MTConnect® Standard

Part 1.0 – Fundamentals

Version 2.1.0

Prepared for: MTConnect Institute
Prepared from: `MTConnectSysMLModel.xml`
Prepared on: January 14, 2023

MTConnect Specification and Materials

The normative XMI is located at the following URL: `MTConnectSysMLModel.xml`

Table of Contents

Table of Figures

List of Tables

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 model* 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 and response* and *publish and subscribe* type of communications. The *request and response* communications structure is used throughout this document to describe the functionality provided by MTConnect. See *Section 5.1.3.1 - Streaming Data* for details describing the functionality of the *publish and subscribe* communications structure available from an *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 model* to address their specific requirements. See *Section D - Extensibility* 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 model*. 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 *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 *agent*.
- Meaning 3: In an MTConnect implementation, the electronic documents generated by an *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 *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 *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 as shown in Figure 1.

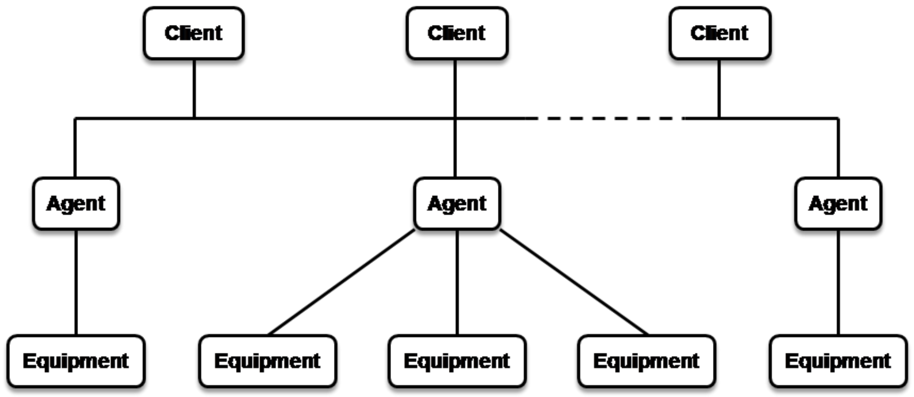

Figure 1: Basic MTConnect Implementation Structure

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.
- 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 *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 *agent*.

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

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

- the method used by a client software application to request information from an *agent*.
- the response that an *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 an *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 *MTConnect Standard: Part 2.0 - Device Information Model* and *MTConnect Standard: Part 3.0 - Observation 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 and response* messaging structure. This *interaction model* is called `Interfaces` which is detailed in *MTConnect Standard: Part 5.0 - Interface Interaction Model* 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 *Asset*. *Assets* are detailed in *MTConnect Standard: Part 4.0 - Asset Information Model*, and its sub-Parts, of the MTConnect Standard.

2 Purpose of This Document

This document, *MTConnect Standard Part 1.0 - Fundamentals* 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 *agent* should organize and structure data that has been collected from a data source.
- Definition and structure of the *response documents* supplied by an *agent*.
- The protocol used by a client software application to communicate with an *agent*.

3 Terminology and Conventions

This section provides a dictionary of terms, reserved language, and document conventions used in the MTConnect Standard.

3.1 General Terms

adapter

optional piece of hardware or software that transforms information provided by a piece of equipment into a form that can be received by an *agent*.

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 model* of a Standard.

alarm limit

limit used to trigger warning or alarm indicators.

application

software or a program that is specific to the solution of an application problem. *Ref ISO/IEC 20944-1:2013*

archetype

archetype provides the requirements, constraints, and common properties for a type of *Asset*.

asset buffer

buffer for *Assets*.

attachment

connection by which one thing is associated with another.

buffer

section of an *agent* that provides storage for information published from pieces of equipment.

cartesian coordinate system

3D orthogonal coordinate system [(]ISO/IEC 19794-5:2011en).

client

application that sends *request* for information to an *agent*.

Note: Examples include software applications or a function that implements the *request* portion of an *interface interaction model*.

controlled vocabulary

restricted set of values that may be published for an observation.

data dictionary

listing of standardized terms and definitions used in *MTConnect Information Model*.

data model

organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities.

data set

key-value pairs where each entry is uniquely identified by the *key*.

data source

piece of equipment that can produce data that is published to an *agent*.

deprecated

indication that specific content in an *MTConnect Document* is currently usable but is regarded as being obsolete or superseded.

deprecation warning

indication that specific content in an *MTConnect Document* may be changed to *deprecated* in a future release of the standard.

document

piece of written, printed, or electronic matter that provides information or evidence that serves as an official record.

electric current

rate of flow of electric charge.

element

constituent part or a basic unit of identifiable and definable data.

extensible

ability for an implementer to extend *MTConnect Information Model* by adding content not currently addressed in the MTConnect Standard.

force

push or pull on a mass which results in an acceleration.

heartbeat

function that indicates to a *client* that the communications connection to an *agent* is still viable during times when there is no new data available to report often referred to as a “keep alive” message.

higher level

nested element that is above a lower level element.

implementation

specific instantiation of the MTConnect Standard.

information model

rules, relationships, and terminology that are used to define how information is structured.

instance

describes a set of *streaming data* in an *agent*. Each time an *agent* is restarted with an empty *buffer*, data placed in the *buffer* represents a new *instance* of the *agent*.

interaction model

model that defines how information is exchanged across an *interface* to enable interactions between independent systems.

interface

means by which communication is achieved between independent systems.

key

unique identifier in a *key-value pair* association.

key-value pair

association between an identifier referred to as the *key* and a value which taken together create a *key-value pair*.

lower camel case

first word is lowercase and the remaining words are capitalized and all spaces between words are removed.

lower level

nested element that is below a higher level element.

lower limit

lower conformance boundary for a variable.

lower warning

lower boundary indicating increased concern and supervision may be required.

major

identifier representing a consistent set of functionalities defined by the MTConnect Standard.

maximum

numeric upper constraint.

message

communication in writing, in speech, or by signals.

metadata

data that provides information about other data.

minimum

numeric lower constraint.

minor

identifier representing a specific set of functionalities defined by the MTConnect Standard.

nominal

ideal or desired value for a variable.

organize

act of containing and owning one or more elements.

organizer

entity that *organizes* one or more elements.

parameter

variable that must be given a value during the execution of a program or a communications command.

part

discrete item that has both defined and measurable physical characteristics including mass, material, and features, and is created by applying one or more manufacturing process steps to a workpiece

pascal case

first letter of each word is capitalized and the remaining letters are in lowercase. All space is removed between letters

persistence

method for retaining or restoring information.

probe

instrument commonly used for measuring the physical geometrical characteristics of an object.

profile

extends a reference metamodel (such as Unified Modeling Language (UML)) by allowing to adapt or customize the metamodel with constructs that are specific to a particular domain, platform, or a software development method.

requester

entity that initiates a *request* for information in a communications exchange.

reset

act of reverting back the accumulated value or statistic to their initial value.

> Note: An *Observation* with a *data set* representation removes all *key-value pairs*, setting the *data set* to an empty set.

responder

entity that responds to a *request* for information in a communications exchange.

response document

electronic *document* published by an *MTConnect Agent* in response to a *probe request*, *current request*, *sample request* or *asset request*.

revision

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

schema

definition of the structure, rules, and vocabularies used to define the information published in an electronic document.

semantic data model

methodology for defining the structure and meaning for data in a specific logical way that can be interpreted by a software system.

sensing element

mechanism that provides a signal or measured value.

sequence number

primary key identifier used to manage and locate a specific piece of *streaming data* in an *agent*.

specification limit

limit defining a range of values designating acceptable performance for a variable.

spindle

mechanism that provides rotational capabilities to a piece of equipment.

Note: Typically used for either work holding, materials or cutting tools.

standard

document established by consensus that provides rules, guidelines, or characteristics for activities or their results.. *Ref ISO/IEC Guide 2:2004*

stereotype

defines how an existing UML metaclass may be extended as part of a *profile*.

subtype

secondary or subordinate type of categorization or classification of information.

table

two dimensional set of values given by a set of *key-value pairs table entries*.

table cell

subdivision of a *table entry* representing a singular value.

table entry

subdivision of a *table* containing a set of *key-value pairs* representing *table cells*.

top level

element that represents the most significant physical or logical functions of a piece of equipment.

type

classification or categorization of information.

upper limit

upper conformance boundary for a variable.

upper warning

upper boundary indicating increased concern and supervision may be required.

version

unique identifier of the administered item. *Ref ISO/IEC 11179-:2015*

3.2 Information Model Terms

Asset Information Model

information model that provides semantic models for *Assets*.

Device Information Model

information model that describes the physical and logical configuration for a piece of equipment and the data that may be reported by that equipment.

Error Information Model

information model that describes the *response document* returned by an *agent* when it encounters an error while interpreting a *request* for information from a *client* or when an *agent* experiences an error while publishing the *response* to a *request* for information.

MTConnect Information Model

information model that defines the semantics of the MTConnect Standard.

Observation Information Model

information model that describes the *streaming data* reported by a piece of equipment.

3.3 Protocol Terms

asset request

HTTP Request to the *agent* regarding *Assets*.

current request

request to an *agent* to produce an *MTConnectStreams Response Document* containing the *Observation Information Model* for a snapshot of the latest observations at the moment of the *request* or at a given *sequence number*.

data streaming

method for an *agent* to provide a continuous stream of information in response to a single *request* from a *client*.

MTConnect Request

request for information issued from a *client* to an *MTConnect Agent*.

MTConnect Response Document

response document published by an *MTConnect Agent*.

MTConnectAssets Response Document

response document published by an *MTConnect Agent* in response to an *asset request*.

MTConnectDevices Response Document

response document published by an *MTConnect Agent* in response to a *probe request*.

MTConnectErrors Response Document

response document published by an *MTConnect Agent* whenever it encounters an error while interpreting an *MTConnect Request*.

MTConnectStreams Response Document

response document published by an *MTConnect Agent* in response to a *current request* or a *sample request*.

probe request

request to an *agent* to produce an *MTConnectDevices Response Document* containing the *Device Information Model*.

protocol

set of rules that allow two or more entities to transmit information from one to the other.

publish

sending of messages in a *publish and subscribe* pattern.

publish and subscribe

asynchronous communication method in which messages are exchanged between applications without knowing the identity of the sender or recipient.

> Note: In the MTConnect Standard, a communications messaging pattern that may be used to publish *streaming data* from an *agent*.

request

communications method where a *client* transmits a message to an *agent*. That message instructs the *agent* to respond with specific information.

request and response

communications pattern that supports the transfer of information between an *agent* and a *client*.

response

response *interface* which responds to a *request*.

sample request

request to an *agent* to produce an *MTConnectStreams Response Document* containing the *Observation Information Model* for a set of timestamped observations made by *Components*.

streaming data

observations published by a piece of equipment defined by the equipment metadata.

subscribe

receiving messages in a *publish and subscribe* pattern.

transport protocol

set of capabilities that provide the rules and procedures used to transport information between an *agent* and a client software application through a physical connection.

3.4 HTTP Terms

HTTP Body

data bytes transmitted in an HTTP transaction message immediately following the headers. *Ref IETF:RFC-2616*

HTTP Error Message

response provided by an *agent* indicating that an *HTTP Request* is incorrectly formatted or identifies that the requested data is not available from the *agent*. *Ref IETF:RFC-2616*

HTTP Header

header of either an *HTTP Request* from a *client* or an *HTTP Response* from an *agent*. *Ref IETF:RFC-2616*

HTTP Header Field

components of the header section of request and response messages in an HTTP transaction. *Ref IETF:RFC-2616*

HTTP Message

consist of requests from client to server and responses from server to client. *Ref IETF:RFC-2616*

> Note: In MTConnect Standard, it describes the information that is exchanged between an *agent* and a *client*.

HTTP Messaging

interface for information exchange functionality. *Ref IETF:RFC-2616*

HTTP Method

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. *Ref IETF:RFC-2616*

HTTP Query

portion of a request for information that more precisely defines the specific information to be published in response to the request. *Ref IETF:RFC-2616*

HTTP Request

request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use. *Ref IETF:RFC-2616*

> Note: In MTConnect Standard, a request issued by a *client* to an *agent* requesting information defined in the *HTTP Request Line*.

HTTP Request Line

begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. A CRLF is allowed in the definition of TEXT only as part of a header field continuation. *Ref IETF:RFC-2616*

> Note: the first line of an *HTTP Request* describing a specific *response document* to be published by an *agent*.

HTTP Request Method

indicates the method to be performed on the resource identified by the Request-URI. *Ref IETF:RFC-2616*

HTTP Request URI

Uniform Resource Identifier that identifies the resource upon which to apply the request. *Ref IETF:RFC-2616*

HTTP Response

after receiving and interpreting a request message, a server responds with an HTTP response message. *Ref IETF:RFC-2616*

> Note: In MTConnect Standard, the information published from an *agent* in reply to an *HTTP Request*.

HTTP Server

server that accepts *HTTP Request* from *client* and publishes *HTTP Response* as a reply to those *HTTP Request*. *Ref IETF:RFC-2616*

HTTP Status Code

3-digit integer result code of the attempt to understand and satisfy the request. *Ref IETF:RFC-2616*

HTTP Version

version of the HTTP protocol. *Ref IETF:RFC-2616*

3.5 XML Terms

abstract element

element that defines a set of common characteristics that are shared by a group of elements. An abstract entity cannot appear in a document. In a specific implementation, an abstract entity is replaced by a derived element that is itself not an abstract entity. The characteristics for the derived element are inherited from the abstract entity.

attribute

additional information or property for an *element*.

child element

element of a data modeling structure that illustrates the relationship between itself and the higher-level *parent element* within which it is contained.

document body

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 *Observations* or *DataItems* reported in a *response document*.

document header

portion of the content of an *MTConnect Response Document* that provides information from an *agent* defining version information, storage capacity, protocol, and other information associated with the management of the data stored in or retrieved from the *agent*.

element name

descriptive identifier contained in both the `start-tag` and `end-tag` of an XML element that provides the name of the element.

namespace

organizes information into logical groups.

parent element

element of a data modeling structure that illustrates the relationship between itself and the lower-level *child element*.

root element

first *structural element* provided in a *response document* encoded using XML.

structural element

element that organizes information that represents the physical and logical parts and sub-parts of a piece of equipment.

XML Document

structured text file encoded using Extensible Markup Language (XML).

XML Schema

schema defining a specific document encoded in XML.

3.6 MTConnect Terms

Asset

asset that is used by the manufacturing process to perform tasks.

> Note 1 to entry: An *Asset* relies upon an *Device* to provide observations and information about itself and the *Device* revises the information to reflect changes to the *Asset* during their interaction. Examples of *Assets* are cutting tools, Part Information, Manufacturing Processes, Fixtures, and Files.

> Note 2 to entry: A singular `assetId,Asset` uniquely identifies an *Asset* throughout its lifecycle and is used to track and relate the *Asset* to other *Devices* and entities.

> Note 3 to entry: *Assets* are temporally associated with a device and can be removed from the device without damage or alteration to its primary functions.

Component

engineered system part of a *Device* composed of zero or more *Components*

Composition

Component belonging to a *Component* and not composed of any *Components*.

Configuration

configuration for a *Component*

DataItem

observable observed by a *Component* that may make *Observations*

Device

Component not belonging to any *Component* that may have assets

MTConnect Agent

agent for the *MTConnect Information Model*.

MTConnect Document

document that represents a Part(s) of the MTConnect Standard.

MTConnect Event

observation of either a state or discrete value of the *Component*.

MTConnect Interface

interaction model for interoperability between pieces of equipment.

Observation

observation that provides telemetry data for a *DataItem*.

3.7 Acronyms

2D

two-dimensional

3D

three-dimensional

AI

artificial intelligence

ALM

application lifecycle management

AMT

The Association for Manufacturing Technology

ANSI

American National Standards Institute

AP

Application Protocol

API

application programming interface

ASME

American Society of Mechanical Engineers

ASTM

American Society for Testing and Materials

AWS

American Welding Society

BDD

block definition diagram

BOM

bill of materials

BST

Board on Standardization and Testing

C&R

cause and remedy

CA

certificate authority

CAD

computer-aided design

CAE

computer-aided engineering

CAI

computer-aided inspection

CAM

computer-aided manufacturing

CAx

computer-aided technologies

CDATA

Character Data

CFD

computational fluid dynamics

CM

configuration management

CMS

coordinate-measurement system

CNC

Computer Numerical Controller

CNRI

Corporation for National Research Initiatives

CPM

Core Product Model

CPM2

Revised Core Product Model

CPSC

Consumer Product Safety Commission

cUAV

configurable unmanned aerial vehicle

DARPA

Defense Advanced Research Projects Agency

DER

designated-engineering representative

DFM

design for manufacturing

DLA

Defense Logistics Agency

DMC

digital manufacturing certificate

DMSC

Dimensional Metrology Standards Consortium

DNS

Domain Name System

DoD

U.S. Department of Defense

DOI

Distributed Object Identifier

DRM

digital rights management

ECR

engineering change request

ERP

enterprise resource planning

FAA

Federal Aviation Administration

FAIR

first article inspection reporting

FDA

Food and Drug Administration

FEA

finite-element analysis

GD&T

geometric dimensions and tolerances

GID

global identifier

HMI

Human Machine Interface

HTML

Hypertext Markup Language

HTTP

Hypertext Transfer Protocol

HTTPS

Hypertext Transfer Protocol over Secure Sockets Layer

I/O

in-out

ID

identifier

IEEE

Institute of Electrical and Electronics Engineers

IIoT

industrial internet of things

INCOSE

International Council on Systems Engineering

IP

intellectual property

ISO

International Standards Organization

ISS

International Space Station

ISV

Independent Software Vendor

IT

information technology

ITU-T

Telecommunication Standardization Sector of the International Telecommunication Union

JSON

JavaScript Object Notation

JT

Jupiter Tesselation

LHS

Lifecycle Handler System

LIFT

Lifecycle Information Framework and Technology

LOI

Lifecycle Object Identifier

MAC

media access control

MADE

Manufacturing Automation and Design Engineering

MBD

model-based definition

MBE

Model-Based Enterprise

MBI

model-based inspection

MBM

model-based manufacturing

MBSD

model-based standards development

MBSE

model-based systems engineering

MEDALS

Military Engineering Data Asset Locator System

MES

manufacturing execution system

MOI

manufacturing object identifier

MOM

Message Orienged Middleware

MQTT

Message Queuing Telemetry Transport

MTC

Manufacturing Technology Centre

NASA

National Aeronautics and Space Administration

NC

numerical control

NIST

National Institute of Standards and Technology

NMTOKEN

Name Token

NNMI

National Network of Manufacturing Innovation

NSF

National Science Foundation

NTSC

National Transportation Safety Board

OASIS

Organization for the Advancement of Structured Information Standards

ODI

Open Data Institute

OEM

original equipment manufacturer

OOI

Ocean Observatories Initiative

OPC

OLE for Process Control

OSLC

Open Services for Lifecycle Collaboration

OSTP

Office of Science and Technology Policy

OT

operational technology

OWL

Ontology Web Language

PDF

Portable Document Format

PDM

product-data management

PDQ

product-data quality

PHM

prognosis and health monitoring

PI

principal investigator

PLC

Programmable Logic Controller

PLCS

Product Life Cycle Support

PLM

product lifecycle management

PLOT

product lifecycle of trust

PMI

product and manufacturing information

PMS

Production Management System

PRC

Product Representation Compact

PSI

Physical Science Informatics

PTAB

Primary Trustworthy Digital Repository Authorization Body Ltd.

QIF

Quality Information Framework

QMS

quality management system

QName

Qualified Name

RDF

Resource Description Framework

REST

Representational State Transfer

RII

receiving and incoming inspection

S/MIME

Secure/Multipurpose Internet Mail Extensions

SaaS

software-as-a-service

SAML

Security Assertion Markup Language

SC

Standards Committee

SCADA

Supervisory Control And Data Acquisition

SDO

Standards Development Organization

SFTP

Secure File Transfer Protocol

SKOS

Simple Knowledge Organization System

SLH

system lifecycle handler

SLR

systematic literature review

SME

small-to-medium enterprise

SMOPAC

Smart Manufacturing Operations Planning and Control

SMS Test Bed

Smart Manufacturing Systems Test Bed

SOA

service-oriented architecture

SPMM

semantic-based product metamodel

SSL

Secure Sockets Layer

STEP

Standard for the Exchange of Product Model Data

STEP AP242

Standard for the Exchange of Product Model Data Application Protocol 242

STL

Stereolithography

SysML

Systems Modeling Language

TCP/IP

Transmission Control Protocol/Internet Protocol

TDP

technical data package

TLS

Transport Layer Security

TSM

Total System Model

UA

Unified Architecture

UAL

Unified Architecture Language

UML

Unified Modeling Language

URI

Uniform Resource Identifier

URL

Uniform Resource Locator

URN

Uniform Resource Name

UTC

Coordinated Universal Time

UUID

Universally Unique Identifier

V&V

verification and validation

W3C

World Wide Web Consortium

WSN

Wirth Syntax Notation

WWW

World Wide Web

X.509-PKI

Public Key Infrastructure

X.509-PMI

Privilege Management Infrastructure

XML

Extensible Markup Language

XPath

XML Path Language

XSD

XML Schema Definitions

3.8 MTConnect References

[MTConnect Part 1.0]	*MTConnect Standard Part 1.0 - Fundamentals*. Version 2.0.
[MTConnect Part 2.0]	*MTConnect Standard: Part 2.0 - Device Information Model*. Version 2.0.
[MTConnect Part 3.0]	*MTConnect Standard: Part 3.0 - Observation Information Model*. Version 2.0.
[MTConnect Part 4.0]	*MTConnect Standard: Part 4.0 - Asset Information Model*. Version 2.0.
[MTConnect Part 5.0]	*MTConnect Standard: Part 5.0 - Interface Interaction Model*. Version 2.0.

4 Fundamentals

The MTConnect Standard defines the normative information model and protocol for retrieving information from manufacturing equipment. This document specifies the *agent* behavior and protocol.

4.1 Agent

The MTConnect Standard specifies the minimum functionality of the *agent*. The functionality is as follows:

- Provides store and forward messaging middleware service.
- Provides key-value information storage and asset retrieval service.
- Implements the REST API for the MTConnect Standard (See *Section 5.1 - REST Protocol*).
  - *Device* metadata.
  - observations collected by the agent.
  - assets collected by the agent.

There are three types of information stored by an *agent* that **MAY** be published in a *response document*. These are as follows:

- equipment metadata specified in *MTConnect Standard: Part 2.0 - Device Information Model*.
- *streaming data* provides the observations specified in *MTConnect Standard: Part 3.0 - Observation Information Model*.
- *Assets* specified in *MTConnect Standard: Part 4.0 - Asset Information Model*.

4.1.1 Agent Instance ID

The *agent* **MUST** set the `instanceId` to a unique value whenever the *sequence number* in the agent is initialized to `1`. (see *Section 4.1.3.1 - Sequence Numbers* and *Section 4.1.3.7 - Persistence and Recovery* below).

4.1.2 Storage of Equipment Metadata

An *agent* **MUST** be capable of publishing equipment metadata for the *agent* as specified in *MTConnect Standard: Part 2.0 - Device Information Model*.

4.1.3 Storage of Streaming Data

The *agent* **MAY** implement a *buffer* with a fixed number of observations. If the `bufferSize` is fixed, the *agent* **MUST** store observations using a first-in-first-out pattern. The *agent* will remove the oldest observation when the *buffer* is full and a new observation arrives.

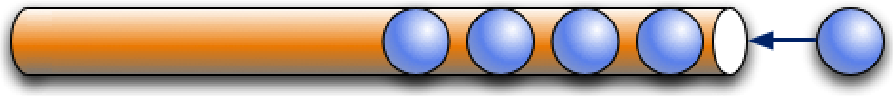

Figure 2: Data Storage in Buffer

In Figure 3, the maximum number of observations that can be stored in the *buffer* of the *agent* is 8. The `bufferSize` in the header reports the maximum number of observations. This example illustrates that when the *buffer* fills up, the oldest piece of data falls out the other end.

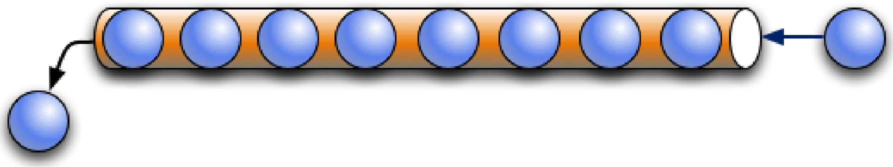

Figure 3: First In First Out Buffer Management

> Note: As an implementation suggestion, the *buffer* should be sized large enough to provide a continuous stream of observations. The implementer should also consider the impact of a temporary loss of communications when determining the size for the *buffer*. A larger *buffer* will allow more time to reconnect to an *agent* without losing data.

4.1.3.1 Sequence Numbers

In an *agent*, each occurrence of an observation in the *buffer* will be assigned a monotonically increasing unsigned 64-bit integer (*sequence number*) when it arrives. The first *sequence number* **MUST** be `1`.

The *sequence number* for each observation **MUST** be unique for an instance of an *agent* identified by an `instanceId`.

Table 1 illustrates the changing of the `instanceId` when an *agent* resets the *sequence number* to `1`.

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

Table 1: instanceId and sequence

Figure 4 shows two additional pieces of information defined for an *agent*:

- `firstSequence` – the oldest observation in the *buffer*. The *agent* removes this observation when it receives the next observation
- `lastSequence` – the newest observation in the *buffer*

`firstSequence` and `lastSequence` provide the range of values for the REST API requests.

The *agent* **MUST** begin evaluating observations with *sample request*'s `from` parameter. Also, the *agent* **MUST** include a maximum number of observations given by the `count` parameter in the *response document*.

In Figure 5, the request specifies the observations start at *sequence number* `15` (`from`) and includes a total of three items (`count`).

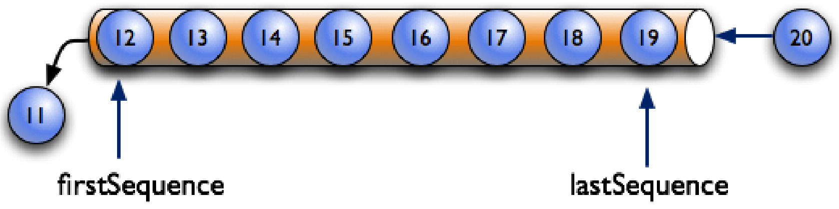

Figure 4: Indentifying the range of data with firstSequence and lastSequence

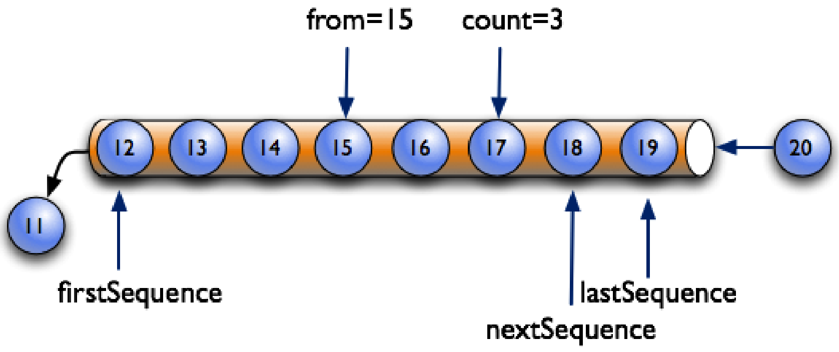

Figure 5: Identifying the range of data with from and count

`nextSequence` header property has the *sequence number* of the next observation in the *buffer* for subsequent *sample requests* providing a contiguous set of observations. In the example in Figure 5, the next *sequence number* (`nextSequence`) will be 18.

As shown in Figure 6, 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 $lastSequence + 1$.

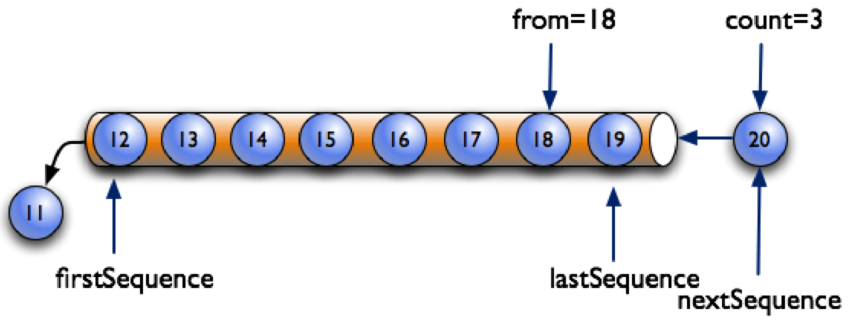

Figure 6: Indentifying the range of data with nextSequence and lastSequence

4.1.3.2 Observation Buffer

An observation has four pieces of information as follows:

1. *sequence number* associated with each observation - `sequence`.
2. The `timestamp` the observation was made. .
3. A reference to the `dataitemid` from the *MTConnect Standard: Part 2.0 - Device Information Model*.
4. The value of the observation.

Table 2 is an example demonstrating the concept of how data may be stored in an *agent*:

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

Table 2: Data Storage Concept

4.1.3.3 Timestamp

observations **MUST** have a `timestamp` giving the most accurate time that the observation occurred.

The timezone of the `timestamp` **MUST** be UTC (Coordinated Universal Time) and represented using ISO 8601 format: e.g., “2010-04-01T21:22:43Z”.

Applications **SHOULD** use the observation’s `timestamp` for ordering as opposed to *sequence number*.

All observations occurring at the same time **MUST** have the same `timestamp`.

4.1.3.4 Recording Occurrences of Streaming Data

The *agent* **MUST** only place observations in the *buffer* if the data has changed from the previous observation for the same `DataItem`.

The *agent* **MUST** place every observation in the *buffer*, without checking for changes, in the following cases:

- The `discrete` attribute is `true` for the `DataItem`.
- The `representation` is `DISCRETE`.
- The `representation` is `TIME_SERIES`.

4.1.3.5 Maintaining Last Value for Data Entities

An *agent* **MUST** retain the most recent observation associated with each `DataItem`, even if the observation is no longer in the *buffer*. This function supports the *current request* functionality.

4.1.3.6 Unavailability of Data

An observation with the value of `UNAVAILABLE` indicates the value is indeterminate.

The *agent* **MUST** initialize every `DataItem`, unless it has a constant value (see below), with an observation with the value of `UNAVAILABLE`. Aditionally, whenever the data source is unreachable, every `DataItem` associated with the data source must have an observation with the value of `UNAVAILABLE` and `timestamp` when the connection was lost.

An `DataItem` that is constrained to a constant value, as defined in *MTConnect Standard: Part 2.0 - Device Information Model*, **MUST** only have an observation with the constant value and **MUST NOT** be set to `UNAVAILABLE`.

4.1.3.7 Persistence and Recovery

The *agent* **MAY** have a fixed size *buffer* and the *buffer* **MAY** be ephemeral.

If the *buffer* is recoverable, the *agent* **MUST NOT** change the `instanceId` and **MUST NOT** set the *sequence number* to `1`. The *sequence number* **MUST** be one greater than the maximum value of the recovered observations. $max(sequence) + 1$

4.1.4 Storage of MTConnect Assets

An *agent* **MAY** only retain a limited number of `Assets` in the *asset buffer*. The `Assets` are stored in first-in-first-out method where the oldest `Asset` is removed when the *asset buffer* is full and a new `Asset` arrives.

Figure 7 illustrates the oldest `Asset` being removed from the *asset buffer* when a new `Asset` is added and the *asset buffer* is full:

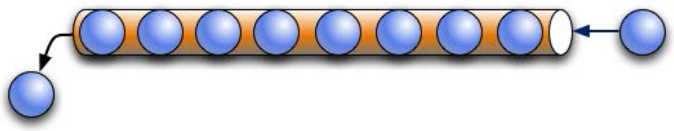

Figure 7: First In First Out Asset Buffer Management

`Assets` are indexed by `assetId`. In the case of `Assets`, Figure 8 demonstrates the relationship between the key (`assetId`) and the stored `Asset`:

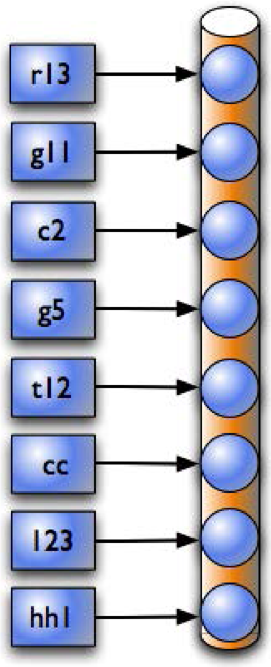

Figure 8: Relationship between assetId and stored Asset documents

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

When the *agent* receives a new `Asset`, one of the following rules **MUST** apply:

- If the `Asset` is not in the *asset buffer*, the *agent* **MUST** add the new `Asset` to the front of the *asset buffer*. If the *asset buffer* is full, the oldest `Asset` will be removed from the *asset buffer*.
- If the `Asset` is already in the *asset buffer*, the *agent* **MUST** replace the existing `Asset` and move the `Asset` to the front of the *asset buffer*.

The number of `Asset` that may be stored in an *agent* is defined by the value for `assetBufferSize`. An `assetBufferSize` of 4,294,967,296 or 2^{32} **MUST** indicate unlimited storage.

The *asset buffer* **MAY** be ephemeral and the `Asset` entities will be lost if the *agent* clears the *asset buffer*. They must be recovered from the data source.

MTConnect Standard: Part 4.0 - Asset Information Model provides additional information on asset management.

4.2 Response Documents

response documents are electronic documents generated by an *agent* in response to a *request* for data.

The *response documents* defined in the MTConnect Standard are:

- *MTConnectDevices Response Document*: Describes the composition and configuration of the *Device* and the data that can be observed. See *Section 5.2 - MTConnectDevices Response Document* and *MTConnect Standard: Part 2.0 - Device Information Model* for details on this information model.
- *MTConnectStreams Response Document*: *Observations* made at a point in time about related *DataItems*. See *Section 5.3 - MTConnectStreams Response Document* and *MTConnect Standard: Part 3.0 - Observation Information Model* for details on this information model.
- *MTConnectAssets Response Document*: *Assets* related to *Devices*. See *Section 5.4 - MTConnectAssets Response Document* and *MTConnect Standard: Part 4.0 - Asset Information Model* for details on this information model.

- *MTConnectErrors Response Document*: Information in response to a failed request. See *Section 6.1 - MTConnectErrors Response Document* for details on this information model.

4.3 Request/Response Information Exchange

The transfer of information between an *agent* and a client software application is based on a *request and response* REST protocol. A client application requests specific information from an *agent* and an *agent* responds with a *response document*.

There are four types of *MTConnect Requests*. These *requests* are as follows:

- *probe request*: Requests information about one more more *Devices* as an `MTConnectDevices` block.
- *current request*: Requests the most recent, or snapshot at a *sequence number*, observations as an `MTConnectStreams` block.
- *sample request*: Requests a series of observations as an `MTConnectStreams` block.
- *asset request*: Requests a set of assets as an `MTConnectAssets` block.

If an *agent* is unable to respond to the request for information or the request includes invalid information, the *agent* will publish an *MTConnectErrors Response Document*. See `MTConnectErrors`.

See *Section 5.1 - REST Protocol* for the details on the normative requirements of the agent.

5 MTConnect Protocol

The *agent* **MUST** support the *Section 5.1 - REST Protocol* and produce XML representations of the information models.

All other protocols and representations are optional.

5.1 REST Protocol

An *agent* **MUST** provide a REST API application programming interface (API) supporting HTTP version 1.0 or greater. This interface **MUST** support HTTP (RFC7230) and use URIs (RFC3986) to identify specific information requested from an *agent*.

The REST API adheres to the architectural principles of a stateless service to retrieve information associated with pieces of equipment. Additionally, the API is read-only and does not produce any side effects on the *agent* or the equipment. In REST state management, the client is responsible for recovery in case of an error or loss of connection.

5.1.1 HTTP Request

An *agent* **MUST** support the `HTTP GET` verb, all other verbs are optional. See IETF RFC 7230 for a complete description of the HTTP request structure.

The HTTP uses Uniform Resource Identifiers (URI) as outlined in IETF RFC 3986 as the *request-target*. IETF RFC 7230 specifies the http URI scheme for the *request-target* as follows:

1. `protocol`: The protocol used for the request. Must be `http` or `https`.
2. `authority`: The network domain or address of the agent with an optional port.
3. `path`: A Hierarchical Identifier following a slash (`/`) and before the optional question-mark (`?`). The `path` separates segments by a slash (`/`).
4. `query`: The portion of the HTTP request following the question-mark (`?`). The query portion of the HTTP request is composed of key-value pairs, = separated by an ampersand (`&`).

5.1.1.1 `path` Portion of an HTTP Request

The `path` portion of the *request-target* has the following segments:

- `device-name` or `uuid`: optional `name` or `uuid` of the `Device`
- `request`: request, must be one of the following: (also see *Section 5.1.4.3 - Operations for Agent*)
  - `probe`
  - `current`
  - `sample`
  - `asset` or `assets`
    - `asset` request has additional optional segment `<asset ids>`

If `name` or `uuid` segement are not specified in the *HTTP Request*, an *agent* **MUST** return information for all pieces of equipment. The following sections will

Examples:

- `http://localhost:5000/my_device/probe`

  The request only provides information about `my_device`.
- `http://localhost:5000/probe`

  The request provides information for all devices.

The following section specifies the details for each request.

5.1.2 MTConnect REST API

An *agent* **MUST** support *probe requests*, *current requests*, *sample requests*, and *asset requests*.

See the operations of the `Agent` for details regarding the *requests*.

5.1.3 HTTP Errors

When an *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 *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 *agent* encounters an error in interpreting or responding to an *HTTP Request*, the *agent* **MUST** also publish an *MTConnectErrors Response Document* that provides additional details about the error. See *Section 6 - Error Information Model* for details on the *MTConnectErrors Response Document*.

5.1.3.1 Streaming Data

HTTP *data streaming* is a method for an *agent* to provide a continuous stream of observations in response to a single *request* using a *publish and subscribe* communication pattern.

When an *HTTP Request* includes an `interval` parameter, an *agent* **MUST** provide data with a minimum delay in milliseconds between the end of one data transmission and the beginning of the next. A value of zero (0) for the `interval` parameter indicates that the *agent* should deliver data at the highest rate possible and is only relevant for *sample requests* .

The format of the response **MUST** use an `x-multipart-replace` encoded message with each section separated by MIME boundaries. Each section **MUST** contain an entire *MTConnectStreams Response Document*.

When streaming for a *current request*, the *agent* produces an *MTConnectStreams Response Document* with the most current observations every `interval` milliseconds.

When streaming for a *sample request*, if there are no available observations after the `interval` time elapsed, the *agent* **MUST** wait for either the `heartbeat` time to elapse or an observation arrives. If the `heartbeat` time elapses and no observations arrive, then an empty *MTConnectStreams Response Document* **MUST** be sent.

Note: For more information on MIME, see IETF RFC 1521 and RFC 822.

An example of the format for an *HTTP Request* that includes an `interval` parameter is:

Example 1: Example for HTTP Request with interval parameter

```
1 http://localhost:5000/sample?interval=1000
```

HTTP Response Header:

Example 2: HTTP Response header

```
1 HTTP/1.1 200 OK
2 Connection: close
3 Date: Sat, 13 Mar 2010 08:33:37 UTC
4 Status: 200 OK
5 Content-Disposition: inline
6 X-Runtime: 144ms
7 Content-Type: multipart/x-mixed-replace;boundary=
8 a8e12eced4fb871ac096a99bf9728425
9 Transfer-Encoding: chunked
```

Lines 1-9 in *Example 2* represent a standard header for a MIME `multipart/x-mixed-replace` message. The boundary is a separator for each section of the stream. Lines 7-8 indicate this is a multipart MIME message and the boundary between sections.

With streaming protocols, the `Content-length` **MUST** be omitted and `Transfer-Encoding` **MUST** be set to `chunked` (line 9). See IETF RFC 7230 for a full description of the HTTP protocol and chunked encoding.

Example 3: HTTP Response header 2

```
10 --a8e12eced4fb871ac096a99bf9728425
11 Content-type: text/xml
12 Content-length: 887
13
14 <?xml version="1.0" ecoding="UTF-8"?>
15 <MTConnectStreams ...>...
```

Each section of the document begins with a boundary preceded by two hyphens (–). The `Content-type` and `Content-length` header fields **MUST** be provided for each section and **MUST** be followed by `<CR><LF><CR><LF>` (ASCII code for `<CR>` is 13 and `<LF>` 10) before the XML document. The header and the `<CR><LF><CR><LF>` **MUST NOT** be included in the computation of the content length.

An *agent* MUST continue to stream results until the client closes the connection. The *agent* MUST NOT stop streaming for any reason other than the following:

- *agent* process stops
- The client application stops receiving data

5.1.3.1.1 Heartbeat

When *streaming data* is requested from a *sample request*, an *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 *agent* by sending an MTConnect *response document* with an empty `Steams` entity (See *MTConnect Standard: Part 3.0 - Observation Information Model* for more details on `Streams`) to the client software application.

The *heartbeat* **MUST** occur on a periodic basis given by the optional `heartbeat` query parameter and **MUST** default to 10 seconds. An *agent* **MUST** maintain a separate *heartbeat* for each client application for which the *agent* is responding to a *data streaming request*.

An *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 *agent* or the client application.

5.1.3.2 References

A `Component` **MAY** include a set of `Reference` entities of the following types that **MAY** alter the content of the *MTConnectStreams Response Documents* published in response to a *current request* or a *sample request* as specified:

- A *Component* reference (`ComponentRef`) modifies the set of *Observations*, limited by a path query parameter of a *current request* or *sample request*, to include the *Observations* associated with the entity whose value for its `id` attribute matches the value provided for the `idRef` attribute of the `ComponentRef` element. Additionally, *Observations* defined for any *lower level* entity(s) associated with the identified entities **MUST** also be returned. The result is equivalent to appending `//[@id=<"idRef">]` to the path query parameters of the *current request* or *sample request*. See *Section 4.1 - Agent* for more details on path queries.
- A *DataItem* reference (`DataItemRef`) modifies the set of resulting *Observations*, limited by a path query parameter of a *current request* or *sample request*, to include the *Observations* whose value for its `id` attribute matches the value provided for the `idRef` attribute of the `DataItemRef` element. The result is equivalent to appending `//[@id=<"idRef">]` to the path query parameters of the *current request* or *sample request*. See *Section 4.1 - Agent* for more details on path queries.

5.1.4 Agent

agent.

An *agent* **MUST** perform the following tasks:

- Collect data from manufacturing equipment.
- Generate *response documents*.
- Provide a REST interface using Hypertext Transfer Protocol (HTTP).

In addition to XML and HTTP, An *agent* **MAY** provide additional protocols and representations. Some representations **MAY** have companion specifications.

5.1.4.1 Value Properties of Agent

Table 3 lists the Value Properties of `Agent`.

Value Property name	Value Property type	Multiplicity
`instanceId`	`uint32`	1
`sequenceNumber`	`uint64`	1
`bufferSize`	`uint32`	1
`maxAssets`	`uint32`	1
`assetCount`	`uint32`	1

Table 3: Value Properties of Agent

Descriptions for Value Properties of `Agent`:

- `instanceId`

  identifier for an *instance* of the *agent*.

  `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.

- `sequenceNumber`

  sequence number.

- `bufferSize`

  maximum number of *Observations* that **MAY** be retained in the *agent* that published the *response document* at any point in time.

- `maxAssets`

  maximum number of *Assets* that **MAY** be retained in the *agent* that published the *response document* at any point in time.

- `assetCount`

  current number of *Assets* that are currently stored in the *agent* as of the `creationTime` that the *agent* published the *response document*.

5.1.4.2 Part Properties of Agent

Table 4 lists the Part Properties of `Agent`.

Part Property name	**Multiplicity**
`Observation` (organized by `buffer`)	0..*
`Asset` (organized by `assetBuffer`)	0..*

Table 4: Part Properties of Agent

Descriptions for Part Properties of `Agent`:

- `Observation`

  abstract entity that provides telemetry data for a `DataItem` at a point in time.

  `buffer` is a *buffer* for `Observation` types.

- `Asset`

  abstract *Asset*.

  `assetBuffer` is an *asset buffer* for `Asset` types.

5.1.4.3 Operations for Agent

- `probe`

  agent **MUST** respond to a successful *probe request* with an `MTConnectDevices` entity containing either one, when a `Device` `name` or `uuid` is given, or all known `Device` entries.

When successful, an `MTConnectDevices` entity is returned and status code of 200. Otherwise an `MTConnectError` and an associated status code.

The parameters for `Agent` are:

- `device`

  if present, specifies that only the `Device` for the given name or uuid will be returned.

  If not present, all associated `Device` for the Agent will be returned.

- `status`

  HTTP Status Code.

  The following *HTTP Status Codes* **MUST** be supported as possible responses to a *probe request*:

  * Status Code: `200`, Code Name: `OK`:
    The *request* succeeded.
  * Status Code: `400`, Code Name: `Bad Request`:
    The *request* was invalid. The *response* **MUST** have an *MTConnectErrors Response Document*.
  * Status Code: `404`, Code Name: `Not Found`:
    The device name or uuid could not be located. The *response* **MUST** have an *MTConnectErrors Response Document*.
  * Status Code: `405`, Code Name: `Method Not Allowed`:
    The *request* specified a method other than `GET`
  * Status Code: `406`, Code Name: `Not Acceptable`:
    The HTTP `Accept` Header in the *request* was not one of the supported representations.
  * Status Code: `431`, Code Name: `Request Header Fields Too Large`:
    The fields in the *HTTP Request* exceed the limit of the implementation of the *agent*.
  * Status Code: `500`, Code Name: `Internal Server Error`:
    There was an unexpected error in the *agent* while responding to a *request*.

- `return`

  agent **MUST** respond to a successful *probe request* with an *HTTP Status Code* `200` (`OK`) and an *MTConnectDevices Response Document*. If the *request* fails, the *agent* **MUST** respond with an *MTConnectErrors Response Document* an *HTTP Status Code* other than 200.

  `MTConnectDevices` if successful, `MTConnectError` otherwise.

• `current`

agent **MUST** respond to a successful *current request* with an `MTConnectStreams` block containing the latest values for the selected observations. If the `at` parameter is given, the values for the observations are a snapshot taken when the `lastSequence` number was equal to the value of the `at` parameter.

When successful, an `MTConnectStreams` entity is returned and status code of 200. Otherwise an `MTConnectError` and an associated status code.

The parameters for `Agent` are:

- `device`

  optional `Device` `name` or `uuid`. If not given, all devices are returned.

- `path`

  XPath evaluated against the *Device Information Model* that references the *Components* and *DataItems* to include in the *MTConnectStreams Response Document*.

  When a `Component` element is referenced by the XPath, all observations for its *DataItems* and related *Components* **MUST** be included in the *MTConnectStreams Response Document*.

- `frequency`

  agent **MUST** stream samples and events to the client application pausing for frequency milliseconds between each part. Each part will contain a maximum of `count` events or samples and from will be used to indicate the beginning of the stream.

  DEPRECATED Version 1.2, replace by `interval`

- `at`

  response documents **MUST** include observations consistent with a specific *sequence number* given by the value of the `at` parameter.

  If the value is either less than the `firstSequence` or greater than the `lastSequence`, the *request* **MUST** return a 404 *HTTP Status Code* and the *agent* **MUST** return an *MTConnectErrors Response Document* with an `OUT_OF_RANGE` `errorCode`.

  The `at` parameter **MUST NOT** be used in conjunction with the `interval` parameter.

- `interval`

  agent **MUST** continuously publish *response documents* pausing for the number of milliseconds given as the value.

  The `interval` value **MUST** be in milliseconds, and **MUST** be a positive integer greater than zero (0).

  The `interval` parameter **MUST NOT** be used in conjunction with the `at` parameter.

- `status`

  HTTP Status Code.

  The following *HTTP Status Codes* **MUST** be supported as possible responses to a *current request*:

  * Status Code: `200`, Code Name: `OK`:
    The *request* succeeded.
  * Status Code: `400`, Code Name: `Bad Request`:
    The *request* was invalid. The *response* **MUST** have an *MTConnectErrors Response Document*.
  * Status Code: `404`, Code Name: `Not Found`:
    One of the following conditions apply:
    · The device name or uuid could not be located.
    · The `at` was `OUT_OF_RANGE` range.

    The *response* **MUST** have an *MTConnectErrors Response Document*.
  * Status Code: `405`, Code Name: `Method Not Allowed`:
    The *request* specified a method other than `GET`
  * Status Code: `406`, Code Name: `Not Acceptable`:
    The HTTP `Accept` Header in the *request* was not one of the supported representations.
  * Status Code: `431`, Code Name: `Request Header Fields Too Large`:
    The fields in the *HTTP Request* exceed the limit of the implementation of the *agent*.
  * Status Code: `500`, Code Name: `Internal Server Error`:
    There was an unexpected error in the *agent* while responding to a *request*.

- `return`

  agent responds to a *current request* with an *MTConnectStreams Response Document* that contains the current value of *Observations* associated with each piece of *streaming data* available from the *agent*, subject to any filtering defined in the *request*.

• `sample`

agent **MUST** respond to a successful *sample request* with an `MTConnectStreams` entity containing the values for the selected observations according to the parameters provided.

When successful, an `MTConnectStreams` entity is returned and status code of 200. Otherwise an `MTConnectError` and an associated status code.

The parameters for `Agent` are:

- `device`

  optional `Device` `name` or `uuid`. If not given, all devices are returned.
- `path`

  XPath evaluated against the *Device Information Model* that references the *Components* and *DataItems* to include in the *MTConnectStreams Response Document*.

  When a `Component` element is referenced by the XPath, all observations for its *DataItems* and related *Components* **MUST** be included in the *MTConnectStreams Response Document*.
- `from`

  designates the *sequence number* of the first observation in the *buffer* the *agent* **MUST** consider publishing in the *response document*.

  If `from` is zero (0), it **MUST** be set to the `firstSequence`, the oldest observation in the *buffer*.

  If `from` and `count` parameters are not given, `from` **MUST** default to the `firstSequence`.

  If the `from` parameter is less than the `firstSequence` or greater than `lastSequence`, the *agent* **MUST** return a `404` *HTTP Status Code* and **MUST** publish an *MTConnectErrors Response Document* with an `OUT_OF_RANGE` `errorCode`.
- `count`

  designates the maximum number of observations the *agent* **MUST** publish in the *response document*.

  The `count` **MUST NOT** be zero (0).

  When the `count` is greater than zero (0), the `from` parameter **MUST** default to the `firstSequence`. The evaluation of observations starts at `from` and moves forward accumulating newer observations until the number of observations equals the `count` or the observation at `lastSequence` is considered.

  When the `count` is less than zero (0), the `from` parameter **MUST** default to the `lastSequence`. The evaluation of observations starts at `from` and moves backward accumulating older observations until the number of observations equals the absolute value of `count` or the observation at `firstSequence` is considered.

  `count` **MUST NOT** be less than zero (0) when an `interval` parameter is given.

  If `count` is not provided, it **MUST** default to `100`.

  If the absolute value of `count` is greater than the size of the *buffer* or equal to zero (0), the *agent* **MUST** return a `404` *HTTP Status Code* and **MUST** publish an *MTConnectErrors Response Document* with an `OUT_OF_RANGE` `errorCode`.

If the `count` parameter is not a numeric value, the *agent* **MUST** return a `400` *HTTP Status Code* and **MUST** publish an *MTConnectErrors Response Document* with an `INVALID_REQUEST` `errorCode`.

- `frequency`

  agent **MUST** stream samples and events to the client application pausing for frequency milliseconds between each part. Each part will contain a maximum of `count` events or samples and from will be used to indicate the beginning of the stream.

  DEPRECATED Version 1.2, replace by `interval`

- `heartbeat`

  sets the time period for the *heartbeat* function in an *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 **MUST** default to 10 seconds.

  `heartbeat` **MUST** only be specified if `interval` is also specified.

- `interval`

  agent **MUST** continuously publish *response documents* when the query parameters include `interval` using the value as the minimum period between adjacent publications.

  The `interval` value **MUST** be in milliseconds, and **MUST** be a positive integer greater than or equal to zero (0).

  If the value for the `interval` parameter is zero (0), the *agent* **MUST** publish *response documents* when any observations become available.

  If the period between the publication of a *response document* and reception of observations exceeds the `interval`, the *agent* **MUST** wait for a maximum of `heartbeat` milliseconds for observations. Upon the arrival of observations, the *agent* **MUST** immediately publish a *response document*. When the period equals or exceeds the `heartbeat`, the *agent* **MUST** publish an empty *response document*.

- `to`

  specifies the *sequence number* of the observation in the *buffer* that will be the upper bound of the observations in the *response document*.

  Rules for `to` are as follows:

  * The value of `to` **MUST** be an unsigned 64-bit integer.
  * The value of `to` **MUST** be greater than the `firstSequence`.
  * The value of `to` **MUST** be less than or equal to the `lastSequence`.

        * The value of `to` **MUST** be greater than `from`.
        * If `to` and `count` are given, the `count` parameter **MUST** be greater than zero.
        * If `to` and `count` are given, the maximum number of observations published in the *response document* **MUST NOT** be greater than the value of `count`.
        * If `to` is not given, see the `from` parameter for default behavior.
        * If the `to` parameter is less than the `firstSequence` or greater than `lastSequence`, the *agent* **MUST** return a `404` *HTTP Status Code* and **MUST** publish an *MTConnectErrors Response Document* with an `OUT_OF_RANGE` `errorCode`.
        * If the `to` parameter is not a positive numeric value, the *agent* **MUST** return a `400` *HTTP Status Code* and **MUST** publish an *MTConnectErrors Response Document* with an `INVALID_REQUEST` `errorCode`.
        * If the `to` parameter is less than the `from` parameter, the *agent* **MUST** return a `400` *HTTP Status Code* and **MUST** publish an *MTConnectErrors Response Document* with an `INVALID_REQUEST` `errorCode`.
        * If the `to` parameter is given and the `count` parameter is less than zero, the *agent* **MUST** return a `400` *HTTP Status Code* and **MUST** publish an *MTConnectErrors Response Document* with an `INVALID_REQUEST` `errorCode`.

    - `status`

      HTTP Status Code.

      The following *HTTP Status Codes* **MUST** be supported as possible responses to a *current request*:

        * Status Code: `200`, Code Name: `OK`:
          The *request* succeeded.
        * Status Code: `400`, Code Name: `Bad Request`:
          The *request* was invalid. The *response* **MUST** have an *MTConnectErrors Response Document*.
        * Status Code: `404`, Code Name: `Not Found`:
          One of the following conditions apply:
            · The device name or UUID could not be located.
            · One of the `asset_ids` could not be found.

          The *response* **MUST** have an *MTConnectErrors Response Document*.
        * Status Code: `405`, Code Name: `Method Not Allowed`:
          The *request* specified a method other than `GET`
        * Status Code: `406`, Code Name: `Not Acceptable`:
          The HTTP `Accept` Header in the *request* was not one of the supported representations.

    * Status Code: `431`, Code Name: `Request Header Fields Too Large`:
      The fields in the *HTTP Request* exceed the limit of the implementation of the *agent*.
    * Status Code: `500`, Code Name: `Internal Server Error`:

  There was an unexpected error in the *agent* while responding to a *request*.

  - `return`

    agent **MUST** respond to a successful *sample request* with an *HTTP Status Code* `200` (`OK`) and an *MTConnectStreams Response Document*. If the *request* fails, the *agent* **MUST** respond with an *MTConnectErrors Response Document* an *HTTP Status Code* other than 200.

- `asset`

  agent **MUST** respond to a successful *asset request* with an `MTConnectAssets` entity with the selected asset entities according to the parameters provided.

  When successful, an `MTConnectAssets` entity is returned and status code of 200. Otherwise an `MTConnectError` and an associated status code.

  The parameters for `Agent` are:

  - `device`

    optional `Device` `name` or `uuid`. If not given, all devices are returned.

  - `assetIds`

    `path` portion is a list of (`asset_id`) for specific *MTConnectAssets Response Documents*.

    In response, the *agent* returns an *MTConnectAssets Response Document* that contains information for the specific assets for each of the `asset_id` values provided in the *request*. Each `asset_id` is separated by a ";".

  - `count`

    specifies the maximum number of *MTConnectAssets Response Documents* returned in an *MTConnectAssets Response Document*.

    If `count` is not given, the default value **MUST** be `100`.

  - `type`

    type of *Asset*. See *MTConnect Standard: Part 4.0 - Asset Information Model*.

  - `removed`

    value for `removed` **MUST** be `true` or `false` and interpreted as follows:

    * `true`: *MTConnectAssets Response Documents* for assets marked as removed **MUST** be included in the *response document*.

        * `false`: *MTConnectAssets Response Documents* for assets marked as removed **MUST NOT** be included in the *response document*.

    If `removed` is not given, the default value **MUST** be `false`.

- `status`

  HTTP Status Code.

  The following *HTTP Status Codes* **MUST** be supported as possible responses to a *asset request*:

    * Status Code: `200`, Code Name: `OK`:
      The *request* succeeded.
    * Status Code: `400`, Code Name: `Bad Request`:
      The *request* was invalid. The *response* **MUST** have an *MTConnectErrors Response Document*.
    * Status Code: `404`, Code Name: `Not Found`:
      One of the following conditions apply:
        · The device name or uuid could not be located.
        · The `from` or `to` was `OUT_OF_RANGE`.

      The *response* **MUST** have an *MTConnectErrors Response Document*.
    * Status Code: `405`, Code Name: `Method Not Allowed`:
      The *request* specified a method other than `GET`
    * Status Code: `406`, Code Name: `Not Acceptable`:
      The HTTP `Accept` Header in the *request* was not one of the supported representations.
    * Status Code: `431`, Code Name: `Request Header Fields Too Large`:
      The fields in the *HTTP Request* exceed the limit of the implementation of the *agent*.
    * Status Code: `500`, Code Name: `Internal Server Error`:
      There was an unexpected error in the *agent* while responding to a *request*.

- `return`

  MTConnectAssets Response 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*.

5.2 MTConnectDevices Response Document

This section provides semantic information for the `MTConnectDevices` entity.

5.2.1 MTConnectDevices

root entity of an *MTConnectDevices Response Document* that contains the *Device Information Model* of one or more `Device` entities.

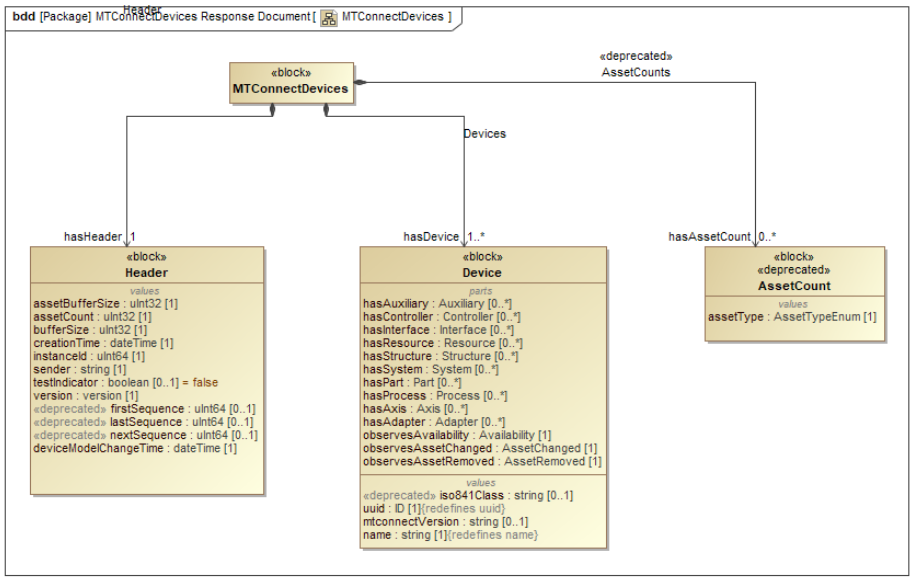
bdd [Package] MTConnectDevices Response Document [MTConnectDevices]
«block» MTConnectDevices
«deprecated» AssetCounts
Devices
hasHeader 1
hasDevice 1..*
hasAssetCount 0..*
«block» Header
values
assetBufferSize : uInt32 [1]
assetCount : uInt32 [1]
bufferSize : uInt32 [1]
creationTime : dateTime [1]
instanceId : uInt64 [1]
sender : string [1]
testIndicator : boolean [0..1] = false
version : version [1]
«deprecated» firstSequence : uInt64 [0..1]
«deprecated» lastSequence : uInt64 [0..1]
«deprecated» nextSequence : uInt64 [0..1]
deviceModelChangeTime : dateTime [1]
«block» Device
parts
hasAuxiliary : Auxiliary [0..*]
hasController : Controller [0..*]
hasInterface : Interface [0..*]
hasResource : Resource [0..*]
hasStructure : Structure [0..*]
hasSystem : System [0..*]
hasPart : Part [0..*]
hasProcess : Process [0..*]
hasAxis : Axis [0..*]
hasAdapter : Adapter [0..*]
observesAvailability : Availability [1]
observesAssetChanged : AssetChanged [1]
observesAssetRemoved : AssetRemoved [1]
values
«deprecated» iso841Class : string [0..1]
uuid : ID [1]{redefines uuid}
mtconnectVersion : string [0..1]
name : string [1]{redefines name}
«block» «deprecated» AssetCount
values
assetType : AssetTypeEnum [1]

Figure 9: MTConnectDevices

> Note: Additional properties of `MTConnectDevices` **MAY** be defined for schema and namespace declaration. See *Section C - Schema and Namespace Declaration Information* for an XML example.

5.2.1.1 Part Properties of MTConnectDevices

Table 5 lists the Part Properties of `MTConnectDevices`.

Part Property name	Multiplicity
`Header`	1
`Device` (organized by `Devices`)	1..*

Table 5: Part Properties of MTConnectDevices

Descriptions for Part Properties of `MTConnectDevices`:

- `Header`

provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

- `Device`

  `Component` composed of a piece of equipment that produces observations about itself.

  `Devices` groups one or more `Device` entities. See *MTConnect Standard: Part 2.0 - Device Information Model* for more detail.

5.2.2 Header

provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

5.2.2.1 Value Properties of Header

Table 6 lists the Value Properties of `Header`.

Value Property name	**Value Property type**	**Multiplicity**
`assetBufferSize`	`uint32`	1
`assetCount`	`uint32`	1
`bufferSize`	`uint32`	1
`creationTime`	`datetime`	1
`instanceId`	`uint64`	1
`sender`	`string`	1
`testIndicator`	`boolean`	0..1
`version`	`version`	1
`<<deprecated>> firstSequence`	`uint64`	0..1
`<<deprecated>> lastSequence`	`uint64`	0..1
`<<deprecated>> nextSequence`	`uint64`	0..1
`deviceModelChangeTime`	`datetime`	1

Table 6: Value Properties of Header

Descriptions for Value Properties of `Header`:

- `assetBufferSize`

  maximum number of `Asset` types that can be stored in the *agent* that published the *response document*.

> Note: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `assetBufferSize`.

- `assetCount`

  current number of `Asset` that are currently stored in the *agent* as of the `creationTime` that the *agent* published the *response document*.

  `assetCount` **MUST NOT** be larger than the value reported for `assetBufferSize`.

- `bufferSize`

  maximum number of *DataItems* that **MAY** be retained in the *agent* that published the *response document* at any point in time.

  > Note 1 to entry: `bufferSize` represents the maximum number of sequence numbers that **MAY** be stored in the *agent*.

  > Note 2 to entry: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `bufferSize`.

- `creationTime`

  timestamp that an *agent* published the *response document*.

- `instanceId`

  identifier for a specific instantiation of the *buffer* associated with the *agent* that published the *response document*.

  `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.

- `sender`

  identification defining where the *agent* that published the *response document* is installed or hosted.

  `sender` **MUST** be either an IP Address or Hostname describing where the *agent* is installed or the URL of the *agent*; e.g., `http://<address>[:port]/`.

  > Note: The port number need not be specified if it is the default HTTP port 80.

- `testIndicator`

  indicates whether the *agent* that published the *response document* is operating in a test mode.

If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `false`.

- `version`

  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 data model*.

  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

- `<<deprecated>> firstSequence`

  sequence number assigned to the oldest piece of *streaming data* stored in the *buffer* of the *agent* immediately prior to the time that the *agent* published the *response document*.

- `<<deprecated>> lastSequence`

  sequence number assigned to the last piece of *streaming data* that was added to the *buffer* of the *agent* immediately prior to the time that the *agent* published the *response document*.

- `<<deprecated>> nextSequence`

  sequence number of the piece of *streaming data* that is the next piece of data to be retrieved from the *buffer* of the *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 *agent* at the time that the document was published, then the value reported for `nextSequence` **MUST** be equal to `lastSequence` + 1.

- `deviceModelChangeTime`

  timestamp of the last update of the `Device` information for any device.

5.2.2.2 Part Properties of Header

Table 7 lists the Part Properties of `Header`.

Part Property name	Multiplicity
`<<deprecated>> AssetCount` (organized by `<<deprecated>> AssetCounts`)	0..*

Table 7: Part Properties of Header

Descriptions for Part Properties of `Header`:

- `<<deprecated>> AssetCount`

  count of each asset type currently in the *agent*.

  `AssetCounts` groups `AssetCount` entities.

5.2.3 <<deprecated>>AssetCount

count of each asset type currently in the *agent*.

5.2.3.1 Value Properties of AssetCount

Table 8 lists the Value Properties of `AssetCount`.

Value Property name	Value Property type	Multiplicity
`assetType`	`string`	1

Table 8: Value Properties of AssetCount

Descriptions for Value Properties of `AssetCount`:

- `assetType`

  type of *Asset*.

5.3 MTConnectStreams Response Document

This section provides semantic information for the `MTConnectStreams` entity.

5.3.1 MTConnectStreams

root entity of an *MTConnectStreams Response Document* that contains the *Observation Information Model* of one or more `Device` entities.

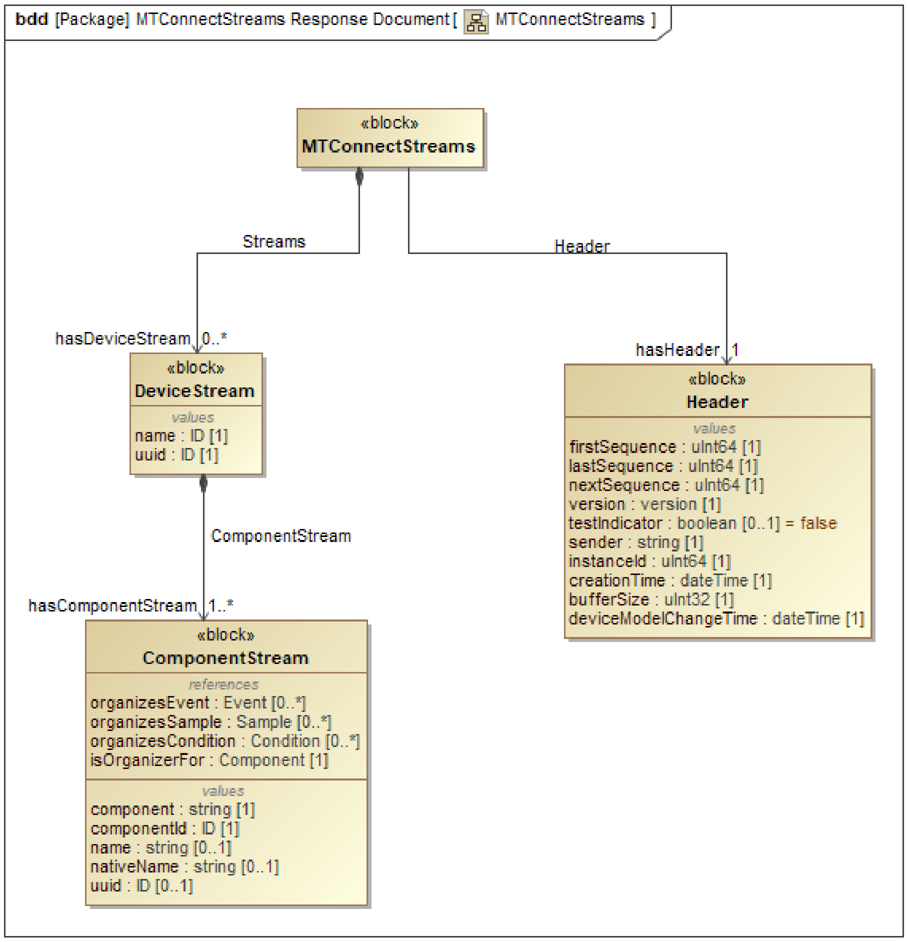
bdd [Package] MTConnectStreams Response Document [MTConnectStreams]
«block» MTConnectStreams
Streams
Header
hasDeviceStream 0..*
hasHeader 1
«block» DeviceStream
values
name : ID [1]
uuid : ID [1]
«block» Header
values
firstSequence : uInt64 [1]
lastSequence : uInt64 [1]
nextSequence : uInt64 [1]
version : version [1]
testIndicator : boolean [0..1] = false
sender : string [1]
instanceId : uInt64 [1]
creationTime : dateTime [1]
bufferSize : uInt32 [1]
deviceModelChangeTime : dateTime [1]
ComponentStream
hasComponentStream 1..*
«block» ComponentStream
references
organizesEvent : Event [0..*]
organizesSample : Sample [0..*]
organizesCondition : Condition [0..*]
isOrganizerFor : Component [1]
values
component : string [1]
componentId : ID [1]
name : string [0..1]
nativeName : string [0..1]
uuid : ID [0..1]

Figure 10: MTConnectStreams

> Note: Additional properties of `MTConnectStreams` **MAY** be defined for schema and namespace declaration. See *Section C - Schema and Namespace Declaration Information* for an XML example.

5.3.1.1 Part Properties of MTConnectStreams

Table 9 lists the Part Properties of `MTConnectStreams`.

Part Property name	Multiplicity
`Header`	1
`DeviceStream` (organized by `Streams`)	0..*

Table 9: Part Properties of MTConnectStreams

Descriptions for Part Properties of `MTConnectStreams`:

- `Header`

  provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

- `DeviceStream`

  organizes data reported from a `Device`.

  `Streams` groups one or more `DeviceStream` entities. See *MTConnect Standard: Part 3.0 - Observation Information Model* for more detail.

5.3.2 Header

provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

5.3.2.1 Value Properties of Header

Table 10 lists the Value Properties of `Header`.

Value Property name	**Value Property type**	**Multiplicity**
`firstSequence`	`uint64`	1
`lastSequence`	`uint64`	1
`nextSequence`	`uint64`	1
`version`	`version`	1
`testIndicator`	`boolean`	0..1
`sender`	`string`	1
`instanceId`	`uint64`	1
`creationTime`	`datetime`	1
`bufferSize`	`uint32`	1
`deviceModelChangeTime`	`datetime`	1

Table 10: Value Properties of Header

Descriptions for Value Properties of `Header`:

- `firstSequence`

  sequence number assigned to the oldest piece of *streaming data* stored in the *buffer* of the *agent* immediately prior to the time that the *agent* published the *response document*.

- `lastSequence`

  sequence number assigned to the last piece of *streaming data* that was added to the *buffer* of the *agent* immediately prior to the time that the *agent* published the *response document*.

- `nextSequence`

  sequence number of the piece of *streaming data* that is the next piece of data to be retrieved from the *buffer* of the *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 *agent* at the time that the document was published, then the value reported for `nextSequence` **MUST** be equal to `lastSequence` + 1.

- `version`

  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 data model*.

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

- `testIndicator`

  indicates whether the *agent* that published the *response document* is operating in a test mode.

  If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `false`.

- `sender`

  identification defining where the *agent* that published the *response document* is installed or hosted.

  `sender` **MUST** be either an IP Address or Hostname describing where the *agent* is installed or the URL of the *agent*; e.g., `http://<address>[:port]/`.

  > Note: The port number need not be specified if it is the default HTTP port 80.

- `instanceId`

  identifier for a specific instantiation of the *buffer* associated with the *agent* that published the *response document*.

  `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.

- `creationTime`

  timestamp that an *agent* published the *response document*.

- `bufferSize`

  maximum number of *DataItems* that **MAY** be retained in the *agent* that published the *response document* at any point in time.

  > Note 1 to entry: `bufferSize` represents the maximum number of sequence numbers that **MAY** be stored in the *agent*.

  > Note 2 to entry: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `bufferSize`.

- `deviceModelChangeTime`

  timestamp of the last update of the `Device` information for any device.

5.4 MTConnectAssets Response Document

This section provides semantic information for the `MTConnectAssets` entity.

5.4.1 MTConnectAssets

root entity of an *MTConnectAssets Response Document* that contains the *Asset Information Model* of `Asset` types.

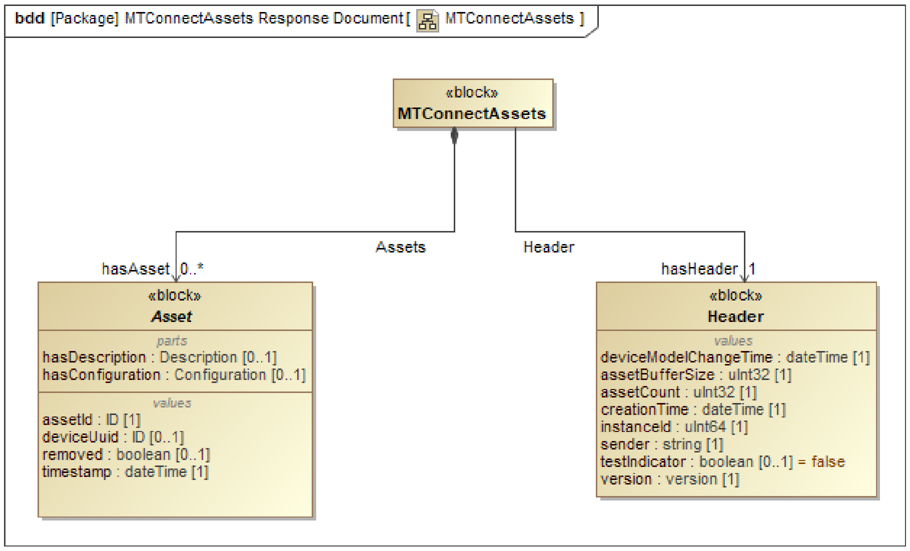

Figure 11: MTConnectAssets

> Note: Additional properties of `MTConnectAssets` **MAY** be defined for schema and namespace declaration. See *Section C - Schema and Namespace Declaration Information* for an XML example.

5.4.1.1 Part Properties of MTConnectAssets

Table 11 lists the Part Properties of `MTConnectAssets`.

Part Property name	Multiplicity
`Header`	1
`Asset` (organized by `Assets`)	0..*

Table 11: Part Properties of MTConnectAssets

Descriptions for Part Properties of `MTConnectAssets`:

- `Header`

  provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

- `Asset`

  abstract *Asset*.

  `Assets` groups one or more `Asset` types. See *MTConnect Standard: Part 4.0 - Asset Information Model* for more details.

5.4.2 Header

provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

5.4.2.1 Value Properties of Header

Table 12 lists the Value Properties of `Header`.

Value Property name	Value Property type	Multiplicity
`deviceModelChangeTime`	`datetime`	1
`assetBufferSize`	`uint32`	1
`assetCount`	`uint32`	1
`creationTime`	`datetime`	1
`instanceId`	`uint64`	1
`sender`	`string`	1
`testIndicator`	`boolean`	0..1
`version`	`version`	1

Table 12: Value Properties of Header

Descriptions for Value Properties of `Header`:

- `deviceModelChangeTime`

  timestamp of the last update of the `Device` information for any device.

- `assetBufferSize`

  maximum number of `Asset` types that can be stored in the *agent* that published the *response document*.

  > Note: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `assetBufferSize`.

- `assetCount`

  current number of `Asset` that are currently stored in the *agent* as of the `creationTime` that the *agent* published the *response document*.

  `assetCount` **MUST NOT** be larger than the value reported for `assetBufferSize`.

- `creationTime`

  timestamp that an *agent* published the *response document*.

- `instanceId`

  identifier for a specific instantiation of the *buffer* associated with the *agent* that published the *response document*.

  `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.

- `sender`

  identification defining where the *agent* that published the *response document* is installed or hosted.

  `sender` **MUST** be either an IP Address or Hostname describing where the *agent* is installed or the URL of the *agent*; e.g., `http://<address>[:port]/`.

  > Note: The port number need not be specified if it is the default HTTP port 80.

- `testIndicator`

  indicates whether the *agent* that published the *response document* is operating in a test mode.

  If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `false`.

- `version`

  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 data model*.

  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

6 Error Information Model

The *Error Information Model* establishes the rules and terminology that describes the *response document* returned by an *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 *agent* provides the information regarding errors encountered when processing a *request* for information by publishing an *MTConnectErrors Response Document* to the client software application that made the *request* for information.

6.1 MTConnectErrors Response Document

This section provides semantic information for the `MTConnectErrors` entity.

6.1.1 MTConnectError

root entity of an *MTConnectErrors Response Document* that contains the *Error Information Model*.

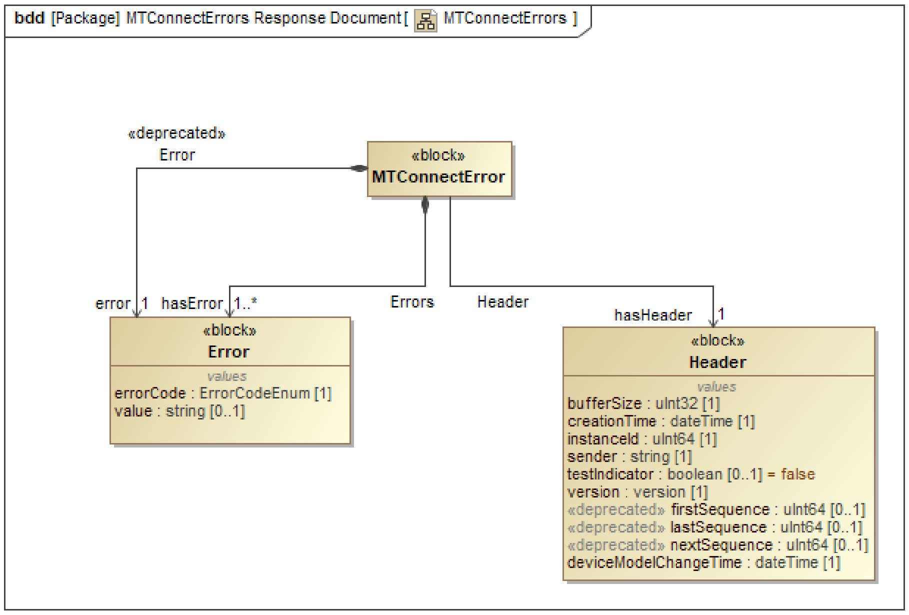
bdd [Package] MTConnectErrors Response Document [MTConnectErrors]
«deprecated»
Error
«block»
MTConnectError
error 1
hasError 1..*
Errors
Header
hasHeader 1
«block»
Error
values
errorCode : ErrorCodeEnum [1]
value : string [0..1]
«block»
Header
values
bufferSize : uInt32 [1]
creationTime : dateTime [1]
instanceId : uInt64 [1]
sender : string [1]
testIndicator : boolean [0..1] = false
version : version [1]
«deprecated» firstSequence : uInt64 [0..1]
«deprecated» lastSequence : uInt64 [0..1]
«deprecated» nextSequence : uInt64 [0..1]
deviceModelChangeTime : dateTime [1]

Figure 12: MTConnectError

> Note: Additional properties of `MTConnectError` **MAY** be defined for schema and namespace declaration. See *Section C - Schema and Namespace Declaration Information* for an XML example.

6.1.1.1 Part Properties of MTConnectError

Table 13 lists the Part Properties of `MTConnectError`.

Part Property name	Multiplicity
`Header`	1
`Error` (organized by `Errors`)	1..*
`<<deprecated>> Error`	1

Table 13: Part Properties of MTConnectError

Descriptions for Part Properties of `MTConnectError`:

- `Header`

  provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

- `Error`

  error encountered by an *agent* when responding to a *request*.

  `Errors` groups one or more `Error` entities. See *Section 6.1.3 - Error*.

  > 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` entity and the `Errors` entity **MUST NOT** appear in the document.

- `Error`

  error encountered by an *agent* when responding to a *request*.

6.1.2 Header

provides information from an *agent* defining version information, storage capacity, and parameters associated with the data management within the *agent*.

6.1.2.1 Value Properties of Header

Table 14 lists the Value Properties of `Header`.

Value Property name	**Value Property type**	**Multiplicity**
`bufferSize`	`uint32`	1
`creationTime`	`datetime`	1
`instanceId`	`uint64`	1
`sender`	`string`	1
`testIndicator`	`boolean`	0..1
`version`	`version`	1
`<<deprecated>> firstSequence`	`uint64`	0..1
`<<deprecated>> lastSequence`	`uint64`	0..1
`<<deprecated>> nextSequence`	`uint64`	0..1
`deviceModelChangeTime`	`datetime`	1

Table 14: Value Properties of Header

Descriptions for Value Properties of `Header`:

- `bufferSize`

  maximum number of *DataItems* that **MAY** be retained in the *agent* that published the *response document* at any point in time.

  > Note 1 to entry: `bufferSize` represents the maximum number of sequence numbers that **MAY** be stored in the *agent*.

  > Note 2 to entry: The implementer is responsible for allocating the appropriate amount of storage capacity required to accommodate the `bufferSize`.

- `creationTime`

  timestamp that an *agent* published the *response document*.

- `instanceId`

  identifier for a specific instantiation of the *buffer* associated with the *agent* that published the *response document*.

  `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.

- `sender`

  identification defining where the *agent* that published the *response document* is installed or hosted.

  `sender` **MUST** be either an IP Address or Hostname describing where the *agent* is installed or the URL of the *agent*; e.g., `http://<address>[:port]/`.

  > Note: The port number need not be specified if it is the default HTTP port 80.

- `testIndicator`

  indicates whether the *agent* that published the *response document* is operating in a test mode.

  If `testIndicator` is not specified, the value for `testIndicator` **MUST** be interpreted to be `false`.

- `version`

  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 data model*.

  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

- `<<deprecated>> firstSequence`

  sequence number assigned to the oldest piece of *streaming data* stored in the *buffer* of the *agent* immediately prior to the time that the *agent* published the *response document*.

- `<<deprecated>> lastSequence`

  sequence number assigned to the last piece of *streaming data* that was added to the *buffer* of the *agent* immediately prior to the time that the *agent* published the *response document*.

- `<<deprecated>> nextSequence`

  sequence number of the piece of *streaming data* that is the next piece of data to be retrieved from the *buffer* of the *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 *agent* at the time that the document was published, then the value reported for `nextSequence` **MUST** be equal to `lastSequence` + 1.

- `deviceModelChangeTime`

  timestamp of the last update of the `Device` information for any device.

6.1.3 Error

error encountered by an *agent* when responding to a *request*.

The value of `Error` **MUST** be `string`.

6.1.3.1 Value Properties of Error

Table 15 lists the Value Properties of `Error`.

Value Property name	Value Property type	Multiplicity
`errorCode`	`ErrorCodeEnum`	1

Table 15: Value Properties of Error

Descriptions for Value Properties of `Error`:

- `errorCode`

  descriptive code that indicates the type of error that was encountered by an *agent*.

  `ErrorCodeEnum` Enumeration:

  - `ASSET_NOT_FOUND`

    request for information specifies an `Asset` that is not recognized by the *agent*.

  - `INTERNAL_ERROR`

    agent experienced an error while attempting to published the requested information.

  - `INVALID_REQUEST`

    request contains information that was not recognized by the *agent*.

  - `INVALID_URI`

    Uniform Resource Identifier (URI) provided was incorrect.

- INVALID_XPATH

  XML Path Language (XPath) identified in the *request* for information could not be parsed correctly by the *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

  identity of the Device specified in the *request* for information is not associated with the *agent*.

- OUT_OF_RANGE

  request for information specifies *streaming data* that includes sequence number(s) for pieces of data that are beyond the end of the *buffer*.

- QUERY_ERROR

  agent was unable to interpret the query.

  The query parameters do not contain valid values or include an invalid parameter.

- TOO_MANY

  count parameter provided in the *request* for information requires either of the following:

  * *streaming data* that includes more pieces of data than the *agent* is capable of organizing in an *MTConnectStreams Response Document*.
  * Assets that include more Asset in an *MTConnectAssets Response Document* than the *agent* is capable of handling.

- UNAUTHORIZED

  requester does not have sufficient permissions to access the requested information.

- UNSUPPORTED

  valid *request* was provided, but the *agent* does not support the feature or type of *request*.

7 Profile

MTConnect Profile is a *profile* that extends the Systems Modeling Language (SysML) metamodel for the MTConnect domain using additional data types and *stereotypes*.

7.1 DataTypes

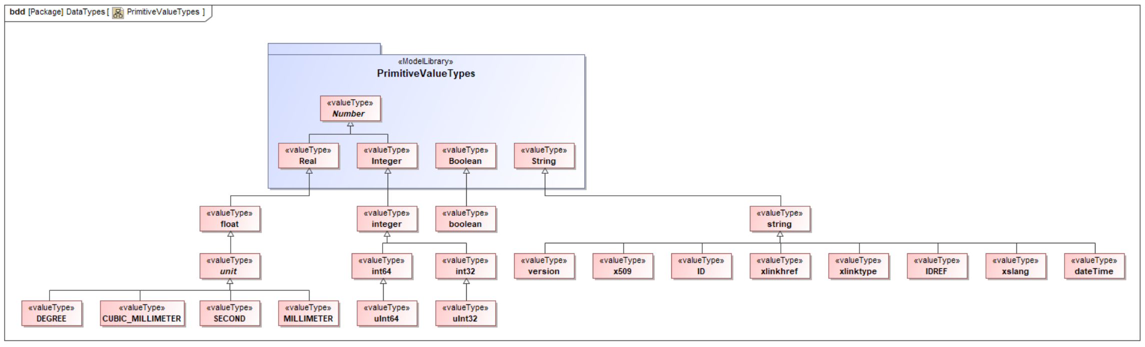

Figure 13: DataTypes

7.1.1 boolean

primitive type.

7.1.2 ID

string that represents an identifier (ID).

7.1.3 string

primitive type.

7.1.4 float

primitive type.

7.1.5 datetime

string that represents timestamp in ISO 8601 format.

7.1.6 integer

primitive type.

7.1.7 xlinktype

string that represents the type of an XLink element. See `https://www.w3.org/TR/xlink11/`.

7.1.8 xslang

string that represents a language tag. See `http://www.ietf.org/rfc/rfc4646.txt`.

7.1.9 SECOND

float that represents time in seconds.

7.1.10 IDREF

string that represents a reference to an `ID`.

7.1.11 xlinkhref

string that represents the locator attribute of an XLink element. See `https://www.w3.org/TR/xlink11/`.

7.1.12 x509

string that represents an `x509` data block. *Ref ISO/IEC 9594-8:2020*.

7.1.13 int32

32-bit integer.

7.1.14 int64

64-bit integer.

7.1.15 version

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*.

7.1.16 uint32

32-bit unsigned integer.

7.1.17 uint64

64-bit unsigned integer.

7.2 Stereotypes

7.2.1 organizer

element that *organizes* other elements of a type.

7.2.2 deprecated

element that has been deprecated.

7.2.3 extensible

enumeration that can be extended.

7.2.4 informative

element that is descriptive and non-normative.

7.2.5 valueType

extends SysML `<<ValueType>>` to include `Class` as a value type.

7.2.6 normative

element that has been added to the standard.

7.2.7 observes

association in which a *Component* makes *Observations* about an observable *DataItem*.

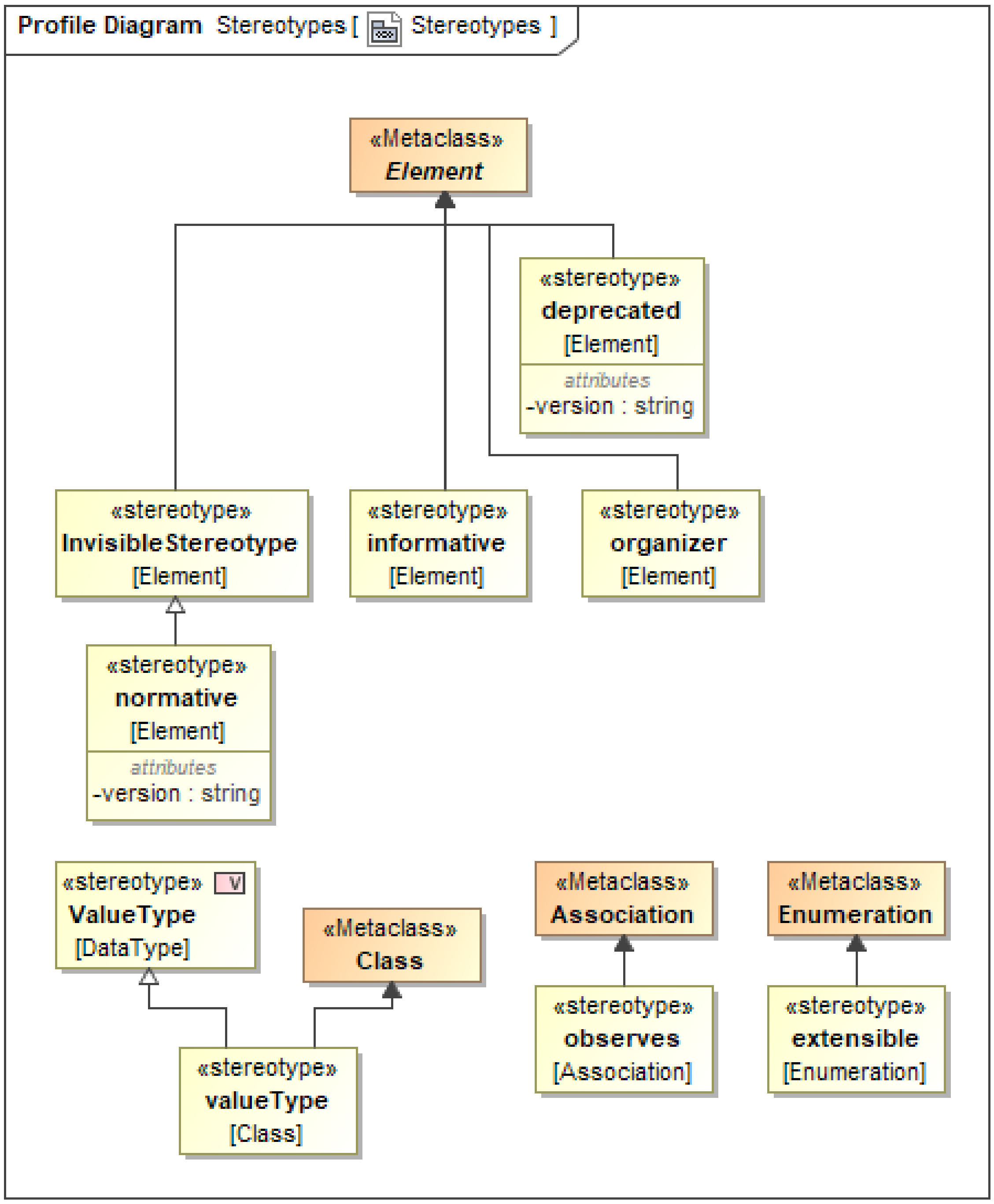
Profile Diagram Stereotypes [Stereotypes]
«Metaclass» Element
«stereotype» deprecated [Element]
attributes
-version : string
«stereotype» InvisibleStereotype [Element]
«stereotype» informative [Element]
«stereotype» organizer [Element]
«stereotype» normative [Element]
attributes
-version : string
«stereotype» ValueType [DataType]
V
«Metaclass» Class
«stereotype» valueType [Class]
«Metaclass» Association
«stereotype» observes [Association]
«Metaclass» Enumeration
«stereotype» extensible [Enumeration]

Figure 14: Stereotypes

Appendices

A Bibliography

Engineering Industries Association. EIA Standard - EIA-274-D, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.

ISO TC 184/SC4/WG3 N1089. ISO/DIS 10303-238: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.

International Organization for Standardization. ISO 14649: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.

International Organization for Standardization. ISO 14649: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.

International Organization for Standardization. ISO 6983/1 – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.

Electronic Industries Association. ANSI/EIA-494-B-1992, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.

National Aerospace Standard. Uniform Cutting Tests - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.

International Organization for Standardization. ISO 10303-11: 1994, Industrial automation systems and integration Product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

International Organization for Standardization. ISO 10303-21: 1996, Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.

H.L. Horton, F.D. Jones, and E. Oberg. Machinery's Handbook. Industrial Press, Inc.

New York, 1984.

International Organization for Standardization. ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature. Geneva, Switzerland, 2001.

ASME B5.57: Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers, 1998.

ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Controlled Machining Centers. 2005.

OPC Foundation. OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.

IEEE STD 1451.0-2007, Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH99684, October 5, 2007.

IEEE STD 1451.4-1994, Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH95225, December 15, 2004.

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 Character Data (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 element-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 is *Example 4*:

Example 4: Example of attributes for an element

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

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

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

An example of CDATA associated with an XML element would be *Example 5*:

Example 5: Example of cdata associated with element

```
<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 *Section C - Schema and Namespace Declaration Information* 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.

Example 6 demonstrates the hierarchy of XML elements and *child elements* used to form an XML document:

Example 6: Example of hierarchy of XML elements

```
<Root Level>     (Parent Element)
  <First Level>   (Child Element to Root Level and
  Parent Element to Second Level)
    <Second Level>  (Child Element to First Level
    and Parent Element to Third Level)
      <Third Level name="N1"></Third Level>
      (Child Element to Second Level)
      <Third Level name="N2"></Third Level>
      (Child Element to Second Level)
      <Third Level name="N3"></Third Level>
      (Child Element to Second Level)
    </Second Level>   (end-tag for Second Level)
  </First Level>   (end-tag for First Level)
</Root Level>   (end-tag for Root Level)
```

In the *Example 6*, *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*.

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 D - 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 *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 *Example 7*, 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 `MTCconnectDevices` that declare the XML *schema* and *namespace* associated with an *MTConnectDevices Response Document*.

Example 7: Example of schema and namespace declaration

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

D 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 *DataItems*.
- 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 *DataItems*,
- **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*, is shown in *Example 8*:

Example 8: Example of extended schema and namespace in declaration

```
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*, *DataItem*, and 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*, *DataItem*, or asset. The prefix identifies the *schema* and *namespace* where that XML Element is defined.