MTConnect Standard
Part 1 - Overview and Protocol
Version 1.0.1

Prepared for: MTConnect Institute
Prepared by: William Sobel
Prepared on: October 2, 2009

MTConnect Specification

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.

The first version of MTConnect will focus on a limited set of the characteristics mentioned above that were selected based on the fact that they can have an immediate affect on the efficiency of operations.

1.1 MTConnect Document Structure

The MTConnect specification is subdivided using the following scheme:

Part 1: Overview and Protocol
Part 2: Components and Data Items
Part 3: Streams, Events and Samples

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.

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.

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 specification, acting as an interface to the device.

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` and `Events` 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 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.

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:

```
<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:

```
<?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. The second part contains the XML document and consists of the rest 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.

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. For example: `ON`, `OFF`, `ACTUAL`, 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 collecting 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 implementor to decide if they are relevant to their device.

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. This will allow for greatest interoperability with other specifications. It is assumed that all MTConnect *Agents* will be responsible for converting the units from the native device units.

Property	Symbol	Unit
Angle	°	decimal degrees
Angular Acceleration	$°/s^2$	degree per second square
Angular Velocity	°/s	degrees per second
Elapsed time	s	seconds with fractions
Force	N	newtons
Length	mm	millimeters
Linear Acceleration	mm/s^2	millimeter per second square
Linear Velocity	mm/s	millimeters per second
Mass	kg	kilograms
Spindle Speed	rev/min	revolutions per minute
Temperature	°C	degree Celsius

Additional units will be added as needed. The decision to require the *Agent* to convert to the standard simplifies the applications and will provide greater interoperability and accuracy.

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 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 on page 15)*
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 section4)*
Streams	A set of samples or events 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	Unexpected or discrete occurrence in a component. This includes state changes and alarms. *(See Events in Part 3)*
Alarms	A type of event that indicates an abnormal behavior. *(See Alarms in Part 3)*

Each of these parts will be covered in detail in the following sections as well as Part 2 and 3 of the Standard.

3.1 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.

Discussion of discovery will be detailed in the *Name Service* related specification.

3.2 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.

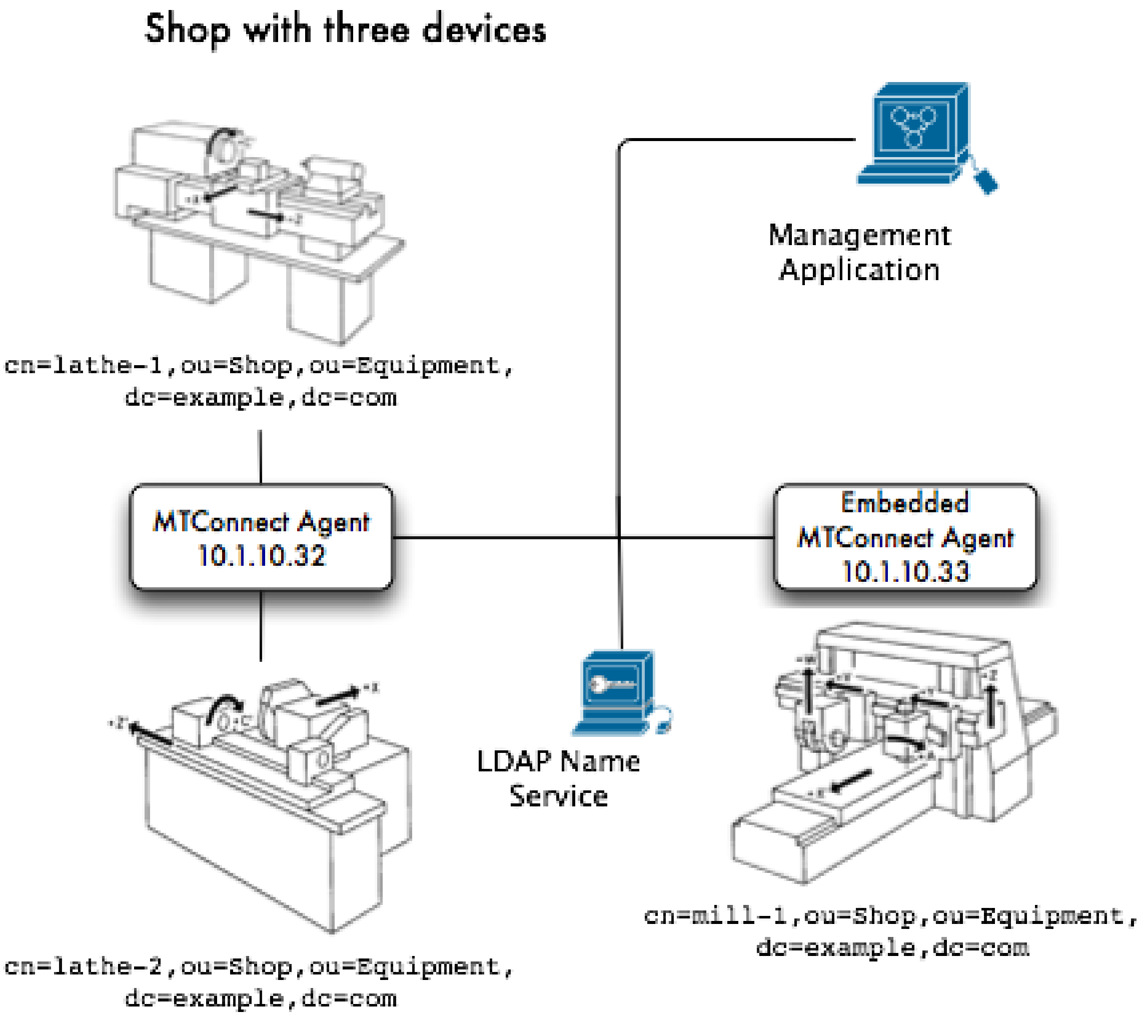

Figure 1: Shop Illustration

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.

3.3 Optional Embedded Architecture

The MTConnect *Agent* can also be deployed as an embedded service with no external network access. Since there is no external network, there is no need for the *Name Service*. As shown in the diagram below, the *Agent* will interconnect the various components of a single device.

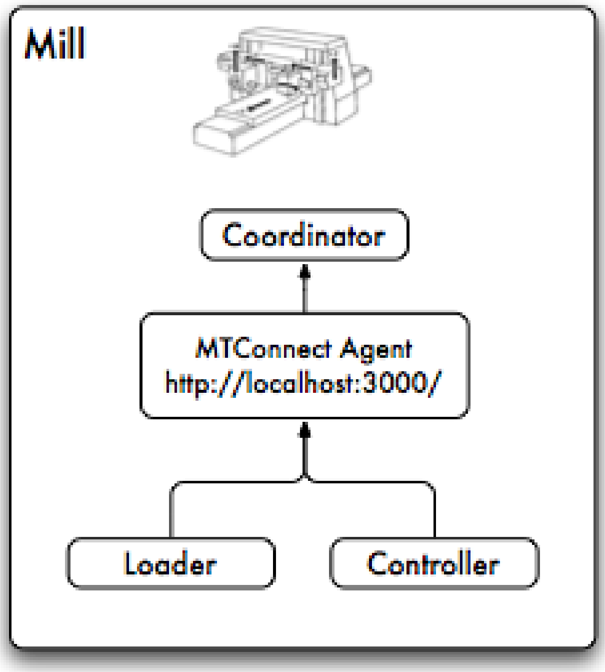

Figure 2: Device Detail

In the above illustration, we present a single device with an embedded *Agent*. The *Agent* is only communicating locally with the Loader and Controller components, there is no external network. The Coordinator is an application that is receiving information from the *Agent* and is making some control decisions based on the information it is receiving.

Although MTConnect is not addressing the real-time aspects of data capture, it **MAY** be used if the requirements of the application do not exceed the response time and performance of the *Agent*. The command and control workflow will be addressed in later version of the specification.

3.4 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.)

3.5 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).

3.5.1 Agent Initialization

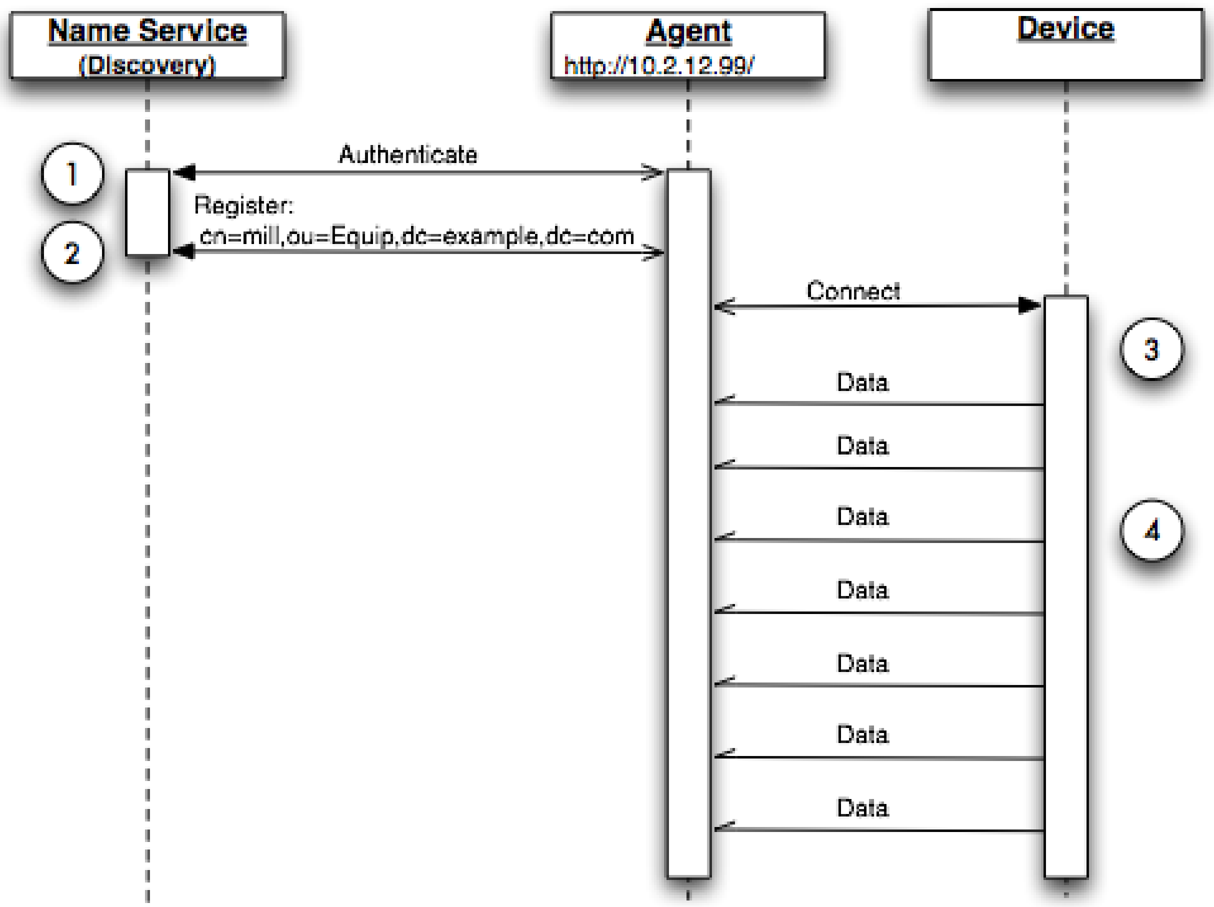

Figure 3: Agent Initialization

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.

3.5.2 Application Communication

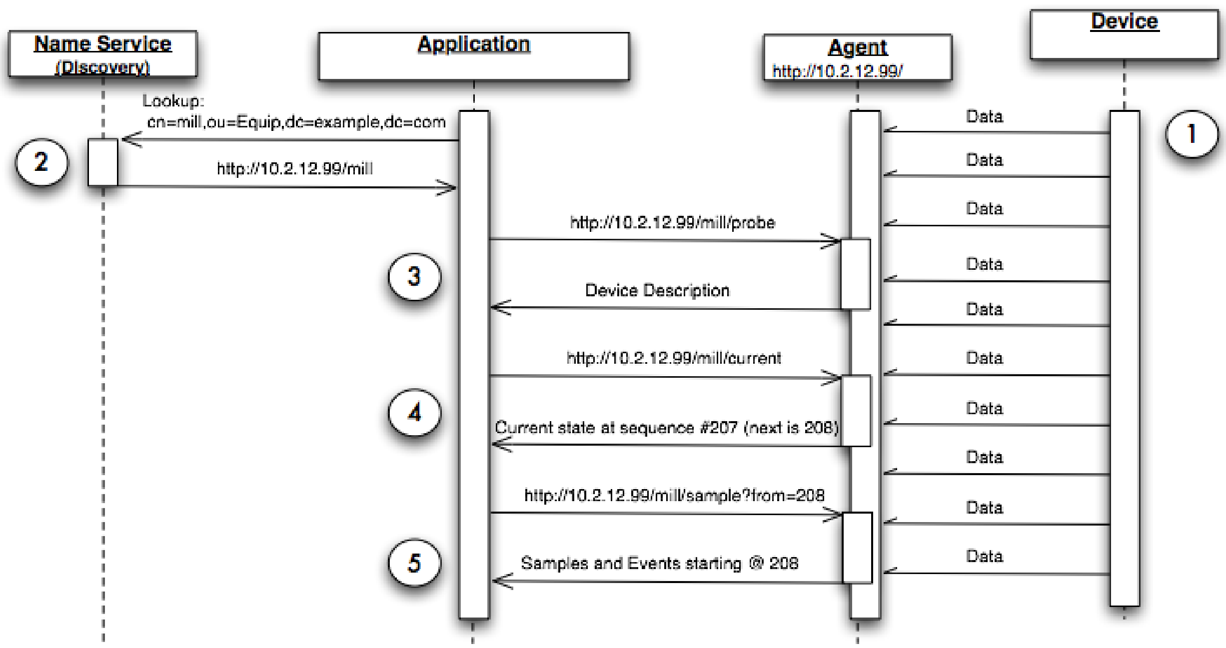

Figure 4: Application Communication

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 persist. The data flow from the adapter 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 or events. 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 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 sections. The first section is the `Header` that provides protocol related information like next sequence number and creation date and the second section the content for `Devices`, `Streams`, or `Error`.

4.1 MTConnectDevices

`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** include a `Devices` section. An `MTConnectDevices` XML Document will have the following structure (the details have been eliminated for illustrative purposes):

```
1. <MTConnectDevices …>
2.   <Header> … </Header>
3.   <Devices> … </Devices>
4. </MTConnectDevices>
```

4.1.1 MTConnectDevices Elements

An `MTConnectDevices` document **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

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

```
1. <MTConnectStreams …>
2.   <Header> … </Header>
3.   <Streams> … </Streams>
```

```
4. </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

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 an Error section:

```
1. <MTConnectError …>
2.   <Header> … </Header>
3.   <Error> … </Error>
4. </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
Error	The error information	1

For the above elements of the XML Document, please refer to section 4.4 for Header and section 5.5 for Error.

4.4 Header

Every MTConnect response **MUST** contain a header as the first element of any MTConnect XML Document sent back to an application. The following information **MUST** be provided in the header:

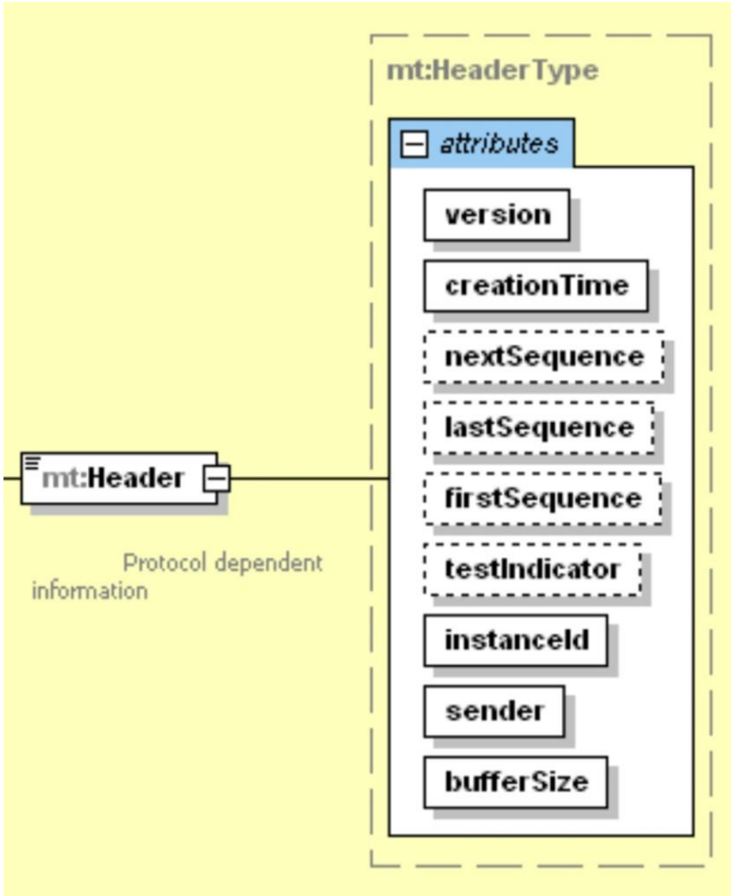

Figure 5: Header Schema Diagram

```
1. <Header instanceId="1" creationTime="2007-12-03T13:23:33"
      sender="http://10.3.1.10" bufferSize="1000000" firstSequence="107"
      lastSequence="3780" />
```

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

Attribute	Description	Occurrence
testIndicator	Optional flag that indicates the system is operating in test mode. This data is only for testing and may be fake.	0..1
sender	The *Agent* identification information.	1
bufferSize	The number of samples and events 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.	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.	1
version	The protocol version number. This will be 1.0 for this specification.	1

The nextSequence, firstSequence, and lastSequence number **MUST** be provided for sample and current requests, but it **MUST NOT** be provided for the probe request. The testIndicator **MAY** be provided as needed. The firstSequence and lastSequence **MUST** be provided for sample and current requests to allow determination by the client application that the required sequence numbers are within range.

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 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 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.

An MTConnect *Agent* **MUST** only 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` protocol error 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
- `current` – to retrieve a snapshot of the data item's most recent values
- `sample` – to retrieve the samples and events in sequence

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.

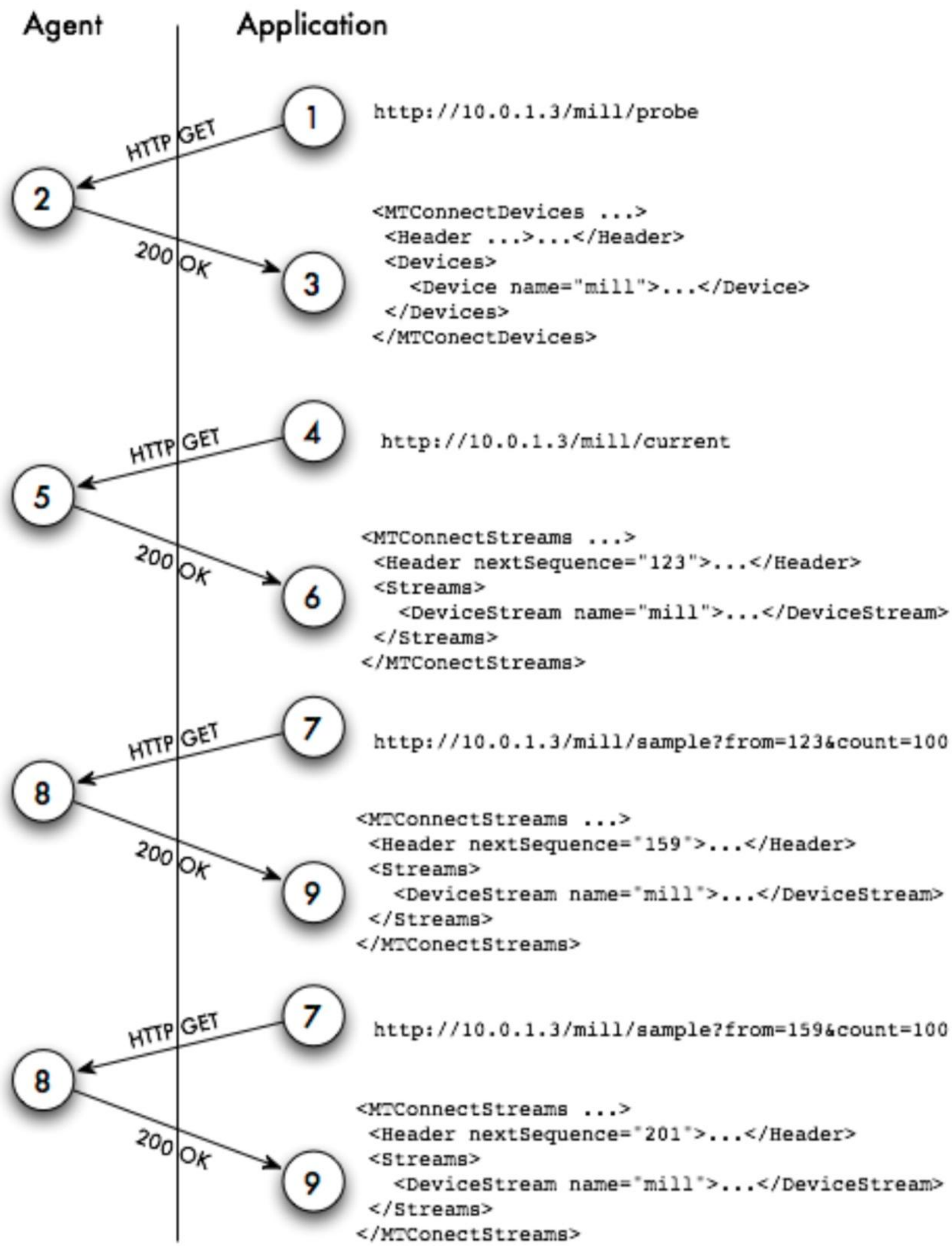

Figure 6: Application and Agent Conversation

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 and events.

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 `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 LinuxCNC:

```
<MTConnectDevices
    xsi:schemaLocation="urn:mtconnect.com:MTConnectDevices:0.9”
    http://www.mtconnect.org/schemas/MTConnectDevices.xsd">
  <Header sender="localhost" bufferSize="100000" creationTime="2008-07-
    06T00:01:05-07:00" version="0.9" instanceId="1214527986"/>
  <Devices>
    <Device iso841Class="6" uuid="linux-01" name="LinuxCNC"
    sampleRate="100.0" id="1">
      <Description manufacturer="NIST" serialNumber="01"/>
      <DataItems>
        <DataItem type="ALARM" name="alarm" category="EVENT" id="10"/>
      </DataItems>
      <Components>
          <Axes name="Axes" id="3">
            <DataItems>
              <DataItem type="PATH_FEEDRATE" name="path_feedrate"
    category="SAMPLE" id="11" nativeUnits="PERCENT" subType="OVERRIDE"
    units="PERCENT"/>
            </DataItems>
            <Components>
              <Spindle name="S" id="7">
                <DataItems>
                  <DataItem type="SPINDLE_SPEED" name="Sspeed"
    category="SAMPLE" id="18" nativeUnits="REVOLUTION/MINUTE"
    subType="ACTUAL" units="REVOLUTION/MINUTE">
                    <Source>spindle_speed</Source>
                  </DataItem>
                  <DataItem type="PRESSURE" name="Jet" id="31"/>
```

```xml
                    </DataItems>
                  </Spindle>
                  <Linear name="X" id="4">
                    <DataItems>
                      <DataItem type="POSITION" name="Xact" category="SAMPLE" id="12" nativeUnits="MILLIMETER" subType="ACTUAL" units="MILLIMETER"/>
                      <DataItem type="POSITION" name="Xcom" category="SAMPLE" id="13" nativeUnits="MILLIMETER" subType="COMMANDED" units="MILLIMETER"/>
                    </DataItems>
                  </Linear>
                  <Linear name="Y" id="5">
                    <DataItems>
                      <DataItem type="POSITION" name="Yact" category="SAMPLE" id="14" nativeUnits="MILLIMETER" subType="ACTUAL" units="MILLIMETER"/>
                      <DataItem type="POSITION" name="Ycom" category="SAMPLE" id="15" nativeUnits="MILLIMETER" subType="COMMANDED" units="MILLIMETER"/>
                    </DataItems>
                  </Linear>
                  <Linear name="Z" id="6">
                    <DataItems>
                      <DataItem type="POSITION" name="Zact" category="SAMPLE" id="16" nativeUnits="MILLIMETER" subType="ACTUAL" units="MILLIMETER"/>
                      <DataItem type="POSITION" name="Zcom" category="SAMPLE" id="17" nativeUnits="MILLIMETER" subType="COMMANDED" units="MILLIMETER"/>
                    </DataItems>
                  </Linear>
                </Components>
              </Axes>
              <Controller name="Controller" id="8">
                <DataItems>
                  <DataItem type="LINE" name="line" category="EVENT" id="19" subType="ACTUAL"/>
                  <DataItem type="CONTROLLER_MODE" name="mode" category="EVENT" id="20"/>
                  <DataItem type="PROGRAM" name="program" category="EVENT" id="21"/>
                  <DataItem type="EXECUTION" name="execution" category="EVENT" id="22"/>
                </DataItems>
              </Controller>
              <Power name="power" id="2">
```

```
              <DataItems>
                <DataItem type="POWER_STATUS" name="power" category="EVENT"
     id="9"/>
              </DataItems>
            </Power>
          </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, three linear and one spindle axes and two data items for each axis. The Controller is capable of providing the execution status and the current block of code. The device has a single power component that will indicate if the device is turned on or off.

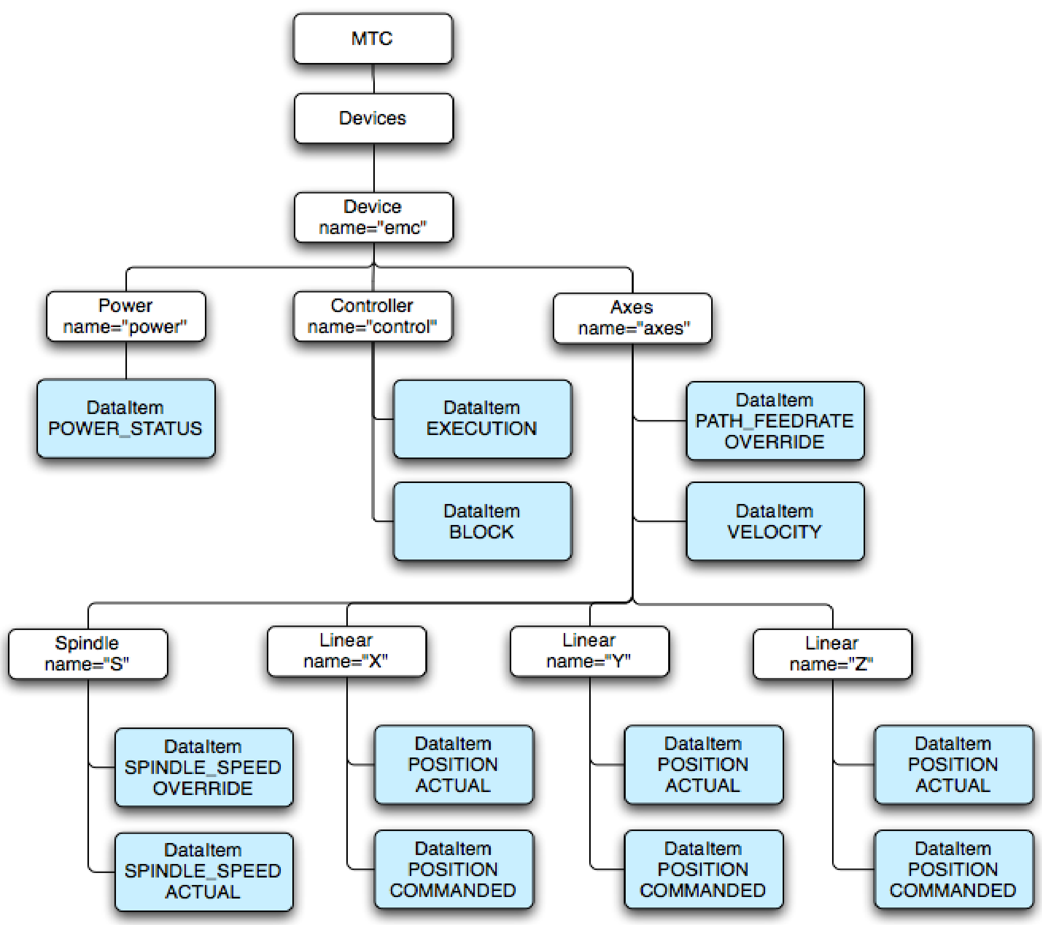

Figure 7: Sample Device Organization

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):

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

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

```
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:

```
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:

```
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:

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

or:

```
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.*

5.3.1 Parameters

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 `Spindle` sub-components **MUST** be included as well.

`from` - This parameter requests events 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 and samples to consider, see detailed explanation below. Events 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 and samples (in aggregate), but fewer events 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 and events to the client application pausing for `frequency` milliseconds between each part. Each part will contain a maximum of `count` events or samples and `from` will be used to indicate the beginning of the stream.

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 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 or devices, in the same way as the `sample` request.

```
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.

`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 and events 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.

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`.

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

5.5 Streaming

When the `frequency` parameter is provided, the MTConnect *Agent* **MUST** check for available events and sample at the frequency specified or at its maximum possible scan rate. The frequency indicates the delay between data deliveries. 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 will use a **MIME** encoded message with each section separated by a **MIME** boundary. Each section of the response will 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:3000/sample?frequency=1000&path=//Power`

Sample response:

```
1. HTTP/1.1 200 OK
2. Connection: close
3. Date: Mon, 01 Dec 2008 21:35:13 GMT
4. Status: 200 OK
5. X-Runtime: 0.12153
6. Content-Transfer-Encoding: binary
7. Cache-Control: private
8. Content-Disposition: inline
9. Server: Mongrel 1.1.5
10. Content-Type: multipart/x-mixed-
replace;boundary=8a89b9e00b810f6de5901cc0014d706d
11. Content-Length: 10737418240
12.

```

Lines 1-12 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.

```
13. --8a89b9e00b810f6de5901cc0014d706d
14. Content-type: text/xml
15. Content-length: 596
16.
```

```
17. <?xml version="1.0" encoding="UTF-8"?>
18. <MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:0.9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.com:MTConnectStreams:0.9"
xsi:schemaLocation="urn:mtconnect.com:MTConnectStreams:0.9
/schemas/MTConnectStreams.xsd">
19. <Header version="0.9" firstSequence="0" lastSequence="20"
sender="localhost" creationTime="2008-12-01T13:35:15-08:00"
bufferSize="100000" instanceId="1228167061" nextSequence="21"/>
20. <Streams>
21. <DeviceStream name="LinuxCNC" uuid="linux-01">
22. </DeviceStream>
23. </Streams>
24. </MTConnectStreams>
```

Lines 13-24 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.

```
25. --8a89b9e00b810f6de5901cc0014d706d
26. Content-type: text/xml
27. Content-length: 850
28.
29. <?xml version="1.0" encoding="UTF-8"?>
30. <MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:0.9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.com:MTConnectStreams:0.9"
xsi:schemaLocation="urn:mtconnect.com:MTConnectStreams:0.9
/schemas/MTConnectStreams.xsd">
31. <Header version="0.9" firstSequence="0" lastSequence="22"
sender="localhost" creationTime="2008-12-01T13:35:29-08:00"
bufferSize="100000" instanceId="1228167061" nextSequence="23"/>
32. <Streams>
33. <DeviceStream name="LinuxCNC" uuid="linux-01">
34.   <ComponentStream name="power" component="Power" componentId="2">
35.     <Events>
36.       <PowerStatus dataItemId="15" sequence="22" name="power"
timestamp="2008-08-14T20:13:14.253192">OFF</PowerStatus>
37.     </Events>
38.   </ComponentStream>
39. </DeviceStream>
40. </Streams>
```

```
41. </MTConnectStreams>
```

Lines 25-41: After a period of time, the power gets turned off and a new mime part is sent with the new status.

```
42. --8a89b9e00b810f6de5901cc0014d706d
43. Content-type: text/xml
44. Content-length: 849
45.
46. <?xml version="1.0" encoding="UTF-8"?>
47. <MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:0.9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.com:MTConnectStreams:0.9"
xsi:schemaLocation="urn:mtconnect.com:MTConnectStreams:0.9
/schemas/MTConnectStreams.xsd">
48. <Header version="0.9" firstSequence="0" lastSequence="24"
sender="localhost" creationTime="2008-12-01T13:35:34-08:00"
bufferSize="100000" instanceId="1228167061" nextSequence="25"/>
49. <Streams>
50. <DeviceStream name="LinuxCNC" uuid="linux-01">
51.   <ComponentStream name="power" component="Power" componentId="2">
52.     <Events>
53.       <PowerStatus dataItemId="15" sequence="24" name="power"
timestamp="2008-08-14T20:13:19.153473">ON</PowerStatus>
54.     </Events>
55.   </ComponentStream>
56. </DeviceStream>
57. </Streams>
58. </MTConnectStreams>
```

Lines 42-58: 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.

```
59. --8a89b9e00b810f6de5901cc0014d706d
60. Content-type: text/xml
61. Content-length: 596
62.
63. <?xml version="1.0" encoding="UTF-8"?>
64. <MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:0.9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.com:MTConnectStreams:0.9"
```

```
xsi:schemaLocation="urn:mtconnect.com:MTConnectStreams:0.9
/schemas/MTConnectStreams.xsd">
65. <Header version="0.9" firstSequence="0" lastSequence="24"
sender="localhost" creationTime="2008-12-01T13:35:45-08:00"
bufferSize="100000" instanceId="1228167061" nextSequence="25"/>
66. <Streams>
67. <DeviceStream name="LinuxCNC" uuid="linux-01">
68. </DeviceStream>
69. </Streams>
70. </MTConnectStreams>
```

Lines 59-70 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. All update will be handled asynchronously and will not impede the other requests both synchronous and asynchronous.

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
200	OK	The request was handled successfully.
400	Bad Request	The request could not be interpreted.
401	Unauthorized	The application has not provided sufficient credentials to access this application.
403	Forbidden	This server cannot fulfill this request because the client is not authorized and no authorization is possible.
500	Internal Error	There was an internal error in processing the request. This will require technical support to resolve.

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

5.6.1 **Error**

The `MTConnectError` and `Error` element **MUST** be returned if the *Agent* cannot handle the request or the *Agent* is not functioning properly. 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.
`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:

```
1. HTTP/1.1 200 Success
2. Content-Type: text/xml; charset=UTF-8
3. Server: Agent
4. Date: Sun, 23 Dec 2007 21:10:19 GMT
5.
6. <?xml version="1.0" encoding="UTF-8"?>
7. <MTConnectError version="0.1"
       xsi:schemaLocation="urn:mtconnect.com:MTConnect:0.2 mtc.xsd"
       xmlns:mt="urn:mtconnect.com:MTConnect:0.2"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
8.  <Header creationTime="2007-12-06T23:18:57-08:00"
  sender=”MTConnect2.Publish”/>
9.  <Error errorCode=”INVALID_PATH”>The path provided was incorrect:
  //Foos</Error>
10. </MTConnectError>
```

5.7 Protocol Details

When an MTConnect *Agent* collects information from the components, 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 in the *Agent*. Each data item **SHOULD** provide a time-stamp indicating when the information was collected from the component. The *Agent* **MAY** provide a time-stamp of its own, but each sample or event **MUST** have a time-stamp. The time-stamps **MUST** be used to determine the ordering of the messages and **MUST** be 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.

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.

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 persist the data.

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

Agent

Seq	Time	Data Item	Value
101	2007-12-13T10:00:00.0002	Position X	10
102	2007-12-13T10:00:00.0002	Position Y	25
103	2007-12-13T10:00:00.0002	Position Z	1
104	2007-12-13T10:00:00.0002	Spindle Speed	0
105	2007-12-13T10:01:01.0012	Power	ON
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	Power	OFF

Figure 8: Sample Data in an Agent

This is a table of 25 data values and a duration of around 12 seconds. The data captures the power status of the device and the position of its axes: the linear axes X, Y, and Z, and the spindle axis S. The only data items collected in this example are the Position (for the sake of this data, we have the actual position) and the Spindle Speed. We are also collecting the device's power status that can be either `ON` or `OFF`. The device is `OFF` 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 above, the application made a request for a sample starting at sequence #103 and retrieves the next eleven items. The response will include all the samples and events in the mill device from 103 to 113. The `nextSequence` number in the header will tell the application it should begin the next request at 114.

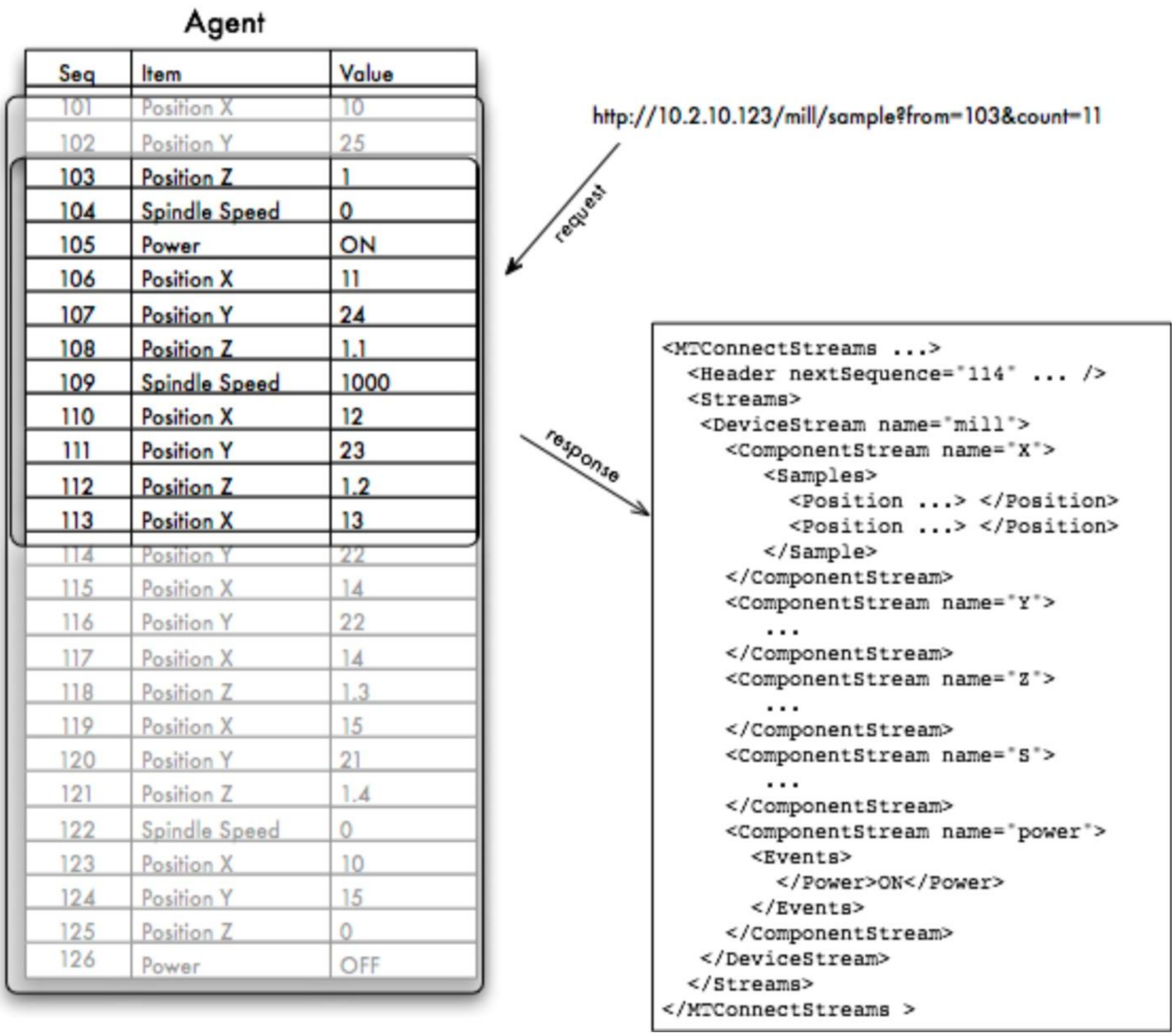

Figure 9: Example #1 for Sample from Sequence #103

In the following illustration, the next request starts at 114 and gets the next ten samples. The response will include the X, Y, Z, and spindle samples and since there are no `Power` events, this component will not be included:

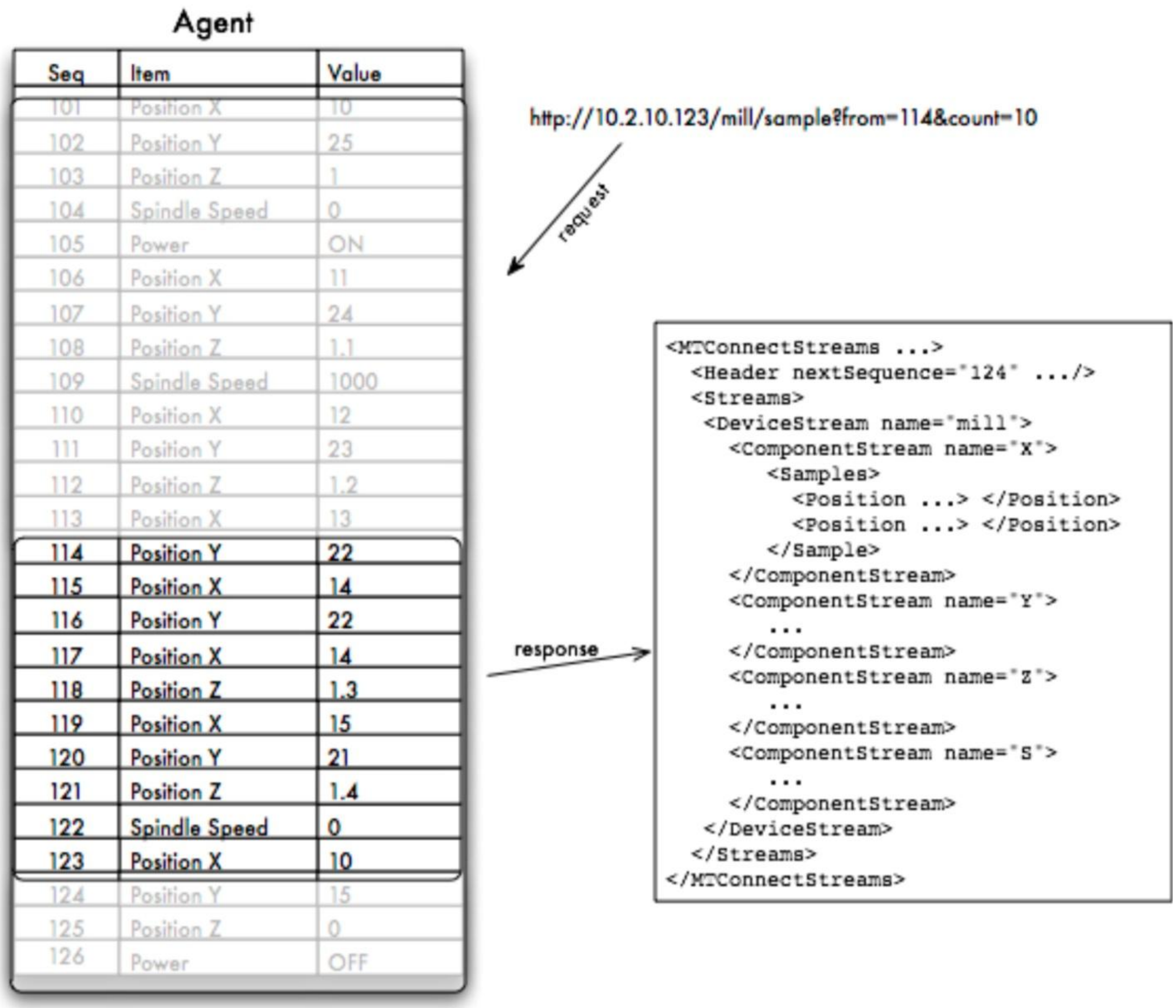

Figure 10: Example #1 for Sample from Sequence #114

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 124.

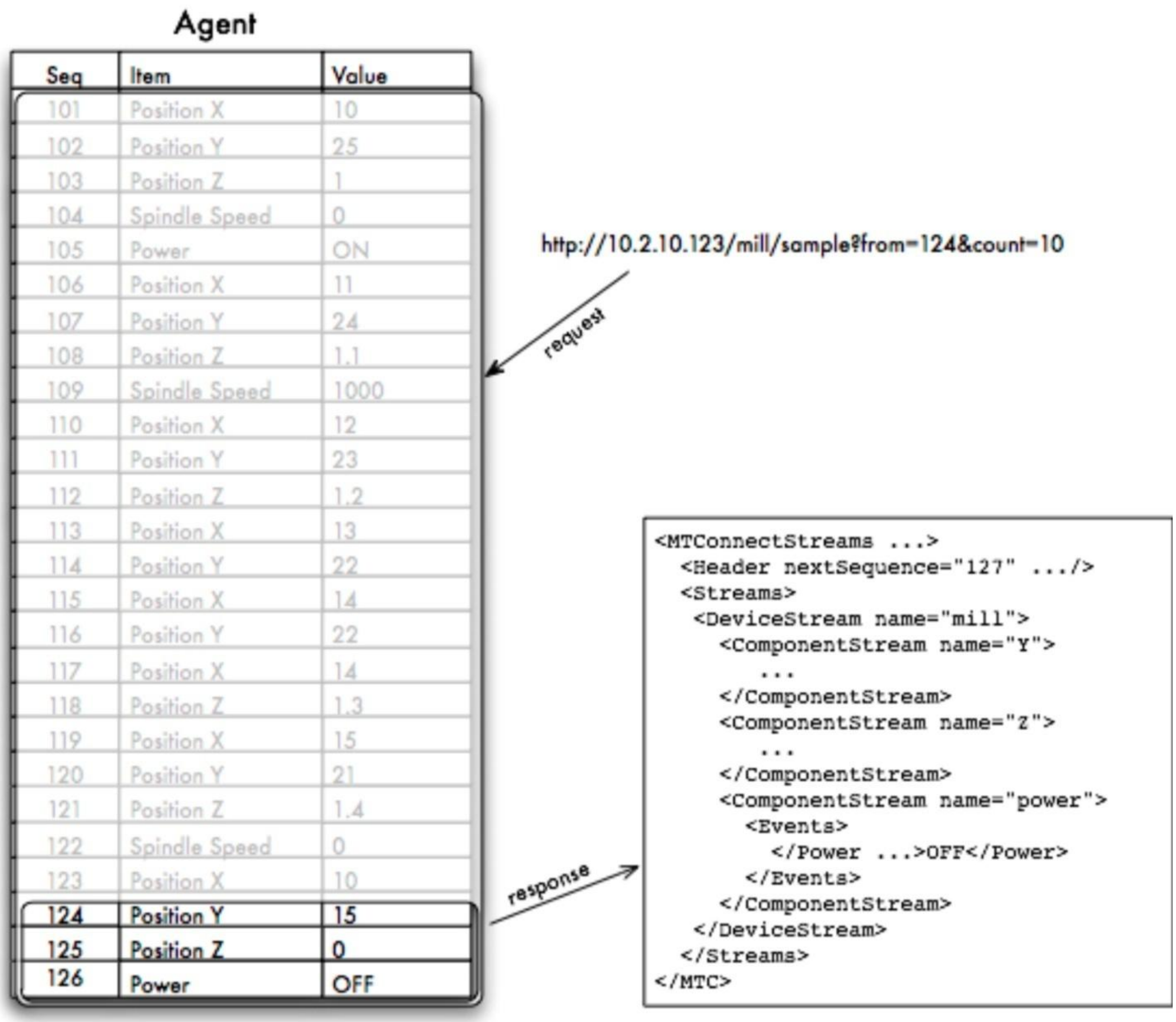

Figure 11: Example #1 for Sample from Sequence #124

In the above illustration, there are only three items available. The first two are axis samples and the third is a power event. The next sequence will indicate that the application must request samples and events 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 and samples, it **MUST** always be one greater than the maximum sequence number assigned.

5.9 Request with Path Parameter

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

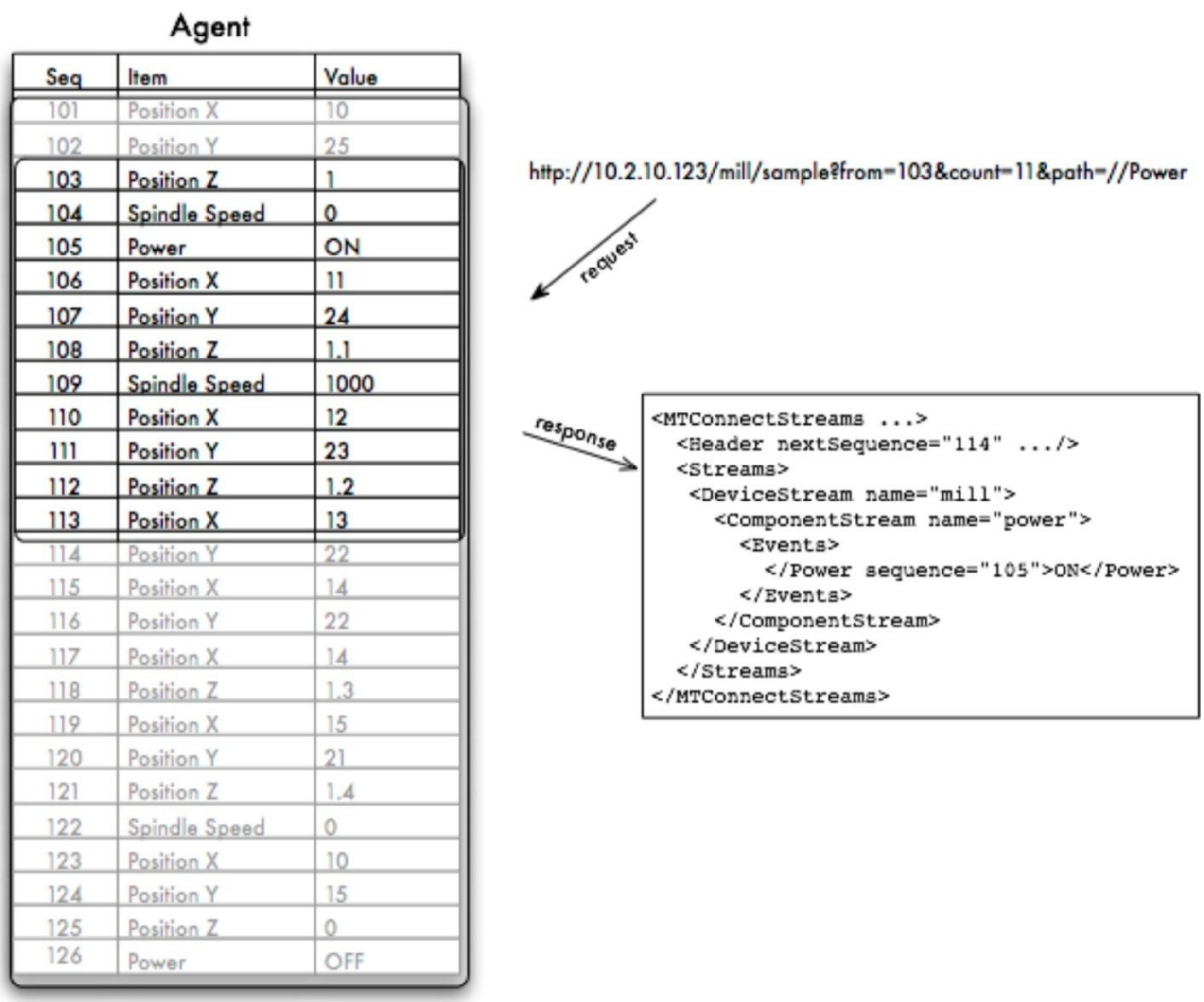

Figure 12: Example #2 for Sample from Sequence #103 with Path

Figure 12 shows that when events are filtered for only the `Power` component, the `Power ON` event will be delivered and nothing else. The `Power ON` event is sequence number 105, but since the other samples and events are considered, the next sequence number is still 114. 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 NOT** consider only the events and samples delivered to the application when computing the next sequence number.

In the next illustration the request is sent as before but now only including Power components:

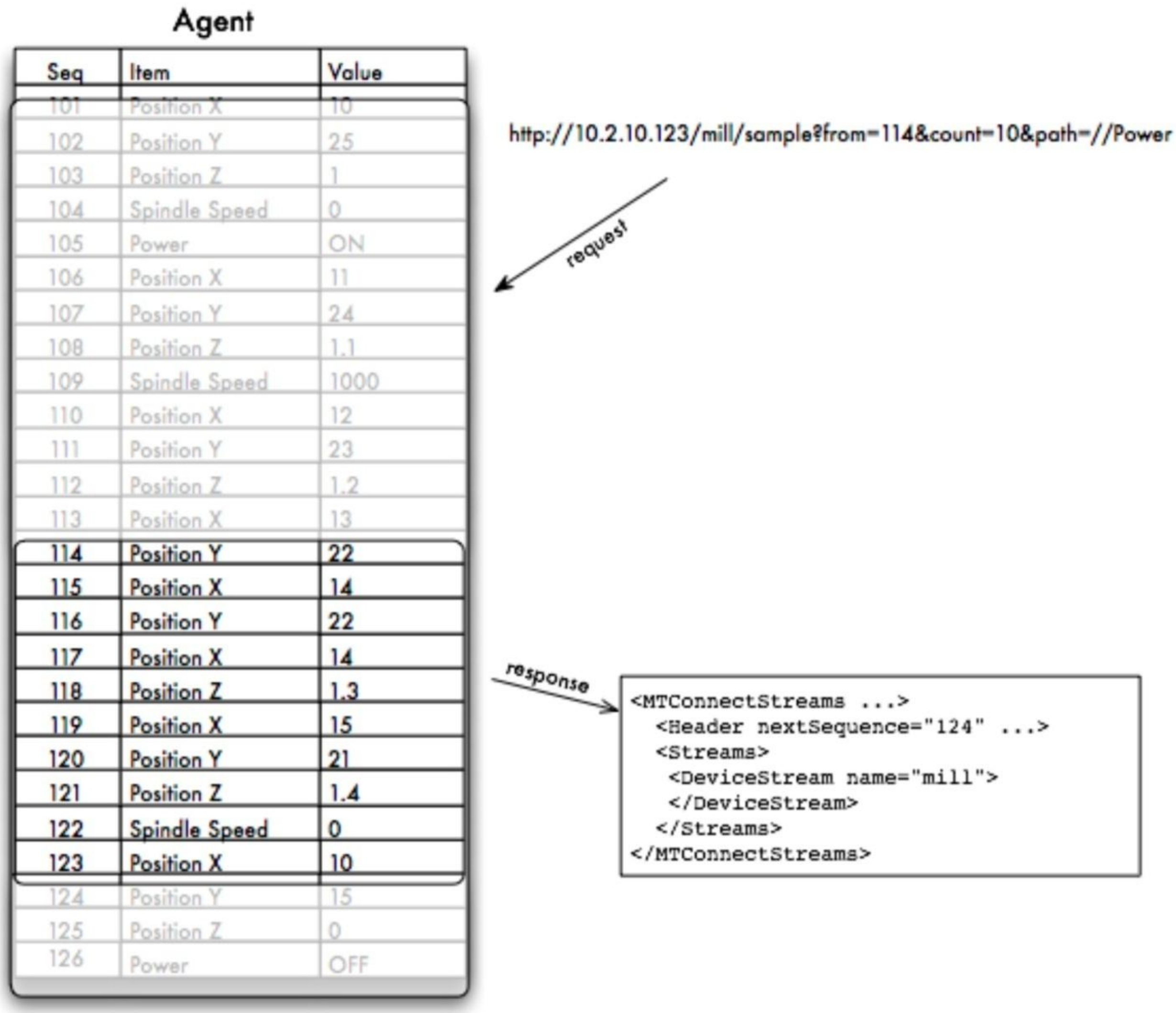

Figure 13: Example #2 for Sample from Sequence #114 with Path

Since there are no Power events in this group of samples and events, an empty element representing the device **MUST** be returned to indicate that the request was valid and no data was found. To further illustrate the `nextSequence` handling, one will notice that `nextSequence` is set to 124 even though no results were returned. If this was not done, the application would continue to scan at 124 and may never move on.

To continue this example, the last request will start at 124 as before and will now request only Power components:

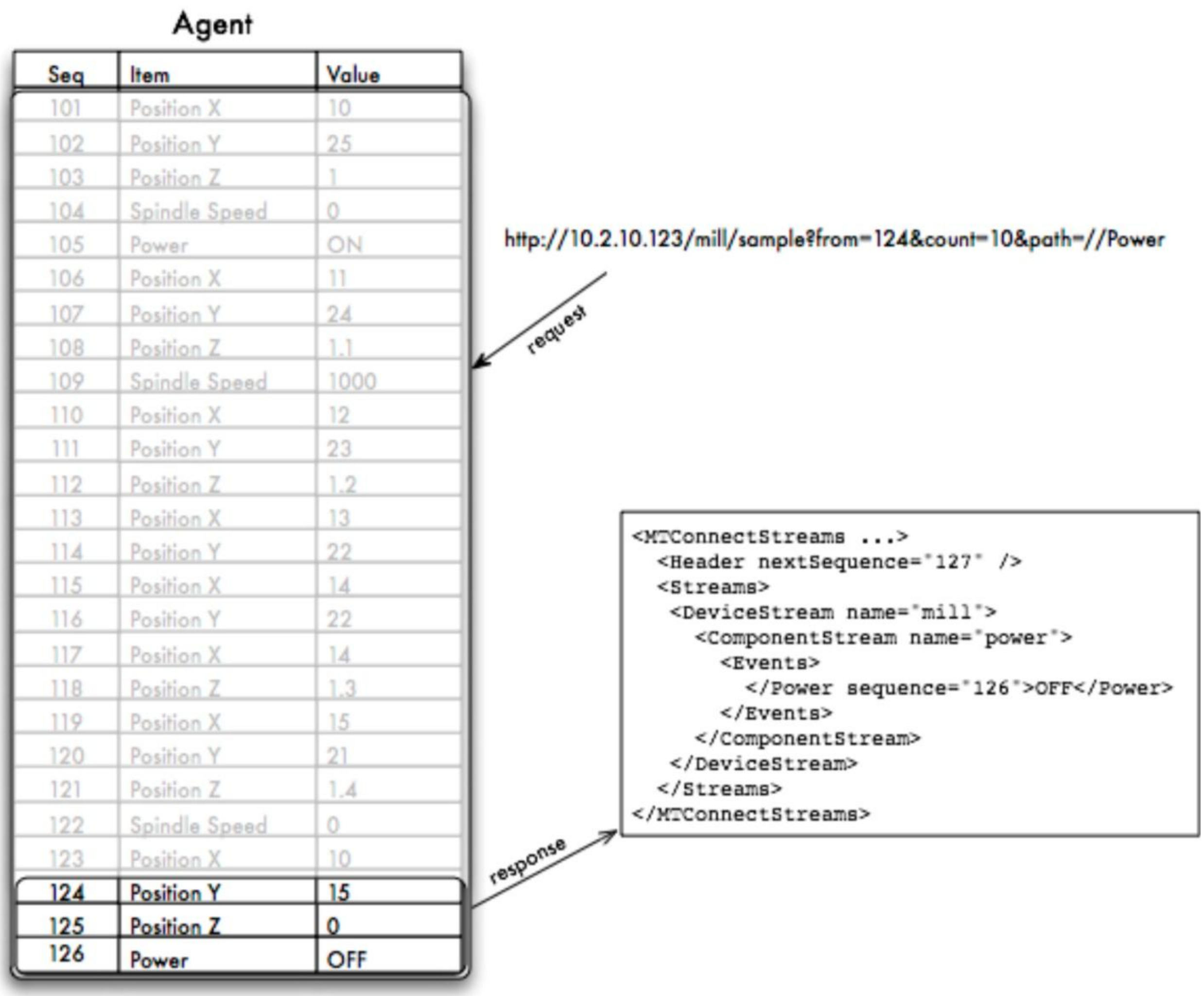

Figure 14: Example #2 for Sample from Sequence #124 with Path

As can be seen, the one Power 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 or samples 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

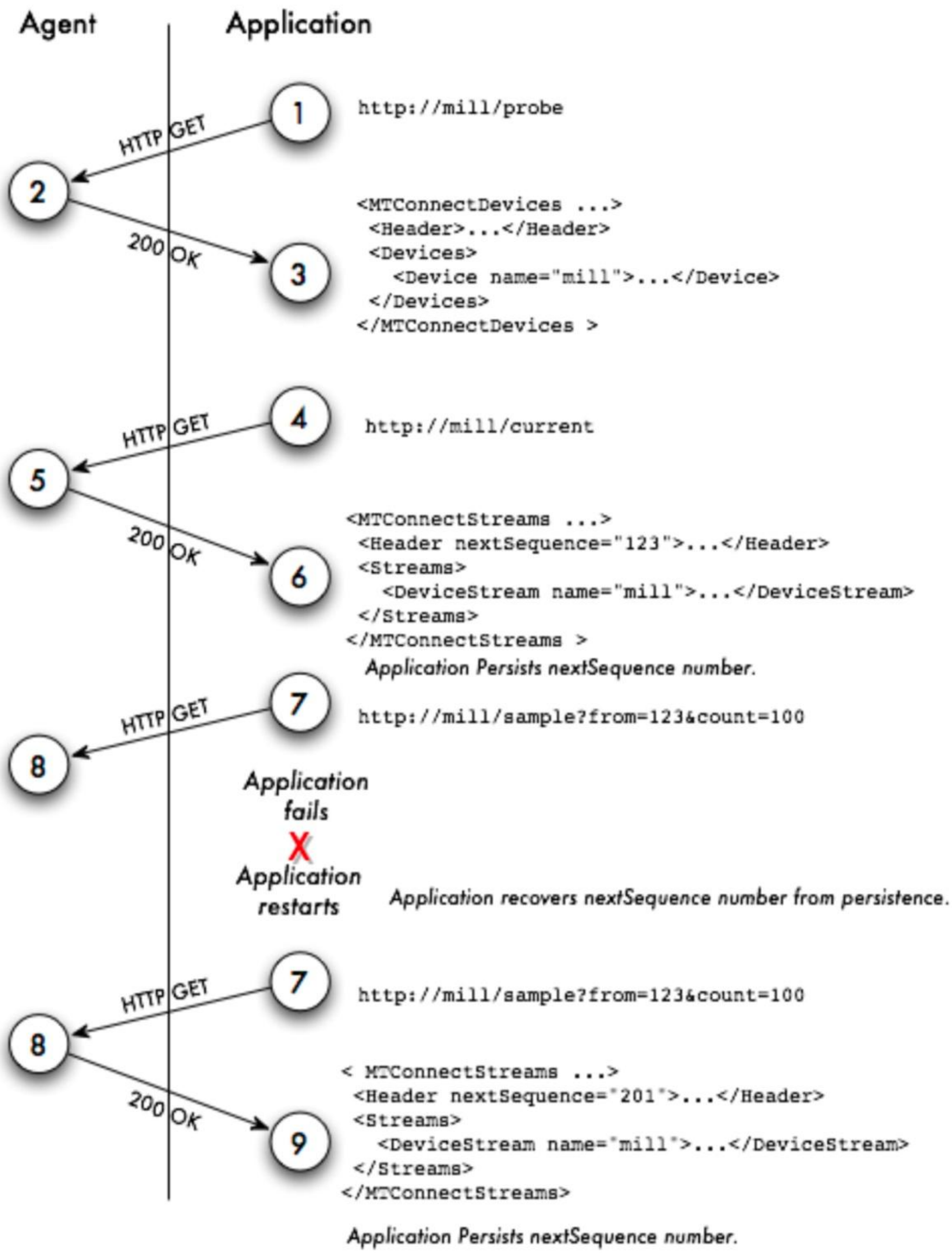

Figure 15: Application Failure and Recovery

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.

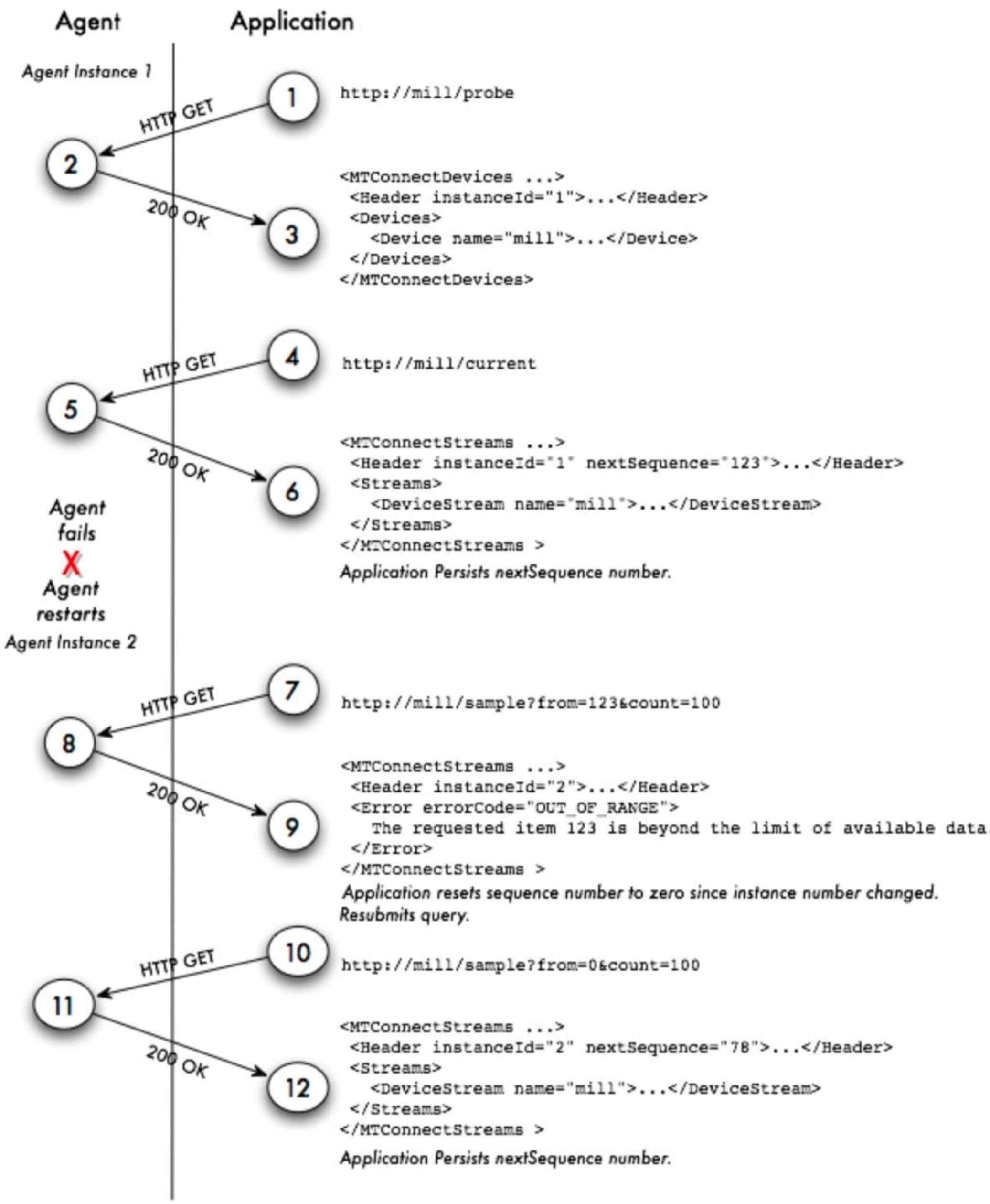

Figure 16: Agent Failure and Recovery

In the above example, the `instanceId` is increased from 1 to 2 indicating that there was a discontinuity in the sequence numbers. When the application detects the change in `instanceId`, it **MUST** reset its next sequence number and retry its request from sequence number 0. 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 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 **MAY** be used for identification of the MTConnect *Agent* in the `instanceId`.

6 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.*