MTConnect® Standard

Part 1 - Overview and Protocol

Version 1.1.0 – Final

Prepared for: MTConnect Institute
Prepared by: William Sobel
Prepared on: April 30, 2010

MTConnect® Specification

If you intend to adopt or implement this MTConnect® Specification in a product, whether hardware, software or firmware, which complies with the MTConnect® Specification, you must agree to the MTConnect® Specification Implementer License Agreement ("Implementer License") or to the MTConnect® Intellectual Property Policy and Agreement ("IP Policy"). The Implementer License and IP Policy each sets forth the license terms and other terms of use for MTConnect® Implementers to adopt or implement the MTConnect® Specifications, including certain license rights covering necessary patent claims for that purpose. These materials can be found at www.MTConnect.org, or by contacting Paul Warndorf at mailto:pwarndorf@mtconnect.hyperoffice.com.

MTConnect® Institute and AMT have no responsibility to identify patents, patent claims or patent applications which may relate to or be required to implement a Specification, or to determine the legal validity or scope of any such patent claims brought to their attention. Each MTConnect® Implementer is responsible for securing its own licenses or rights to any patent or other intellectual property rights that may be necessary for such use, and neither AMT nor MTConnect® Institute have any obligation to secure any such rights.

The MTConnect® Specification is provided "as is" and MTConnect® Institute and AMT, and each of their respective members, officers, affiliates, sponsors and agents, make no representation or warranty of any kind relating to these materials or to any implementation of the MTConnect® Specification in any product, including, without limitation, any express or implied warranty of noninfringement, merchantability, or fitness for particular purpose, or of the accuracy, reliability, or completeness of information contained herein. In no event shall MTConnect® Institute or AMT be liable to any user or implementer of the MTConnect® Specification for the cost of procuring substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, indirect, special or punitive damages or other direct damages, whether under contract, tort, warranty or otherwise, arising in any way out of access, use or inability to use the MTConnect® Specification or other MTConnect® Materials, whether or not they had advance notice of the possibility of such damages.

Table of Contents

Table of Figures

1 Overview

MTConnect is a standard based on an open protocol for data integration. MTConnect® is not intended to replace the functionality of existing products, but it strives to enhance the data acquisition capabilities of devices and applications and move toward a plug-and-play environment to reduce the cost of integration.

MTConnect® is built upon the most prevalent standards in the manufacturing and software industry, maximizing the number of tools available for its implementation and providing the highest level of interoperability with other standards and tools in these industries.

To facilitate this level of interoperability, a number of objectives are being met. Foremost is the ability to transfer data via a standard protocol which includes:

- A device identity (i.e. model number, serial number, calibration data, etc.).
- The identity of all the independent components of the device.
- Possibly a device's design characteristics (i.e. axis length, maximum speeds, device thresholds, etc.).
- Most importantly, data captured in real or near-real-time (i.e. current speed, position data, temperature data, program block, etc.) by a device that can be utilized by other devices or applications (e.g. utilized by maintenance diagnostic systems, management production information systems, CAM products, etc.).

The types of data that may need to be addressed in MTConnect® could include:

- Physical and actual device design data
- Measurement or calibration data
- Near-real-time data from the device

To accommodate the vast amount of different types of devices and information that may come into play, MTConnect® will provide a common high-level vocabulary and structure.

1.1 MTConnect® Document Structure

The MTConnect® specification is subdivided using the following scheme:

Part 1: Overview and Protocol – Version 1.1.0 Final
Part 2: Components and Data Items – Version 1.1.0 Final
Part 3: Streams, Events, Samples, and Condition – Version 1.1.0, Final

Extensions to the standard will be made according to this scheme and new sections will be added as new areas are addressed. Documents will be named as follows: MTC_Part_<Number>_<Description>.doc. All documents will be developed in Microsoft® Word format and released in Adobe® PDF format. For example, this document is MTC_Part_1_Overview.doc.

1.2 MTConnect Versions and Backward Compatibility

MTConnect® uses a three digit version numbering system consisting of a *major.minor.revision*, for example, a version number 1.1.4 would be major=1, minor=2, and revision=4. The major revision changes indicate that major changes to the standard have been made and backward compatibility **MAY** not be possible. This means that the schema may have changed in ways that will require the applications to change the way the request and interpret the data so they **MUST** be fully version aware and using the same requests across major versions **MAY NOT** work. The standard will still try to maintain as much backward compatibility as possible to preserve the investment in existing software development.

A minor version will introduce new components and data items and minor structural changes, additions only. With a minor release applications will only require minor changes to accept the changes and will still be able to function with older agents. Protocol changes will be kept to a minimum so application can use the same request semantics across versions. A minor version change will only DEPRECATE existing content and mark it for remove in future major version changes. This allows previous implementations to use new components and still function correctly.

Both major and minor changes **MUST** require a ninety day review of the standard by the technical advisory group (TAG). This requirement is to ensure that the additional are free from any intellectual property or copyright violations.

Revision changes will be editorial corrections and will introduce no new functionality. These changes **MUST NOT** require any changes to the application and implementation of the supporting software. Revisions **MUST NOT** require any review period since there is no new structure or functionality introduced.

2 Purpose of This Document

This document is intended to:

- define the MTConnect® standard;
- specify the requirements for compliance with the MTConnect® standard;
- provide engineers with sufficient information to implement *Agents* for their devices;
- provide developers with the necessary guidelines to use the standard to develop applications.

Part 1 of the MTConnect Standard provides an overview of the MTConnect Structure and Overview of the Protocol; including the communication between devices, fault tolerance, connectivity handling, and error handling.

The document is organized as follows:

- Section 3 discusses the architecture and the MTConnect® standard in relation to the other devices and processes. A brief discussion of the high level data flow is also given to frame the scope of the standard.
- Section 4 provides the structure of the protocol header which will be discussed in detail in section 5.
- Section 5 provides detailed information on the MTConnect® protocol and how processes will communicate and recover from failure.

2.1 Terminology

Adapter An optional software component that connects the Agent to the Device.

Agent A process that implements the MTConnect® HTTP protocol, XML generation, and MTConnect protocol.

Alarm An alarm indicates an event that requires attention and indicates a deviation from normal operation.

Application A process or set of processes that access the MTConnect® *Agent* to perform some task.

Attribute A part of an element that provides additional information about that element. For example, the `name` element of the `Device` is given as `<Device` **`name="mill-1"`**`>...</Device>`

CDATA The text in a simple content element. For example, *`This is some text`*, in `<mt:Alarm ...>This is some text</mt:Alarm>`.

Component A part of a device that can have sub-components and data items. A component is a basic building block of a device.

Controlled Vocabulary The value of an element or attribute is limited to a restricted set of possibilities. Examples of controlled vocabularies are country codes: US, JP, CA, FR, DE, etc…

Current A snapshot request to the *Agent* to retrieve the current values of all the data items specified in the path parameter. If no path parameter is given, then the values for all components are provided.

Data Item A data item provides the descriptive information regarding something that can be collected by the *Agent*.

Device A piece of equipment capable of performing an operation. A device is composed of a set of components that provide data to the application. The device is a separate entity with at least one Controller managing its operation.

Discovery Discovery is a service that allows the application to locate *Agents* for devices in the manufacturing environment. The discovery service is also referred to as the *Name Service.*

Element An XML element is the central building block of any XML Document. For example, in MTConnect® the Device element is specified as `<`**`Device`**` >...</`**`Device`**`>`

Event An event represents a change in state that occurs at a point in time. Note: An event does not occur at predefined frequencies.

HTTP Hyper-Text Transport Protocol. The protocol used by all web browsers and web applications.

Instance When used in software engineering, the word *instance* is used to define a single physical example of that type. In object-oriented models, there is the class that describes the thing and the instance that is an example of that thing.

LDAP Lightweight Directory Access Protocol, better known as Active Directory in Microsoft Windows. This protocol provides resource location and contact information in a hierarchal structure.

MIME Multipurpose Internet Mail Extensions. A format used for encoding multipart mail and http content with separate sections separated by a fixed boundary.

Probe A request to determine the configuration and reporting capabilities of the device.

REST REpresentational State Transfer. A software architecture where the client and server move through a series of state transitions based solely on the request from the client and the response from the server.

Results A general term for the `Samples`, `Events`, and `Condition` contained in a `ComponentStream` as a response from a `sample` or `current` request.

Sample A sample is a data point from within a continuous series of data points. An example of a Sample is the position of an axis.

Socket When used concerning interprocess communication, it refers to a connection between two end-points (usually processes). Socket communication most often uses TCP/IP as the underlying protocol.

Stream A collection of Events, Condition, and Samples organized by devices and components.

Service An application that provides necessary functionality.

Tag Used to reference an instance of an XML element.

TCP/IP TCP/IP is the most prevalent stream-based protocol for interprocess communication. It is based on the IP stack (Internet Protocol) and provides the flow-control and reliable transmission layer on top of the IP routing infrastructure.

URI Universal Resource Identifier. This is the official name for a web address as seen in the address bar of a browser.

UUID Universally unique identifier.

XPath XPath is a language for addressing parts of an XML Document. See the XPath specification for more information. http://www.w3.org/TR/xpath

XML Extensible Markup Language. http://www.w3.org/XML/

XML Schema The definition of the XML structure and vocabularies used in the XML Document.

XML Document An instance of an XML Schema which has a single root element and conforms to the XML specification and schema.

XML NMTOKEN The data type for XML identifiers. It must start with a letter, an underscore "_" or a colon ":" and then it **MUST** be followed by a letter, a number, or one of the following ".", "-", "_", ":". An NMTOKEN cannot have any spaces or special characters.

2.2 XML Terminology

In the document there will be references to XML constructs, including elements, attributes, CDATA, and more. XML consists of a hierarchy of elements. The elements can contain sub-elements, CDATA, or both. For this specification, however, an element never contains mixed content or both sub-elements and CDATA. Attributes are additional information associated with an *element*. The textual representation of an element is referred to as a *tag*. In the example:

```
1. <Foo name=”bob”>Ack!</Foo>
```

An *element* consists of a named opening and closing tag. In the above example, <Foo...> is referred to as the opening tag and </Foo> is referred to as the closing tag. The text Ack! in between the opening and closing tags is called the CDATA. CDATA can be restricted to certain formats, patterns, or words. In the document when it refers to an element having CDATA, it indicates that the element has no sub-elements and only contains data.

When one looks at an XML Document there are two parts. The first part is typically referred to as an XML declaration and is only a single line. It looks something like this:

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

This line indicates the XML version being used and the character encoding. Though it is possible to leave this line off, it is usually considered good form to include this line in the beginning of the document.

Every XML Document contains one and only one root element. In the case of MTConnect, it is the `MTConnectDevices`, `MTConnectStreams`, or `MTConnectError` element. When these root elements are used in the examples, you will sometimes notice that it is prefixed with `mt:` as in `mt:MTConnectDevices`. The `mt:` is what is referred to as a namespace. In XML, to allow for multiple XML Schemas to be used within the same XML Document, a namespace will indicate which XML Schema is in effect for this section of the document. This convention allows for multiple XML Schemas to be used within the same XML Document, even if they have the same element names. The namespace is optional and is only required if multiple schemas are required.

An *attribute* is additional data that can be included in each XML element. For example, in the following MTConnect® `DataItem`, there are several attributes describing the data item:

```
<DataItem name=”Xpos” type=”POSITION” subType=”ACTUAL”
category=”SAMPLE” />
```

The `name`, `type`, `subType`, and `category` are attributes of the element. Each attribute can only occur once within an element declaration, and it can either be required or optional.

An element can have any number of sub-elements. The XML Schema specifies which sub-elements and how many times a given sub-element can occur. Here’s an example:

```
<TopLevel>
  <FirstLevel>
    <SecondLevel>
      <ThirdLevel name=”first”></ThirdLevel>
      <ThirdLevel name=”second”></ThirdLevel>
    </SecondLevel>
  </FirstLevel>
</TopLevel>
```

In the above example, the `FirstLevel` has a sub-element `SecondLevel` which in turn has two sub-elements, `ThirdLevel`, with different names. Each level is an element and its children are its sub-elements and so forth.

In XML we sometimes use elements to organize parts of the document. A few examples in MTConnect® are `Streams`, `DataItems`, and `Components`. These elements have no attributes or data of their own; they only provide structure to the document and allow for various parts to be addressed easily.

```
…
<Device id=”d” name=”Device”>
  <DataItems>
    <DataItem …/>
    …
  </DataItems>
  <Components>
    <Axes … >…</Axes>
  </Components>
</Device>
```

In the previous example `DataItems` and `Components` are only used to contain certain types of elements and provide structure to the documents. These elements will be referred to as *Containters* in the standard.

An XML Document can be validated. The most basic check is to make sure it is well-formed, meaning that each element has a closing tag, as in `<foo>...</foo>` and the document does not contain any illegal characters (<>) when not specifying a tag. If the closing `</foo>` was left off or an extra > was in the document, the document would not be well-formed and may be rejected by the receiver. The document can also be validated against a schema to ensure it is valid. This second level of analysis checks to make sure that required elements and attributes are present and only occur the correct number of times. A valid document must be well-formed.

All MTConnect® documents must be valid and conform to the XML Schema provided along with this specification. The schema will be versioned along with this specification. The greatest possible care will be taken to make sure that the schema is backward compatible.

For more information, visit the w3c website for the XML Standards documentation: http://www.w3.org/XML/

2.3 Markup Conventions

MTConnect® follows industry conventions on tag format and notations when developing the XML schema. The general guidelines are as follows:

1. All tag names will be specified in Pascal case (first letter of each word is capitalized). For example: <ComponentEvents />
2. Attribute names will also be camel case, similar to Pascal case, but the first letter will be lower case. For example: <MyElement attributeName=”bob”/>
3. All values that are part of a limited or controlled vocabulary will be in upper case with an _ (underscore) separating words. For example: `ON`, `OFF`, `ACTUAL`, `COUNTER_CLOCKWISE`, etc…
4. Dates and times will follow the W3C ISO 8601 format with arbitrary fractions of a second allowed. Refer to the following specification for details: http://www.w3.org/TR/NOTE-datetime The format will be YYYY-MM-DDThh:mm:ss.ffff, for example 2007-09-13T13:01.213415. The accuracy and number of fractional digits of the timestamp is determined by the capabilities of the device collect-ing the data. All times will be given in UTC (GMT).
5. Element names will be spelled-out and abbreviations will be avoided. The one exception is the word `identifier` that will be abbreviated `Id`. For example: `SequenceNumber` will be used instead of `SeqNum`.

2.4 Document Conventions

The following documentation conventions will be used in the text:

- The word **MUST** is used to indicate provisions that are mandatory. Any deviation from those provisions will not be permitted.
- The word **SHOULD** is used to indicate a provision that is recommended but the exclusion of which will not invalidate the implementation.
- The word **MAY** will be used to indicate provisions that are optional and are up to the implementer to decide if they are relevant to their device.
- The word **NOT** will be added to any of the previous words to emphasize the negation of this provision.

In the tables where elements are described, the Occurrence column indicates if the attribute or sub-elements are required by the specification.

For attributes:

1. If the Occurrence is 1, the attribute **MUST** be provided.
2. If the Occurrence is 0..1, the attribute **MAY** be provided, and at most one occurrence of the attribute may be given.

For elements:

1. If the Occurrence is 1, the element **MUST** be provided.
2. If the Occurrence is 0..1, the element **MAY** be provided, and at most one occurrence of the element may be given.
3. If the Occurrence is 1..INF, one or more elements **MUST** be provided.
4. If the Occurrence is a number, e.g. 2, exactly that number of elements **MUST** be provided.

Font styles used:

Code samples as well as any XML elements or attributes will always be given in `fixed width fonts`. References to other *Documents* or *Sections* will be presented in italics.

2.5 Units

MTConnect® will adopt the units common to most standards specifications for exchanging data items. These units have been selected by the working group as giving the greatest interoperability and common acceptance.

Property	Symbol	Unit
Angle	°	decimal degree
Angular Acceleration	°/s^2	degree per second squared
Angular Velocity	°/s	degrees per second
Elapsed time	s	seconds with fractions

Property	Symbol	Unit
Force	N	newtons
Length	mm	millimeters
Linear Acceleration	mm/s^2	millimeter per second squared
Linear Velocity	mm/s	millimeter per second
Mass	kg	kilogram
Rotary Velocity	rev/min	revolution per minute
Spindle Speed	rev/min	revolution per minute
Temperature	°C	degree Celsius
Time	Sec	second
Torque	N m	newton meter

2.6 Referenced Standards and Specifications

A large number of specifications are being used to normalize and harmonize the schema and the vocabulary (names of tags and attributes) specified in MTConnect® *(See Appendix A: Bibliography for complete references).*

3 Architectural Overview

MTConnect® is built upon the most prevalent standards in the industry. This maximizes the number of tools available for implementation and provides the highest level of interoperability with other standards and protocols.

MTConnect® **MUST** use the HTTP protocol as the underlying transport for all messaging. The data **MUST** be sent back in valid XML, according to this standard. Each MTConnect® *Agent* **MUST** represent at least one device. The Agent **MAY** represent more than one device if desired.

MTConnect® is composed of a few basic conceptual parts. They are as follows:

Header Protocol related information. (*See Header in Part 1 Section 4)*

Components The building blocks of the device. *(See Components in Part 2 Section 3)*

DataItems The description of the data available from the device. *(See DataItems in Part 2 Section 4)*

Streams A set of Samples, Events, or Conditon for components and devices. *(See Streams in Part 3)*

Samples A point-in-time measurement of a data item that is continuously changing. *(See Samples in Part 3)*

Events Discrete changes in state that can have no intermediate value. They indicate the state of a specific attribute of a component. *(See Events in Part 3)*

Condition A piece of information the device provides as an indicator of its health and ability to function. A condition can be one of `Normal`, `Warning`, `Fault`, or `Unavailable`. A single condition type can have multiple Faults or Warnings at any given time. This behavior is different from Events and Samples where a data item **MUST** only have a single value at a given time. *(See Condition in Part 3).*

3.1 Request Structure

An MTConnect® request **SHOULD NOT** include any body in the HTTP request. If the *Agent* receives any additional data, the *Agent* **MAY** ignore it. There will be no cookies or additional information considered; the only information the *Agent* **MUST** consider is the URI in the HTTP GET (Type a URI into the browser's address bar, hit return, and a GET is sent to the server. In fact, with MTConnect® one can do just that. To test the Agent, one can type the Agent's URI into the browser's address bar and view the results.)

3.2 Process Workflow

What follows is the typical interaction between four entities in the MTConnect® architecture: the *Name Service* (an LDAP server that translates device names to the Agent's URI), the *Application* (a user application that makes special use of the device's data), the *Agent* (the process collecting data from the device and delivering it to the applications), and the *Device* (the physical piece of equipment).

Note: Refer to Appendix B for more information on LDAP and the requirements for its use.

3.2.1 Agent Initialization

For this example, the agent first authenticates itself with the Name Server (if used). In the second part of the example, it shows how the entities interrelate in an architecture.

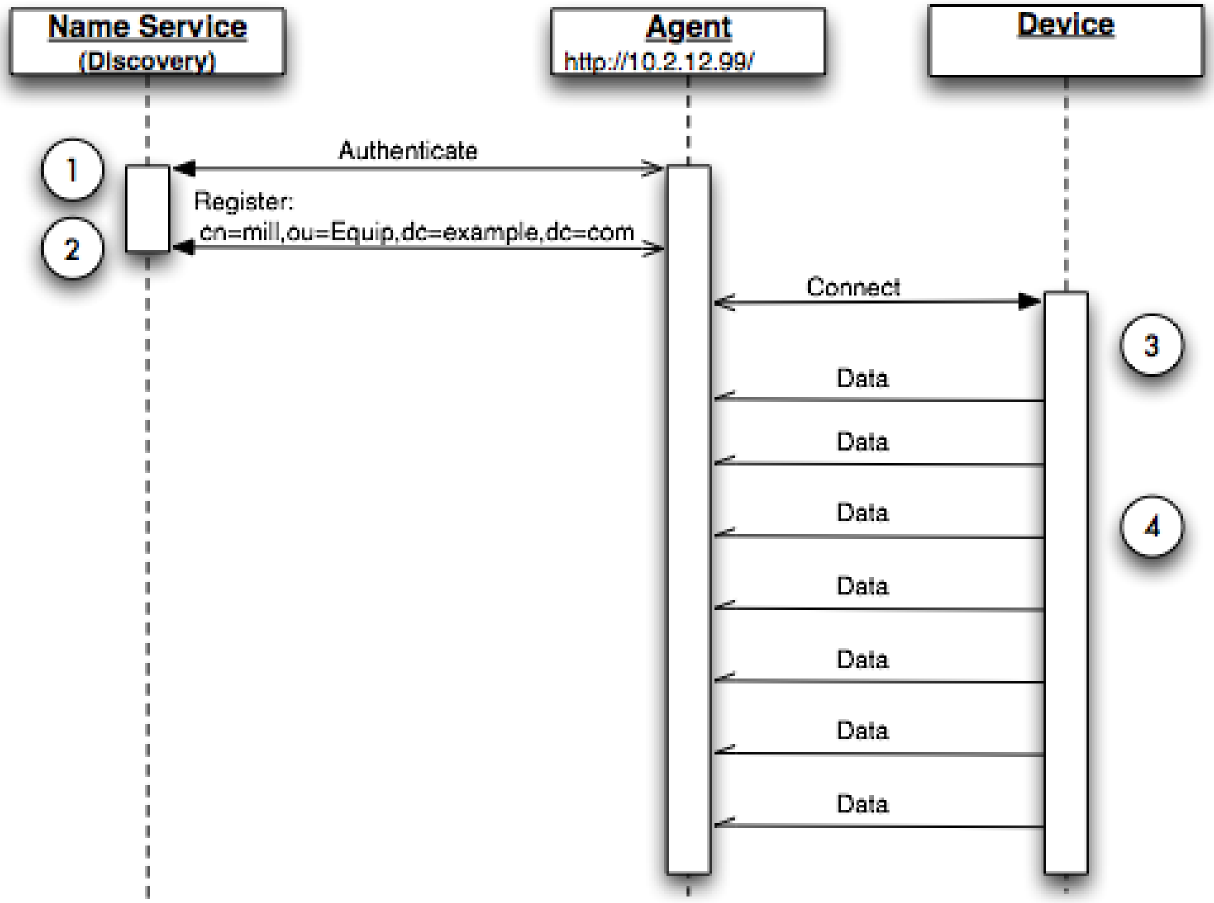


Figure 1: Agent Initialization

The diagram above illustrates the initialization of the *Agent* and communication with the device. *Implementors Note:* This is the recommended architecture and implementations **SHOULD** refer to this when developing their MTConnect® Agents.

Step 1 The Agent connects and authenticates itself with the Name Service (LDAP server).

Step 2 The Agent registers its URI with the Name Service so it can be located.

Step 3 The Agent connects to the Device using the device's API or another specialized process.

Step 4 The device sends data to the Agent or the Agent polls the device for data.

3.2.2 Application Communication

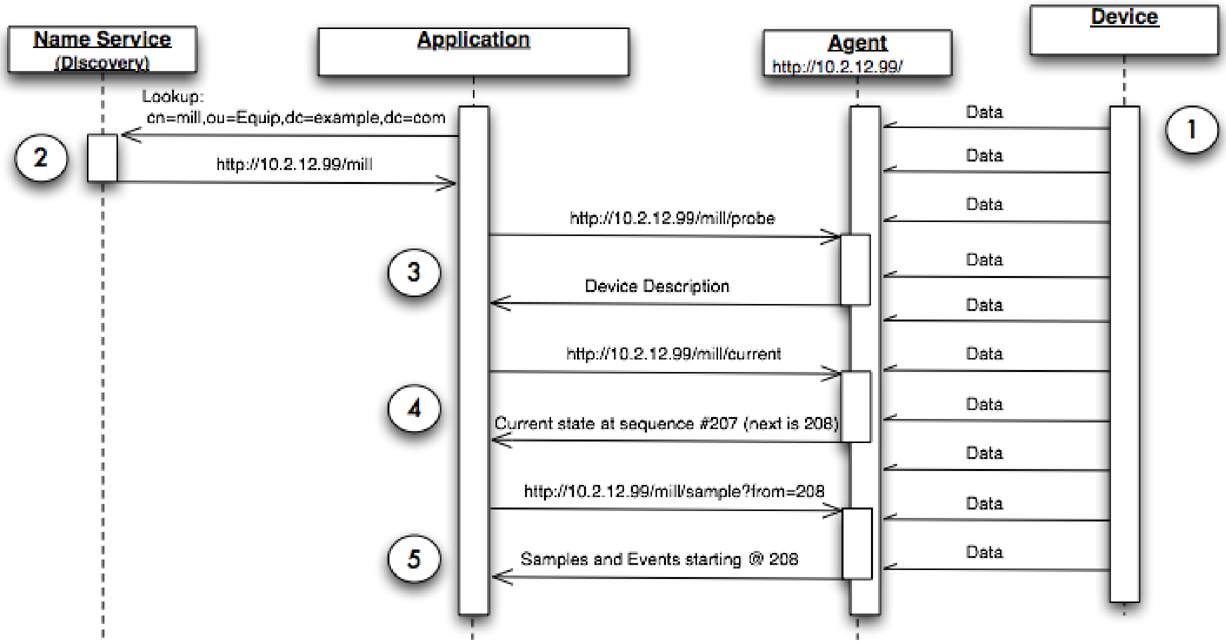

Name Service (Discovery)
Application
Agent
http://10.2.12.99/
Device
Lookup:
cn=mill,ou=Equip,dc=example,dc=com
2
http://10.2.12.99/mill
1
Data
Data
Data
http://10.2.12.99/mill/probe
3
Device Description
Data
Data
http://10.2.12.99/mill/current
Data
4
Current state at sequence #207 (next is 208)
Data
Data
http://10.2.12.99/mill/sample?from=208
Data
5
Samples and Events starting @ 208
Data

Figure 2: Application Communication

The preceding diagram shows how all major components of an MTConnect® architecture inter-relate and how the four basic operations are used to locate and communicate with the *Agent* regarding the device.

Step 1 The device is continually sending information to the Agent. The Agent is collecting the information and saving it based on its ability to store information. The data flow from the device to the agent is implementation dependant. The data flow can begin once a request has been issued from a client application at the discretion of the agent.

Step 2 The Application locates the device using the *Name Service* with the standard LDAP syntax that is interpreted as follows: the mill is in the organizational unit of Equip which is in the example.com domain. The LDAP record for this device will contain a URI that the Application can use to contact the Agent.

Step 3 The Application has the URI to contact the Agent for the mill device. The first step is a request for the device's descriptive information using the `probe` request. The `probe` will return the component composition of the device as well as all the data items available.

Step 4 The Application requests the `current` state for the device. The results will contain the device stream and all the component streams for this device. Each of the data items will report their values as Samples, Events or Condition. The application will receive the `nextSequence` number from the *Agent* to use in the subsequent sample request.

Step 5 The Application uses the `nextSequence` number to `sample` the data from the Agent starting at sequence number 208. The results will be Events, Condition, and Samples; and the count is not specified, so it defaults to 100.

This will be discussed in more detail in the *Protocol* section of the document. The remainder of this document will assume the *Name Service* discovery has already been completed.

4 Reply XML Document Structure

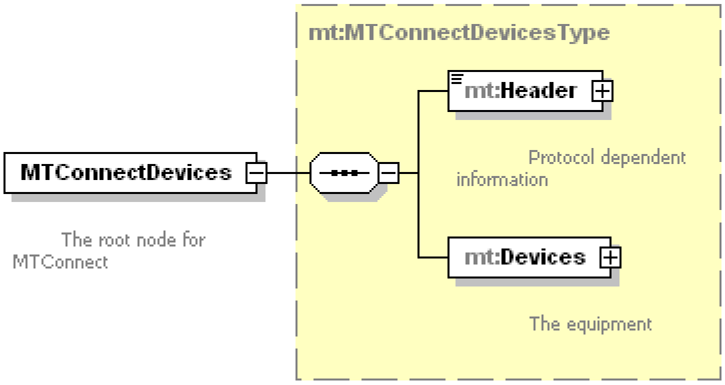
At the top level of all MTConnect® XML Documents there **MUST** be one of the following elements: MTConnectDevices, MTConnectStreams, or MTConnectError. This element will be the root for all MTConnect® responses and contains all sub-elements for the protocol.

All MTConnect® XML Documents are broken down into two parts. The first element is the Header that provides protocol related information like next sequence number and creation date and the second section provides the content for Devices, Streams, or Errors.

The top level elements **MUST** contain references to the XML schema URN and the schema location. This is the standard XML schema attributes:

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

4.1 MTConnectDevices



Generated by XMLSpy www.altova.com

Figure 3: MTConnectDevices structure

MTConnectDevices provides the descriptive information about each device served by this *Agent* and specifies the data items that are available. In an MTConnectDevices XML Document, there **MUST** be a Header and it **MUST** be followed by Devices section. An MTConnectDevices XML Document **MUST** have the following structure (the details have been eliminated for illustrative purposes):

```
<MTConnectDevices xmlns:m="urn:mtconnect.com:MTConnectDevices:1.1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="urn:mtconnect.com:MTConnectDevices:1.1"
   xsi:schemaLocation="urn:mtconnect.com:MTConnectDevices:1.1
      http://www.mtconnect.org/schemas/MTConnectDevices_1.1.xsd">
  <Header> … </Header>
  <Devices> … </Devices>
</MTConnectDevices>
```

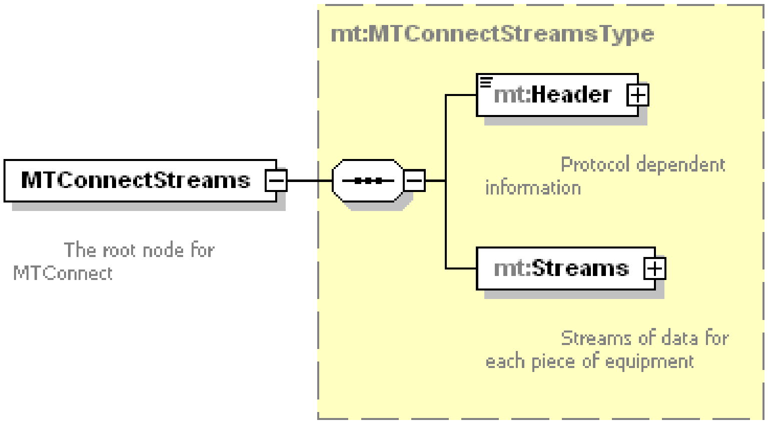
4.1.1 MTConnectDevices Elements

An MTConnectDevices element **MUST** include the Header for all documents and the Devices element.

Element	Description	Occurrence
Header	A simple header with next sequence and creation time	1
Devices	The root of the descriptive data	1

For the above elements of the XML Document, please refer to Part 1 section 4.4 for Header and Part 2 section 3 Components and Devices.

4.2 MTConnectStreams


mt:MTConnectStreamsType
mt:Header
Protocol dependent information
MTConnectStreams
The root node for MTConnect
mt:Streams
Streams of data for each piece of equipment

Generated by XMLSpy www.altova.com

Figure 4: MTConnectStreams structure

MTConnectStreams contains a timeseries of Samples, Events, and Condition from devices and their components. In an MTConnectStreams XML Document, there **MUST** be a Header and it **MUST** be followed by a Streams section. An MTConnectStreams XML Document will have the following structure (the details have been eliminated for illustrative purposes):

```
<MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:1.1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="urn:mtconnect.com:MTConnectStreams:1.1"
   xsi:schemaLocation="urn:mtconnect.com:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams.xsd">
  <Header> … </Header>
```

```
   <Streams> … </Streams>
</MTConnectStreams>
```

4.2.1 **MTConnectStreams Elements**

An `MTConnectStreams` document **MUST** include a Header and a `Streams` element.

Element	Description	Occurrence
`Header`	A simple header with next sequence and creation time	1
`Streams`	The root of the sample and event data	1

For the above elements of the XML Document, please refer to Part 1 section 4.4 for `Header` and Part 3 section 3 for `Streams`.

4.3 MTConnectError

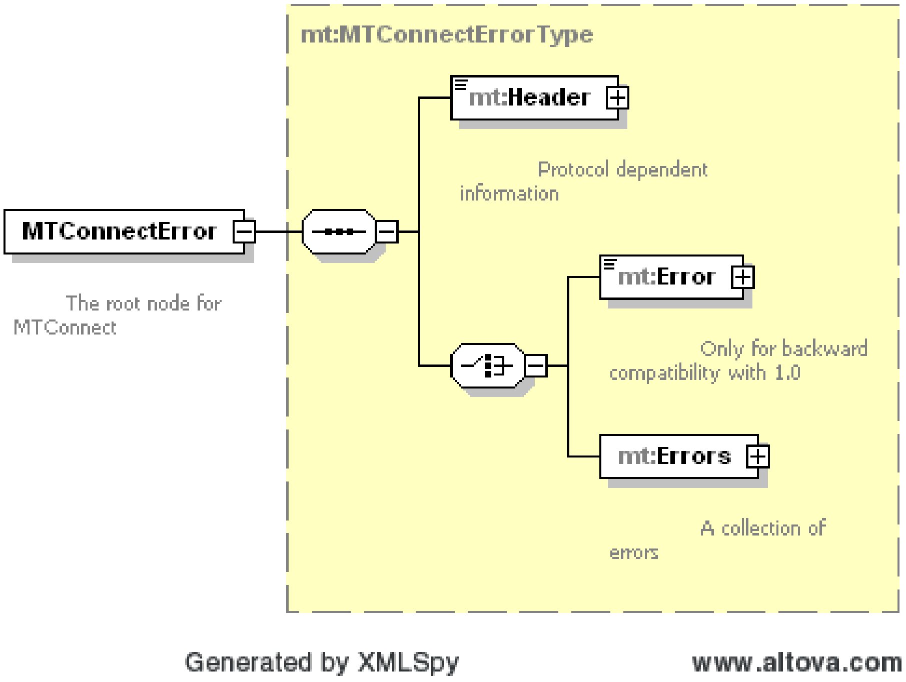


Figure 5: MTConnectError structure

An `MTConnectError` document contains information about an error that occurred in processing the request. In an `MTConnectError` XML Document, there **MUST** be a `Header` and it must be followed by an `Errors` container that can contain a series of `Error` elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectError xmlns="urn:mtconnect.org:MTConnectError:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mtconnect.org:MTConnectError:1.1
http://www.mtconnect.org/schemas/MTConnectError_1.1.xsd">
```

```
  <Header creationTime="2010-03-12T12:33:01" sender="localhost"
version="1.1" bufferSize="131072" instanceId="1268463594" />
  <Errors>
    <Error errorCode="OUT_OF_RANGE" >Argument was out of range</Error>
    <Error errorCode="INVALID_PATH" >Bad path</Error>
  </Errors>
</MTConnectError>
```

For purposes of backward compatibility, a single error can have a single `Error` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectError xmlns="urn:mtconnect.org:MTConnectError:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mtconnect.org:MTConnectError:1.1
http://www.mtconnect.org/schemas/MTConnectError_1.1.xsd">
  <Header creationTime="2010-03-12T12:33:01" sender="localhost"
version="1.1" bufferSize="131072" instanceId="1268463594" />
  <Error errorCode="OUT_OF_RANGE" >Argument was out of range</Error>
</MTConnectError>
```

4.3.1 **`MTConnectError`** Elements

An MTConnect® document **MUST** include the Header for all documents and one `Error` element.

Element	Description	Occurrence
`Header`	A simple header with next sequence and creation time	1
`Errors`	A collection of `Error` elements.	1

For the above elements of the XML Document, please refer to section 4.4 for `Header` and section 5.6 for `Error`.

4.4 Header

Every MTConnect® response **MUST** contain a header as the first element below the root element of any MTConnect® XML Document sent back to an application. The following information **MUST** be provided in the header: `creationTime`, `instanceId`, `sender`, `bufferSize`, and `version`. If the document is an `MTConnectStreams` document it **MUST** also contain the `nextSequence`, `firstSequence`, and `lastSequence` attributes as well.

The `MTConnectDevices` and `MTConnectError` header is as follows:

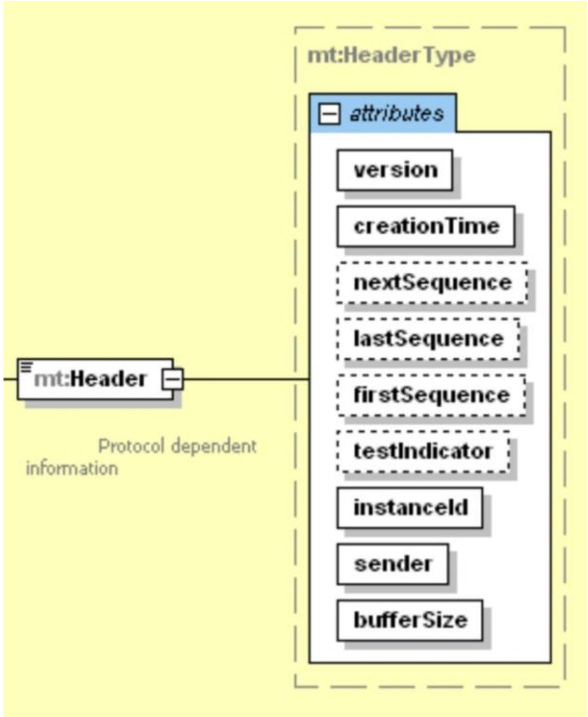


Figure 6: Header Schema Diagram for `MTConnectDevices` and `MTConnectError`

The second header is for `MTConnectStreams` where the protocol sequence information **MUST** be provided:

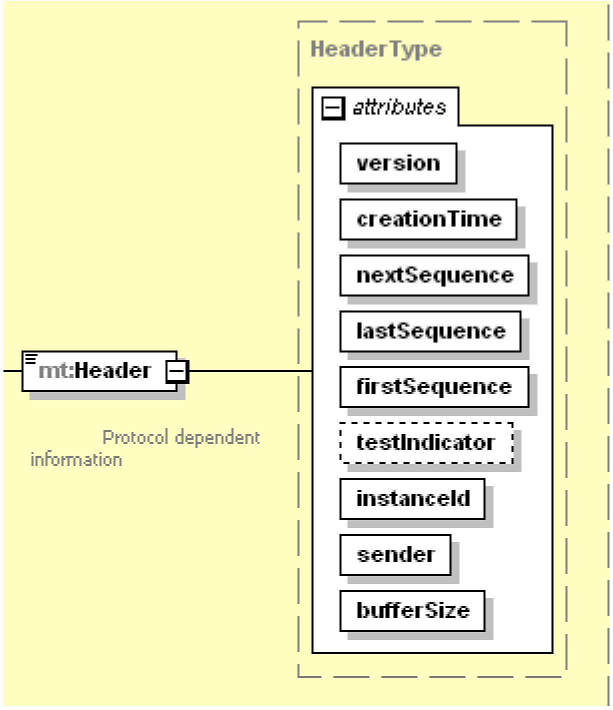


Figure 7: Header Schema Diagram for `MTConnectStreams`

```
<Header creationTime="2010-03-13T07:59:11+00:00" sender="localhost"
        instanceId="1268463594" bufferSize="131072" version="1.1"
        nextSequence="154" firstSequence="1" lastSequence="153" />
```

4.4.1 Header Attributes

Attribute	Description	Occurrence
`creationTime`	The time the response was created.	1
`nextSequence`	The sequence number to use for the next request. Used for sample and current requests. Not used in probe request. This value **MUST** have a maximum value of 2^63-1 and **MUST** be stored in an signed 64 bit integer.	0..1
`instanceId`	A number indicating which invocation of the *Agent*. This is used to differentiate between separate instances of the *Agent*. This value **MUST** have a maximum value of 2^63-1 and **MUST** be stored in an signed 64 bit integer.	1
`testIndicator`	Optional flag that indicates the system is operating in test mode. This data is only for testing and indicates that the data is simulated.	0..1
`sender`	The *Agent* identification information.	1
`bufferSize`	The number of Samples, Events, and Condition that will be retained by the *Agent*. The buffersize **MUST** be a positive integer value with a maximum value of 2^31-1.	1
`firstSequence`	The sequence number of the first sample or event available. This value **MUST** have a maximum value of 2^63-1 and **MUST** be stored in an signed 64 bit integer.	0..1
`lastSequence`	The sequence number of the last sample or event available. This value **MUST** have a maximum value of 2^63-1 and **MUST** be stored in an signed 64 bit integer.	0..1
`version`	The protocol version number. This is the major and minor version number of the MTConnect standard being used. For example if the `version` number is current `10.21.33`, the version will be `10.21`.	1

The `nextSequence`, `firstSequence`, and `lastSequence` number **MUST** be included in sample and current responses. These values **MAY** be used by the client application to determine if the sequence values are within range.The `testIndicator` **MAY** be provided as needed.

Details on the meaning of various fields and how they relate to the protocol are described in detail in the next section on *Protocol (section 5)*. The standard specifies how the protocol **MUST** be implemented to provide consistent MTConnect® *Agent* behavior.

The `instanceId` **MAY** be implemented using any unique information that will be guaranteed to be different each time the sequence number counter is reset. This will usually happen when the MTConnect® *Agent* is restarted. If the *Agent* is implemented with the ability to recover the event stream and the next sequence number when it is restarted, then it **MUST** use the same `instanceId` when it restarts.

The `instanceId` allows the MTConnect® *Agents* to forgo persistence of Events, Condition, and Samples and restart clean each time. Persistence is a decision for each implementation to be determined. This will be discussed further in the section on *Fault Tolerance (in section 5.10).*

The `sender` **MUST** be included in the header to indicate the identity of the *Agent* sending the response. The sender **MUST** be in the following format: `http://<address>[:port]/`. The `port` **MUST** only be specified if it is **NOT** the default HTTP port 80.

The `bufferSize` **MUST** contain the maximum number of results that can be stored in the *Agent* at any one instant. This number can be used by the application to determine how frequently it needs to sample and if it can recover in case of failure. It is the decision of the implementer to determine how large the buffer should be.

As a general rule, the buffer **SHOULD** be sufficiently large to contain at least five minutes' worth of Events, Condition, and Samples. Larger buffers are more desirable since they allow longer application recovery cycles. If the buffer is too small, data can be lost. The *Agent* **SHOULD NOT** be designed so it becomes burdensome to the device and could cause any interruption to normal operation.

5 Protocol

The MTConnect® *Agent* collects and distributes data from the components of a device to other devices and applications. The standard requires that the protocol **MUST** function as described in this section; the tools used to implement the protocol are the decision of the developer.

MTConnect® provides a RESTful interface. The term REST is short for ***RE****presentational* ***S****tate* ***T****ransfer* and provides an architectural framework that defines how state will be managed within the application and *Agent*. REST dictates that the server is unaware of the clients state and it is the responsibility of the client application to maintain the current read position or next operation. This removes the server's burden of keeping track of client sessions. The underlying protocol is HTTP, the same protocol as used in all web browsers.

The MTConnect® *Agent* **MUST** support HTTP version 1.0 or greater. The only requirement for an MTConnect® *Agent* is that it **MUST** support the `HTTP GET` verb. The response to an MTConnect® request **MUST** always be in XML. The `HTTP` request **SHOULD NOT** include a body. If the *Agent* receives a body, the *Agent* **MAY** ignore it. The *Agent* **MAY** ignore any cookies or additional information. The only information the *Agent* **MUST** consider is the URI in the `HTTP GET`.

If the `HTTP GET` verb is not used, the *Agent* must respond with a `HTTP 400` Bad Request indicating that the client issued a bad request. See section 5.6 for further discussion on error handling.

5.1 Standard Request Sequence

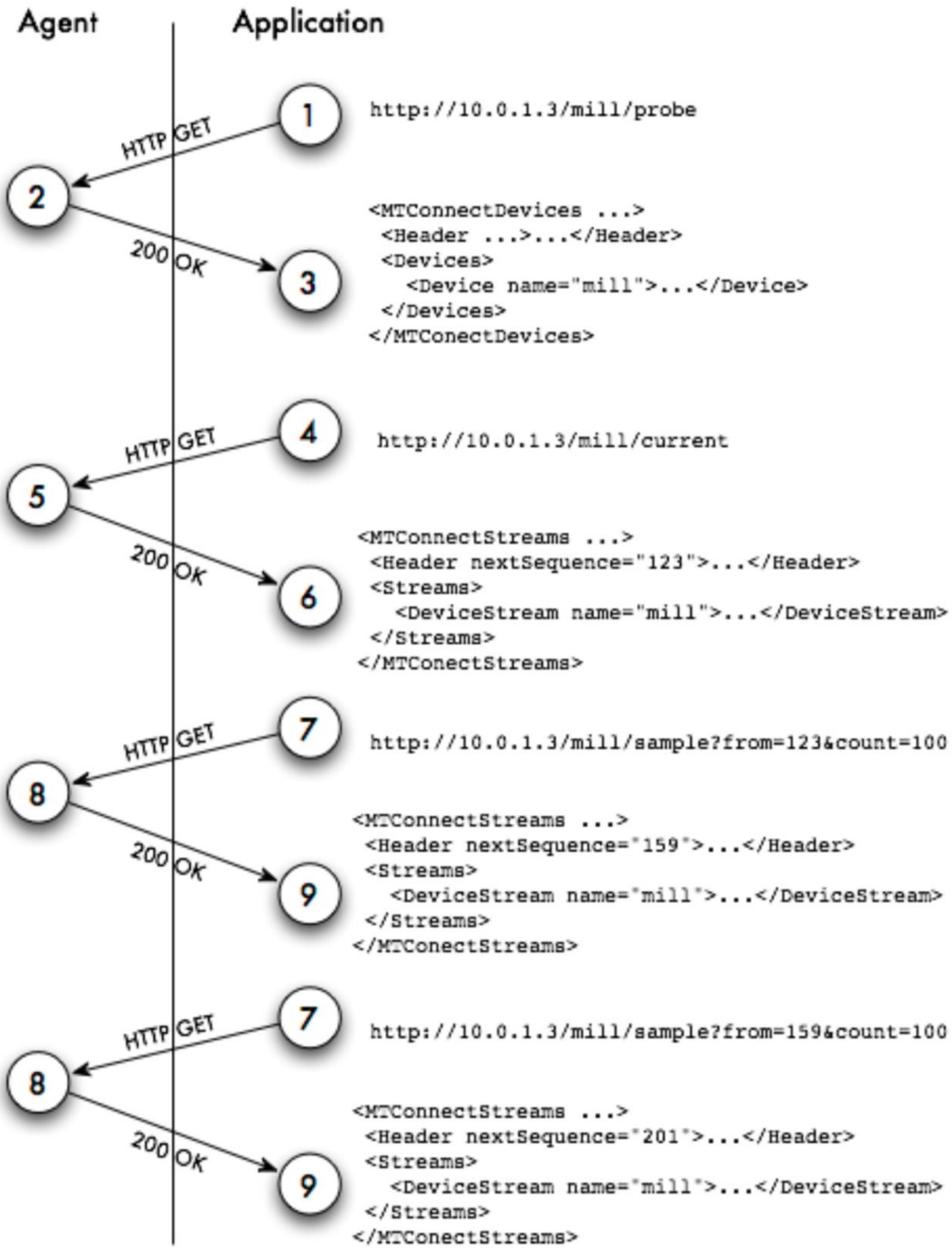
MTConnect® *Agent* **MUST** support three types of requests:

- `probe` – to retrieve the components and the data items for the device. Returns a `MTConnectDevices` XML document.
- `current` – to retrieve a snapshot of the data item's most recent values or the state of the device at a point in time. Returns an `MTConnectStreams` XML document.
- `sample` – to retrieve the Samples, Events, and Condition in time series. Returns an `MTConnectStreams` XML document.

The sequence of requests for a standard MTConnect® conversation will typically begin with the application issuing a `probe` to determine the capabilities of the device. The result of the `probe` will provide the component structure of the device and all the available data items for each component.

Once the application determines the necessary data items are available from the *Agent*, it can issue a `current` request to acquire the latest values of all the data items and the next sequence number for subsequent `sample` requests. The application **SHOULD** also record the `instanceId` to know when to reset the sequence number in the eventuality of *Agent* failure. *(See Fault Tolerance (Section 5.10) for a complete discussion of the use of* `instanceId`*).*

Once the current state has been retrieved, the *Agent* can be sampled at a rate determined by the needs of the application. After each request, the application **SHOULD** save the `nextSequence` number for the next request. This allows the application to receive all results without missing a single sample or event and removes the need for the application to compute the value of the `from` parameter for the next request.



```
http://10.0.1.3/mill/probe
```

```
<MTConnectDevices ...>
 <Header ...>...</Header>
 <Devices>
   <Device name="mill">...</Device>
 </Devices>
</MTConectDevices>
```

```
http://10.0.1.3/mill/current
```

```
<MTConnectStreams ...>
 <Header nextSequence="123">...</Header>
 <Streams>
   <DeviceStream name="mill">...</DeviceStream>
 </Streams>
</MTConectStreams>
```

```
http://10.0.1.3/mill/sample?from=123&count=100
```

```
<MTConnectStreams ...>
 <Header nextSequence="159">...</Header>
 <Streams>
   <DeviceStream name="mill">...</DeviceStream>
 </Streams>
</MTConectStreams>
```

```
http://10.0.1.3/mill/sample?from=159&count=100
```

```
<MTConnectStreams ...>
 <Header nextSequence="201">...</Header>
 <Streams>
   <DeviceStream name="mill">...</DeviceStream>
 </Streams>
</MTConectStreams>
```

Figure 8: Application and Agent Conversation

The above diagram illustrates a standard conversation between an application and an MTConnect® Agent. The sequence is very simple because the entire protocol is an HTTP request/response. The next sequence number handling is shown as a guideline for capturing the stream of Samples, Events, and Condition.

5.2 Probe Requests

The MTConnect® *Agent* **MUST** provide a `probe` response that describes this *Agent*'s devices and all the devices' components and data items being collected. The response to the `probe` **MUST** always provide the most recent information available. A probe request **MUST NOT** supply any parameters. If any are supplied, they **MUST** be ignored. The response from the `probe` will be static as long as the machine physical composition and capabilities do not change, therefore it is acceptable to `probe` very infrequently. In many cases, once a week may be sufficient.

The `probe` request **MUST** support two variations:

- The first provides information on only one device. The device's name **MUST** be specified in the first part of the path. This example will only retrieve components and data items for the mill-1 device.

```
http://10.0.1.23/mill-1/probe
```

- The second does not specify the device and therefore retrieves information for all devices:

```
http://10.0.1.23/probe
```

5.2.1.1 Example

The following is an example `probe` response for 4 Axis Simulator:

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectDevices xmlns:m="urn:mtconnect.org:MTConnectDevices:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectDevices:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectDevices:1.1
http://www.mtconnect.org/schemas/MTConnectDevices_1.1.xsd">
 <Header creationTime="2010-03-13T08:02:38+00:00" sender="localhost"
instanceId="1268463594" bufferSize="131072" version="1.1" />
 <Devices>
  <Device id="dev" name="VMC-4Axis" uuid="XXX111">
   <DataItems>
    <DataItem category="EVENT" id="avail" type="AVAILABILITY" />
   </DataItems>
   <Components>
    <Axes id="axes" name="axes">
     <Components>
      <Linear id="x" name="X">
       <DataItems>
        <DataItem category="SAMPLE" id="Xact" nativeUnits="MILLIMETER"
subType="ACTUAL" type="POSITION" units="MILLIMETER" />
        <DataItem category="SAMPLE" id="Xload" nativeUnits="PERCENT"
type="LOAD" units="PERCENT" />
        <DataItem category="CONDITION" id="Xtravel" type="POSITION" />
        <DataItem category="CONDITION" id="Xovertemp"
type="TEMPERATURE" />
        <DataItem category="CONDITION" id="Xservo" type="ACTUATOR" />
```

```
         </DataItems>
        </Linear>
        <Linear id="y" name="Y">
         <DataItems>
          <DataItem category="SAMPLE" id="Yact" nativeUnits="MILLIMETER" subType="ACTUAL" type="POSITION" units="MILLIMETER" />
          <DataItem category="SAMPLE" id="Yload" nativeUnits="PERCENT" type="LOAD" units="PERCENT" />
          <DataItem category="CONDITION" id="Ytravel" type="POSITION" />
          <DataItem category="CONDITION" id="Yovertemp" type="TEMPERATURE" />
          <DataItem category="CONDITION" id="Yservo" type="ACTUATOR" />
         </DataItems>
        </Linear>
        <Linear id="z" name="Z">
         <DataItems>
          <DataItem category="SAMPLE" id="Zact" nativeUnits="MILLIMETER" subType="ACTUAL" type="POSITION" units="MILLIMETER" />
          <DataItem category="SAMPLE" id="Zload" nativeUnits="PERCENT" type="LOAD" units="PERCENT" />
          <DataItem category="CONDITION" id="Ztravel" type="POSITION" />
          <DataItem category="CONDITION" id="Zovertemp" type="TEMPERATURE" />
          <DataItem category="CONDITION" id="Zservo" type="ACTUATOR" />
         </DataItems>
        </Linear>
        <Rotary id="a" name="A">
         <DataItems>
          <DataItem category="SAMPLE" id="Aact" nativeUnits="DEGREE" subType="ACTUAL" type="ANGLE" units="DEGREE" />
          <DataItem category="SAMPLE" id="Aload" nativeUnits="PERCENT" type="LOAD" units="PERCENT" />
          <DataItem category="CONDITION" id="Atravel" type="POSITION" />
          <DataItem category="CONDITION" id="Aovertemp" type="TEMPERATURE" />
          <DataItem category="CONDITION" id="Aservo" type="ACTUATOR" />
         </DataItems>
        </Rotary>
        <Rotary id="c" name="C" nativeName="S1">
         <DataItems>
          <DataItem category="SAMPLE" id="S1speed" nativeUnits="REVOLUTION/MINUTE" type="SPINDLE_SPEED" units="REVOLUTION/MINUTE" />
          <DataItem category="EVENT" id="S1mode" type="ROTARY_MODE">
           <Constraints>
            <Value>SPINDLE</Value>
           </Constraints>
          </DataItem>
          <DataItem category="SAMPLE" id="S1load" nativeUnits="PERCENT" type="LOAD" units="PERCENT" />
          <DataItem category="CONDITION" id="spindle" type="SYSTEM" />
         </DataItems>
        </Rotary>
       </Components>
      </Axes>
      <Controller id="cont" name="controller">
       <DataItems>
      <DataItem category="CONDITION" id="logic" type="LOGIC_PROGRAM" />
      <DataItem category="EVENT" id="estop" type="EMERGENCY_STOP" />
      <DataItem category="CONDITION" id="servo" type="ACTUATOR" />
      <DataItem category="EVENT" id="message" type="MESSAGE" />
      <DataItem category="CONDITION" id="comms" type="COMMUNICATIONS" />
     </DataItems>
     <Components>
      <Path id="path" name="path">
       <DataItems>
        <DataItem category="SAMPLE" id="SspeedOvr" nativeUnits="PERCENT" subType="OVERRIDE" type="SPINDLE_SPEED" units="PERCENT" />
        <DataItem category="EVENT" id="block" type="BLOCK" />
        <DataItem category="EVENT" id="execution" type="EXECUTION" />
        <DataItem category="EVENT" id="program" type="PROGRAM" />
        <DataItem category="SAMPLE" id="path_feedrate" nativeUnits="MILLIMETER/SECOND" type="PATH_FEEDRATE" units="MILLIMETER/SECOND" />
        <DataItem category="EVENT" id="mode" type="CONTROLLER_MODE" />
        <DataItem category="EVENT" id="line" type="LINE" />
        <DataItem category="SAMPLE" id="path_pos" nativeUnits="MILLIMETER_3D" subType="ACTUAL" type="PATH_POSITION" units="MILLIMETER_3D" />
        <DataItem category="SAMPLE" id="probe" nativeUnits="MILLIMETER_3D" subType="PROBE" type="PATH_POSITION" units="MILLIMETER_3D" />
        <DataItem category="EVENT" id="part" type="PART_ID" />
        <DataItem category="CONDITION" id="motion" type="MOTION_PROGRAM" />
        <DataItem category="CONDITION" id="system" type="SYSTEM" />
       </DataItems>
      </Path>
     </Components>
    </Controller>
   </Components>
  </Device>
 </Devices>
</MTConnectDevices>
```

5.3 Sample Request

The `sample` request retrieves the values for the component’s data items. The reponse to a `sample` request **MUST** be a valid `MTConnectStreams` XML Document.

The diagram below is an example of all the components and data items in relation to one another. The device has one Controller with a single Path, three linear and one rotary axis. The Controller’s Path is capable of providing the execution status and the current block of code. The device has a data item, Availability, that indicates the device is available to communicate.

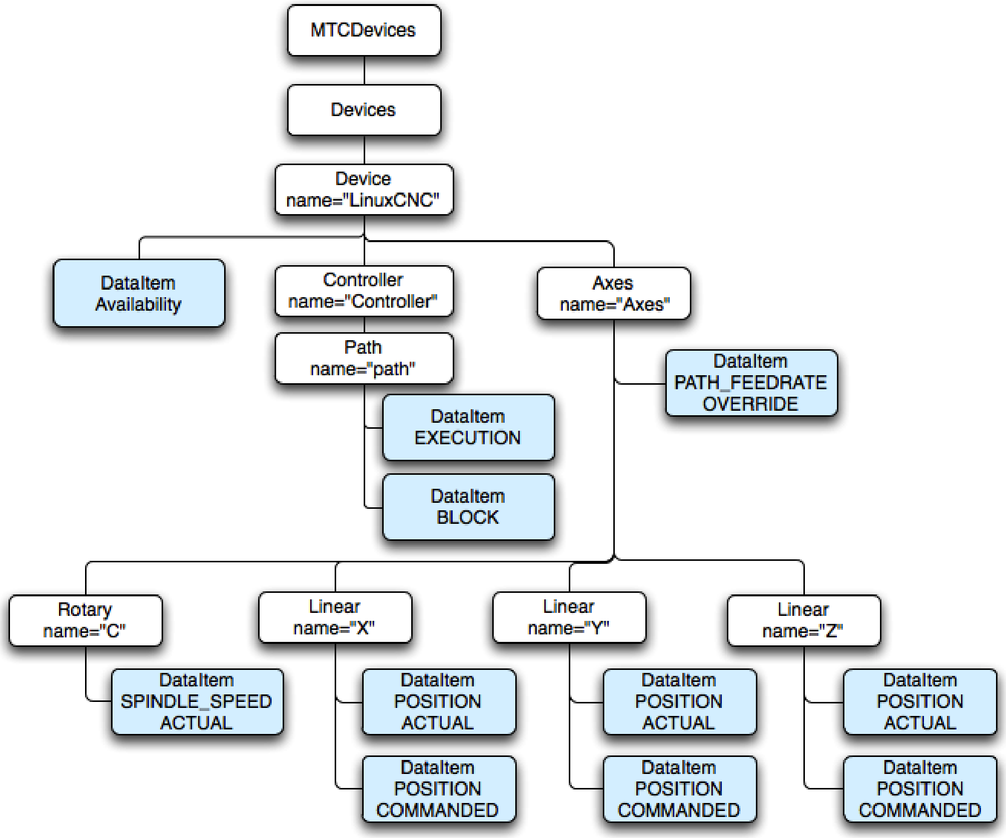


Figure 9: Sample Device Organization

The following path will request the data items for all components in mill-1 with regards to the example above (note that the path parameter refers to the XML Document structure from the probe request, not the XML Document structure of the sample):

```
10. http://10.0.1.23:3000/mill-1/sample
```

This is equivalent to providing a path-based filter for the device named mill-1:

```
11. http://10.0.1.23:3000/sample?path=//Device[@name=”mill-1”]
```

To request all the axes’ data items the following path expression is used:

```
12. http://10.0.1.23:3000/mill-1/sample?path=//Axes
```

To specify only certain data items to be included (e.g. the positions from the axes), use this form:

```
13. http://10.0.1.23:3000/mill-
1/sample?path=//Axes//DataItem[@type=”POSITION”]
```

To retrieve only actual positions instead of both the actual and commanded, the following path syntax can be used:

```
14. http://10.0.1.23:3000/mill-
1/sample?path=//Axes//DataItem[@type=”POSITION” and @subType=”ACTUAL”]
```

or:

```
15. http://10.0.1.23:3000/mill-
1/sample?path=//Axes//DataItem[@type=”POSITION” and
@subType=”ACTUAL”]&from=50&count=100
```

The above example will retrieve all the axes' positions from sample 50 to sample 150. The actual number of items returned will depend on the contents of the data in the *Agent* and the number of results that are actual position samples.

A more complete discussion of the protocol can be found in the section on *Protocol Details – Part 1, Section 5.7.*

5.3.1 Parameters

All parameters **MUST** only be given once and the order of the parameters is not important. The MTConnect® *Agent* **MUST** accept the following parameters for the `sample` request:

`path` - This is an xpath expression specifying the components and/or data items to include in the sample. If the path specifies a component, all data items for that component and any of its sub-components **MUST** be included. For example, if the application specifies the `path=//Axes`, then all the data items for the `Axes` component as well as the `Linear` and `Rotary` sub-components **MUST** be included as well.

`from` - This parameter requests Events, Condition, and Samples starting at this sequence number. The sequence number can be obtained from a prior `current` or `sample` request. The response **MUST** provide the `nextSequence` number. If the value is 0 the first available sample or event **MUST** be used. If the value is less than 0 (< 0) an `INVALID_REQUEST` error **MUST** be returned.

`count` - The maximum number of Events, Condition, and Samples to consider, see detailed explanation below. Events, Condition, and Samples will be considered between `from` and `from + count`, where the latter is the lesser of `from + count` and the last sequence number stored in the agent. The *Agent* **MUST NOT** send back more than this number of Events, Condition, and Samples (in aggregate), but fewer Events, Condition, and Samples **MAY** be returned. If the value is less than 1 (< 1) an `INVALID_REQUEST` error **MUST** be returned**.**

`frequency` – The *Agent* **MUST** stream Samples, Events, and Condition to the client application pausing for `frequency` milliseconds between each part. Each part will contain a maximum of `count` Events, Samples, and Condition and `from` will be used to indicate the beginning of the stream.

The `nextSequence` number in the header **MUST** be set to the sequence number following the largest sequence number (highest sequence number + 1) of all the Events, Condition, and Samples considered when collecting the results.

If no parameters are given, the following defaults **MUST** be used:

The `path` **MUST** default to all components in the device or devices if no device is specified.

The `count` **MUST** default to 100 if it is not specified.

The `from` **MUST** default to 0 and return the first available event or sample. If the latest state is desired, see `current`.

5.4 Current Request

The `current` request retrieves the values for the components' data items at the point the request is received. The response to the request **MUST** contain the most current values for all data items specified in the request path. If the path is not given, it **MUST** respond with all data items for the device(s), in the same way as the `sample` request.

`current` **MUST** return the `nextSequence` number for the event or sample directly following the point at which the snapshot was taken. This **MUST** be determined by finding the sequence number of the last event or sample in the *Agent* and adding one (+1) to that value. The `nextSequence` number **MAY** be used for subsequent `sample`s.

The Samples, Events, and Condition returned from the `current` request **MUST** have the time-stamp and the sequence number that was assigned at the time the data was collected. The *Agent* **MUST NOT** alter the original time, sequence, or values that were assigned when the data was collected.

```
http://10.0.1.23:3000/mill-1/current?path=//Axes//DataItem[@type=”POSITION”
and @subType=”ACTUAL”]
```

This example will retrieve the current actual positions for all the axes, as with a `sample`, except with `current`, there will always be a sample or event for each data item if at least one piece of data was retrieved from the device.

```
http://10.0.1.23:3000/mill-1/current?path=//Axes//DataItem[@type=”POSITION”
and @subType=”ACTUAL”]&at=1232
```

The above example retrieves the axis actual position at a specific earlier point in time - in this case, `at` Sequence Number 1232.

5.4.1 Parameters

The MTConnect® *Agent* **MUST** accept the following parameter for the `current` request:

`path` - same requirements as `sample`.

`freqency` - same requirements as `sample`. **MUST NOT** be used with `at`.

`at` - an optional argument specifying the MTConnect protocol sequence number. If supplied, the most current values on or before the sequence number **MUST** be provided. If `at` is not provided, the latest values **MUST** be provided. `at` **MUST NOT** be used with the `frequency` as this will just return the same data set repeatedly.

If no parameters are provided for the current request, all data items **MUST** be retrieved with their latest values.

5.4.2 Getting the State `at` a Sequence Number

The current `at` allows an application to monitor real-time conditions and then perform causal analysis by requesting the current values for all the data items at the sequence number of interest. This removes the requirement that the application continually poll for all states and burden the server and the network with unneeded information associated with faults or other abnormal conditions.

An example of the current request using the `at` parameter with a very simple machine configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectDevices xmlns="urn:mtconnect.org:MTConnectDevices:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mtconnect.org:MTConnectDevices:1.1
http://www.mtconnect.org/schemas/MTConnectDevices_1.1.xsd">
  <Header creationTime="2010-04-01T21:22:43" sender="host" version="1.1" buf-
ferSize="1" instanceId="1"/>
  <Devices>
    <Device name="minimal" uuid="1" id="d">
      <DataItems>
        <DataItem type="AVAILABILITY" category="EVENT" id="avail" />
      </DataItems>
      <Components>
        <Controller name="controller" id="c1">
          <DataItems>
            <DataItem id="estop" type="EMERGENCY_STOP" category="EVENT"/>
            <DataItem id="system" type="SYSTEM" category="CONDITION" />
          </DataItems>
          <Components>
            <Path id="p1" name="path" >
              <DataItems>
                <DataItem id="execution" type="EXECUTION" category="EVENT"/>
              </DataItems>
            </Path>
          </Components>
         </Controller>
       </Components>
     </Device>
  </Devices>
</MTConnectDevices>
```

Here is a series of events and condition:

Time Offset	Sequence	Name	Value
06:19:25.089023	1	estop	UNAVAILABLE
06:19:25.089023	2	execution	UNAVAILABLE
06:19:25.089023	3	avail	UNAVAILABLE
06:19:25.089023	4	system	Unavailable
06:19:35.153141	5	avail	AVAILABLE
06:19:35.153141	6	execution	STOPPED
06:19:35.153141	7	estop	ACTIVE
06:19:35.153370	8	system	Normal
06:20:05.153230	9	estop	RESET
06:20:05.153230	10	execution	ACTIVE

Time Offset	Sequence	Name	Value
06:20:35.153716	11	system	Fault
06:21:05.153587	12	execution	STOPPED
06:21:35.153784	13	system	Normal
06:22:05.153741	14	execution	ACTIVE

If a `current` request is made after this sequence of events, the result will be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
  <Header creationTime="2010-04-06T06:53:34+00:00" sender="localhost" instan-
ceId="1270534765" bufferSize="16" version="1.1" nextSequence="19" firstSe-
quence="3" lastSequence="18" />
  <Streams>
    <DeviceStream name="minimal" uuid="1">
      <ComponentStream component="Device" name="minimal" componentId="d">
        <Events>
          <Availability dataItemId="avail" sequence="5" timestamp="2010-04-
06T06:19:35.153141">AVAILABLE</Availability>
        </Events>
      </ComponentStream>
      <ComponentStream component="Controller" name="controller" componen-
tId="c1">
        <Events>
          <EmergencyStop dataItemId="estop" sequence="9" timestamp="2010-04-
06T06:20:05.153230">RESET</EmergencyStop>
        </Events>
        <Condition>
          <Normal dataItemId="system" sequence="13" timestamp="2010-04-
06T06:21:35.153784" type="SYSTEM" />
        </Condition>
      </ComponentStream>
      <ComponentStream component="Path" name="path" componentId="p1">
        <Events>
          <Execution dataItemId="execution" sequence="14" timestamp="2010-04-
06T06:22:05.153741">ACTIVE</Execution>
        </Events>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```

If we want to inspect the state of the machine at the point the fault occurred, sequence number 11, we can issue a request: `http://localhost:5000/current?at=11`. This will return the following response:

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

```
<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
  <Header creationTime="2010-04-06T07:05:49+00:00" sender="localhost" instan-
ceId="1270534765" bufferSize="16" version="1.1" nextSequence="19" firstSe-
quence="3" lastSequence="18" />
  <Streams>
    <DeviceStream name="minimal" uuid="1">
      <ComponentStream component="Device" name="minimal" componentId="d">
        <Events>
          <Availability dataItemId="avail" sequence="5" timestamp="2010-04-
06T06:19:35.153141">AVAILABLE</Availability>
        </Events>
      </ComponentStream>
      <ComponentStream component="Controller" name="controller" componen-
tId="c1">
        <Events>
          <EmergencyStop dataItemId="estop" sequence="9" timestamp="2010-04-
06T06:20:05.153230">RESET</EmergencyStop>
        </Events>
        <Condition>
          <Fault dataItemId="system" sequence="11" timestamp="2010-04-
06T06:20:35.153716" type="SYSTEM" />
        </Condition>
      </ComponentStream>
      <ComponentStream component="Path" name="path" componentId="p1">
        <Events>
          <Execution dataItemId="execution" sequence="10" timestamp="2010-04-
06T06:20:05.153230">ACTIVE</Execution>
        </Events>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```

With MTConnect you can replay the history and move forward a single sequence to see what happened immediately after the fault occurred: http://localhost:5000/current?at=12.

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
  <Header creationTime="2010-04-06T07:05:55+00:00" sender="localhost" instan-
ceId="1270534765" bufferSize="16" version="1.1" nextSequence="19" firstSe-
quence="3" lastSequence="18" />
  <Streams>
    <DeviceStream name="minimal" uuid="1">
      <ComponentStream component="Device" name="minimal" componentId="d">
        <Events>
```

```
          <Availability dataItemId="avail" sequence="5" timestamp="2010-04-
06T06:19:35.153141">AVAILABLE</Availability>
        </Events>
      </ComponentStream>
      <ComponentStream component="Controller" name="controller" componen-
tId="c1">
        <Events>
          <EmergencyStop dataItemId="estop" sequence="9" timestamp="2010-04-
06T06:20:05.153230">RESET</EmergencyStop>
        </Events>
        <Condition>
          <Fault dataItemId="system" sequence="11" timestamp="2010-04-
06T06:20:35.153716" type="SYSTEM" />
        </Condition>
      </ComponentStream>
      <ComponentStream component="Path" name="path" componentId="p1">
        <Events>
          <Execution dataItemId="execution" sequence="12" timestamp="2010-04-
06T06:21:05.153587">STOPPED</Execution>
        </Events>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```

Here one can see that execution state has now transitioned to STOPPED and the Fault is still active. The application is free to scroll through the buffer from the first sequence number to the last sequence number.

It should also be noted that the first sequence number is 3 and a request before this first sequence number is not allowed. If, for example, a request is made at sequence 2: `http://localhost:5000/current?at=2`, an error will be returned:

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectError xmlns:m="urn:mtconnect.org:MTConnectError:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectError:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectError:1.1
http://www.mtconnect.org/schemas/MTConnectError_1.1.xsd">
  <Header creationTime="2010-04-06T22:01:17+00:00" sender="localhost" instan-
ceId="1270534765" bufferSize="16" version="1.1" />
  <Errors>
    <Error errorCode="QUERY_ERROR">'at' must be greater than or equal to
3.</Error>
  </Errors>
</MTConnectError>
```

5.4.3 Determining Event Duration

A common requirement is to determine the duration of an event, such as how long the machine has been actively executing a program. The addition of `current` with the `at` parameter facilitates this operation. The following is an example based on the value of the `Execution` tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
  <Header creationTime="2010-04-17T08:05:10+00:00" sender="localhost" instanceId="1267747762"
bufferSize="131072" version="1.1" nextSequence="746859061" firstSequence="746727989" lastSe-
quence="746859060" />
  <Streams>
    <DeviceStream name="VMC-3Axis" uuid="000">
      <ComponentStream component="Path" name="path" componentId="pth">
        <Samples>
          <PathFeedrate dataItemId="Fovr" sequence="746803687" timestamp="2010-04-
17T08:01:45.149887">100.0000000000</PathFeedrate>
          <PathFeedrate dataItemId="Frt" sequence="746859054" timestamp="2010-04-
17T08:05:09.829551">0</PathFeedrate>
        </Samples>
        <Events>
          <Block dataItemId="cn2" name="block" sequence="746858893" timestamp="2010-04-
17T08:05:08.597481">G0Z1</Block>
          <ControllerMode dataItemId="cn3" name="mode" sequence="746803685" timestamp="2010-04-
17T08:01:45.149887">AUTOMATIC</ControllerMode>
          <Line dataItemId="cn4" name="line" sequence="746859056" timestamp="2010-04-
17T08:05:09.861553">0</Line>
          <Program dataItemId="cn5" name="program" sequence="746803684" timestamp="2010-04-
17T08:01:45.149887">FLANGE_CAM.NGC</Program>
          <Execution dataItemId="cn6" name="execution" sequence="746859059" timestamp="2010-
04-17T08:05:09.905555">READY</Execution>
        </Events>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```

When the execution value changes to `READY` after it was in the `ACTIVE` state, we can determine the duration by performing a `current` with `at` set to one minus the sequence number of the event in question. In this case `Execution` has a sequence number `746859059`, so one would perform a request as follows:

`http://agent.mtconnect.org:5000/current?path=//Path&at=746859058`

This will result in the following response:

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
  <Header creationTime="2010-04-17T08:05:33+00:00" sender="localhost" instanceId="1267747762"
bufferSize="131072" version="1.1" nextSequence="746859061" firstSequence="746727989" lastSe-
quence="746859060" />
  <Streams>
    <DeviceStream name="VMC-3Axis" uuid="000">
```

```
   <ComponentStream component="Path" name="path" componentId="pth">
    <Samples>
     <PathFeedrate dataItemId="Fovr" sequence="746803687" timestamp="2010-04-
17T08:01:45.149887">100.0000000000</PathFeedrate>
     <PathFeedrate dataItemId="Frt" sequence="746859054" timestamp="2010-04-
17T08:05:09.829551">0</PathFeedrate>
    </Samples>
    <Events>
     <Block dataItemId="cn2" name="block" sequence="746858893" timestamp="2010-04-
17T08:05:08.597481">G0Z1</Block>
     <ControllerMode dataItemId="cn3" name="mode" sequence="746803685" timestamp="2010-04-
17T08:01:45.149887">AUTOMATIC</ControllerMode>
     <Line dataItemId="cn4" name="line" sequence="746859056" timestamp="2010-04-
17T08:05:09.861553">0</Line>
     <Program dataItemId="cn5" name="program" sequence="746803684" timestamp="2010-04-
17T08:01:45.149887">FLANGE_CAM.NGC</Program>
     <Execution dataItemId="cn6" name="execution" sequence="746803674" timestamp="2010-
04-17T08:01:45.149887">ACTIVE</Execution>
    </Events>
   </ComponentStream>
  </DeviceStream>
 </Streams>
</MTConnectStreams>
```

The previous event shows the `Execution` in the `ACTIVE` state. The next step is to take the difference between the two time-stamps:

2010-04-17T08:05:09.905555 - 2010-04-17T08:01:45.149887 =
204.755668 Seconds or 00:03:24.755668

The technique can be used for any observed values in MTConnect since only the changes are recorded, the previous state will always be available using the current at the previous sequence number, even if the previous event is no longer in the buffer, but the previous sequence number is greater than the `firstSequence` number.

5.5 Streaming

When the `frequency` parameter is provided, the MTConnect® *Agent* **MUST** find all available events, samples, and condition that match the current filter criteria specified by the path at the frequency given or at its maximum possible scan rate. The frequency indicates the delay between the end of one data transmission and the beginning of the next data transmission. A frequency of zero indicates the *Agent* deliver data at its highest possible frequency.

The frequency **MUST** be given in milliseconds. If there are no available events or samples, the *Agent* **MAY** delay sending an update for **AT MOST** ten (10) seconds. The *Agent* **MUST** send updates at least once every ten (10) seconds to ensure the receiver that the *Agent* is functioning correctly. The content of the streams **MUST** be empty if no data is available for a given interval.

The format of the response **MUST** use a MIME encoded message with each section separated by a MIME boundary. Each section of the response **MUST** contain an entire `MTConnectStreams` document.

For more information on MIME see rfc1521 and rfc822. This format is in use with most streaming web media protocols.

Request:

```
http://localhost/sample?frequency=1000&path=//DataItem[@type=”AVAILABILITY”]
```

Sample response:

```
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=a8e12eced4fb871ac096a99bf9728425
8.

```

Lines 1-8 are a standard header for a MIME multipart message. The boundary is a separator for each section of the stream. The content length is set to some arbitrarily large number or omitted. Line 10 indicates this is a multipart MIME message and the boundary between sections.

```
9.  --a8e12eced4fb871ac096a99bf9728425
10. Content-type: text/xml
11. Content-length: 887
12.
13. <?xml version="1.0" encoding="UTF-8"?>
14. <MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
15.   <Header creationTime="2010-03-13T08:33:37+00:00" sender="localhost"
instanceId="1268469210" bufferSize="131072" version="1.1" nextSequence="43"
firstSequence="1" lastSequence="42" />
16.   <Streams/>
17. </MTConnectStreams>
```

Lines 9-17 are the first section of the stream. Since there was no activity in this time period there are no component streams included. Each section presents the content type and the length of the section. The boundary is chosen to be a string of characters that will not appear in the message.

```
18. --a8e12eced4fb871ac096a99bf9728425
19. Content-type: text/xml
20. Content-length: 545
```

```
21.
22. <?xml version="1.0" encoding="UTF-8"?>
23. <MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
24.   <Header creationTime="2010-03-13T08:33:38+00:00" sender="localhost"
instanceId="1268469210" bufferSize="131072" version="1.1" nextSequence="43"
firstSequence="1" lastSequence="42" />
25.   <Streams>
26.     <DeviceStream name="VMC-4Axis" uuid="XXX111">
27.       <ComponentStream component="Device" name="VMC-4Axis"
componentId="dev">
28.         <Events>
29.           <Availability dataItemId="avail" sequence="25"
timestamp="2010-03-13T08:33:30.555235">UNAVAILABLE</Availability>
30.         </Events>
31.       </ComponentStream>
32.     </DeviceStream>
33.   </Streams>
34. </MTConnectStreams>
```

Lines 18-34: After a period of time, the power gets turned off and a new mime part is sent with the new status.

```
35. --a8e12eced4fb871ac096a99bf9728425
36. Content-type: text/xml
37. Content-length: 883
38.
39. <?xml version="1.0" encoding="UTF-8"?>
40. <MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
41.   <Header creationTime="2010-03-13T08:34:18+00:00" sender="localhost"
instanceId="1268469210" bufferSize="131072" version="1.1" nextSequence="98"
firstSequence="1" lastSequence="97" />
42.   <Streams>
43.     <DeviceStream name="VMC-4Axis" uuid="XXX111">
44.       <ComponentStream component="Device" name="VMC-4Axis"
componentId="dev">
45.         <Events>
```

```
46.           <Availability dataItemId="avail" sequence="65"
timestamp="2010-03-13T08:34:16.0312">AVAILABLE</Availability>
47.         </Events>
48.       </ComponentStream>
49.     </DeviceStream>
50.   </Streams>
51. </MTConnectStreams>
```

Lines 34-51: Approximately six seconds later the machine is turned back on and a new message is generated. Even though we have a scan frequency of one second, the *Agent* waited for ten seconds to send a new message.

```
52. --a8e12eced4fb871ac096a99bf9728425
53. Content-type: text/xml
54. Content-length: 545
55.
56. <?xml version="1.0" encoding="UTF-8"?>
57. <MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
58.   <Header creationTime="2010-03-13T08:34:27+00:00" sender="localhost"
instanceId="1268469210" bufferSize="131072" version="1.1" nextSequence="98"
firstSequence="1" lastSequence="97" />
59.   <Streams />
60. </MTConnectStreams>
```

Lines 52-60 demonstrate a heartbeat sent out 10 seconds after the previous message. Since there is no activity there is no content in the device streams element.

The *Agent* **MUST** continue to stream results until the client closes the connection. The *Agent* **MUST NOT** stop the streaming for any other reason other than the *Agent* process shutting down.

5.6 HTTP Response Codes and Error

MTConnect® uses the HTTP response codes to indicate errors where no XML document is returned because the request was malformed and could not be handled by the *Agent*. These errors are serious and indicate the client application is sending malformed requests or the *Agent* has an unrecoverable error. The error code **MAY** also be used for HTTP authentication with the 401 request for authorization. The HTTP protocol has a large number of codes defined[1]; only the following mapping **MUST** be supported by the MTConnect® *Agent*:

HTTP Status	Name	Description

[1] For a full list of HTTP response codes see the following document: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP Status	Name	Description
200	OK	The request was handled successfully.
400	Bad Request	The request could not be interpreted.
500	Internal Error	There was an internal error in processing the request. This will require technical support to resolve.
501	Not Implemented	The request cannot be handled on the server because the specified functionality is not implemented.

5.6.1 MTConnectError

The MTConnectError document **MUST** be returned if the *Agent* cannot handle the request. The Error contains an errorCode and the CDATA of the element is the complete error text. The classification for errors is expected to expand as the standard matures.

For backward compatibility, MTConnectError can contain a single Error element. If there are more than one error to report, it is up to the implementation of the *Agent* to determine the most important error to include.

5.6.2 Errors

The MTConnectError element **MUST** contain all relevant errors for the given request. The Errors element **MUST** contain at least one Error element. There are no attributes for this element.

5.6.3 Error

The Error contains an errorCode and the CDATA of the element is the complete error text. The classification for errors is expected to expand as the standard matures.

Attributes	Description	Occurrence
errorCode	An error code	1

The CDATA of the Error element is the textual description of the error and any additional information the *Agent* wants to send. The Error element **MUST** contain one of the following error codes:

Error Code	Description
UNAUTHORIZED	The request did not have sufficient permissions to perform the request.
NO_DEVICE	The device specified in the URI could not be found.
OUT_OF_RANGE	The sequence number was beyond the end of the buffer.

Error Code	Description
TOO_MANY	The count given is too large.
INVALID_URI	The URI provided was incorrect.
INVALID_REQUEST	The request was not one of the three specified requests.
INTERNAL_ERROR	Contact the software provider, the *Agent* did not behave correctly.
INVALID_PATH	The xpath could not be parsed. Invalid syntax.
UNSUPPORTED	A valid request was provided, but the *Agent* does not support the feature or request type.

Here is an example of an HTTP error:

```
HTTP/1.1 200 Success
Content-Type: text/xml; charset=UTF-8
Server: Agent
Date: Sun, 23 Dec 2007 21:10:19 GMT

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

5.7 Protocol Details

When an MTConnect® *Agent* collects information from the device, it assigns each piece of information a unique sequence number. The sequence number **MUST** be assigned in monotonically increasing numbers in the order they arrive at the *Agent*. Each source **SHOULD** provide a time-stamp indicating when the information was collected from the component. If no time-stamp is provided, the Agent **MUST** provide a time-stamp of its own. The time-stamps reported by the Agent **MUST** be used as the means for the ordering of the messages as opposed to using the sequence number for this purpose.

Note: It is assumed the time-stamp is the best available estimate of when the data was recorded.

If two data items are sampled at the same exact time, they **MUST** be given the same time stamp. It is assumed that all events or samples with the same timestamp occurred at the same moment. A sample is considered to be valid until the time of the next sample for the same data item. If no new samples are present for a data item, the last value is maintained for the entire period between the samples. **Important:** MTConnect® only records data when it changes. If the value remains the same, MTConnect **MUST NOT** record a duplicate value with a new sequence number and time stamp. There **MUST NEVER** be two identical adjacent values for the same data item in the same component.

For example, if the Xact is 0 at 12:00.0000 and Yact is 1 at 12:00.0000, these two samples were collected at the same moment. If Yact is 2 at 12:01.0000 and there is no value at this point for Xact, it is assumed that Xact is still 0 and has not moved.

The sequence number **MUST** be unique for this instance of the MTConnect® *Agent*, regardless of the device or component the data came from. The MTConnect® *Agent* provides the sequence numbers in series for all devices using the same counter. This allows for multi-device responses without sequence number collisions and unnecessary protocol complexity.

As an implementation warning, it is the applications responsibility to make sure it does not miss information from the *Agent*. The *Agent* has no awareness of the application or the application's requirements for data, and it therefore does not guarantee the application receive all pieces of data. The *Agent* protocol makes it easy for the application developers to determine if they have received all pieces of data by scrolling through the buffer. If they ever receive an `OUT_OF_RANGE` error due to providing a `from` argument that references a sequence number prior to the beginning of the retained data, they know they have missed some information.

If the application only uses `current` requests, it may miss information since it will only be receiving a snapshot at various points in time. For some display application that do not need to store or reason on the data, this may be adequate, but if more in-depth analysis is to be performed, it is advised that the application make requests based on their data requirements using filtering and streams to get all vital information. For example, the application can request all condition types and controller events, and then sample other pieces of data for which they have less strict requirements. Breaking things out like this will allow for continuous data flow and minimal bandwidth utilization.

The application may request any sequence of data within the buffer at any time using either the `sample from` or the `current at` semantics. With these two calls it is easy for the application to go back in time and find data prior to an occurrence. It is of course limited to the size of the buffer and rate of incoming data.

5.7.1 Buffer Semantics

The MTConnect buffer can be thought of as a tube that can hold a finite set of balls. As balls are inserted in one end they fill the tube until there is no more room for additional balls at which point any new balls inserted will push the oldest ball out the back of the tube. The tube will continue to shift in this manor with monotonically increasing sequence numbers being assigned as each ball gets inserted. The sequence numbers will never be reused for one instance of the *Agent* process. Since the sequence number is a 64 bit integer, the numbers will never (at least within the next 100,000 years) wrap around or be exhausted.

The follow example is a contrived agent with only 8 slots and two data item types, a Line (**Line)** event and a Position (**Pos**) sample. The Position sample at sequence number 19 was just inserted and the event at sequence number 11 was just removed.

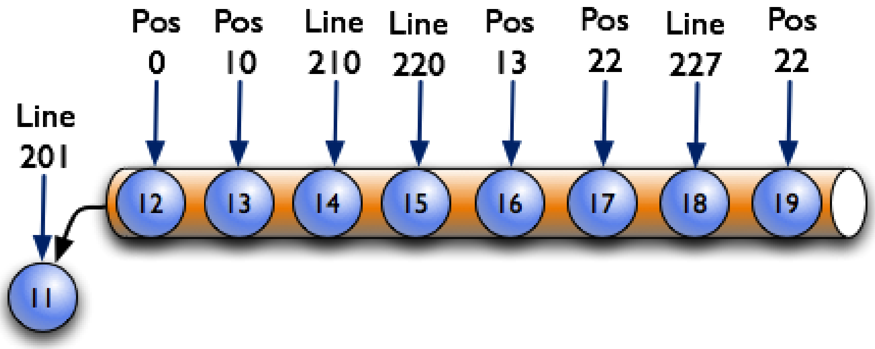


Figure 10: Example Buffer 1

If we perform a `current` request, we will receive Line 227 and Pos 22. If the `at` parameter is given to the `current` request and is set to 12, we will receive Line 201 and Position 0, and as follows at 13 will retrieve Line 201 and Position 10. Note: The last value for all Events, Samples, and each Condition will be preserved until they are replaced. Therefore, Line 201 is returned since it has not been replaced until sequence number 14 where Line is 210.

If a `current` request is made for a sequence number prior to 12, the agent **MUST** return a `OUT_OF_RANGE` error. For example, a request for `current` at 11 will result in `OUT_OF_RANGE` error. The same error **MUST** be given if a sequence number is requested that is greater than the end of the buffer. For example, a request for current at 20 will result in an `OUT_OF_RANGE` error.

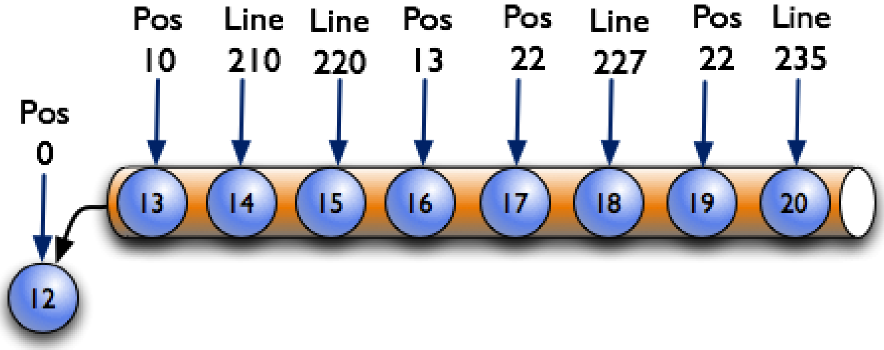


Figure 11: Buffer Semantics 2

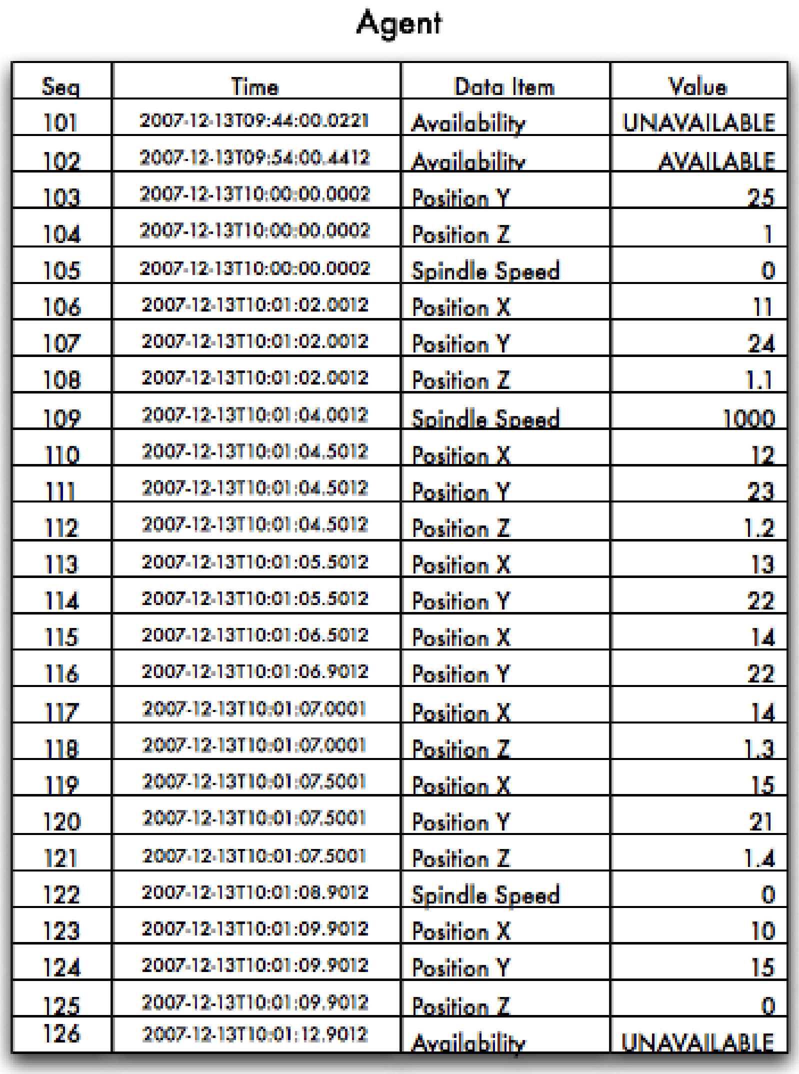
The above illustration show what happens when another Line event is added at sequence number 20. The Pos 0 is sample is pushed out the back of the pipe and the first available sequence number is now 13. A request for the `current at` 13 will still retrieve a Line of 201, since the first value for line has not been replaced.

5.7.2 Buffer Windows

The information in MTConnect® can be thought of as a four column table of data where the first column is a sequence number increasing by increments of one, the second column is the time, the third column is the data item it is associated with, and the fourth column is the value. The storage, internal representation, and implementation is not part of this standard. The implementer can choose to store as much or as little information as they want, as long as they can support the requirements of the standard. They can also decide if it is necessary to locally store the data.

The following examples will use only a single device. Multiple devices are treated the same as single devices. We will document the multiple device scenarios in more depth in future versions of this standard.

The following table is an example of a small window of data collected from a device:



Agent

Seq	Time	Data Item	Value
101	2007-12-13T09:44:00.0221	Availability	UNAVAILABLE
102	2007-12-13T09:54:00.4412	Availability	AVAILABLE
103	2007-12-13T10:00:00.0002	Position Y	25
104	2007-12-13T10:00:00.0002	Position Z	1
105	2007-12-13T10:00:00.0002	Spindle Speed	0
106	2007-12-13T10:01:02.0012	Position X	11
107	2007-12-13T10:01:02.0012	Position Y	24
108	2007-12-13T10:01:02.0012	Position Z	1.1
109	2007-12-13T10:01:04.0012	Spindle Speed	1000
110	2007-12-13T10:01:04.5012	Position X	12
111	2007-12-13T10:01:04.5012	Position Y	23
112	2007-12-13T10:01:04.5012	Position Z	1.2
113	2007-12-13T10:01:05.5012	Position X	13
114	2007-12-13T10:01:05.5012	Position Y	22
115	2007-12-13T10:01:06.5012	Position X	14
116	2007-12-13T10:01:06.9012	Position Y	22
117	2007-12-13T10:01:07.0001	Position X	14
118	2007-12-13T10:01:07.0001	Position Z	1.3
119	2007-12-13T10:01:07.5001	Position X	15
120	2007-12-13T10:01:07.5001	Position Y	21
121	2007-12-13T10:01:07.5001	Position Z	1.4
122	2007-12-13T10:01:08.9012	Spindle Speed	0
123	2007-12-13T10:01:09.9012	Position X	10
124	2007-12-13T10:01:09.9012	Position Y	15
125	2007-12-13T10:01:09.9012	Position Z	0
126	2007-12-13T10:01:12.9012	Availability	UNAVAILABLE

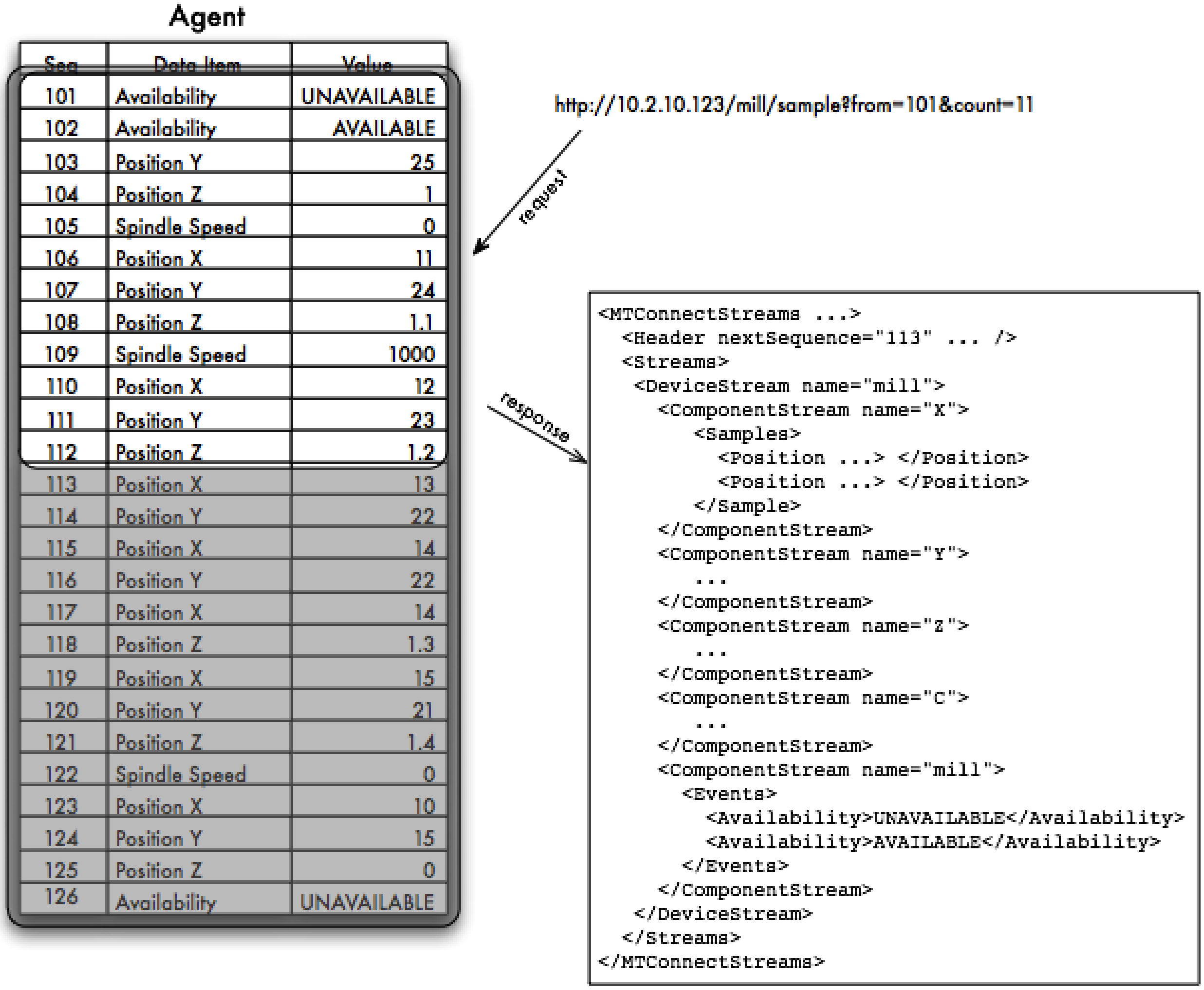
Figure 12: Sample Data in an Agent

This is a table of 25 data values and a duration of around 12 seconds. The data captures the availability of the device and the position of its axes: the linear axes X, Y, and Z, and the rotary axis C. The only data items collected in this example are the Position (for the sake of this data, we have the actual position) and the rotary axis C Spindle Speed. We are also collecting the device's availability state that can be either `AVAILABLE` or `UNAVAILABLE`. The device is `UNAVAILABLE` when the sample starts.

For the remainder of the examples we will be excluding the time column to save space.

5.8 Request without Filtering

In the example below, the application made a request for a sample starting at sequence #101 and retrieves the next eleven items. The response will include all the Samples, Events, and Condition in the mill device from 101 to 112. The `nextSequence` number in the header will tell the application it should begin the next request at 113. (The response is abbreviated and for illustration purpose only.)

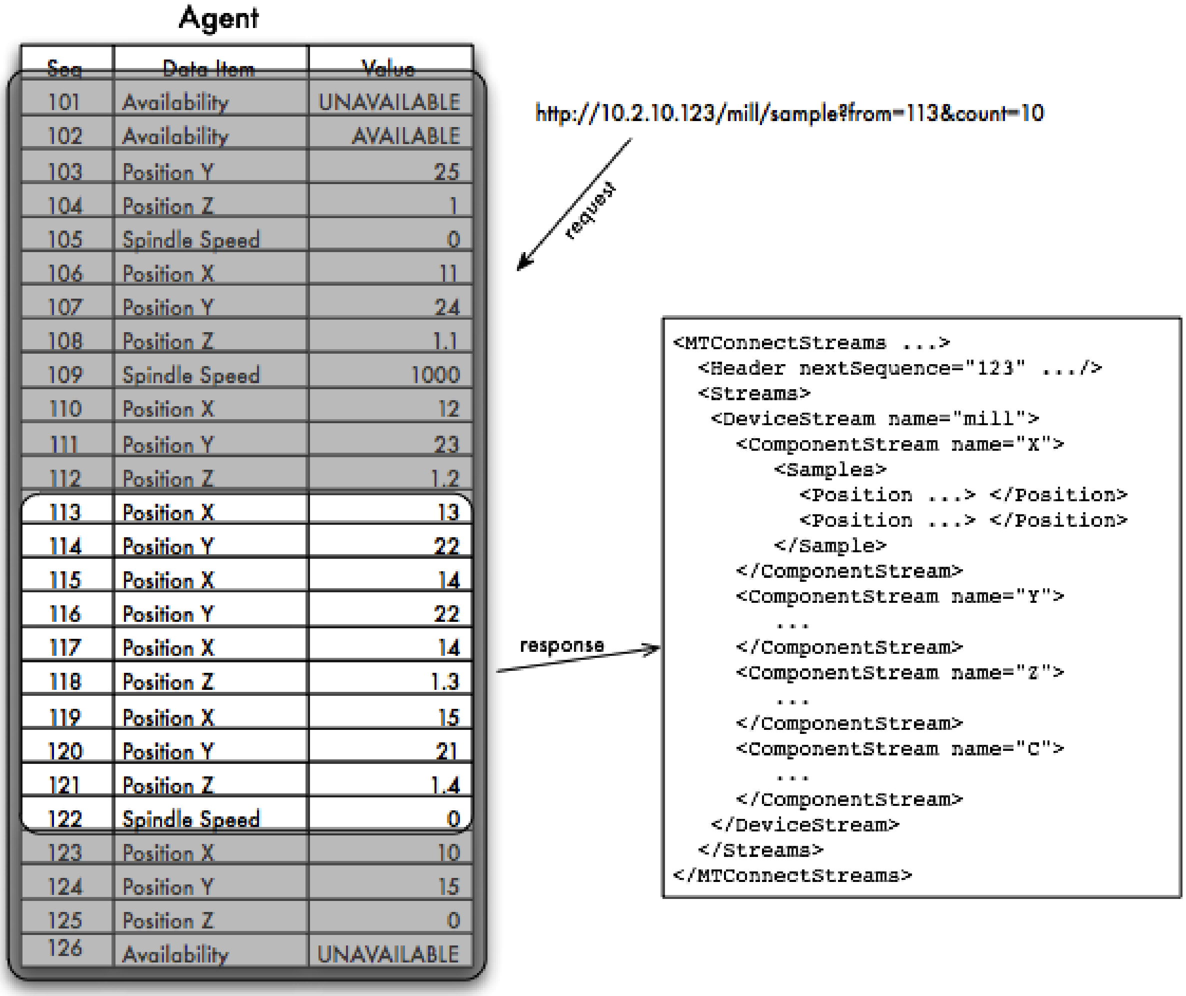


Seq	Data Item	Value
101	Availability	UNAVAILABLE
102	Availability	AVAILABLE
103	Position Y	25
104	Position Z	1
105	Spindle Speed	0
106	Position X	11
107	Position Y	24
108	Position Z	1.1
109	Spindle Speed	1000
110	Position X	12
111	Position Y	23
112	Position Z	1.2
113	Position X	13
114	Position Y	22
115	Position X	14
116	Position Y	22
117	Position X	14
118	Position Z	1.3
119	Position X	15
120	Position Y	21
121	Position Z	1.4
122	Spindle Speed	0
123	Position X	10
124	Position Y	15
125	Position Z	0
126	Availability	UNAVAILABLE

```
<MTConnectStreams ...>
  <Header nextSequence="113" ... />
  <Streams>
   <DeviceStream name="mill">
     <ComponentStream name="X">
        <Samples>
          <Position ...> </Position>
          <Position ...> </Position>
        </Sample>
     </ComponentStream>
     <ComponentStream name="Y">
        ...
     </ComponentStream>
     <ComponentStream name="Z">
        ...
     </ComponentStream>
     <ComponentStream name="C">
        ...
     </ComponentStream>
     <ComponentStream name="mill">
       <Events>
         <Availability>UNAVAILABLE</Availability>
         <Availability>AVAILABLE</Availability>
       </Events>
     </ComponentStream>
   </DeviceStream>
  </Streams>
</MTConnectStreams>
```

Figure 13: Example #1 for Sample from Sequence #101

In the following illustration, the next request starts at 113 and gets the next ten samples. The response will include the X, Y, Z, and C samples and since there are no `Availablity` events, this component will not be included:



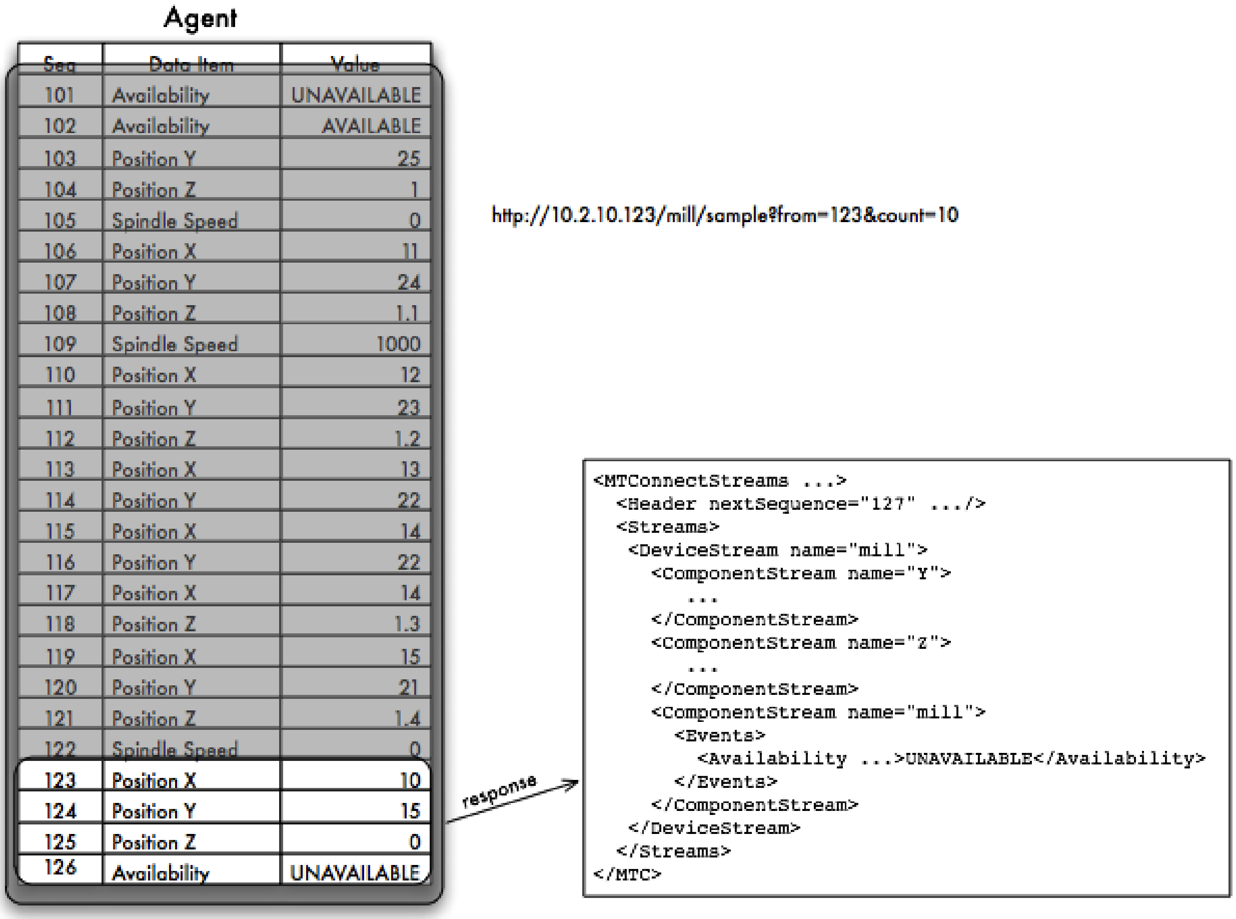
Agent

Seq	Data Item	Value
101	Availability	UNAVAILABLE
102	Availability	AVAILABLE
103	Position Y	25
104	Position Z	1
105	Spindle Speed	0
106	Position X	11
107	Position Y	24
108	Position Z	1.1
109	Spindle Speed	1000
110	Position X	12
111	Position Y	23
112	Position Z	1.2
113	Position X	13
114	Position Y	22
115	Position X	14
116	Position Y	22
117	Position X	14
118	Position Z	1.3
119	Position X	15
120	Position Y	21
121	Position Z	1.4
122	Spindle Speed	0
123	Position X	10
124	Position Y	15
125	Position Z	0
126	Availability	UNAVAILABLE

```
<MTConnectStreams ...>
  <Header nextSequence="123" .../>
  <Streams>
   <DeviceStream name="mill">
     <ComponentStream name="X">
        <Samples>
          <Position ...> </Position>
          <Position ...> </Position>
        </Sample>
     </ComponentStream>
     <ComponentStream name="Y">
        ...
     </ComponentStream>
     <ComponentStream name="Z">
        ...
     </ComponentStream>
     <ComponentStream name="C">
        ...
     </ComponentStream>
   </DeviceStream>
  </Streams>
</MTConnectStreams>
```

Figure 14: Example #1 for Sample from Sequence #113

In the above illustration, only the four axis components have samples. One will only get samples or events if they occur in the window being requested. In the next illustration, the application will request the next ten items starting at sequence number 123.



Seq	Data Item	Value
101	Availability	UNAVAILABLE
102	Availability	AVAILABLE
103	Position Y	25
104	Position Z	1
105	Spindle Speed	0
106	Position X	11
107	Position Y	24
108	Position Z	1.1
109	Spindle Speed	1000
110	Position X	12
111	Position Y	23
112	Position Z	1.2
113	Position X	13
114	Position Y	22
115	Position X	14
116	Position Y	22
117	Position X	14
118	Position Z	1.3
119	Position X	15
120	Position Y	21
121	Position Z	1.4
122	Spindle Speed	0
123	Position X	10
124	Position Y	15
125	Position Z	0
126	Availability	UNAVAILABLE

http://10.2.10.123/mill/sample?from=123&count=10

```
<MTConnectStreams ...>
  <Header nextSequence="127" .../>
  <Streams>
   <DeviceStream name="mill">
     <ComponentStream name="Y">
       ...
     </ComponentStream>
     <ComponentStream name="Z">
       ...
     </ComponentStream>
     <ComponentStream name="mill">
       <Events>
         <Availability ...>UNAVAILABLE</Availability>
       </Events>
     </ComponentStream>
   </DeviceStream>
  </Streams>
</MTC>
```

Figure 15: Example #1 for Sample from Sequence #124

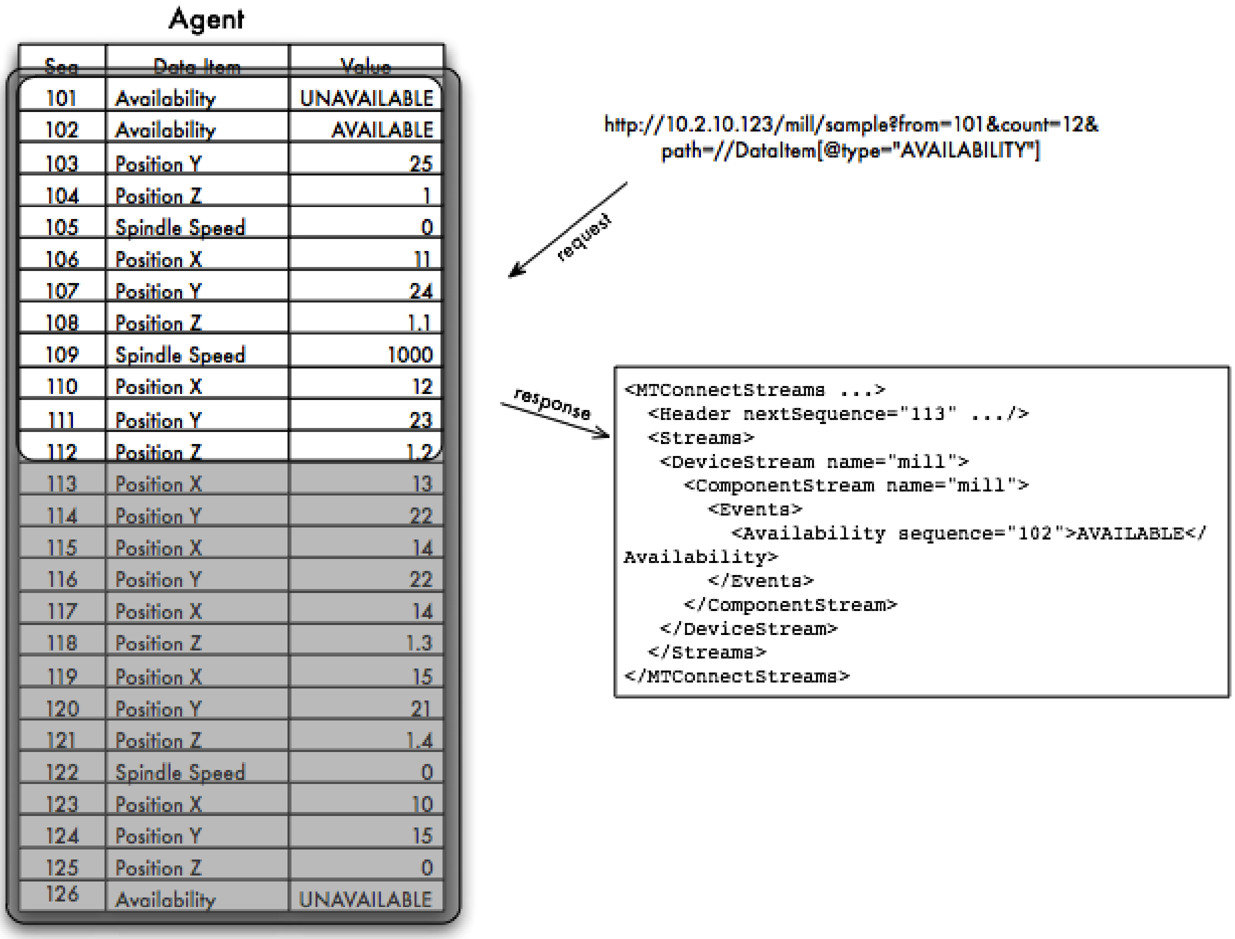
In the above illustration, there are only three items available. The first two are axis samples and the third is a availability event. The next sequence will indicate that the application must request Samples, Events, and Condition starting at 127 for the next group. If the application were to do this, it would receive an empty response with the nextSequence of 127 indicating that no data was available.

The next sequence number **MUST** always be the largest sequence number of available items in the selection window plus one. If the request indicated a from of 10 and a count of 10, the MTConnect® **MUST** consider at most 10 items if available. If the value for from is larger than the last item's sequence number + 1, an OUT_OF_RANGE error **MUST** be returned from the *Agent*.

The same rule will be applied to the current request as well. In the instance of the current request, the next sequence **MUST** be set to the one greater than the last item's sequence number in the table of data values. Since current always considers all Events, Condition, and Samples , it **MUST** always be one greater than the maximum sequence number assigned.

5.9 Request with Filtering using Path Parameter

The next set of examples will show the behavior when a path parameter is provided.



Agent

Seq	Data Item	Value
101	Availability	UNAVAILABLE
102	Availability	AVAILABLE
103	Position Y	25
104	Position Z	1
105	Spindle Speed	0
106	Position X	11
107	Position Y	24
108	Position Z	1.1
109	Spindle Speed	1000
110	Position X	12
111	Position Y	23
112	Position Z	1.2
113	Position X	13
114	Position Y	22
115	Position X	14
116	Position Y	22
117	Position X	14
118	Position Z	1.3
119	Position X	15
120	Position Y	21
121	Position Z	1.4
122	Spindle Speed	0
123	Position X	10
124	Position Y	15
125	Position Z	0
126	Availability	UNAVAILABLE

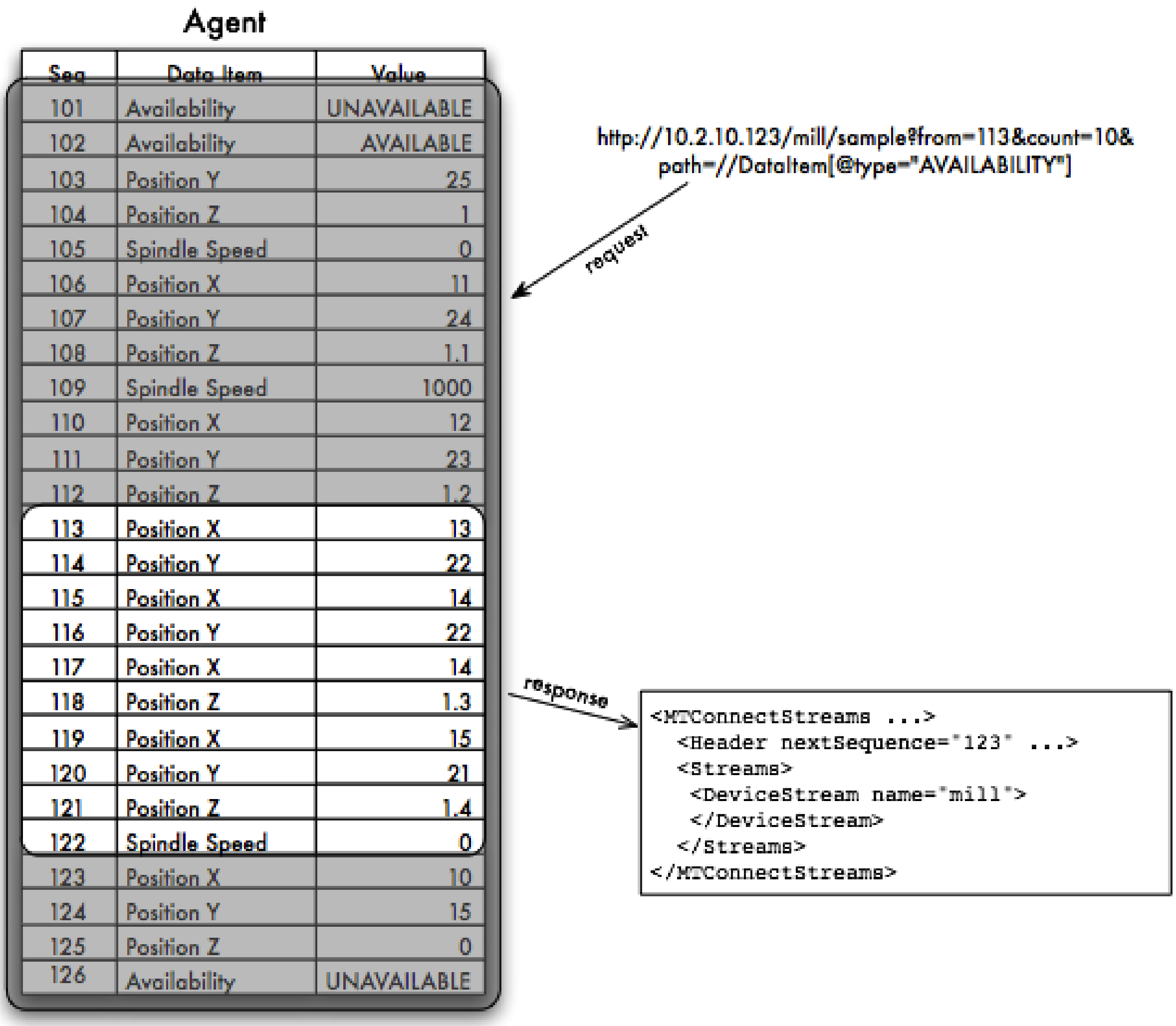
http://10.2.10.123/mill/sample?from=101&count=12&
path=//DataItem[@type="AVAILABILITY"]

```
<MTConnectStreams ...>
  <Header nextSequence="113" .../>
  <Streams>
   <DeviceStream name="mill">
     <ComponentStream name="mill">
       <Events>
         <Availability sequence="102">AVAILABLE</
Availability>
       </Events>
     </ComponentStream>
   </DeviceStream>
  </Streams>
</MTConnectStreams>
```

Figure 16: Example #2 for Sample from Sequence #101 with Path

Figure 16 shows that when events are filtered for only the `Availability` DataItem, the `Availability` is `UNAVAILABLE` event will be delivered and nothing else. The `Availability AVAILABLE` event is sequence number 101, but since the other Samples, Events, and Condition are considered, the next sequence number is still 113. The MTConnect® *Agent* **MUST** set the next sequence number to one greater (+1) than the last event or sample in the window of items being considered. The *Agent* **MUST** consider all the Events, Condition, and Samples evaluated in the process of formulating the response to the application.

In the next illustration the request is sent as before but now only including Availability data items:



Seq	Data Item	Value
101	Availability	UNAVAILABLE
102	Availability	AVAILABLE
103	Position Y	25
104	Position Z	1
105	Spindle Speed	0
106	Position X	11
107	Position Y	24
108	Position Z	1.1
109	Spindle Speed	1000
110	Position X	12
111	Position Y	23
112	Position Z	1.2
113	Position X	13
114	Position Y	22
115	Position X	14
116	Position Y	22
117	Position X	14
118	Position Z	1.3
119	Position X	15
120	Position Y	21
121	Position Z	1.4
122	Spindle Speed	0
123	Position X	10
124	Position Y	15
125	Position Z	0
126	Availability	UNAVAILABLE

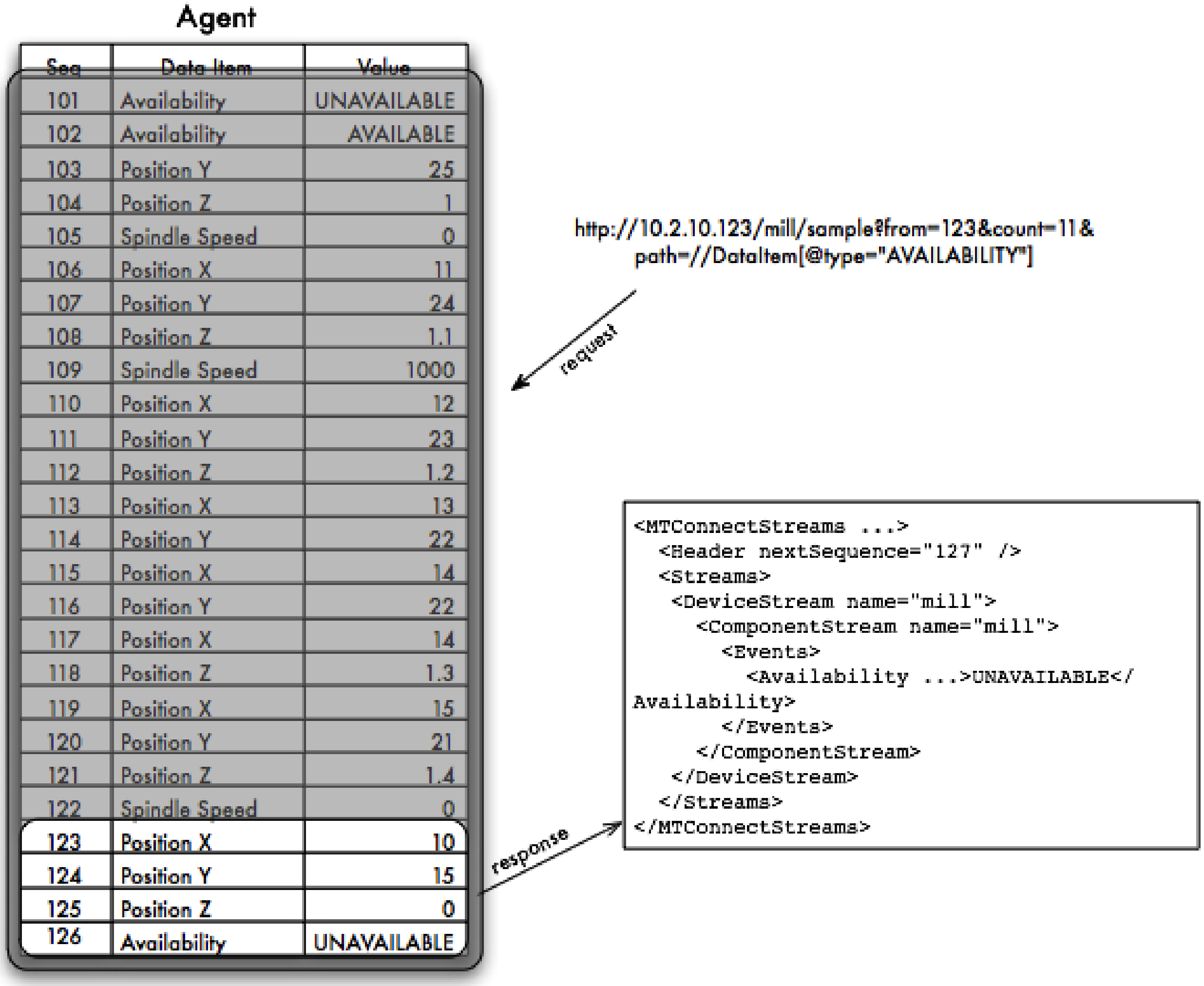
```
http://10.2.10.123/mill/sample?from=113&count=10&
path=//DataItem[@type="AVAILABILITY"]
```

```
<MTConnectStreams ...>
  <Header nextSequence="123" ...>
  <Streams>
   <DeviceStream name="mill">
   </DeviceStream>
  </Streams>
</MTConnectStreams>
```

Figure 17: Example #2 for Sample from Sequence #112 with Path

An empty element representing the device **MUST** be returned to indicate that the request was valid and no data was found since there were no `AVAILABILITY` events in the given range. The nextSequence in the case **MUST** be set to `113` even though no results were returned. If this was not done, the application would continue to request sequence starting at `113` indefinitely.

To continue this example, the last request will start at 123 as before and will now request only Availability data item:



Seq	Data Item	Value
101	Availability	UNAVAILABLE
102	Availability	AVAILABLE
103	Position Y	25
104	Position Z	1
105	Spindle Speed	0
106	Position X	11
107	Position Y	24
108	Position Z	1.1
109	Spindle Speed	1000
110	Position X	12
111	Position Y	23
112	Position Z	1.2
113	Position X	13
114	Position Y	22
115	Position X	14
116	Position Y	22
117	Position X	14
118	Position Z	1.3
119	Position X	15
120	Position Y	21
121	Position Z	1.4
122	Spindle Speed	0
123	Position X	10
124	Position Y	15
125	Position Z	0
126	Availability	UNAVAILABLE

http://10.2.10.123/mill/sample?from=123&count=11&
path=//DataItem[@type="AVAILABILITY"]

```
<MTConnectStreams ...>
  <Header nextSequence="127" />
  <Streams>
   <DeviceStream name="mill">
     <ComponentStream name="mill">
       <Events>
         <Availability ...>UNAVAILABLE</
Availability>
       </Events>
     </ComponentStream>
   </DeviceStream>
  </Streams>
</MTConnectStreams>
```

Figure 18: Example #2 for Sample from Sequence #123 with Path

As can be seen, the one Availability event is returned and the next sequence is now 127. This will indicate that the application must request from 127 on for the next set of events. If no events are available, the `nextSequence` will again be set to 127 and an empty `DeviceStream` will be returned.

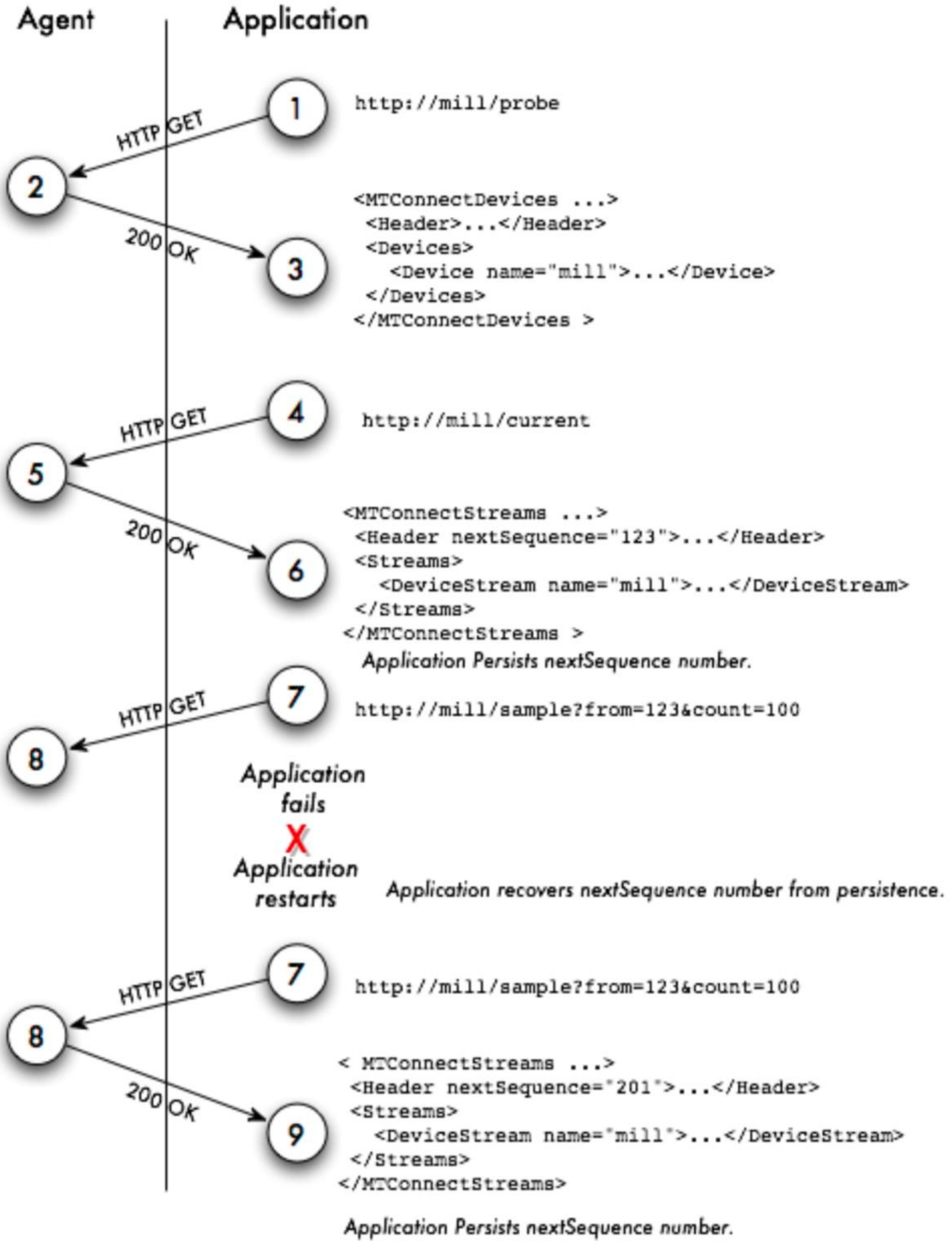
5.10 Fault Tolerance and Recovery

MTConnect® does not provide a guaranteed delivery mechanism. The protocol places the responsibility for recovery on the application.

5.10.1 **Application Failure**

The application failure scenario is easy to manage if the application persists the next sequence number after it processes each response. The MTConnect® protocol provides a simple recovery strategy that only involves reissuing the previous request with the recovered next sequence number.

There is the risk of missing some Events, Samples, and Condition if the time between requests exceeds the capacity of the *Agent*'s buffer. In this case, there is no record of the missing information and it is lost. If the application automatically restarts after failure, the intervening data can be quickly recovered



```
http://mill/probe
```

```
<MTConnectDevices ...>
 <Header>...</Header>
 <Devices>
   <Device name="mill">...</Device>
 </Devices>
</MTConnectDevices >
```

```
http://mill/current
```

```
<MTConnectStreams ...>
 <Header nextSequence="123">...</Header>
 <Streams>
   <DeviceStream name="mill">...</DeviceStream>
 </Streams>
</MTConnectStreams >
```

Application Persists nextSequence number.

```
http://mill/sample?from=123&count=100
```

Application recovers nextSequence number from persistence.

```
http://mill/sample?from=123&count=100
```

```
< MTConnectStreams ...>
 <Header nextSequence="201">...</Header>
 <Streams>
   <DeviceStream name="mill">...</DeviceStream>
 </Streams>
</MTConnectStreams>
```

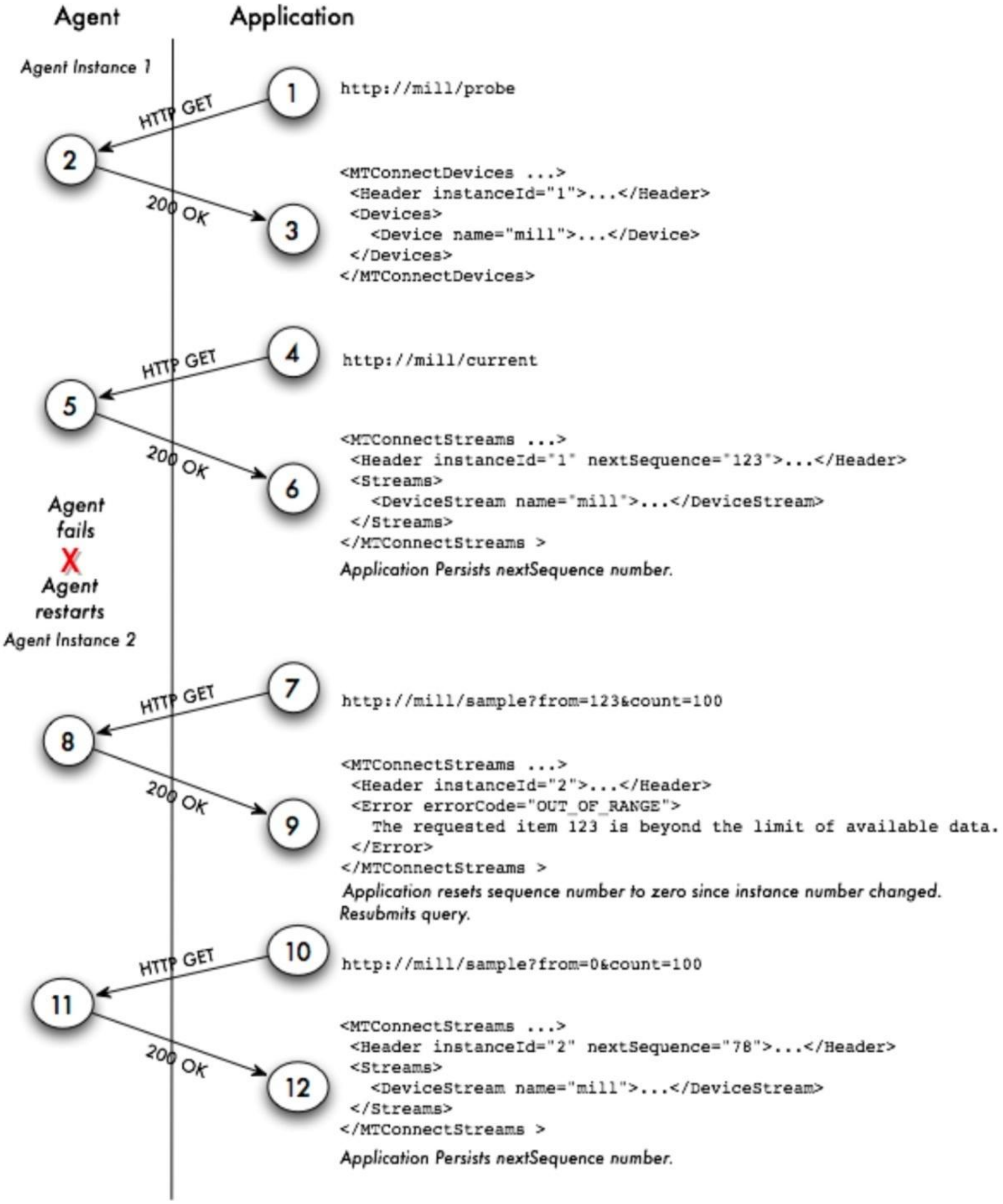
Application Persists nextSequence number.

Figure 19: Application Failure and Recovery

If this cannot be done, the current state of the device can be retrieved and the application can continue from that point onward.

5.10.2 **Agent Failure**

Agent failure is the more complex scenario and requires the use of the `instanceId`. The `instanceId` was created to facilitate recovery when the *Agent* fails and the application is unaware. Since HTTP is a connectionless protocol, there is no way for the application to easily detect that the *Agent* has restarted, the buffer has been lost, and the sequence number has been reset to 1. It should also be noted that all values will be reinitialized to UNAVAILABLE upon agent restart except for data items that are constrained to single values. *See Part 1, Section 5.11 on Unavailability of Data* for a full explanation.



```
http://mill/probe
```

```
<MTConnectDevices ...>
 <Header instanceId="1">...</Header>
 <Devices>
   <Device name="mill">...</Device>
 </Devices>
</MTConnectDevices>
```

```
http://mill/current
```

```
<MTConnectStreams ...>
 <Header instanceId="1" nextSequence="123">...</Header>
 <Streams>
   <DeviceStream name="mill">...</DeviceStream>
 </Streams>
</MTConnectStreams >
```

Application Persists nextSequence number.

```
http://mill/sample?from=123&count=100
```

```
<MTConnectStreams ...>
 <Header instanceId="2">...</Header>
 <Error errorCode="OUT_OF_RANGE">
   The requested item 123 is beyond the limit of available data.
 </Error>
</MTConnectStreams >
```

Application resets sequence number to zero since instance number changed. Resubmits query.

```
http://mill/sample?from=0&count=100
```

```
<MTConnectStreams ...>
 <Header instanceId="2" nextSequence="78">...</Header>
 <Streams>
   <DeviceStream name="mill">...</DeviceStream>
 </Streams>
</MTConnectStreams >
```

Application Persists nextSequence number.

Figure 20: Agent Failure and Recovery

In the above example, the `instanceId` is increased from 1 to 2 indicating that there was a discontinuity in the sequence numbers and all values for the data items are reset to `UNAVAILABLE`. When the application detects the change in `instanceId`, it **MUST** reset its next sequence number and retry its request from sequence number 1. The next request will retrieve all data starting from the first available event or sample.

5.10.3 Data Persistence and Recovery

The implementer of the *Agent* can decide on the strategy regarding the storage of Events, Condition, and Samples. In the simplest form, the *Agent* can persist no data and hold all the results in volatile memory. If the *Agent* has a method of persisting the data fast enough and has sufficient storage, it **MAY** save as much or as little data as is practical in a recoverable storage system.

If the *Agent* can recover data and sequence numbers from a storage system, it **MUST NOT** change the `instanceId` when it restarts. This will indicate to the application that it need not reset the next sequence number when it requests the next set of data from the *Agent*.

If the *Agent* persists no data, then it **MUST** change the `instanceId` to a different value when it restarts. This will ensure that every application receiving information from the *Agent* will know to reset the next sequence number.

The `instanceId` can be any unique number that will be guaranteed to change every time the *Agent* restarts. If the *Agent* will take longer than one second to start, the UNIX time (seconds since January 1, 1970) **MAY** be used for identification an instance of the MTConnect® *Agent* in the `instanceId`.

5.11 Unavailability of Data

Every time the *Agent* is initialized all values **MUST** be set to `UNAVAILABLE` unless they are constant valued data items as described in 5.11.2 below. Even during restarts this must occur so that the application can detect a discontinuity of data and easily determine that gap between the last reported valid values.

In the event no data is available, the value for the data item in the stream **MUST** be `UNAVAILABLE`. This value indicates that the value is currently indeterminate and no assumptions are possible. MTConnect® supports multiple data sources per device, and for that reason, every data item **MUST** be considered independent and **MUST** maintain its own connection status.

In the following example, the data source for a temperature sensor becomes temporarily disconnected from the *Agent*. At this point the value changes from the current temperature to `UNAVAILABLE` since the temperature can no longer be determined.

In figure 17, the temperatures range around 100 until it becomes disconnected and then in the future it reconnects and the temperature is 30. Between these two points assumptions **SHOULD NOT** be made as to the temperature since no information was available.

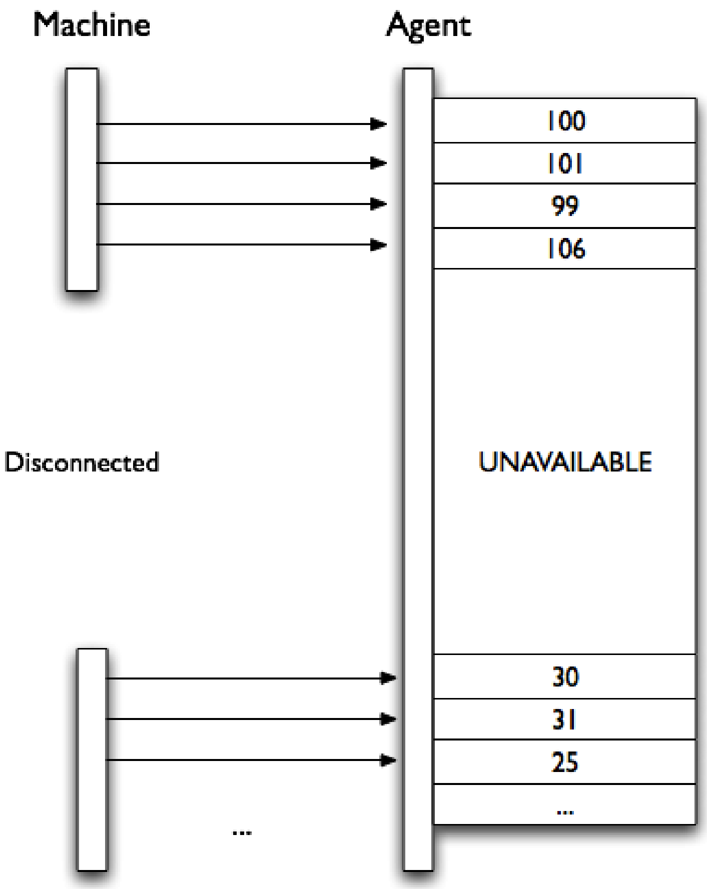


Figure 21: Unavailable Data from Machine

If data for multiple data items are delivered from one source and that source becomes unavailable, all data items associated with that source **MUST** have the value `UNAVAILABLE`. This **MUST** be a synchronous operation where all related data items will get that value with the same time stamp. The value will remain `UNAVAILABLE` until the data source has reconnected.

5.11.1 Examples

```
<Linear name=”X” id=”x”>
  <DataItems>
    <DataItem type=”POSITION” category=”SAMPLE” id=”Xpos” … />
    <DataItem type=”TEMPERATURE” category=”SAMPLE” id=”Ctemp” … />
  </DataItems>
</Linear>
```

When the *Agent* is started and has no initial information about the device, all data item value **MUST** have the value `UNAVAILABLE`. This will produce the following results to a current request:

```
<ComponentStream component="Linear" componentId="x" name="X">
  <Samples>
   <Position timestamp="2010-03-01T11:59:09.001" dataItemId="Xpos" sequence="99" >UNAVAILABLE</Position>
```

```
    <Temperature timestamp="2010-03-01T11:59:09.001" dataItemId="Xpos" se-
quence="100" >UNAVAILABLE</Temperature>
  </Samples>
</ComponentStream>
```

Once the adapters are connected, the values will no longer be `UNAVAILABLE`. The results from the `current` once again:

```
<ComponentStream component="Linear" componentId="x" name="X">
  <Samples>
    <Position timestamp="2010-03-01T12:09:31.021" dataItemId="Xpos" se-
quence="122" >13.0003</Position>
    <Temperature timestamp="2010-03-01T12:07:22.031" dataItemId="Xpos" se-
quence="113" >102</Temperature>
  </Samples>
</ComponentStream>
```

If the temperature sensor should lose power and become disconnected, as shown in figure 17, the following response will be given by `current`.

```
<ComponentStream component="Linear" componentId="x" name="X">
  <Samples>
    <Position timestamp="2010-03-01T12:12:19.311" dataItemId="Xpos" se-
quence="212" >1.0003</Position>
    <Temperature timestamp="2010-03-01T12:15:41.121" dataItemId="Xpos" se-
quence="199" >UNAVAILABLE</Temperature>
  </Samples>
</ComponentStream>
```

The X position has a valid value and only the Temperature is unknown. When a `sample` is requested, the value `UNAVAILABLE` will be treated the same as any other value for the data item.

```
<ComponentStream component="Linear" componentId="x" name="X">
  <Samples>
    <Position timestamp="2010-03-01T11:59:09" dataItemId="Xpos" sequence="212"
>1.0003</Position>
    <Position timestamp="2010-03-01T11:59:09" dataItemId="Xpos" sequence="212"
>2.2103</Position>
    <Position timestamp="2010-03-01T11:59:09" dataItemId="Xpos" sequence="212"
>4.3303</Position>
    <Temperature timestamp="2010-03-01T11:59:09" dataItemId="Xpos" se-
quence="199" >101</Temperature>
    <Temperature timestamp="2010-03-01T11:59:09" dataItemId="Xpos" se-
quence="199" >103</Temperature>
    <Temperature timestamp="2010-03-01T11:59:09" dataItemId="Xpos" se-
quence="199" >UNAVAILABLE</Temperature>
  </Samples>
</ComponentStream>
```

5.11.2 Constant valued data items

If the data item is constrained to one value, the initial value for this data item **MUST** be that value. For example:

```
<Rotary name=”C” id=”C” nativeName=”S”>
  <DataItems>
    <DataItem type=”ROTARY_MODE” category=”EVENT” id=”Cmode”>
     <Constraints><Value>SPINDLE</Value></Constraints>
    </DataItem>
    <DataItem type=”SPINDLE_SPEED” category=”SAMPLE” id=”Cspeed”/>
  </DataItems>
</Rotary>
```

In this example, the `RotaryMode` **MUST** be initialized to `SPINDLE`. If an application was to request data from this device before the adapter was connect, the result **MUST** be the following:

```
<ComponentStream component="Rotary" componentId="c" name="C">
  <Events>
    <RotaryMode timestamp="2010-03-01T11:58:09" dataItemId="Cmode" se-
quence="1" >SPINDLE</Position>
  <Events>
  <Samples>
    <SpindleSpeed timestamp="2010-03-01T11:59:09" dataItemId="Cspeed" se-
quence="113" >UNAVAILABLE</Temperature>
  </Samples>
</ComponentStream>
```

The `SpindleSpeed` shows `UNAVAILABLE` as described above, but the `RotaryMode` is assigned the constant value `SPINDLE` since it can only have one value. The value for `RotaryMode` **MAY NOT** be delivered by the *Adapter* and if it is, it **MUST** be `SPINDLE`.

For more information on `Constraints`, see *MTConnect Part 2, Section 4.1 – Data Item Element.*

Appendices

A. Bibliography

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

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

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

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

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

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

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

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

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

10. H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.

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

12. *ASME B5.59-2 Version 9c: Data Specification for Properties of Machine Tools for Milling and Turning. 2005.*

13. *ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers. 2005.*

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

15. View the following site for RFC references: http://www.faqs.org/rfcs/ .

B. Discovery

The deployment of MTConnect® **SHOULD** use a separate service to aid applications in locating and communicating with devices. If discovery is employed, the MTConnect® Agent **MUST** register all the devices in an LDAP server so each device's *Agent* can be located on the network with an HTTP URI. The `device` entry in LDAP **MUST** include a `labeledURIObject` and **MUST** specify the `labeledURI` field. Other information **MAY** be added to the LDAP `device` record depending on the needs of the application and the organization.

Applications **MAY** require the ability to locate devices and it is best handled by the discovery service. The implementation **SHOULD NOT** assume that one *Agent* will be providing data for all the devices. If one wants to find all the devices available for data collection using the MTConnect® protocol, they **SHOULD** use an LDAP server to organize their equipment and resolve the machine names into valid URIs.

If discovery is not provided or used, the application **MUST** know the URI for the device's *Agent* and address it directly.

B.1. Physical Architecture

The diagram below is an example of a shop floor with three devices, one management application, and one *Name Service*. There are two MTConnect® *Agents* in this deployment. One of the MTConnect® *Agents* is serving two pieces of equipment (lathe-1 and lathe-2) and the other *Agent* is embedded in the controller of the mill. The management application is monitoring all three pieces of equipment.

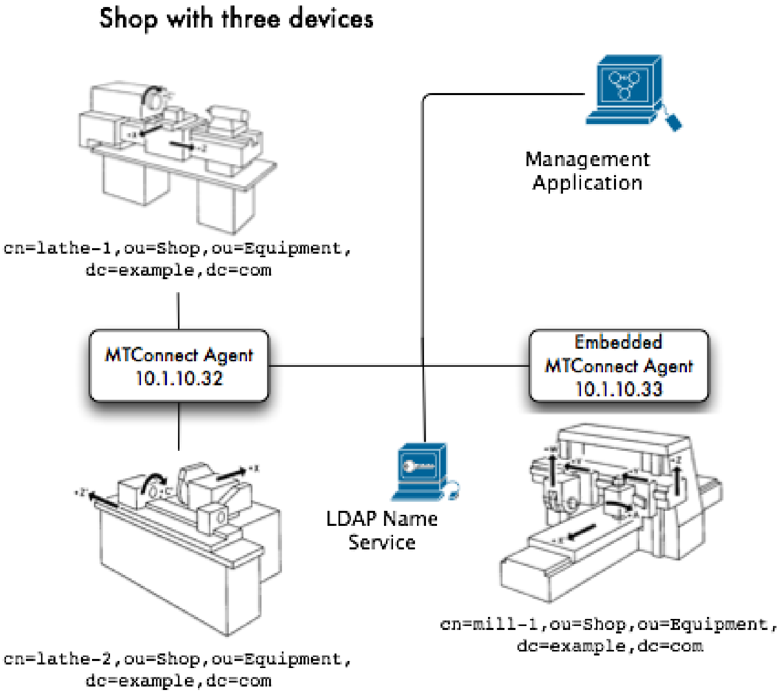

Shop with three devices
Management Application
cn=lathe-1,ou=Shop,ou=Equipment, dc=example,dc=com
MTConnect Agent 10.1.10.32
Embedded MTConnect Agent 10.1.10.33
LDAP Name Service
cn=mill-1,ou=Shop,ou=Equipment, dc=example,dc=com
cn=lathe-2,ou=Shop,ou=Equipment, dc=example,dc=com

Figure 22: Shop Illustration

One can look up the three devices using the *Name Service*. The application would search for all devices in the Equipment organization unit (ou=Equipment,dc=example,dc=com). The application would get back three device names: lathe-1, lathe-2, and mill-1. These would be have the following URIs: http://10.1.10.32/lathe-1, http://10.1.10.32/lathe-2, and http://10.1.10.33/mill-1.

The application can thereafter use the URIs to query the devices for the components and the data they can supply.