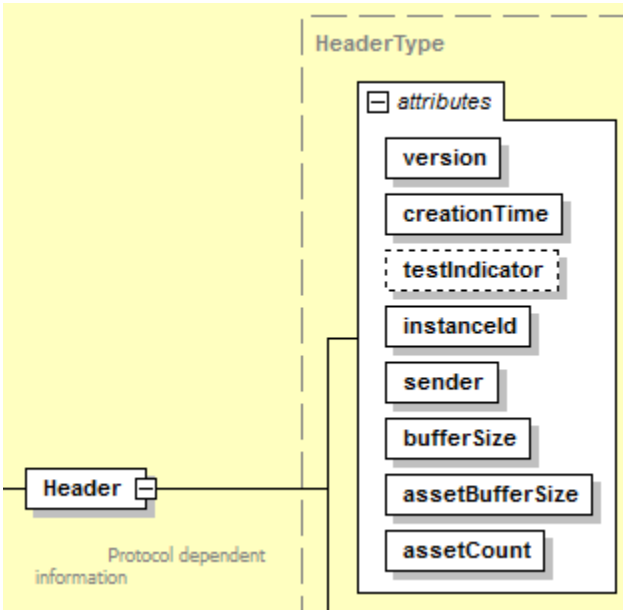


Figure 10: `Header` *Schema* Diagram for `MTConnectDevices`

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*.

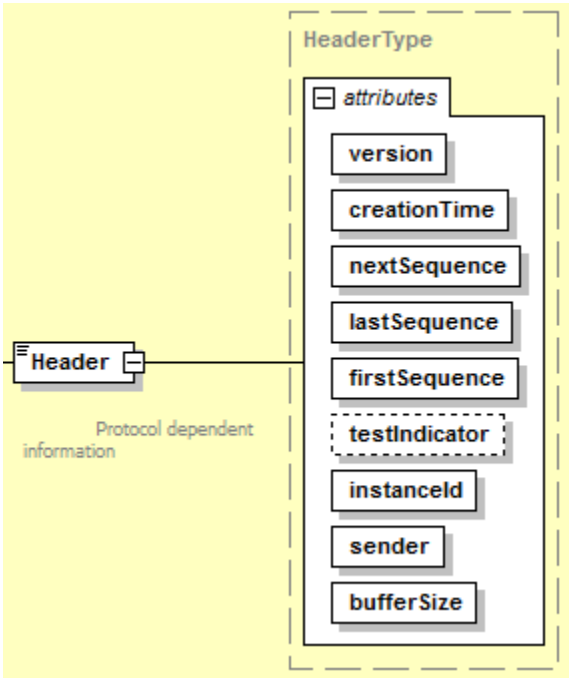


Figure 11: `Header` *Schema* Diagram for `MTConnectStreams`

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

1115

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*.

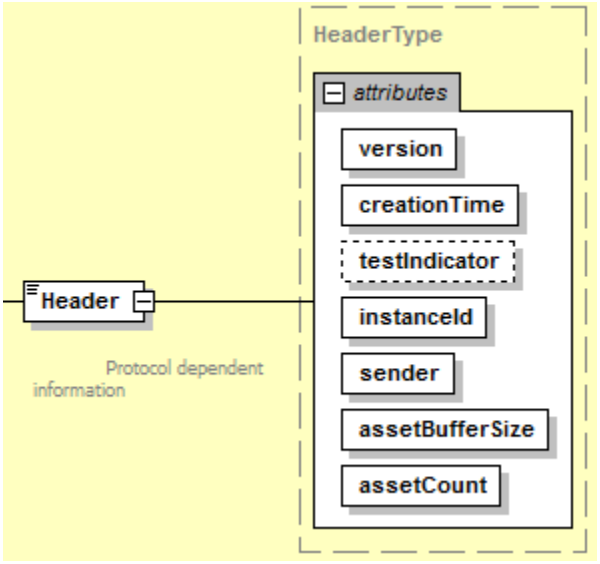


Figure 12: `Header` *Schema* Diagram for `MTConnectAssets`

1133 #### 6.5.3.2 `Attributes` for `Header` for `MTConnectAssets`

1134 The following table defines the attributes that may be used to provide additional information in
1135 the `Header` element for an *MTConnectAssets Response Document*.

1136

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

1137
1138

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*.

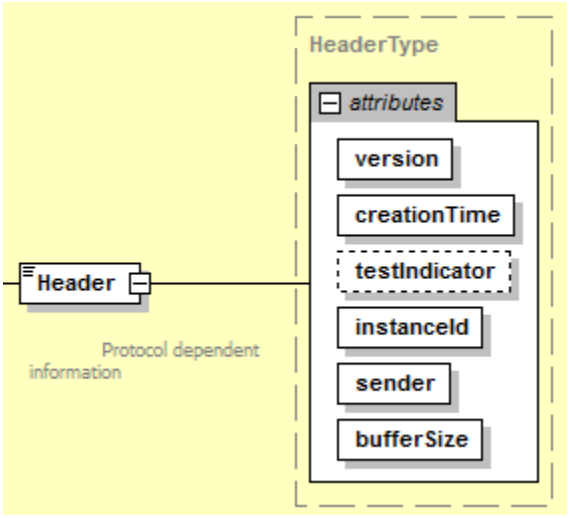


Figure 13: `Header` *Schema* Diagram for `MTConnectError`

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

1158

1159 The following is an example of a `Header` XML element for an *MTConnectError Response*
1160 *Document*:

```
1161   1.  <Header creationTime="2017-02-16T16:44:27Z" sender="MyAgent"
1162   2.    instanceId="1268463594" bufferSize="131072" version="1.4.0.10"/>
```

1163

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.

1406 The following `query` parameters **MUST** be supported in an *HTTP Request Line* for a *Current*
1407 *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*.

1408

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* **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. 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.

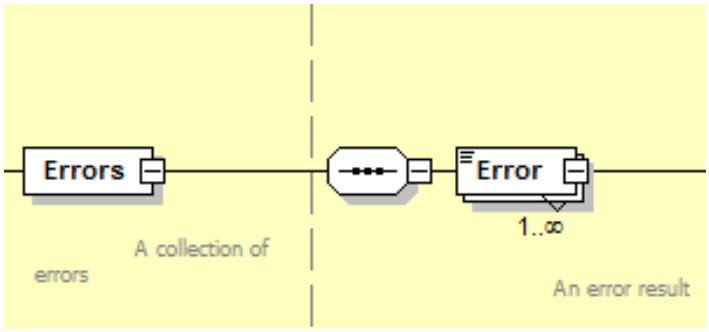


Figure 14: `Errors` *Schema* Diagram

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*:

```
1.  <MTConnectError>
2.    <Header/>
3.    <Errors>
4.      <Error/>
5.      <Error/>
6.      <Error/>
7.    </Errors>
8.  </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`.

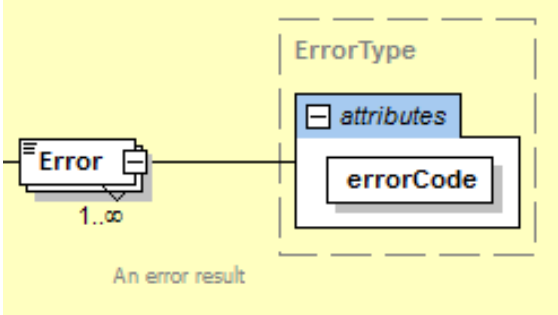


Figure 15: `Error` Schema Diagram

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:

```
1. <DataItem category="SAMPLE" id="S1load"nativeUnits="PERCENT"
2.   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:

```
1. <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*.

```
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.     xsi:schemaLocation="urn:mtconnect.org:
7.       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`.