MTConnect Draft Standard

Version 0.9.10

Initial draft for review
Prepared for: AMT
Prepared by: William Sobel et al.
Date: May 15, 2008

Table of Contents

Revision History

Date	Description	Author	Version
12/12/07	Added component and stream information. Incorporated changes suggested by Andrew Dugenske	Will Sobel	Prerelease
12/13/07	Incorporated more feedback and clarified overall document structure.	Will Sobel	Prerelease
12/19/07	Added protocol and fault tolerance sections to specification	Will Sobel	Prerelease
12/20/07	Changed from specification to standard. Added instance id for fault tolerance.	Will Sobel	Prerelease
12/21/07	Incorporated Paul W. comments into standard.	Will Sobel	Prerelease
12/22/07	Added fault tolerance section and cleaned up protocol section.	Will Sobel	Prerelease
12/23/07	Added error codes.	Will Sobel	Prerelease
12/23/07	Initial release version	Will Sobel	0.9.1
12/27/07	Integrated comments from P. Warndorf	Will Sobel	0.9.2
12/29/07	Added sample frequency to component. Many edits. Integrated comments from P. Wicks	Will Sobel	0.9.3b
12/30/07	Integrated comments and suggestions from Rosie Liebe.	Will Sobel	0.9.4
1/1/08	Integrated comments from P. Warndorf, P. Wicks, Rosie Liebe	Will Sobel	0.9.4a
1/2/08	Added physical architecture	Will Sobel	0.9.4b
1/7/08	Integrated comments from P Wicks and P Warndorf. Added future directions and embedded architecture.	Will Sobel	0.9.5
1/9/08	Integrated comments from Andrew Dugenske	Will Sobel	0.9.5a
1/10/08	Changes from review meeting.	Will Sobel	0.9.5b
1/20/08	Integrated comments from Scott Hibbard and Rosie Liebe. Final draft version. Added bibliography.	Will Sobel	0.9.6
3/28/08	Incorporated feedback from P. Warndorf	Will Sobel	0.9.7
4/4/08	Incorporated new schema changed by A.Dugenske. Added controller mode to appendix A.	Will Sobel	0.9.8
4/20/08	Added reference to 841 and ASME B5	Will Sobel	0.9.9
5/15/08	Changes and recommendations from A. Dugenske	Will Sobel	0.9.10

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.

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.
- Sections 6, 7, and 8 give the structure for the device descriptions and construction.
- Section 9 provides the format for data retrieval from the *Agent*, referred to as streams.
- Section 10 gives examples of various device configurations and data retrieved from an *Agent*.
- Section 11 discusses the future direction of the standard.

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.
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 for continuous data items, that is, the value of a data item at a point in time.

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.
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 MTConnectStreams element. When MTC is used in the examples, you will 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.

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:

```
1. <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:

```
1. <TopLevel>
2.   <FirstLevel>
3.     <SecondLevel>
4.       <ThirdLevel name="first"></ThirdLevel>
5.       <ThirdLevel name="second"></ThirdLevel>
6.     </SecondLevel>
7.   </FirstLevel>
8. </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 camel case (first letter of each word is capitalized). For example: <ComponentEvents />
2. Attribute names will also be camel 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.s[TZD], for example 2007-09-13T13:01.213415-9. The resolution of the timestamp is determined by the capabilities of the device collecting the data. The TZD portion of the timestamp is optional.
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. The following is a list of current standards being referenced or considered for reference:

1.CAMX (IPC-2501 & IPC-2541)
2.OPC (Note: authors of MTConnect have not received UA specification)
3.OMAC-HMI
4.Various Vendor APIs
5.ASME B5.59.2.v10a (draft)

6.ASME B5.54-2005
7.STEP & STEP NC
8.LinuxCNC
9.ISO 841-2001

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.
Components	The building blocks of the device.
DataItems	The description of the data available from the device.
Streams	A set of samples or events for components and devices.
Samples	A point-in-time measurement of a data item that is continuously changing.
Events	Unexpected or discrete occurrence in a component. This includes state changes and alarms.
Alarms	A type of event that indicates an abnormal behavior.

Each of these parts will be covered in detail in the following sections.

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.

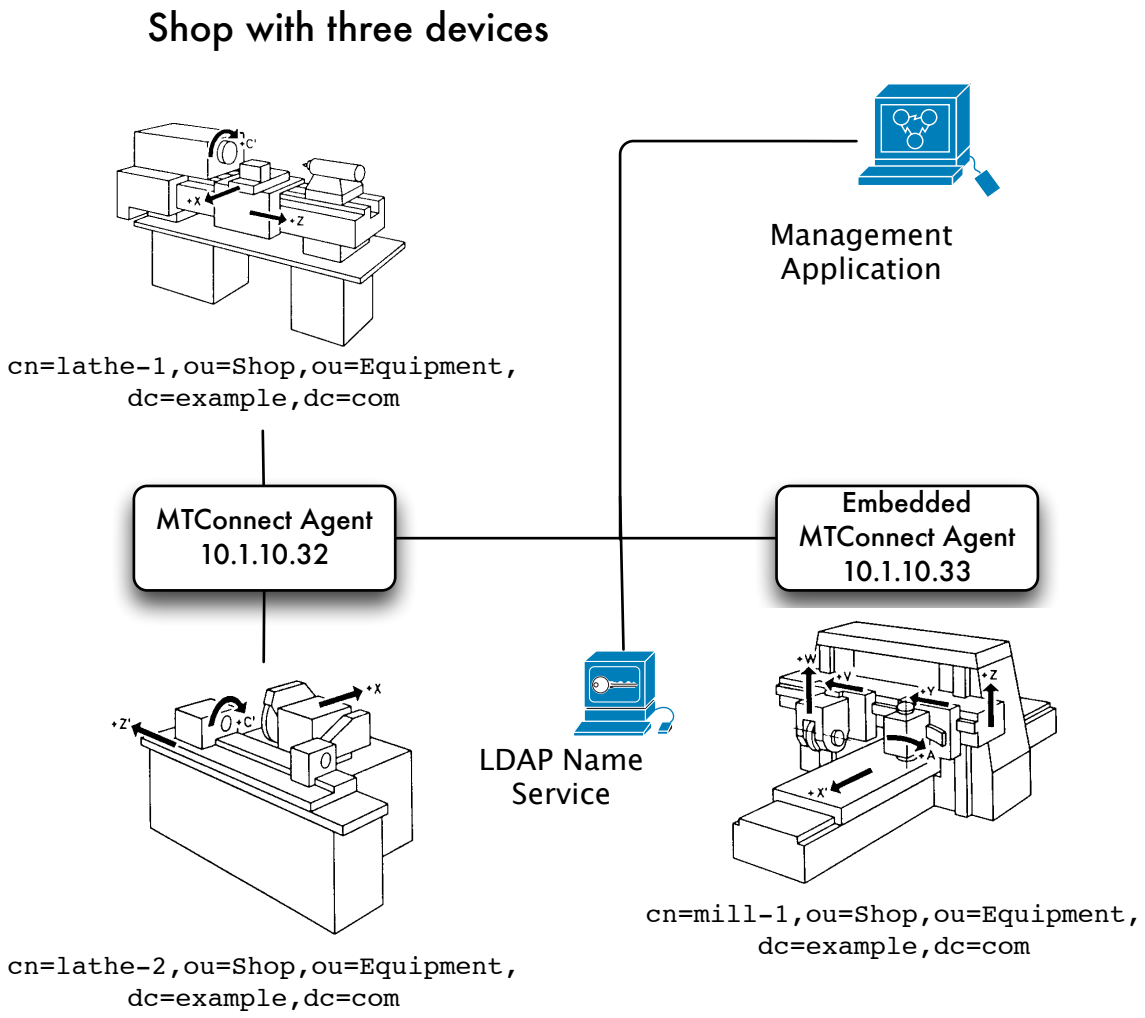
If discovery is not provided or used, the application **MUST** know the URI for the device's *Agent* and address it directly.

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 third

Agent is embedded in the controller of the mill. The management application is monitoring all three pieces of equipment.

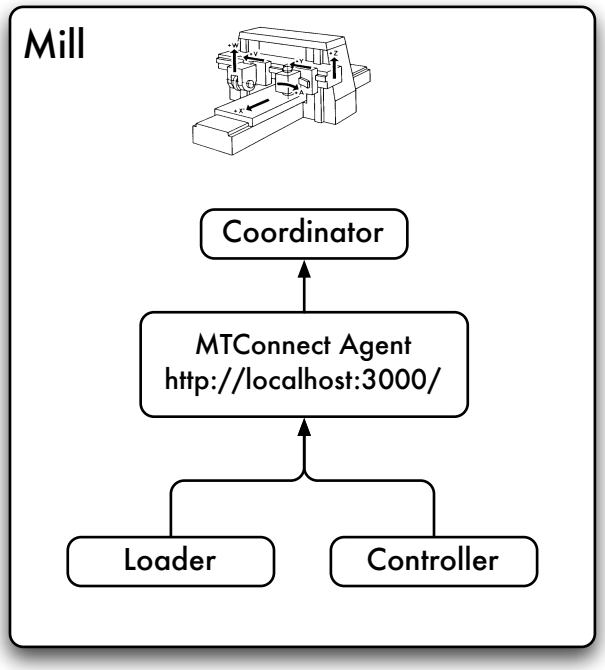


One can look up the three devices using the *Name Service*. The application would search for all devices in the Equipment organization unit (`ou=Equipment,dc=example,dc=com`). The application would get back three device names: `lathe-1`, `lathe-2`, and `mill-1`. These would be have the following URIs: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



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

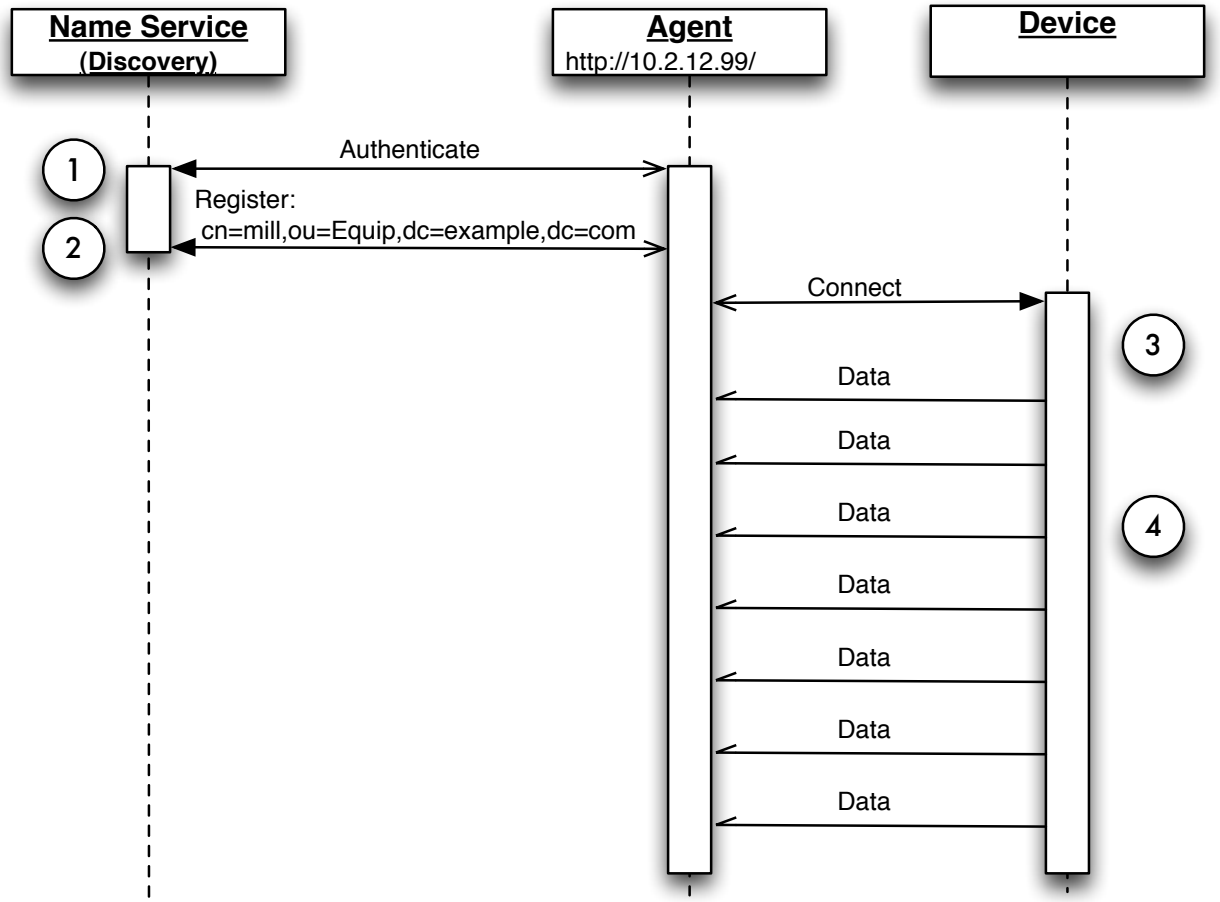
An MTConnect request **SHOULD** not include any body in the HTTP request. If the *Agent* receives any additional data, the *Agent* **MAY** ignore it. There will be no cookies or additional information considered; the only information the *Agent* **MUST** consider is the URI in the HTTP GET (Type a URI 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



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.

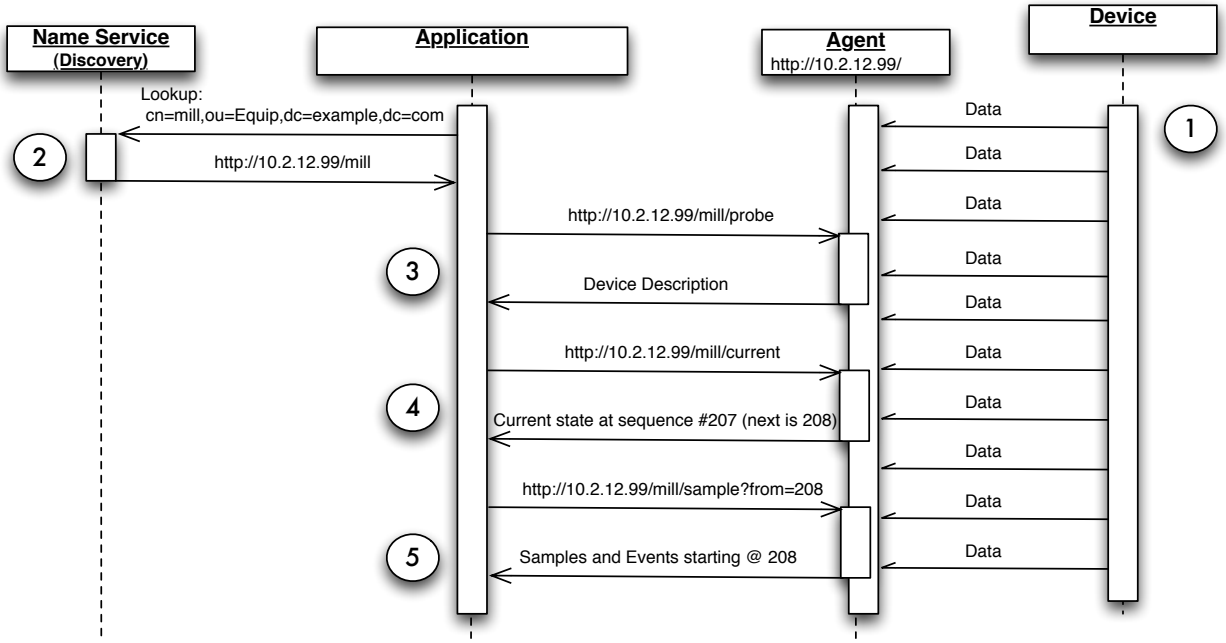
Step 1 The Agent connects and authenticates itself with the Name Service (LDAP server).

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



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.

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`. These 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. <mt:MTConnectDevices version="0.9">
2.   <Header> … </Header>
3.   <Devices> … </Devices>
4. </mt: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 section 4 for `Header` and section 6 for `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 `Devices` section. An `MTConnectStreams` XML Document will have the following structure (the details have been eliminated for illustrative purposes):

```
1. <mt:MTConnectStreams version="0.9">
2.   <Header> … </Header>
3.   <Streams> … </Streams>
```

```
4. </mt:MTConnectStreams>
```

4.2.1. MTConnectStreams Elements

An MTConnectDevices 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 section 4 for Header and section 10 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. <mt:MTConnectError version="0.9">
2.   <Header> … </Header>
3.   <Errror> … </Error>
4. </mt:MTConnectError>
```

4.3.1. MTConnectError Elements

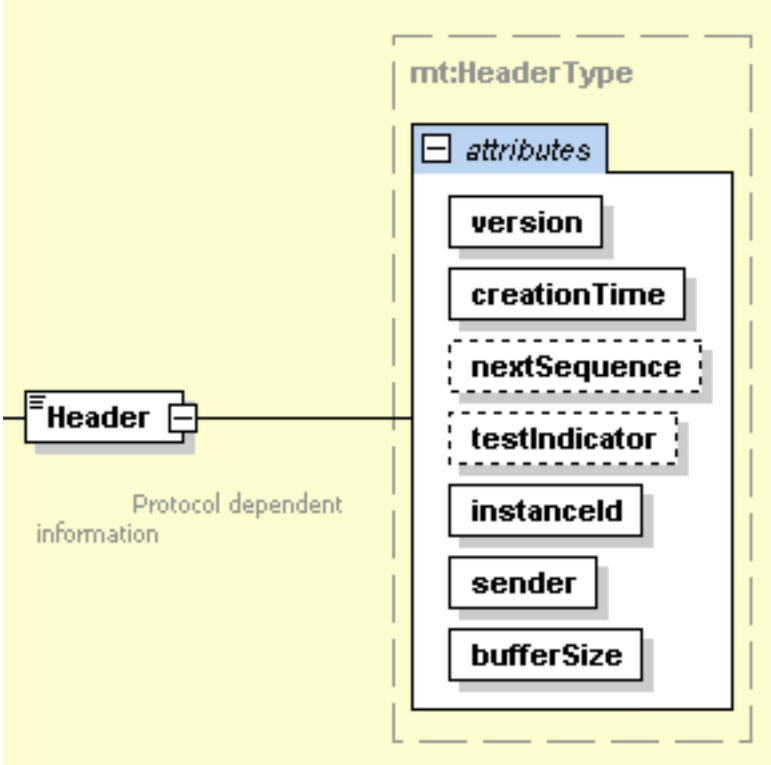
An MTConnect document **MUST** include the Header for all documents and one Error element.

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 for Header, section 6 and section 5.2.4 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:




```
1. <Header instanceId="1" creationTime="2007-12-03T13:23:33">
2.   <Sender>http://10.3.1.10/</Sender>
3.   <BufferSize>10000</BufferSize>
4. </Header>
```

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.	0..1
`instanceId`	A number indicating which invocation of the *Agent*. This is used to differentiate between separate instances of the *Agent*.	1
`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*.	1

The `nextSequence` number **MUST** be provided for `sample` and `current` requests, but it **MUST** be ignored for the `probe` request. 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 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 (section 5.8)*.

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*. 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`.

5.1. Standard Request Sequence

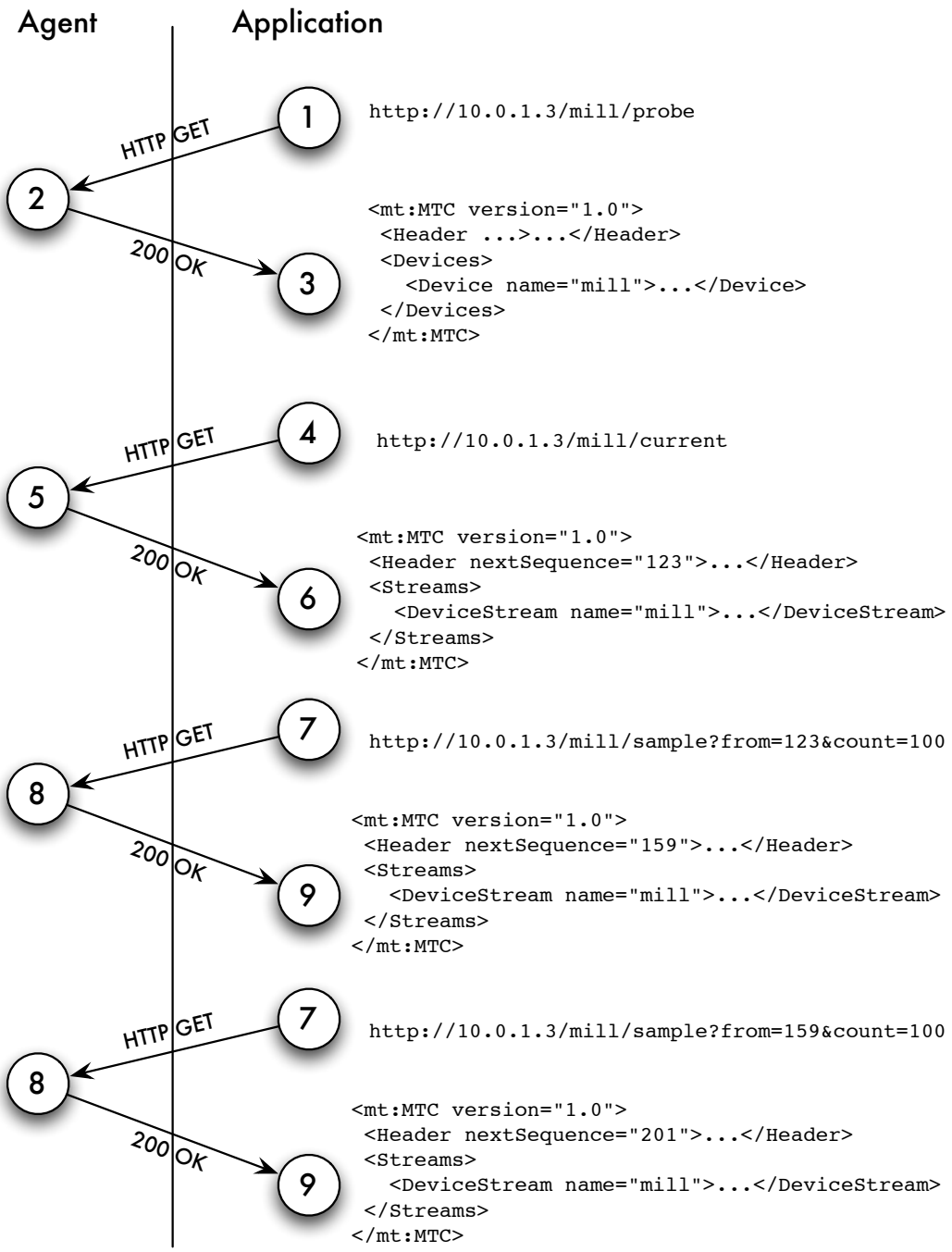
MTConnect *Agent* **MUST** support three types of requests:

- `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 confirms that the necessary data items are available from the *Agent*, it can issue a `current` request to acquire the current values of all the components' data items and the next sequence number on the event queue 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 5.8) 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.



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 mill-1:

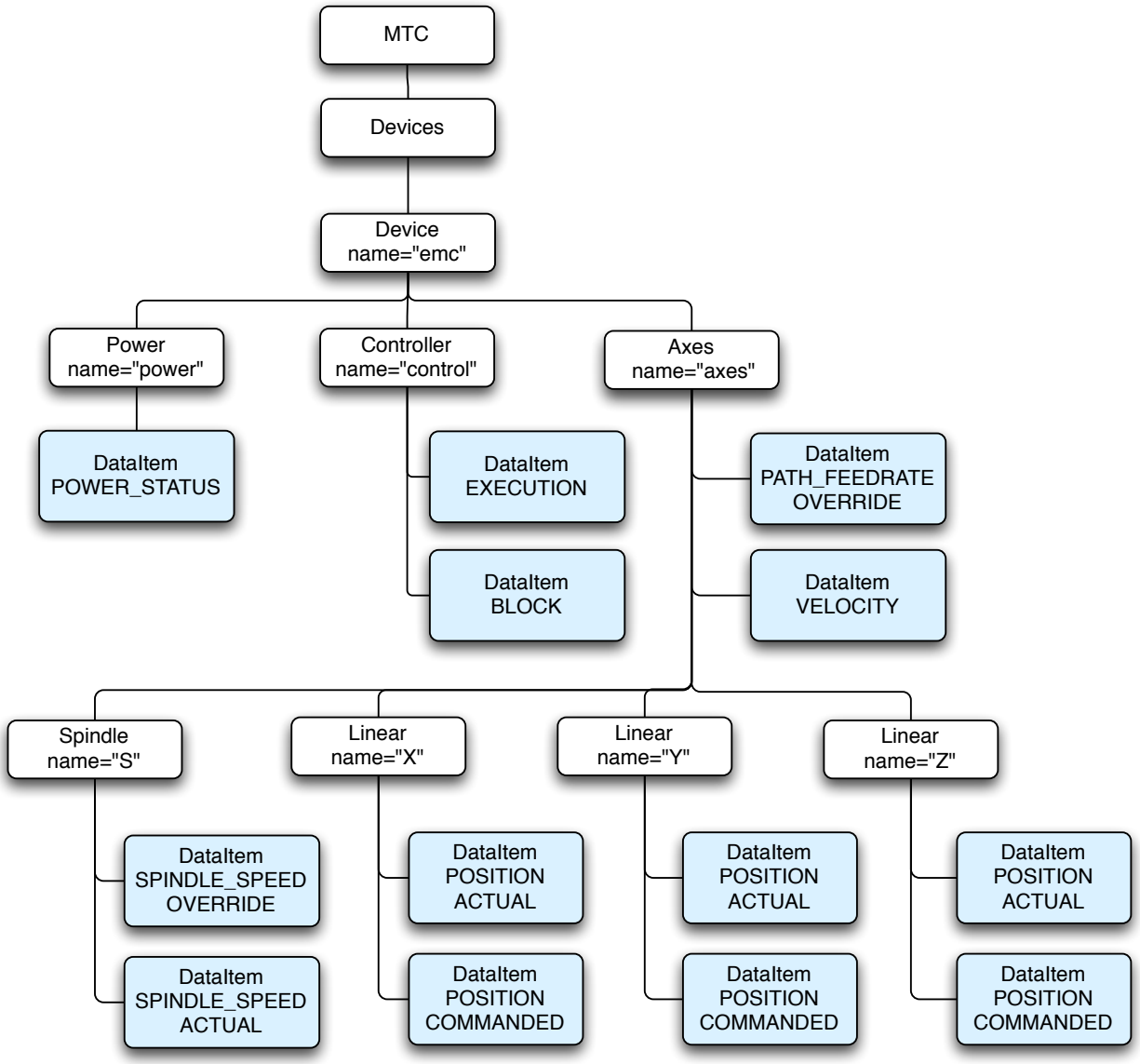
```
<mt:MTConnectDevices>
 <Header>...</Header>
 <Devices>
  <Device name="mill-1" uuid="xxx" iso841Class="6">
   <Components>
    <Power name="power">
     <DataItems>
      <DataItem name="power" type="POWER_STATUS" />
     </DataItems>
    </Power>
    <Controller name="control">
     <DataItems>
      <DataItem name="block" type="BLOCK" />
      <DataItem name="execution" type="EXECUTION" />
     </DataItems>
    </Controller>
    <Axes>
     <DataItems>
      <DataItem name="path_feedrateovr" type="PATH_FEEDRATE"
    subType="OVERRIDE" units="PERCENT" />
      <DataItem name="velocity" type="VELOCITY" units="MILLIMETER/
    SECOND" />
     </DataItems>
     <Components>
      <Linear name="X">
```

```
24.       <DataItems>
25.       <DataItem name="Xabs" type="POSITION" subType="ACTUAL"
     units="MILLIMETER" />
26.       <DataItem name="Xcom" type="POSITION" subType="COMMANDED"
     units="MILLIMETER" />
27.      </Linear>
28.      <Linear name="Y">
29.       <DataItems>
30.        <DataItem name="Yabs" type="POSITION" subType="ACTUAL"
     units="MILLIMETER" />
31.        <DataItem name="Ycom" type="POSITION" subType="COMMANDED"
     units="MILLIMETER" />
32.       </DataItems>
33.      </Linear>
34.      <Linear name="Z">
35.       <DataItems>
36.        <DataItem name="Zabs" type="POSITION" subType="ACTUAL"
     units="MILLIMETER" />
37.        <DataItem name="Zcom" type="POSITION" subType="COMMANDED"
     units="MILLIMETER" />
38.       </DataItems>
39.      </Linear>
40.      <Spindle name="S">
41.       <DataItems>
42.        <DataItem name="Srpm" type="SPINDLE_SPEED" subType="ACTUAL"
     units="REVOLUTION/MINUTE" />
43.        <DataItem name="Sovr" type="SPINDLE_SPEED" subType="OVERRIDE"
     units="REVOLUTION/MINUTE" />
44.       </DataItems>
45.      </Linear>
46.     </Components>
47.    </Axes>
48.   </Components>
49.  </Device>
50. </Devices>
51.<mt: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.



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.

`count` - The maximum number of events and samples to send back. The *Agent* **MUST NOT** send back more than this number of events and samples (in aggregate), but fewer events and samples **MAY** be returned.

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 the first event or sample available in the event set. 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 `samples`.

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`.

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

5.5. HTTP Response Codes and `Error`

MTConnect uses the HTTP response codes and XML based upon the `MTConnectError` schema to indicate success and failure. 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.
401	Unauthorized	The application has not provided sufficient credentials to access this application.
403	Forbidden	The application has requested a resource that cannot be accessed.
404	Not Found	The path or the device could not be found on this *Agent*.

[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
500	Internal Error	There was an internal error in processing the request. This will require technical support to resolve.

5.5.1. Error

The Error element is provided in the MTConnectError XML Document request cannot be handled by the *Agent* 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. If the MTConnect status code is not OK (200), 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.

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.
7. <?xml version="1.0" encoding="UTF-8"?>
8. <mt: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">
9.   <Header creationTime="2007-12-06T23:18:57-08:00"
      sender="MTConnect2.Publish"/>
```

```
10.  <Error errorCode="INVALID_PATH">The path provided was incorrect: //
     Foos</Error>
11.</mt:MTConnectError>
```

5.6. 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.

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 necessary 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:

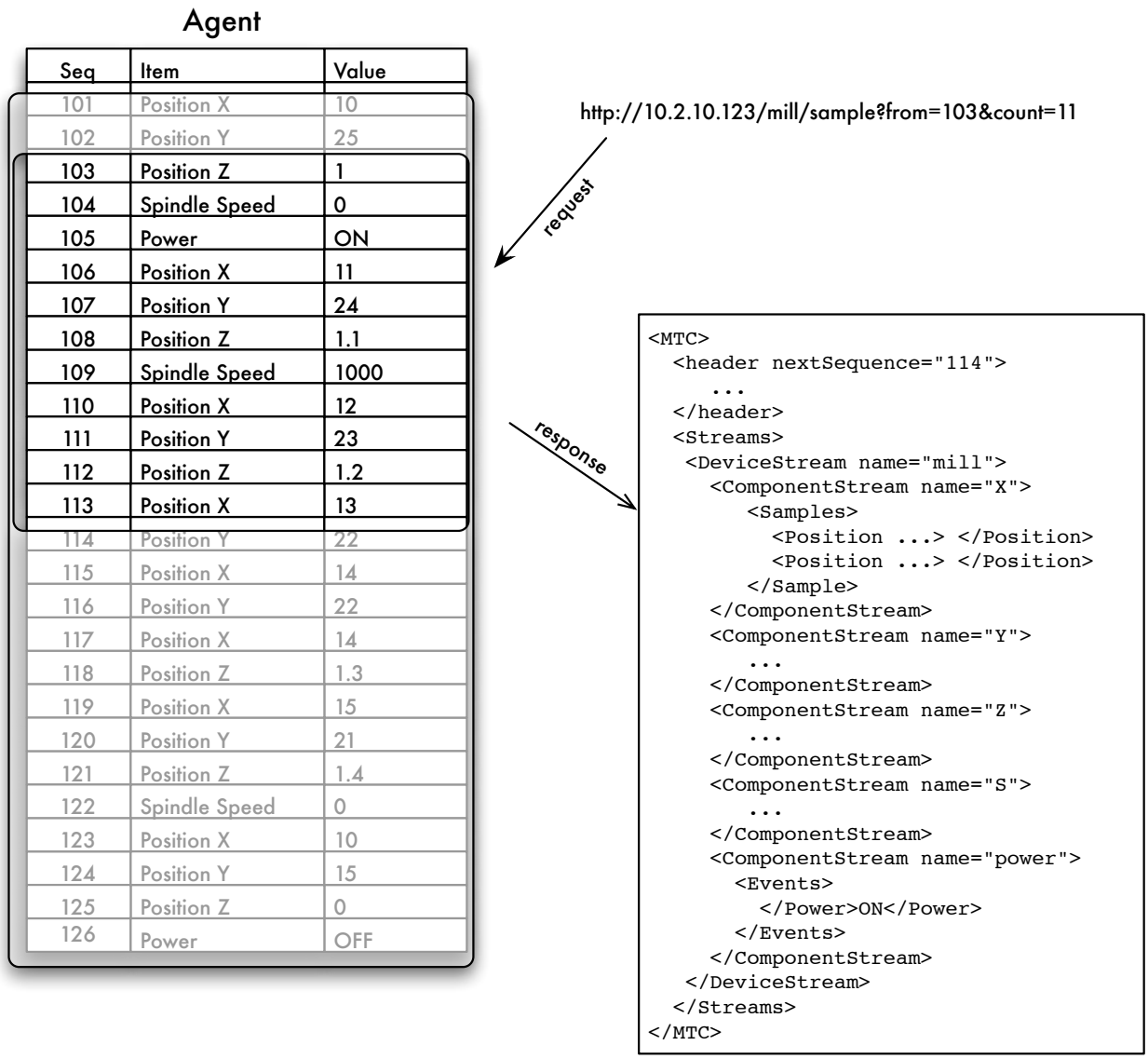
Agent

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

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.

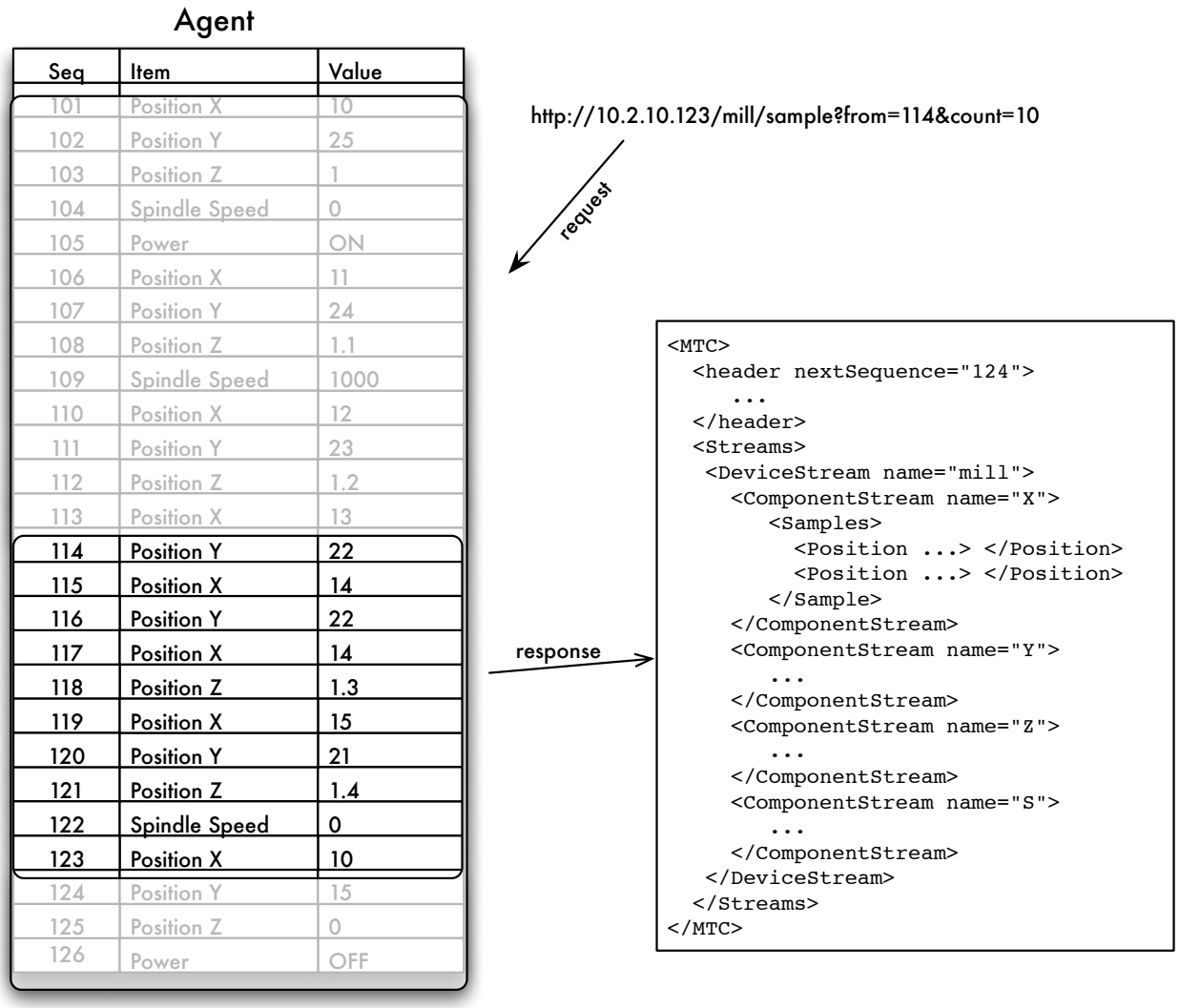
For the remainder of the examples we will be excluding the time column to save space.

5.7. Request without Filtering

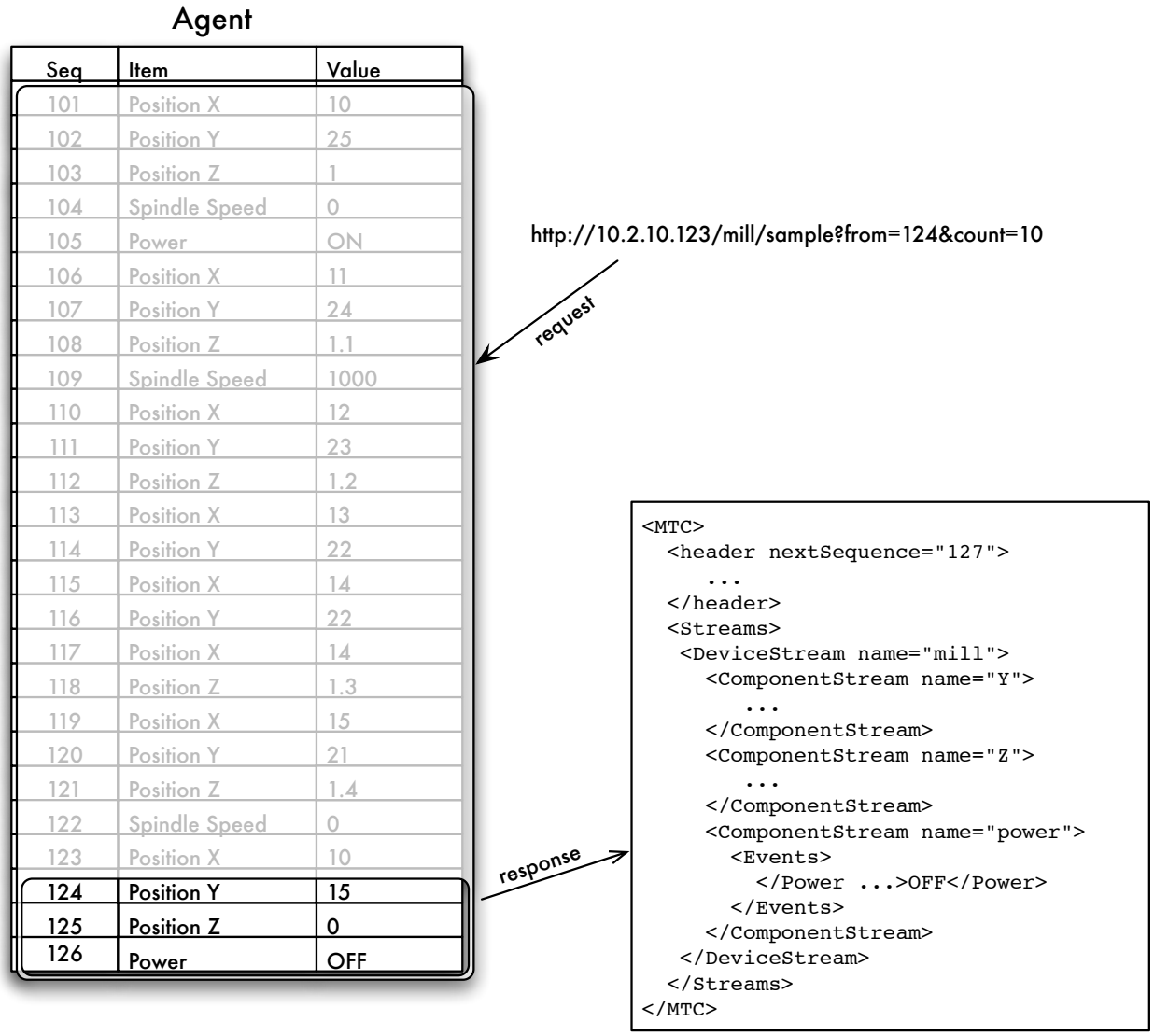


In the example above, the application made a request for a sample starting at sequence #103 and retrieves the next eleven items. The response will include all the samples and events in the mill device from 103 to 113. The `nextSequence` number in the header will tell the application it should begin the next request at 114.

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:



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.



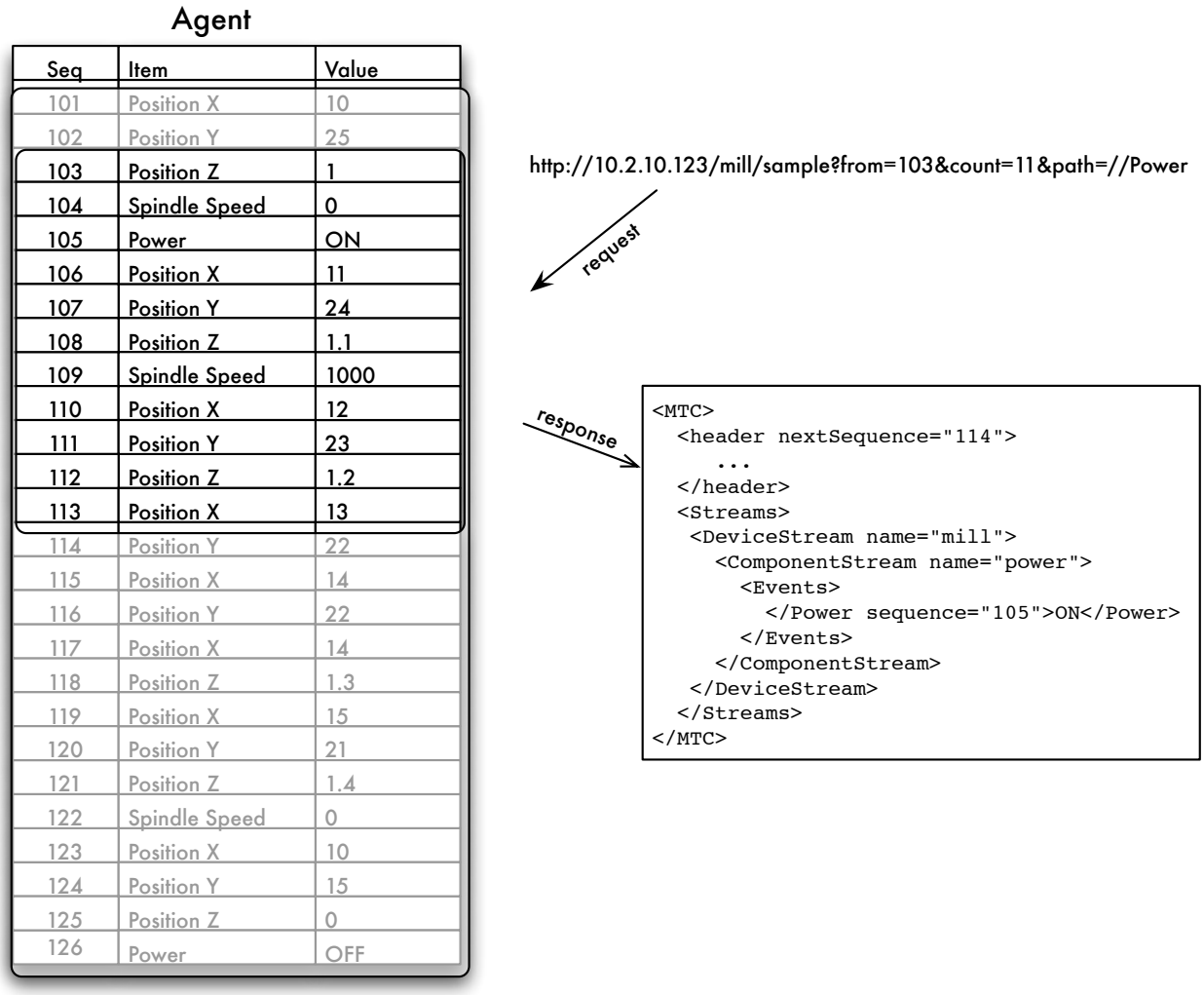
In the above illustration, there are only three items available. The first two are axis samples and the third is a power event. The next sequence will indicate that the application must request samples and events starting at 127 for the next group. If the application were to do this, it would receive 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.

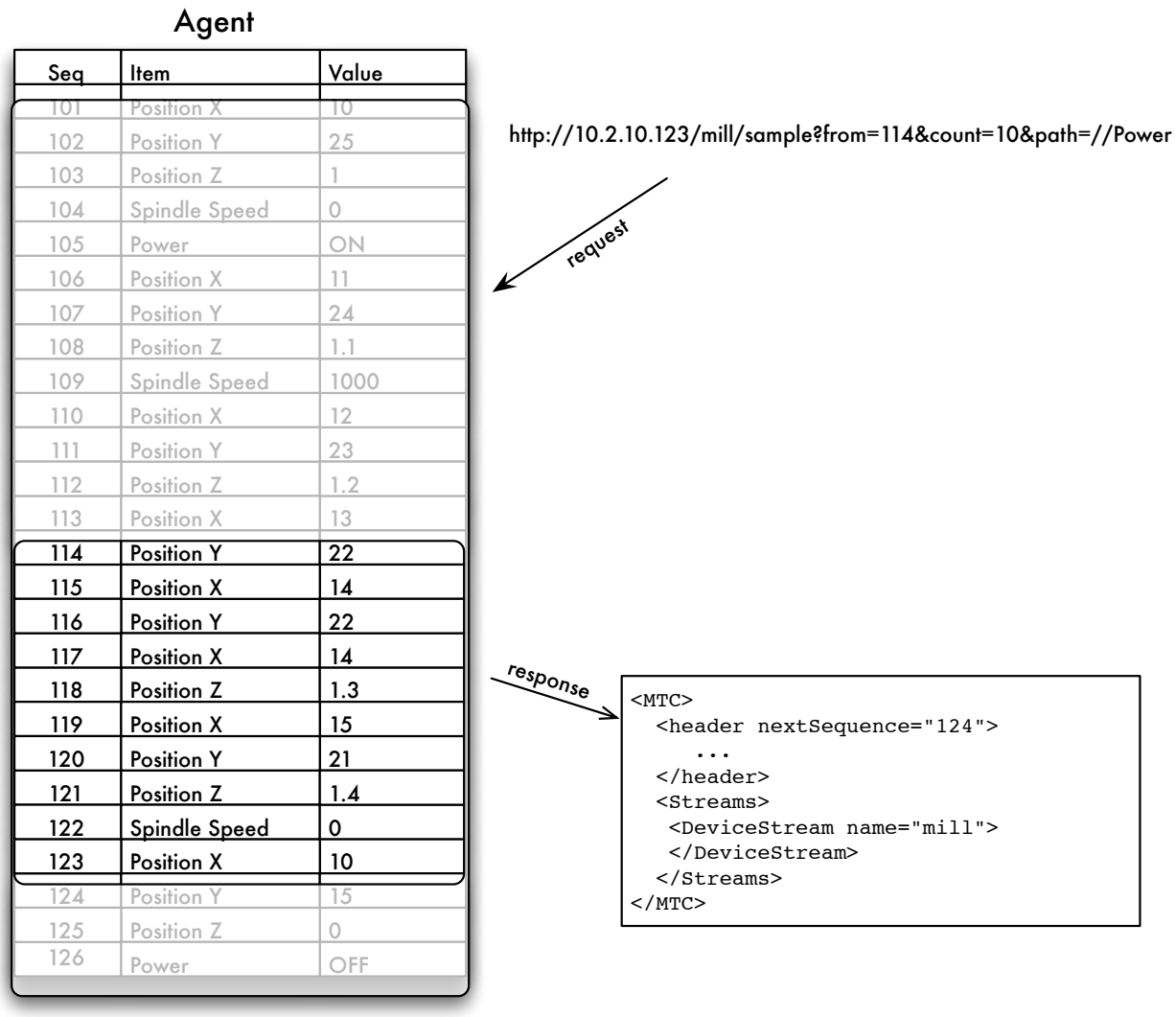
5.8. Request with Path Parameter

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



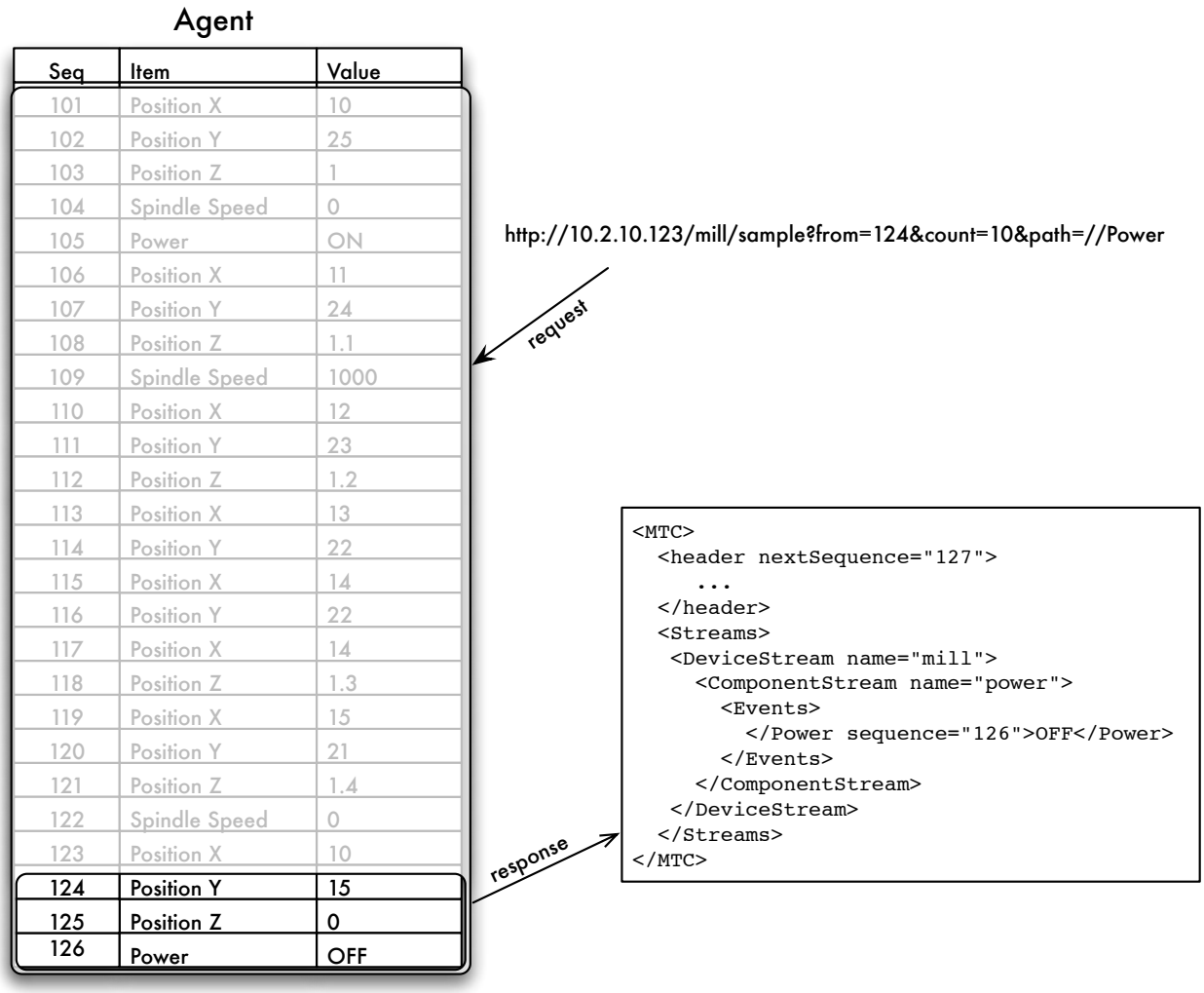
The preceding illustration shows that when events are filtered for only the `Power` component, the `Power ON` event will be delivered and nothing else. The `Power ON` event is sequence number 105, but since the other samples and events are considered, the next sequence number is still 114. The MTConnect *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:



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:



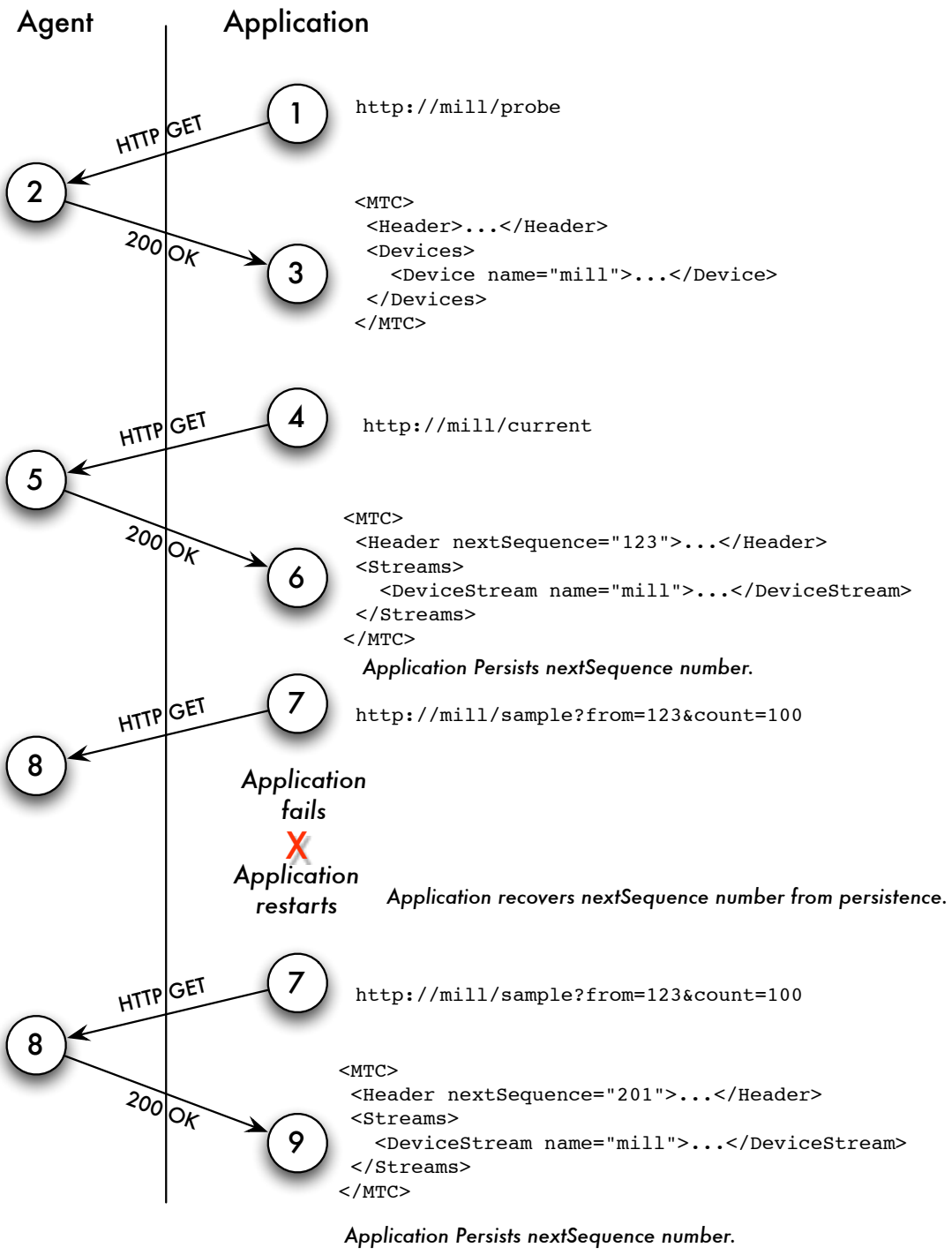
As can be seen, the one Power event is returned and the next sequence is now 127. This will indicate that the application must request from 127 on for the next set of events. If no events are available, the `nextSequence` will again be set to 127 and an empty `DeviceStream` will be returned.

5.9. Fault Tolerance and Recovery

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

5.9.1. 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.

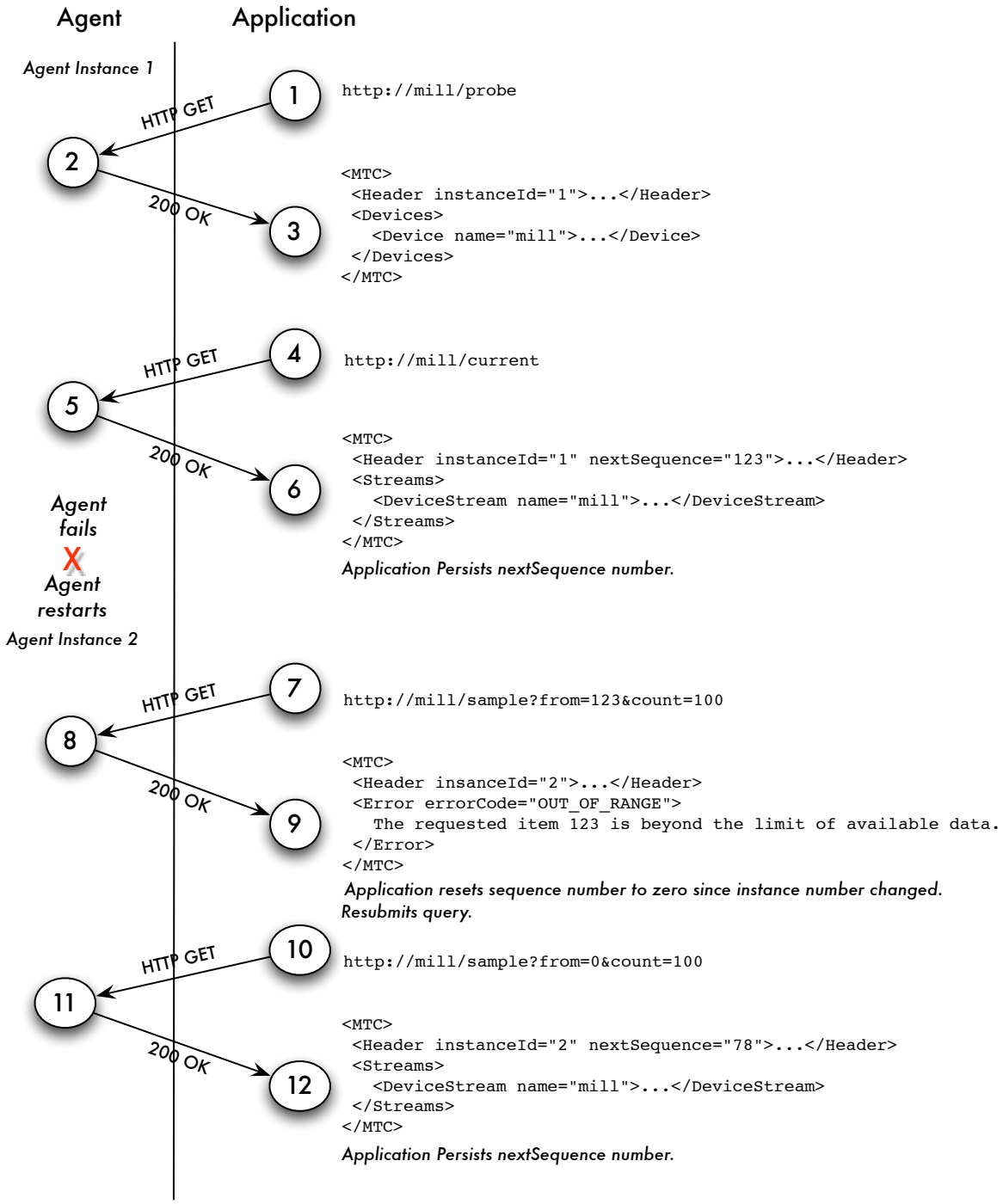


There is the risk of missing some events or samples if the time between requests exceeds the capacity of the *Agent*'s buffer. In this case, there is no record of the missing information and it is lost. If the application automatically restarts after failure, the intervening data can be quickly recovered.

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

5.9.2. 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.



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.

5.9.3. 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. Devices and Components

A device can be thought of as a group of components. For example, the Device is a three axis mill. The mill has components, one of the components is a Power component, often thought of as the main power supply. The mill also has sub-components of the Axes component; these are the three Linear axes and a Spindle. The last component is the Controller, the component that controls the axes and runs the program. These are all sub-components of the Device.

Multiple devices are contained in a top level aggregation element called Devices. The aggregations do not contain any additional attributes.

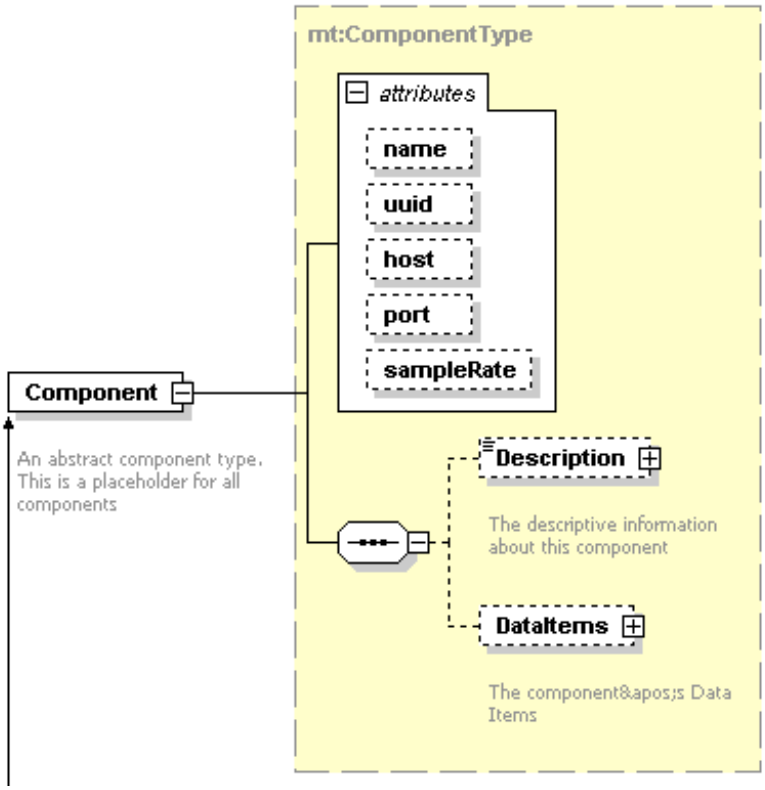
6.1. Devices

Devices is a container for all Devices which is returned from a probe request. The probe response will only return an XML document that is a valid MTConnectDevices document.

Elements	Description	Occurrence
Device	The root of each device. The Device is contained within the top level Devices container. There can be multiple Device elements.	1..INF

6.2. Component

The *Agent* is capable of delivering data associated with each component to the application. The description of these pieces of information is referred to as DataItems and will be discussed in the next section. The actual values for those data items are delivered in Streams and will be discussed in the section on *Streams, Samples, and Events*.



6.3. Component Schema

```
1. <mt:MTConnectDevices version="1.0">
2.   <Header> … </Header>
3.   <Devices>
4.     <Device name="mill-1" uuid="112312321" sampleRate="10" iso841Class="6"
     >
5.       <Components>
6.         <Power name="power">
7.           <DataItems>
8.             <DataItem name="power" type="POWER_STATUS" category="EVENT" />
9.           </DataItems>
10.        </Power>
11.      </Components>
12.    </Device>
13.  </Devices>
14.</mt:MTConnectDevices>
```

6.3.1. Common Component Attributes

An abstract component has the following composition:

Attribute	Description	Occurrence
`uuid`	A unique identifier that will only refer to this component. For example, this can be the manufacturer code and the serial number.	0..1*
`name`	The name of the component. This name should be unique within the machine to allow for easier data integration.	1
`sampleRate`	The rate in milliseconds that data is obtained from the component. This is the number of milliseconds between data captures, can be fractional.	0..1**

Notes: * The `uuid` **MUST** be provided for the `Device`, it is optional for all other elements.
** If this component is receiving data, it **MUST** supply a `sampleRate` to aid the application in interpolating values. This is the desired sample rate and may vary depending on the capabilities of the device.

6.3.2. Component Elements

Element	Description	Occurrence
`Description`	An element that can contain any descriptive content. This can contain configuration information and manufacturer specific details.	0..1
`Components`	Sub-components of this component.	0..1*
`DataItems`	The data items this component provides. The data items are descriptions of the data events for reporting.	0..1*

Notes: * One of either `Components` or `DataItems` **MUST** be provided. Both **MAY** be provided.

6.3.2.1. `Description`

Attribute	Description	Occurrence
`Manufacturer`	The name of the manufacturer of the component	0..1
`SerialNumber`	The device's serial number	0..1
`Station`	The station the device is located at. When a device is part of a manufacturing unit or cell with multiple stations that share the same physical controller.	0..1

The CDATA of the Description is any additional descriptive information the implementor chooses to include regarding the component.

6.3.2.2. `Components`

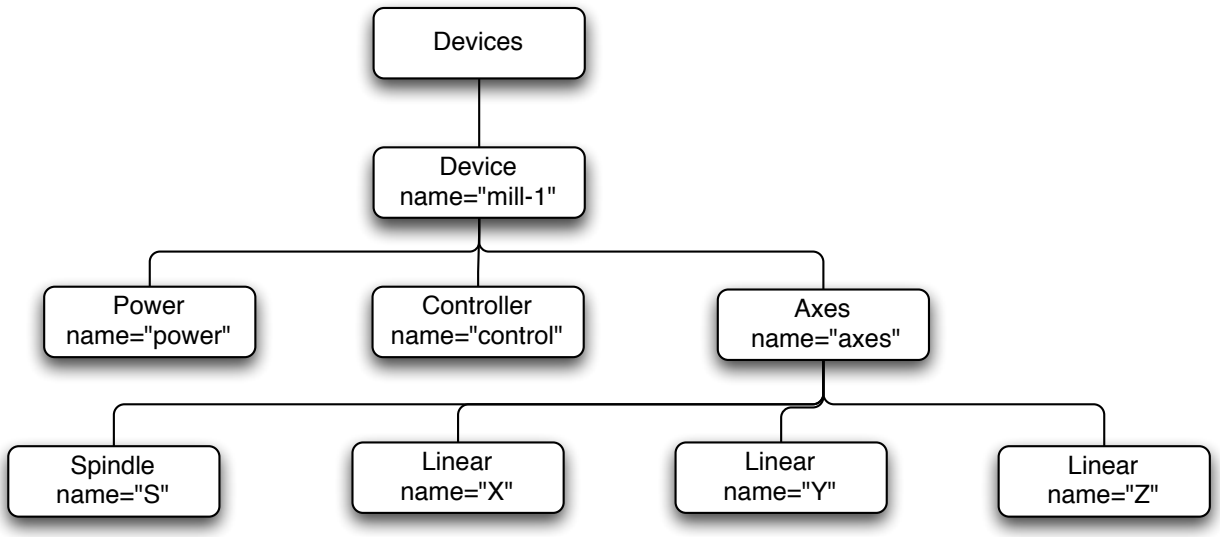
Element	Description	Occurrence
`Component`	One or more components. This can also include the subtypes of Component like Axes, Linear, Power, Thermostat, etc...	1..INF

6.3.2.3. DataItems

Element	Description	Occurrence
`DataItem`	Only elements of types DataItem can be specified	1..INF

6.4. Types of Components

For example, this three axis mill is modeled as a device that has a power supply, a controller, three linear axes and one spindle:



All the elements in the above diagram are subtypes of `Component`. A component is an abstract type that allows for extensibility. As the specification progresses, more component types will be added, like `Joint` (for robotics) and `Tool` (for presetter).

6.4.1. `Device`

At the top of the component tree there **MUST** be the root element `Device`. A `Device` is a container that holds all the components associated with this piece of equipment. The `Device` **MUST** have an alarm data item that provides a place for all `Device` general alarms that cannot be assigned to a sub-component.

6.4.1.1. `Device` Attributes

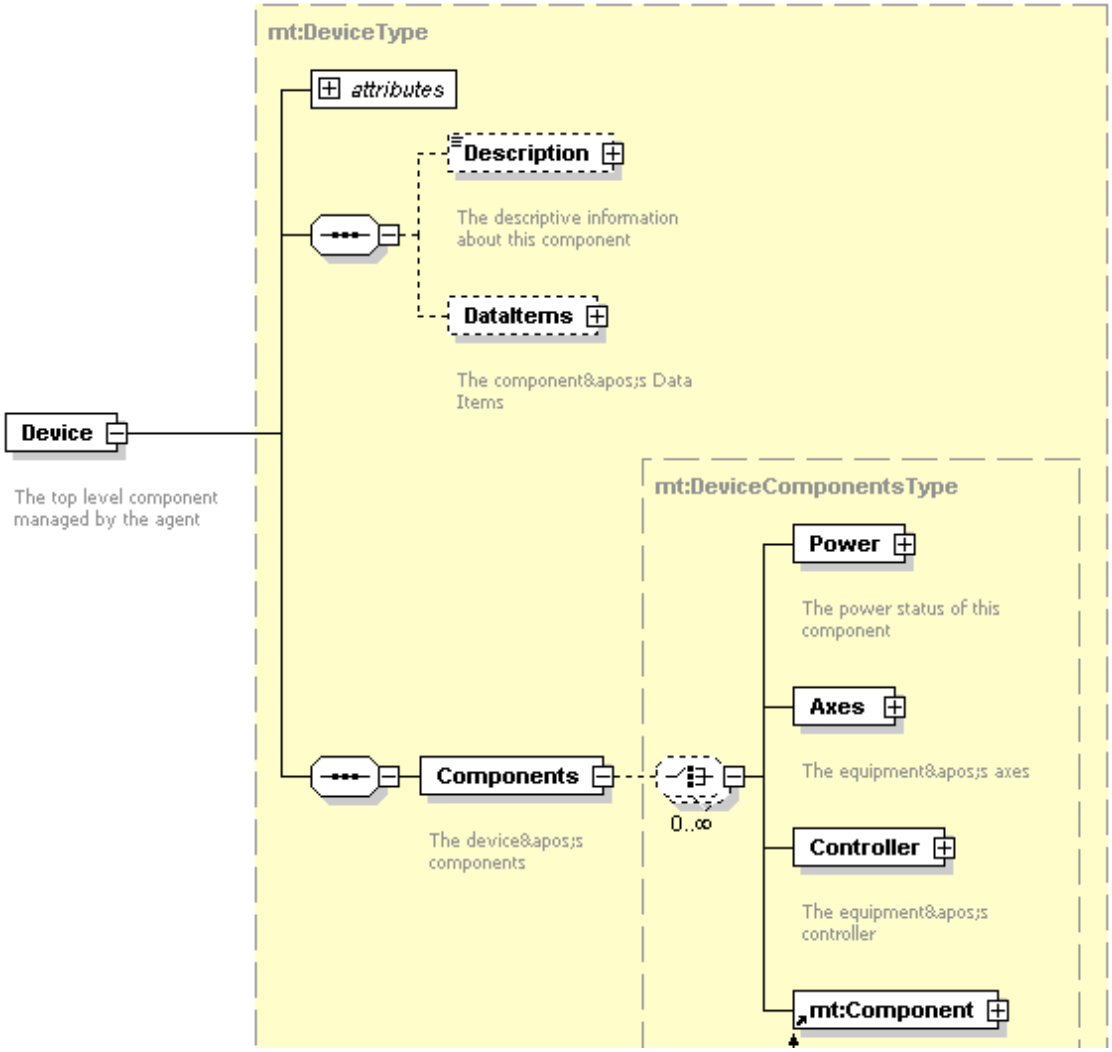
Attribute	Description	Occurrence
`iso841Class`	The ISO 841 classification for the device.	1

A device **MUST** be classified using one of the following identifiers from the ISO 841 specification. The following classification is taken from the appendix of the ISO 841 specification, please use the diagram that best matches the figures in the appendix of ISO 841. If there is no diagram that matches the device, use `iso841Class="1"`. Please provide us with a diagram of your device and its respective components and we attempt to create a new classification for the device.

MTC ISO 841 Classification	Description	Figure
1	Other (Device not included in list)	
2	Parallel lathe (engine lathe)	A.2
3	Twin turret lathe with programmable tailstock	A.3
4	Vertical turning and boring lathe	A.4
5	Milling machine with horizontal spindle	A.5
6	Milling machine with vertical spindle (with W axis)	A.6
7	Boring and milling machine with horizontal spindle	A.7
8	Milling machine with vertical spindle	A.8
9	Portal-type milling machine	A.9
10	Gantry-type milling machine	A.10
11	Planer-type horizontal boring machine	A.11
12	Profile and contouring milling machine with movable table	A.12
13	Profile and contour milling machine with horizontal spindle	A.13
14	Profile and contour milling machine with tilting head	A.14
15	Profile and contour milling machine with tilting table	A.15
16	External cylindrical grinding machine	A.16
17	Tool and cutter grinding machine	A.17
18	Openside planer	A.18
19	Vertical filament winding machine	A.19

MTC ISO 841 Classification	Description	Figure
20	Horizontal filament winding machine	A.20
21	Flame cutting machine	A.21
22	Punch press	A.22
23	Drafting machine	A.23
24	Right-hand tube bender	A.24
25	Surface grinding machine with vertical grinding wheel	A.25
26	Cavity sinking EDM machine	A.26
27	Surface grinding machine	A.27
28	Coordinate measuring machine	A.28
29	Press brake	A.29
30	Wire electrical discharge machine	A.30
31	Laser cutting machine	A.31
32…	Reserved for future use.	

6.4.1.2. `Device` Structure

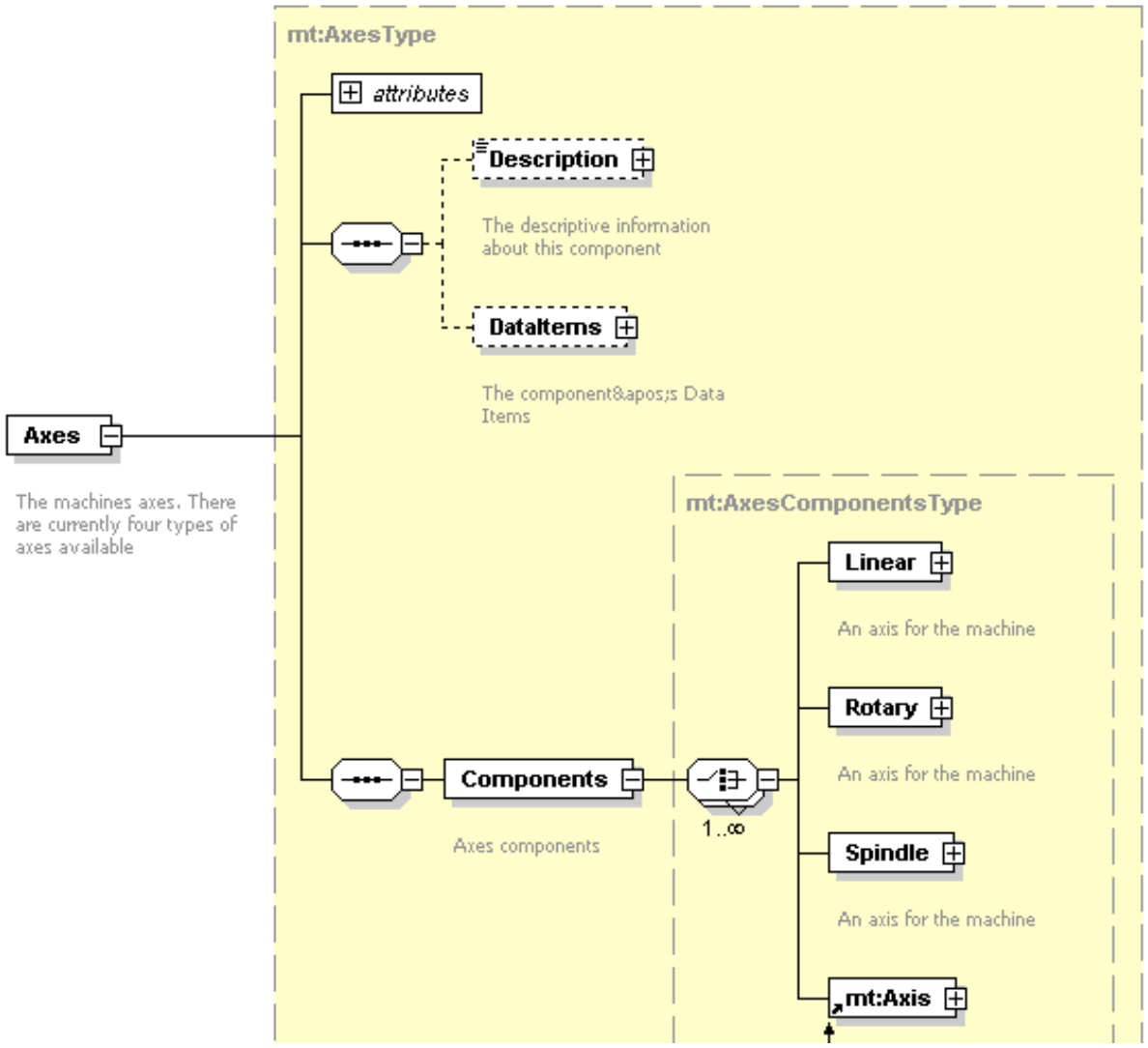


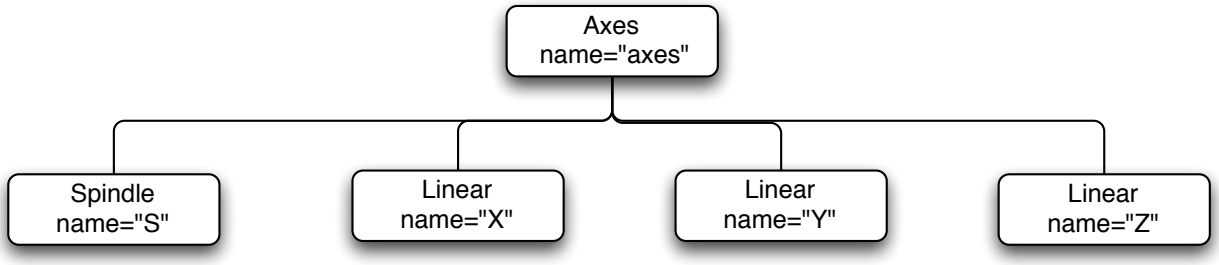
6.4.2. `Axes`

There can be an arbitrary number of axes. This flexibility will accommodate the more complex multi-axis, multi-spindle machines in the future. An `Axis` can be one of three different types: `Linear`, `Rotary`, and `Spindle`. The `Linear` axes MUST be named `X`, `Y`, `Z` and `U`, `V`, `W` as defined in the ISO-841-2001 specification. Rotary axes MUST be named as `A`, `B`, and `C` and rotate around the Linear `X`, `Y`, and `Z` axes respectively as defined in ISO-841-2001.

Note: The convention to be used for multiple linear, rotary, and spindle axes having the same designation is to index the letter with a number. For this standard the number starts at 2 (i.e. X,

X2, X3, … or S, S2, S3, S4, …). This is in compliance with the ISO-841-2001. Please refer to that specification for more details.





The `Axes` component **MUST** contain at least one `Axis` component. The possible axis components are as follows:

Linear A linear axis moves in the direction parallel to the motion direction of a linearly moving component. Because of various errors, the direction of the linear axis can best be defined as a least-squared fit of a straight line to the appropriate straightness data.

Rotary An axis whose function is to provide rotary motion either for the purposes of positioning and can be used for continuous-path contour cutting in a rotary direction or for repositioning different faces of the part for the purpose of metal removal.

Spindle Device that provides an axis of rotation for the purpose of rapidly rotating a part or a tool to provide sufficient surface speed for cutting operations.

6.4.3. Controller

The `Controller` component represents the `CNC` (Computer Numerical Control) or `PAC` (Programmable Automation Control) which has been referred to as a *Motion Control* or *General Purpose Motion Control*. The Control provides information regarding the execution of a control program and the execution state of the device. There are no required sub-components of the `Controller`.

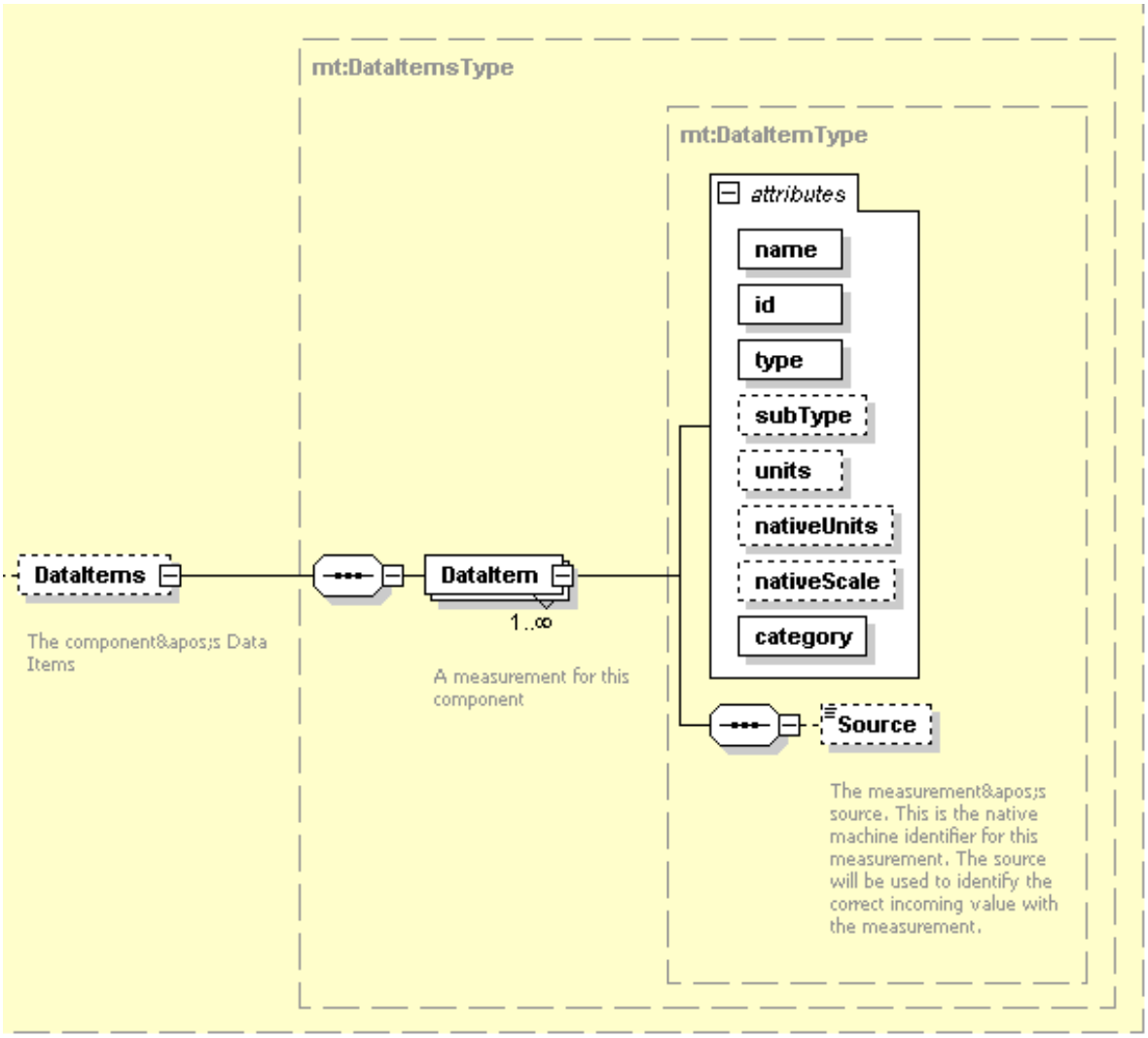
For more complex devices and controllers, it has been considered splitting out the individual execution program for each channel. This may require multiple `Control` components of a single device associated with specific axes. The modeling will be deferred to later revisions of the standard.

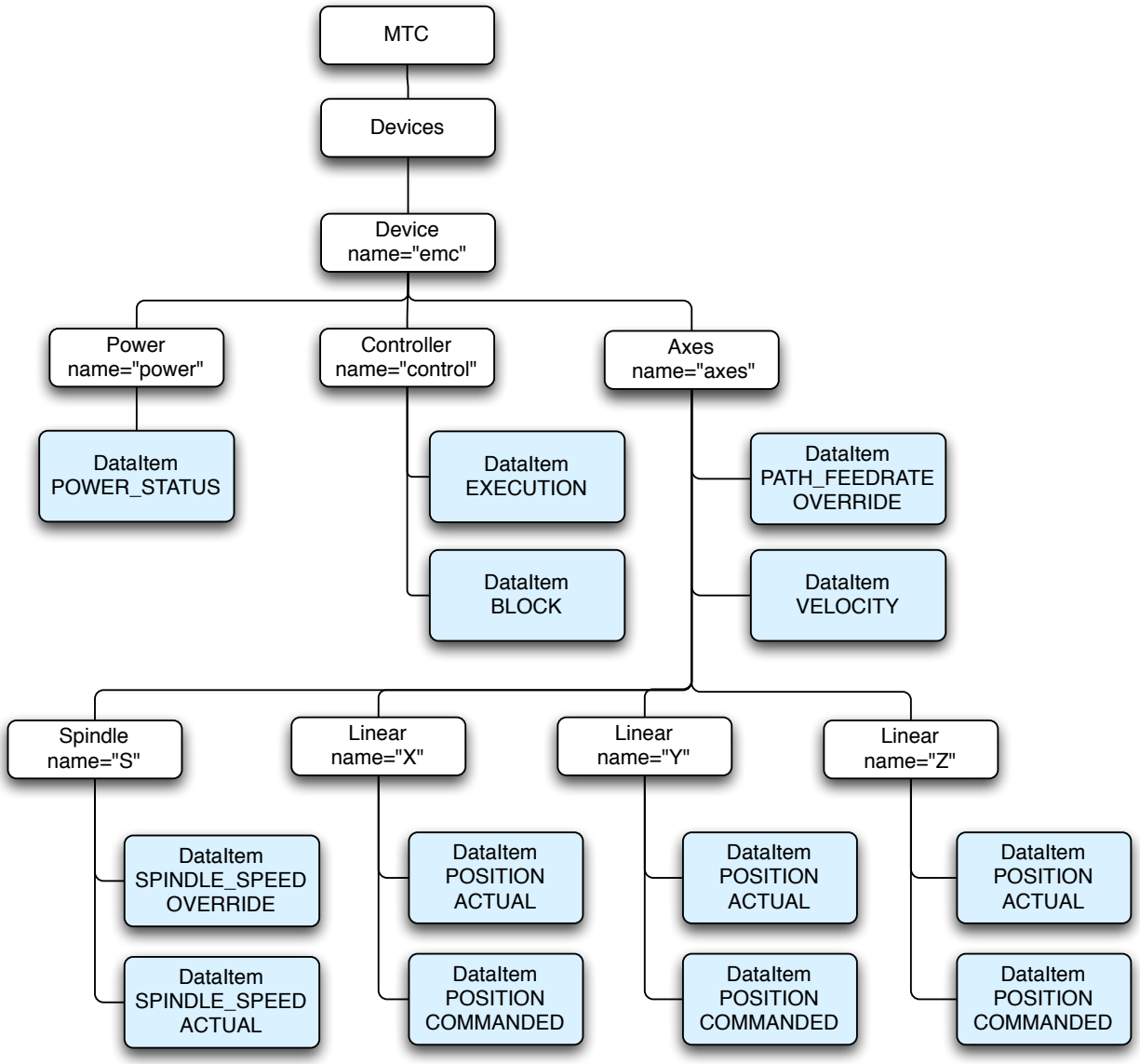
6.4.4. Power

The `Power` component is provided to report on the power status and possibly the voltage associated with its parent component. The device **MUST** contain a power component and **MUST** only contain the `POWER_STATUS` (on/off status). Any other data items **MAY** be added. Any other component, such as a spindle, that can be switched on or off separately from the `Device` **MUST** have a power component. There are no sub-components of `Power`.

7. Data Items

A `DataItem` describes a piece of information that can be collected from a component. The data item **MUST** specify the `type` of data being collected, the `name` of the data item, and the `category` of the item. There will only be one category for each `type`, but it **MUST** be included to aid the application in determining the location for the data stream. The data item **MAY** specify a `Source` sub-element to provide the native name for the data feed.





A `DataItem` **MAY** also specify the `subType`, to further qualify the type of data being requested. Some data item types **MUST** have subtypes and some **MAY** not. A list of the types and subtypes will be provided later in this section. As illustrated in the diagram above, the `POSITION` has two subtypes: `ACTUAL` and `COMMANDED`. These are two separate data items that can be reported independently.

The `nativeUnits` **MAY** be specified if they apply to the type of data and if they differ from the `units`. The `units` **MUST** be specified for any numeric data type. The *Agent* is responsible for converting the `nativeUnits` to the `units` before sending them to the applications. In addition, `nativeUnits` **MAY** be scaled using the `nativeScale` attribute; for example, if the device measures velocity in 100 ft/min, MTConnect would represent it with the following attributes: `nativeUnits="FEET/MINUTE"` and `nativeScale="100"`.

7.1. DataItem Element

7.1.1. Data Item Attributes

Attribute	Description	Occurrence
id	The unique identifier for this data item. This **SHOULD** be used to simplify the correspondence with the events and samples. See itemId in Event and Sample.	0..1
name	The name of the data item. A data item will have a unique name within the component. If there are multiple data items of the same type, like Position, the name will distinguish the data item.	1
type	The type of data being measured. Examples of types are POSITION, VELOCITY, ANGLE, CODE, BLOCK, SPINDLE SPEED, etc. The types are part of a controlled vocabulary that will be fixed for the first version.	1
subType	A sub-categorization of the data item type. Examples of position subtypes of POSITION are ACTUAL and COMMANDED. Not all types have subtypes and this can be left off.	0..1
category	This is how the data item will be sampled. The two options are SAMPLE and EVENT.	1
nativeUnits	The native units used by the component. These units will be converted before they are delivered to the application.	0..1
units	The units delivered to the application. These will always be the same for this data item type. This **MUST** be specified for all numeric values.	0..1
nativeScale	The multiplier for the native units. The received data **MAY** be divided by this value before conversion.	0..1
significantDigits	The number of significant digits in the reported value. This is used by applications to determine accuracy of values. This **MUST** be specified for all numeric values.	0..1

7.1.2. Data Item Elements

Element	Description	Occurrence
`Source`	Source is an optional element that contains the long name of the data item if it is too complex for the dname attribute. For example, if the data item has the name `Xact` and the axis position is delivered as `Axis.channel.0.position` from the device. The `name` attribute is `Xact` and the source is `Axis.channel.0.position`. If the source is not specified, it will be assumed to be the same as the `name`.	0..1

7.1.3. Data Item attribute: `category`

MTConnect provides two different categories of data items, `SAMPLE` and `EVENT`. The `category` will indicate where the results will be reported in the XML Document as a response to a `sample` or `current` request. See the section on *Streams, Samples, and Events (Section 9)* for more information.

`SAMPLE` A sample data item has a value that is in motion or moves between different values in a manner that can be interpolated. A continuous value can be sampled at any point-in-time and will always produce a result. An example of a continuous data item is the S axis spindle speed.

Continuous data items are always scalar floating point or integers that can have an infinite number of possible values. This is different from state or discrete data items that have a limited number of possible values.

`EVENT` Unexpected or discrete occurrence in a component. This includes state changes and alarms. Events do not have intermediate values that differ at intermediate times, as do samples.

7.1.4. `units`

Unit	Description
`AMPERE`	Amps
`CELCIUS`	Degrees Celsius
`DEGREE`	Angle in degrees
`DEGREE/SECOND`	Degrees per second
`DEGREE/SECOND^2`	Acceleration in degrees per second squared
`HERTZ`	Frequency measured in cycles per second
`KILOGRAM`	Kilograms
`LITER`	Liters
`MILLIMETER`	Millimeters
`MILLIMETER/SECOND`	Millimeters per second
`MILLIMETER/SECOND^2`	Acceleration in millimeters per second squared

Unit	Description
NEWTON	Force in newtons
PASCAL	Pressure in Newtons per square meter
PERCENT	Percent
REVOLUTION/MINUTE	Revolutions per minute
STATUS	A status that conforms to the data item's controlled vocabulary. Used in events to indicate states or status.
TICK	An instance of a counted event
VOLT	Volts
WATT	Watts

7.2. Types and Subtypes of Data Items

What follows is the association between the various types and subtypes of data items. Each data item type **MUST** be translated into a `Sample` or `Event` with the following rules: The type and sub-type name will be all in capitals with an underscore (_) between words. The element of the event or sample will be the transformation of the data item type by capitalizing the first character of each word and then removing the underscore. For example the data item type `POWER_STATUS` is `PowerStatus`, `POSITION` is `Position`, and `SPINDLE_SPEED` is `SpindleSpeed`.

The `category` will indicate if the data item is reported in the `Samples` or `Events` section of the XML Document. The following are the currently supported sample and event data items.

7.2.1. Data Item Types for `SAMPLE` Category

The types are given in **bold** and the subtypes are indented and in plain text.

Data Item type/subtype	Description	Units
ACCELERATION	Rate of change of velocity	MILLIMETER/SECOND^2
ANGULAR_ ACCELERATION	Rate of change of angular velocity.	DEGREE/SECOND^2
ANGULAR_ VELOCITY	Rate of change of angular position.	DEGREE/SECOND
AMPERAGE	The line current	AMPERE
ANGLE	The angular position of a component relative to the parent.	DEGREE
ACTUAL	The angular position as read from the physical component	DEGREE
COMMANDED	The angular position the Controller has instructed the component to have	DEGREE
AXIS_FEEDRATE	The feedrate of the axis.	MILLIMETER/SECOND
ACTUAL	The single dimension feedrate.	MILLIMETER/SECOND
COMMANDED	The feedrate as specified in the program.	MILLIMETER/SECOND

Data Item type/subtype	Description	Units
OVERRIDE	The operator's overridden value. Percent of commanded.	PERCENT
GLOBAL_POSITION	The position in three-dimensional space. The X, Y, and Z positions will be provided.	MILLIMETER
ACTUAL	The position of the component as read from the device.	MILLIMETER
COMMANDED	The position as given by the Controller.	MILLIMETER
LOAD	The load on the component.	NEWTON
PATH_FEEDRATE	The feedrate of the tool path.	MILLIMETER/SECOND
ACTUAL	The three-dimensional feedrate derived from all components.	MILLIMETER/SECOND
COMMANDED	The feedrate as specified in the program	MILLIMETER/SECOND
OVERRIDE	The operator's overridden value. Percent of commanded.	PERCENT
PRESSURE	The pressure on the component	PASCAL
POSITION	The position of the component. Always in absolute coordinates.	MILLIMETER
ACTUAL	The position of the component as read from the device.	MILLIMETER
COMMANDED	The position as given by the Controller.	MILLIMETER
SPINDLE_SPEED	The rotational speed of the spindle.	REVOLUTION/MINUTE
ACTUAL	The rotational speed the spindle is spinning at.	REVOLUTION/MINUTE
COMMANDED	The rotational speed the spindle has been instructed to spin at.	REVOLUTION/MINUTE
OVERRIDE	The operator's overridden value. Percent of commanded.	PERCENT
TEMPERATURE	The temperature	CELSIUS
VELOCITY	The rate of change of position.	MILLIMETER/SECOND
VIBRATION	The vibration	HERTZ
VOLTAGE	The voltage	VOLT
WATTAGE	The wattage	WATT

7.2.2. Data Item Types for EVENT Category

Note: The Event does not have any units since these values are not scalers.

Data Item type/subtype	Description
BLOCK	The block of code being executed. The block contains the entire expression of the step in the program.
CODE	The programmatic code being executed

Data Item type/subtype	Description
DIRECTION	The rotational direction of the Axis. `CLOCKWISE or COUNTER_CLOCKWISE`
EXECUTION	The execution status of the Controller. `IDLE, EXECUTING, or PAUSED`
LINE	The current line of code being executed
POWER_STATUS	The ON/OFF status of the component
PROGRAM	The name of the program being executed
TICK	A simple counter event
TRANSFER	A workpiece event that occurs when material enters or leaves the component. `IN or OUT`
ALARM	An alarm is a special data item that will report any alarm for this component. An alarm **MUST** be included as a `DataItem` for the `Device`
CONTROLLER_MODE	The current controller's mode. `AUTOMATIC, MANUAL, or MANUAL_DATA_ENTRY`

8. Component and Data Item Relationships

This section will discuss the association between `Component`, `DataItems`, and `Events` and `Samples`. For each component, there are a limited set of allowable sub-components and a limited set of data items. For example, an `Axes` component may not have a `Device` or a `Controller` as a child, and it may not have as a `Block` `DataItem` type, since it is incapable of running a program.

8.1. Overview

At the top level, a device **MUST** always contain a `Power` component as the main power supply. Every component that is capable of managing its own power supply, **SHOULD** have a `Power` sub-component. For example, a spindle **SHOULD** have a `Power` sub-component if it can be turned off separately from the device.

Any component **MAY** also include an arbitrary set of sensors as sub-components. The sensor is currently a placeholder for extensible data collection devices and is not modeled in this version of the specification. A sensor will be an external device that will collect data and report it to the *Agent*. The sensor **MUST** be correctly associated with its most relevant component. The rules governing this association will be covered in a later version of this specification.

8.2. `Device`

The `Device` is the only top level element in the component tree. Since an MTConnect *Agent* can manage multiple devices, the schema provides a top level container `Devices` to hold the `Device` elements.

8.2.1. DataItem types

- `ALARM` - An alarm placeholder for all alarms that are not associated with another component.

8.2.2. Sub-components of Device

- `Power`
- `Controller`
- `Axes`

8.3. `Axes`

The `Axes` component serves two functions: it is a container for the actual axes as well the global data items for kinematics, path feedrate and other aggregates of all the `Axis` components below it. An `Axes` **MAY** have one or more of these:

8.3.1. DataItem types

- `GLOBAL_POSITION`
- `PATH_FEEDRATE`
- `ACCELERATION`
- `VELOCITY`

8.3.2. Sub-components of Axes

- Linear
- Rotary
- Spindle

8.4. Linear

A linear axis represents travel along a straight line. The name of the linear axis **SHOULD** follow the conventions of the industry.

8.4.1. DataItem types

- POSITION
- ACCELERATION
- VELOCITY
- LOAD
- AXIS_FEEDRATE

8.5. Rotary

A rotary axis revolves around a point.

8.5.1. DataItem types

- ANGLE
- ANGULAR_ACCELERATION
- ANGULAR_VELOCITY
- LOAD
- AXIS_FEEDRATE

8.6. Spindle

The spindle is a rotational axis that revolves at high speed and has its speed expressed in REVOLUTION/MINUTE. The spindle can also have additional data items. Spindle speed has been specified as a separate data item since it receives special treatment in many applications. Velocity is used for linear axes other than spindle.

8.6.1. DataItem types

- SPINDLE_SPEED
- LOAD
- DIRECTION

8.7. Controller

The controller component is the component that controls a device, executes a program, and sends instructions to the other components of the machine. It is the brains of the machine and can be asked for its current execution state and program name.

8.7.1. DataItem types

- PROGRAM
- EXECUTION
- LINE
- BLOCK
- CODE
- CONTROLLER_MODE

8.8. Power

The power component represents the electrical activation of the component. The data items the power component can collect are a simple status (on/off) and three power related measurements, voltage, amperage and watts. There are no sub-components of Power. The reason for making this a separate component is the need to support legacy equipment.

8.8.1. DataItem types

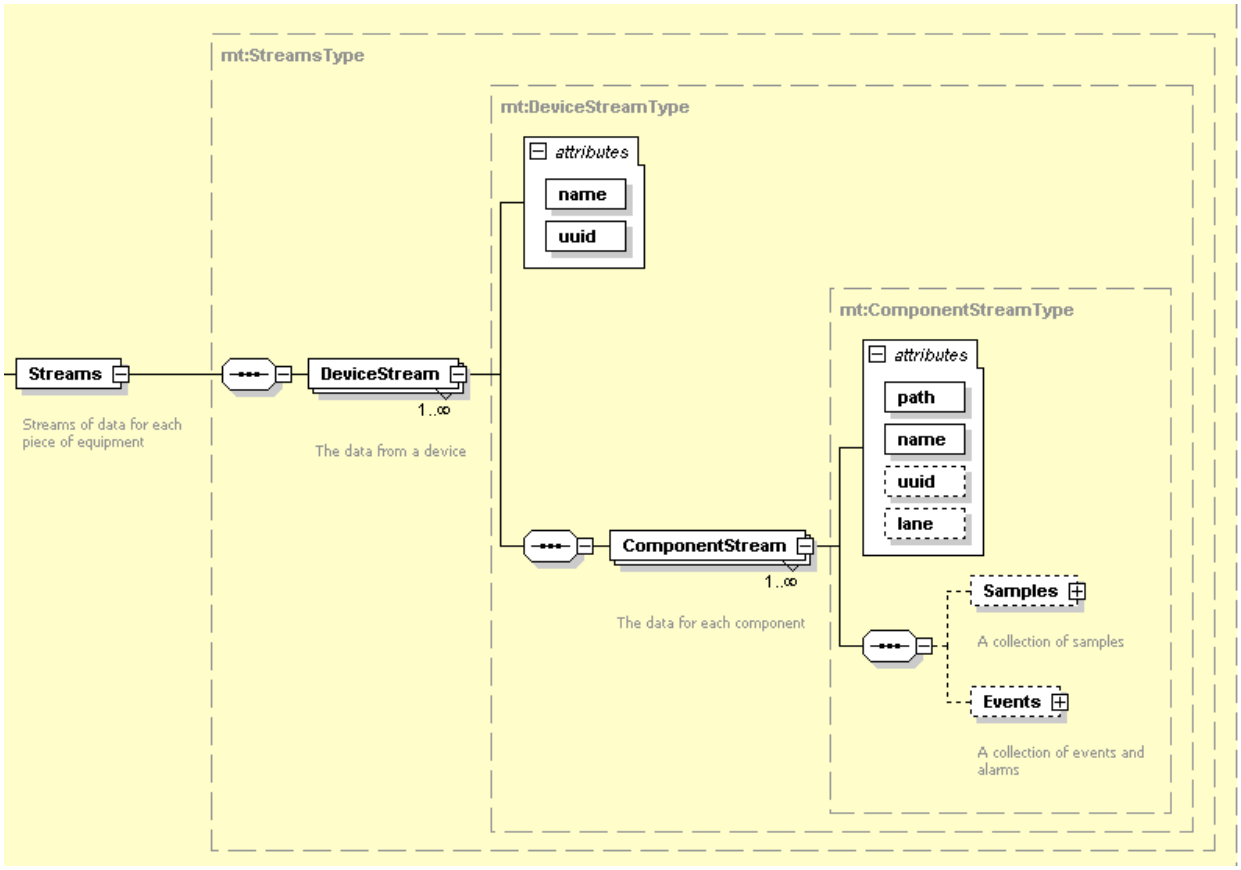
- POWER_STATUS
- VOLTAGE
- AMPERAGE
- WATTS

9. Streams, Samples and Events

The MTConnect *Agent* collects data from various sources and delivers it to applications in response to `sample` or `current` requests. (See *Protocol* section.) All the data are collected into streams and organized by device and then by component. A component stream has two parts: `Samples` and `Events`. `Samples` are point-in-time readings from a component reporting what the value is at that instant.

An `Event` changes state to a limited set of values. It is assumed that an event remains at a state until the next event occurs; it cannot have any intermediate values between the reported values. Alarms are classified as events. The following are examples of `Events`: `Block`, `Code`, `Execution`, `PowerStatus`, etc.

If two adjacent samples for the same component and data item have the same value, the second sample **MUST NOT** be sent to the client application and does not need to be retained by the MTConnect *Agent*. This will greatly reduce the amount of information sent to the application. The application can always assume that if the sample is not present, it has the previous value. If the application needs the present value, it can always ask for the `current` values (see *Protocol)*.



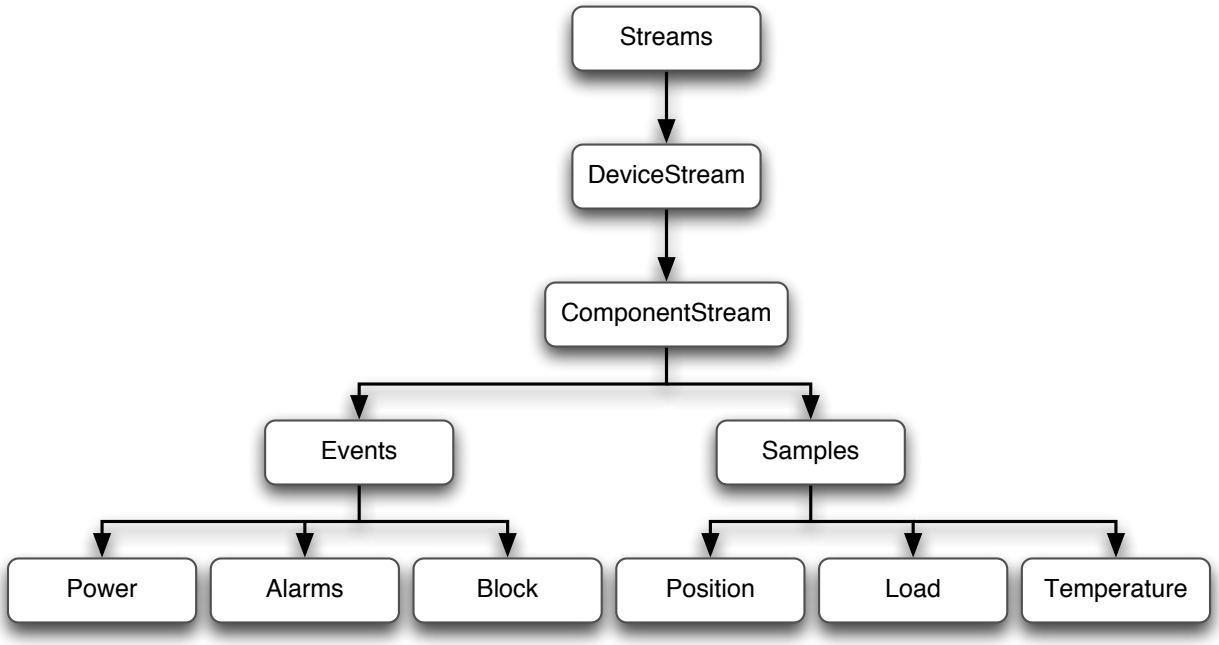
9.1. Streams

A `Stream` is the high level container for all device streams. It serves no other purpose than to have `DeviceStream` sub-elements. There **MUST** be no attributes or elements within this element.

Elements	Description	Occurrence
`DeviceStream`	The stream of samples and events for each device.	1..INF

9.2. Structure

This diagram illustrates the structure of the streams with some samples and events at the lowest level:



A `Stream` **MUST** have at least one `DeviceStream` and the `DeviceStream` **MAY** have one or more `ComponentStream` elements, depending on whether there are events or samples available for the component. If there are no `ComponentStream` elements, then no data was available for this request.

Below is an example XML Document response for an *Agent* with two devices, mill-1 and mill-2. The data will be reported in two separate device streams. The sequence numbers will be consistent across the two devices. The application **MUST NOT** assume that the event and sample sequence numbers will be strictly in sequence. The sequence numbers **MAY** skip due to filtering.

```
1. <mt:MTConnectStreams version="0.9">
2.   <Header instanceId="11" nextSequence="10">…</Header>
3.   <Streams>
```

```
4.     <DeviceStream name="mill-1" uuid="123231341">
5.       <ComponentStream name="power">
6.         <Events>
7.           <PowerStatus sequence="1" timestamp="…">OFF</PowerStatus>
8.         </Events>
9.       </ComponentStream>
10.    </DeviceStream>
11.    <DeviceStream name="mill-2" uuid="123231342">
12.      <ComponentStream name="power">
13.        <Events>
14.          <PowerStatus sequence="4" timestamp="…">ON</PowerStatus>
15.        </Events>
16.      </ComponentStream>
17.    </DeviceStream>
18.  </Streams>
19.</mt:MTConnectStreams>
```

9.3. DeviceStream

A DeviceStream is created to hold the device-specific information so it does not need to be repeated for every event and sample. This is done to reduce the size of each event and sample so they only carry the information that is being reported. A DeviceStream **MAY** contain one or more ComponentStream elements. If the request is valid and there are no events or samples that match the criteria, an empty DeviceStream element **MUST** be created to indicate that the device exists, but there was no data available.

9.3.1. DeviceStream Attributes

Attributes	Description	Occurrence
name	The device's name	1
uuid	The device's unique identifier	1

9.3.2. DeviceStream Elements

Element	Description	Occurrence
ComponentStream	One component's stream for each component with data	0..INF

9.4. ComponentStream

A ComponentStream is similar to the DeviceStream. It contains the information specific to the component within the Device. There is one area that might be confusing: if the Device is being reported on, it **MUST** have a ComponentStream that repeats the name and uuid of the Device.

9.4.1. `ComponentStream` Attributes

Attribute	Description	Occurrence
`name`	This components name within the device	1
`path`	The xpath to the component	1
`component`	The name of the component	1
`uuid`	The component's unique identifier	0..1

The Elements of the `ComponentStream` classify the data into `Events` and `Samples`. *(The classification is discussed below)*. The `ComponentStream` **MUST NOT** be empty. It **MUST** include an `Events` and/or a `Samples` element.

9.4.2. `ComponentStream` Elements

Element	Description	Occurrence
`Events`	The events for this component stream	0..1
`Samples`	The samples for this component	0..1

9.5. `Samples`

The `Samples` element must contain at least one `Sample` element. This element acts only as a container for all the `Samples` to provide a logical structure to the XML Document.

Element	Description	Occurrence
`Sample`	The subtype of `Sample` for this component stream	1..INF

9.6. `Sample`

9.6.1. A sample has following attributes:

Attribute	Description	Occurrence
`name`	The name **MUST** match the name of the `DataItem` this sample is associated with.	1
`sequence`	The sequence number of this event.	1
`timestamp`	The timestamp of the event.	1
`lane`	This is the internal production lane (not external) the machine is taking material from.	0..1
`itemID`	The id of the data item that this sample is associated with. Only supply if the `DataItem` supplied an `id`.	0..1
`workpieceID`	The unique identifier of the workpiece being machined	0..1
`partID`	The part number of the workpiece.	0..1

A sample **MUST** contain CDATA as the content between the element tags. A position will be formatted like this:

```
1. <mt:Position seq="112" timestamp="2007-08-09T12:32:45.1232"
      name="Xabs">123.3333</mt:Position>
```

In this example the 123.3333 is the CDATA for the position. All the CDATA in a sample is typed, meaning that it can be validated using an XML parser. This restricts the format of the values to a specific pattern.

9.6.2. Sample Elements

Acceleration The acceleration of the component MUST always be reported in MILLIMETER/SECOND^2. An acceleration **MUST** have a numeric value.

Amperage The current in an electrical circuit. The amperage **MUST** have a numeric value and **MUST** be reported in AMPS.

Angle An angle **MUST** always be reported in DEGREE and **MUST** always have a numeric CDATA value as a floating point number.

AngularAcceleration The angular acceleration of the component as measured in DEGREE/SECOND^2. An acceleration **MUST** have a numeric value.

AngularVelocity A angular velocity represents the rate of change in angle. An angular velocity **MUST** always be reported in DEGREE/SECOND^2 and **MUST** always have a numeric CDATA value as a floating point number.

AxisFeedrate Axis Feedrate is defined as the rate of motion of the feed axis of the tool relative to the workpiece[2]. An axis feedrate **MUST** always be reported in MILLIMETER/SECOND or PERCENT for override and **MUST** always have a numeric CDATA value as a floating point number.

PathFeedrate Path Feedrate is defined as the rate of motion of the feed path of the tool relative to the workpiece[3]. A path feedrate **MUST** always be reported in MILLIMETER/SECOND or PERCENT for override and **MUST** always have a numeric CDATA value as a floating point number.

GlobalPosition The global position is the three space coordinate of the tool. A global position **MUST** always be reported in MILLIMETER and **MUST** always have a numeric CDATA value as three floating point numbers (x, y, and z). Position **MUST** always be given in absolute coordinates.

---

[2] From ASME B5.54 - 2005

[3] From ASME B5.54 - 2005

Load The load on a component. The load **MUST** always be reported in NEWTON and **MUST** always have a numeric CDATA value as a floating point number.

Position A position represents the location along a linear axis. A position **MUST** always be reported in MILLIMETER and **MUST** always have a numeric CDATA value as a floating point number. Position **MUST** always be given in absolute coordinates.

Pressure The pressure on a component. The pressure **MUST** be a numeric value and **MUST** be provided in PASCALS.

SpindleSpeed The rate of rotation of a machine spindle [4]. A spindle speed **MUST** always be reported in REVOLUTION/MINUTE and **MUST** always have a numeric CDATA value as a floating point number.

Temperature Temperature **MUST** always be reported in degrees CELSIUS and **MUST** always have a numeric CDATA value as a floating point number.

Velocity A velocity represents the rate of change in position along one or more axis. When given as a Sample for the Axes component, it represents the magnitude of the velocity vector for all given axis, similar to a path feedrate. A velocity **MUST** always be reported in MILLIMETER/SECOND and **MUST** always have a numeric CDATA value as a floating point number.

Vibration The rate at which a component is vibrating. The vibration **MUST** have a numeric value and **MUST** be reported in HERTZ.

Volts The potential difference as measured across an electrical circuit. The voltage **MUST** have a numeric value and **MUST** be reported in VOLTS.

Watts The electrical power (volt-amps) of an electrical circuit. The watts **MUST** have a numeric value and **MUST** be reported in WATTS.

9.6.3. Extensibility

Additional sample types can be added by extending the Sample type in the XML schema. The samples presented here are the official sample types that will be supported by all MTConnect *Agents*. Any non-sanctioned extensions will not be guaranteed to have consistency across implementations.

9.7. Events

The Events element must contain at least one Event element. This element acts only as a container for all the Events to provide a logical structure to the XML Document.

[4] From ASME B5.54 - 2005

Element	Description	Occurrence
Event	The subtype of Event for this component stream	1..INF

9.8. Event

Attribute	Description	Occurrence
name	The name **MUST** match the name of the DataItem this event is associated with	1
sequence	The sequence number of this event	1
timestamp	The time-stamp of the event	1
lane	This is the internal production lane (not external) the machine is taking material from.	0..1
itemID	The id of the data item that this event is associated with. Only supply if the DataItem supplied an id	0..1
workpieceID	The unique identifier of the workpiece being machined	0..1
partID	The part number of the workpiece	0..1

An event is similar to a sample, but its values are going to be changing with unpredictable frequency. Events do not have intermediate values. When a power status transitions from OFF to ON, there is no intermediate state that can be inferred. Therefore, most events have a controlled vocabulary as their content.

An event does not add any additional attributes or elements to the Sample. It is a placeholder in the schema type hierarchy for elements that are events. This relationship will be enforced by the schema.

9.8.1. Event Elements

Block A Block of code is a command being executed by the Controller. The Block **MUST** include the entire command with all the parameters.

Code The code is just the G, M, or NC code being executed. The Code **MUST** only contain the simplest form of the executing command.

ControllerMode The Mode of the Controller. The CDATA **MUST** be one of the following:

```
AUTOMATIC
MANUAL
MANUAL_DATA_INPUT
```

Direction A Direction indicates the direction of rotation. The CDATA **MUST** be either CLOCKWISE or COUNTER_CLOCKWISE.

Execution The Execution state of the Controller. The CDATA **MUST** be one of the following:

IDLE
EXECUTING
PAUSED

Line The current line number of the program being executed. The CDATA **MUST** be a numeric value.

PowerStatus Power status **MUST** be either ON or OFF.

Program The name of the program executing in the controller. This is usually the name of the file containing the program instructions.

Tick A discrete event that indicates that something has occurred. A Tick can be used for counting an occurrence like a part count. The CDATA is application-defined.

Transfer A Transfer indicates that material has entered or left the device. The CDATA **MUST** be either IN or OUT.

9.9. Alarms

The Alarm event adds some additional fields to the standard Event schema. The following additional attributes are used for the alarm:

Attribute	Description	Occurrence
code	The type of alarm. This is a high level classification for all codes.	1
severity	The severity of the alarm, currently we have CRITICAL, ERROR, WARNING, or INFORMATION.	1
nativeCode	The native code for the piece of equipment. This is the way the alarm is represented on the component.	1
state	Either INSTANT, ACTIVE or CLEARED. When the Alarm occurs, it will be created with an ACTIVE state. Once it has been addressed, the state will be changed to CLEARED. An INSTANT alarm does not need to be cleared.	1

The code can have one of the following values:

Enumeration	Description
CRASH	A spindle crashed
JAM	A component jammed.
FAILURE	The component failed.

Enumeration	Description
FAULT	A fault occurred on the component.
STALLED	The component has stalled and cannot move.
OVERLOAD	The component is overloaded.
ESTOP	The ESTOP button was pressed.
MATERIAL	There is a problem with the material.
MESSAGE	A system message.
OTHER	The alarm is not in any of the above categories.

The CDATA of the Alarm is the human-readable text from the component that raised the alarm. The device should specify this text so it can be logged.

10. Annotated XML Examples

10.1. Simplest Device

For the simplest possible device we are modeling a saw that has only a power status (the minimal set of components). To retrieve this information we send the following request to the *Agent*:

http://10.1.23.10/saw/probe

The *Agent* responds as follows:

```
2. <?xml version="1.0" encoding="UTF-8"?>
3. <mt:MTConnectDevices xsi:schemaLocation="urn:mtconnect.com:MTConnect:0.9
     MTConnectDevices.xsd" version="0.9"
     xmlns:mt="urn:mtconnect.com:MTConnect:0.9" xmlns:xsi="http://www.w3.org/
     2001/XMLSchema-instance">
4.   <Header creationTime="2008-01-08T18:06:46-08:00" instanceId="1199844370">
```

Line 4 provides the `instanceId` as a unique number for this run. For this example, the *Agent* does not persist the samples and events, therefore, this number will change every time.

```
5.     <Sender>10.1.23.10</Sender>
6.     <BufferSize>100000</BufferSize>
7.   </Header>
```

Line 6 indicates that this *Agent* is capable of storing 100,000 samples and events.

```
8.   <Devices>
9.     <Device name="saw" uuid="321044" sampleRate="1000.0" iso841Class=”6” >
10.      <Description serialNumber="" manufacturer=""/>
```

The above device description includes the unique id and a sample rate of once per second. Since there are no telemetry data being collected, once a second is adequate.

```
11.      <DataItems>
12.        <DataItem type="ALARM" category="EVENT" name="alarm" id="20"/>
13.      </DataItems>
```

On line 12 we define the catch-all alarm for this device.

```
14.      <Components>
15.        <Power name="power">
16.          <DataItems>
17.            <DataItem type="POWER_STATUS" category="EVENT" name="power"
     id="13"/>
18.          </DataItems>
19.        </Power>
```

As was stated before, the device is only required to have one `Power` component which **MUST** report its status. The DataItem on line 15 has an id number of 13. This will allow events

responding to this data item to be easily associated. One can also see that this has been categorized as an Event and the application should expect `PowerStatus` in the `Events` collection of the `ComponentStream`.

```
20.      </Components>
21.    </Device>
22.  </Devices>
23.</mt:MTConnectDevices>
```

10.2. More Complex Example of `probe`

The sample was generated with the following request:

```
http://10.1.23.5/mill/probe
```

The following is an example of a 3 axis mill simulation. The mill has three linear axes and one spindle. The Controller is capable of providing the program name, block, and the current line being executed:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <mt:MTConnectDevices xsi:schemaLocation="urn:mtconnect.com:MTConnect:0.9
     MTConnectDevices.xsd" version="0.9"
     xmlns:mt="urn:mtconnect.com:MTConnect:0.9" xmlns:xsi="http://www.w3.org/
     2001/XMLSchema-instance">
3.   <Header creationTime="2008-01-09T10:10:58-08:00" instanceId="1199844370">
4.     <Sender>localhost</Sender>
5.     <BufferSize>100000</BufferSize>
6.   </Header>
7.   <Devices>
8.     <Device name="mill" uuid="32102904" sampleRate="10.0" iso841Class=”6”>
```

Here we provide the top level container `Devices` and the information on the `Device`.

```
9.       <Description serialNumber="32102904" manufacturer="Linux CNC"/>
10.      <DataItems>
11.        <DataItem type="ALARM" category="EVENT" name="alarm" id="14">
12.          <Source>machine_alarm</Source>
13.        </DataItem>
14.      </DataItems>
15.      <Components>
16.        <Axes name="axes">
17.          <DataItems>
18.            <DataItem type="FEEDRATE" subType="OVERRIDE" category="SAMPLE"
     name="feedrate" id="2"/>
19.
```

```
20.           <DataItem type="VELOCITY" nativeUnits="MILLIMETER/SECOND"
    category="SAMPLE" name="velocity" units="MILLIMETER/SECOND" id="18"/>
21.         </DataItems>
```

On line 16 we introduce the collection of Axes. The Axes component is a special component that acts as an abstract component as well as a collection. The Axes component contains various data items that have a global context; they are not associated with any one data item, but they go across all axes.

```
22.         <Components>
23.           <Spindle name="S">
24.             <DataItems>
25.               <DataItem type="SPINDLE_SPEED" subType="ACTUAL"
    category="SAMPLE" name="Srpm" units="REVOLUTION/MINUTE" id="3"/>
26.               <DataItem type="DIRECTION" category="EVENT" name="Sdir"
    id="16"/>
27.             </DataItems>
28.           </Spindle>
```

The spindle component declared on line 23 is the S axis and has spindle-specific data items.

```
29.           <Linear name="X">
30.             <DataItems>
31.               <DataItem type="POSITION" nativeUnits="MILLIMETER"
    subType="ACTUAL" category="SAMPLE" name="Xact" units="MILLIMETER"
    id="4"/>
32.               <DataItem type="POSITION" nativeUnits="MILLIMETER"
    subType="COMMANDED" category="SAMPLE" name="Xcom" units="MILLIMETER"
    id="5"/>
33.             </DataItems>
34.           </Linear>
35.           <Linear name="Y">
36.             <DataItems>
37.               <DataItem type="POSITION" nativeUnits="MILLIMETER"
    subType="ACTUAL" category="SAMPLE" name="Yact" units="MILLIMETER"
    id="6"/>
38.               <DataItem type="POSITION" nativeUnits="MILLIMETER"
    subType="COMMANDED" category="SAMPLE" name="Ycom" units="MILLIMETER"
    id="7"/>
39.             </DataItems>
40.           </Linear>
41.           <Linear name="Z">
42.             <DataItems>
```

```
43.                <DataItem type="POSITION" nativeUnits="MILLIMETER"
    subType="ACTUAL" category="SAMPLE" name="Zact" units="MILLIMETER"
    id="8"/>
44.                <DataItem type="POSITION" nativeUnits="MILLIMETER"
    subType="COMMANDED" category="SAMPLE" name="Zcom" units="MILLIMETER"
    id="9"/>
45.              </DataItems>
46.            </Linear>
```

Lines 29, 35, and 41 define the three linear axes X, Y, and Z respectively. In this example device the *Agent* is only collecting the actual and commanded positions.

```
47.          </Components>
48.        </Axes>
49.        <Controller name="control">
50.          <DataItems>
51.            <DataItem type="PROGRAM" category="EVENT" name="file" id="10"/
    >
52.            <DataItem type="BLOCK" category="EVENT" name="block" id="11">
53.              <Source>command</Source>
54.            </DataItem>
55.            <DataItem type="EXECUTION" category="EVENT" name="execution"
    id="12"/>
56.            <DataItem type="ALARM" category="EVENT"
    name="controller_alarm" id="15"/>
57.            <DataItem type="LINE" subType="ACTUAL" category="EVENT"
    name="line" id="17"/>
58.          </DataItems>
59.        </Controller>
60.        <Power name="power">
61.          <DataItems>
62.            <DataItem type="POWER_STATUS" category="EVENT" name="power"
    id="1"/>
63.          </DataItems>
64.        </Power>
65.      </Components>
66.    </Device>
67.  </Devices>
68.</mt:MTConnectDevices>
```

10.3. Example of a `sample` Request

The sample was generated with the following request:

```
http://10.1.23.5/mill/sample?from=1&count=3000&path=//Control|//Machine/
    Components/Power
```

The response is as follows:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <m:MTConnectStreams version="0.2" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="urn:mtconnect.com:MTConnect:0.1
    http://localhost:3000/public/schemas/mtc.xsd"
    xmlns:m="urn:mtconnect.com:MTConnect:0.2">
3.   <Header instanceId=”1199256590” nextSequence="2858"
    creationDate="2007-12-06" sender=”MTConnect” bufferSize=”10000”>
4.   </Header>
5.   <Streams>
```

Events are grouped by equipment:

```
6.     <DeviceStream name="mill" uuid="11322343434">
```

All the events are then grouped by components. The path includes the most relevant parts of the xpath with only the `Components` containers removed here for brevity. The only element that **MUST** be removed is `Components`. The `name` selector makes the component unique within the path:

```
7.      <ComponentStream path="//Device[@name='mill']" name="control"
    component=”Controller”>
8. <Events>
9.         <m:Block timestamp="2007-12-06T02:20:51.938" sequence="10">None</
    m:Block>
```

The operator has selected a program to run:

```
10.         <m:Program timestamp="2007-12-06T02:20:51.938" sequence="11">/usr/
    share/emc/ncfiles/skeleton.ngc</m:Program>
```

The control execution is now idle:

```
11.         <m:Execution timestamp="2007-12-06T02:20:51.938"
    sequence="15">IDLE</m:Execution>
12.         <m:Line timestamp="2007-12-06T02:20:51.938" sequence="17">0</
    m:Line>
```

```
13.       <m:Block timestamp="2007-12-06T02:21:14.842" sequence="20">N0050
     M05 M09 (spindle and coolant off)</m:Block>
```

The execution unit is now running:

```
14.       <m:Execution timestamp="2007-12-06T02:21:14.842"
     sequence="21">RUNNING</m:Execution>
15.       <m:Line timestamp="2007-12-06T02:21:14.842" sequence="22">6</
     m:Line>
16.       <m:Block timestamp="2007-12-06T02:21:16.749" sequence="395">N0060
     (MSG,Load tool #1)</m:Block>
17.       <m:Line timestamp="2007-12-06T02:21:16.749" sequence="396">8</
     m:Line>
18.       <m:Block timestamp="2007-12-06T02:21:16.851" sequence="397">N0070
     M00 (don't move until the operator presses the S key)</m:Block>
```

We now pause for a tool insertion:

```
19.       <m:Execution timestamp="2007-12-06T02:21:16.851"
     sequence="398">PAUSED</m:Execution>
20.       <m:Line timestamp="2007-12-06T02:21:16.851" sequence="399">9</
     m:Line>
21.       <m:Block timestamp="2007-12-06T02:21:20.719" sequence="400">N0080
     T1 M06 G43 H1 (change to tool 1 and get its length from the tool
     table)</m:Block>
22.       <m:Execution timestamp="2007-12-06T02:21:20.719"
     sequence="401">RUNNING</m:Execution>
23.       <m:Line timestamp="2007-12-06T02:21:20.719" sequence="402">10</
     m:Line>
24.       <m:Block timestamp="2007-12-06T02:21:24.768" sequence="771">N0090
     G04 P.001 G0 X1.0 Y1.0 S1000 M3 M8 (rapid to the starting XY, spindle
     CW, coolant ON, change as required)</m:Block>
25.       <m:Line timestamp="2007-12-06T02:21:24.768" sequence="773">12</
     m:Line>
26.       <m:Block timestamp="2007-12-06T02:21:24.860" sequence="774">N0100
     G04 P.001 G0 Z0.25 (rapid to .25" above the part, change as required)</
     m:Block>
27.       <m:Line timestamp="2007-12-06T02:21:24.860" sequence="775">13</
     m:Line>
28.       <m:Block timestamp="2007-12-06T02:21:25.837" sequence="1165">N0120
     M05 M09 (spindle and coolant off)</m:Block>
```

```
29.       <m:Line timestamp="2007-12-06T02:21:25.837" sequence="1167">15</
     m:Line>
30.       <m:Block timestamp="2007-12-06T02:21:26.072" sequence="1216">N0130
     G53 G0 Z0 (temporarily cancel offsets, retract quill)</m:Block>
31.       <m:Line timestamp="2007-12-06T02:21:26.072" sequence="1217">16</
     m:Line>
32.       <m:Block timestamp="2007-12-06T02:21:26.163" sequence="1218">N0150
     T0 M6 (remove tool)</m:Block>
33.       <m:Line timestamp="2007-12-06T02:21:26.163" sequence="1219">18</
     m:Line>
34.       <m:Block timestamp="2007-12-06T02:21:33.274" sequence="2849">N0170
     M00 (don't do anything until operator presses the S key)</m:Block>
35.       <m:Execution timestamp="2007-12-06T02:21:33.274"
     sequence="2850">PAUSED</m:Execution>
36.       <m:Line timestamp="2007-12-06T02:21:33.274" sequence="2851">20</
     m:Line>
```

The program has now completed and the interpreter is now idle:

```
37.       <m:Block timestamp="2007-12-06T02:21:34.714" sequence="2852">N0150
     M2 (end program)</m:Block>
38.       <m:Execution timestamp="2007-12-06T02:21:34.714"
     sequence="2853">IDLE</m:Execution>
39.       <m:Line timestamp="2007-12-06T02:21:34.714" sequence="2854">22</
     m:Line>
40.       <m:Line timestamp="2007-12-06T02:21:34.739" sequence="2856">0</
     m:Line>
41.       <m:Block timestamp="2007-12-06T02:21:53.604" sequence="2857"></
     m:Block>
42.         </Events>
43.     </ComponentEvents>
```

These events represent the machine power transitions:

```
44.     <ComponentStream path="//Machine[@name='machine']//
     Power[@name='Power']" name="Power" uuid="">
45.         <Events>
46.       <m:Power timestamp="2007-12-06T02:20:51.938" sequence="3">OFF</
     m:Power>
47.       <m:Power timestamp="2007-12-06T02:20:59.869" sequence="19">ON</
     m:Power>
48.         </Events>
```

```
49.       </ComponentStream>
50.     </DeviceStream>
51.   </Streams>
52.</m:MTConnectStreams>
```

11. Future Direction

With regard to documentation, we will be completing the following documents: The *Name Service* specification as a guide for LDAP configuration; the *SDK Manual* for application developers; and the *Implementation Guide* for people developing *Agent*s and adapters.

The following areas will be added to the standard in the next versions of the specification:

11.1. Tools

The next modeling area will address the Tool domain. This includes the following items: the current tool or tools being used by the device. The tool changer and placement of the tool. Pre-setters and a unique identification of each tool to coordinate the information between the devices.

11.2. Subscription Requests

Currently the protocol only supports a simple request/response model of communication. To make the protocol more efficient, an application should be able to request a stream of data from the *Agent*. There are some complexities with streaming-based protocols as a result of a HTTP request that will need to be addressed.

11.3. Robotics

A full modeling of robotic devices will be scheduled. The special considerations required by robotics could not be handled in time for release of the draft standard.

11.4. Programable Automation Control

The future revisions of the specification will model devices with multiple controllers, where each controller is responsible for other components of the device.

12. Appendix A

12.1. Samples

SAMPLE Category											
Device	Component	Component name	Sub-Component	Sub-Component name	DataItem type	DataItem name	Stream Element	SubType	Units Refer to Chart 7.1.4	CDATA value	
	Power	power			VOLTAGE	voltage	Voltage		VOLT	floating point	
					AMPERAGE	amperage	Amperage		AMPERE	floating point	
					WATTS	watts	Watts		WATT	floating point	
	Axes	axes			GLOBAL_POSITION	global_positionact	GlobalPosition	ACTUAL	MILLIMETER	floating point	
						global_positioncom		COMMANDED	MILLIMETER	floating point	
					PATH_FEEDRATE	path_feedratefrt	PathFeedrate	ACTUAL	MILLIMETER/SECOND	floating point	
						path_feedratecom		COMMANDED	MILLIMETER/SECOND	floating point	
						path_feedrateovr		OVERRIDE	PERCENT	floating point	
					ACCELERATION	acceleration			MILLIMETER/SECOND^2	floating point	
					VELOCITY	velocity	Velocity		MILLIMETER/SECOND	floating point	
			Linear	X, Y, Z, etc.*	POSITION	Xabs, Yabs, Zabs, etc.	Position	ACTUAL	MILLIMETER	floating point, absolute	
						Xcom, Ycom, Zcom, etc.		COMMANDED	MILLIMETER	floating point	
					ACCELERATION	Xacc, Yacc, Zacc, etc.	Acceleration		MILLIMETER/SECOND^2	floating point	
					VELOCITY	Xvel, Yvel, Zvel, etc.	Velocity		DEGREE/SECOND	floating point	
					LOAD	Xlod, Ylod, etc.	Load		NEWTON	floating point	
					AXIS_FEEDRATE	Xfrt, Yfrt, etc.	AxisFeedrate	ACTUAL	MILLIMETER/SECOND	floating point	
						Xfrtcom, Yfrtcom, etc.		COMMANDED	MILLIMETER/SECOND	floating point	
						Xovr, Yovr, etc.		OVERRIDE	PERCENT	floating point	
			Rotary	A, B, C, etc.*	ANGLE	Adeg, Bdeg, etc.	Angle	ACTUAL	DEGREE	floating point	
						Acom, Bcom, etc.		COMMANDED	DEGREE	floating point	
					ANGULAR_ACCELERATION	Aacc, Bacc, etc.	AngularAcceleration		DEGREE/SECOND^2	floating point	
					ANGULAR_VELOCITY	Avel, Bvel, etc.	AngularVelocity		DEGREE/SECOND	floating point	
					LOAD	Alod, Blod, etc.	Load		NEWTON	floating point	
					AXIS_FEEDRATE	Afrt, Bfrt, etc.	AxisFeedrate	ACTUAL	MILLIMETER/SECOND	floating point	
						Afrtcom, Bfrtcom, etc.		COMMANDED	MILLIMETER/SECOND	floating point	
						Aovr, Bovr, etc.		OVERRIDE	PERCENT	floating point	
			Spindle	S, S2, etc.*	SPINDLE_SPEED	Srpm, S2rpm, etc.	SpindleSpeed	ACTUAL	REVOLUTION/MINUTE	floating point	
						Scom, S2com, etc.		COMMANDED	REVOLUTION/MINUTE	floating point	
						Sovr, S2ovr, etc.		OVERRIDE	PERCENT	floating point	
					LOAD	Slod, S2lod, etc.	Load		NEWTON	floating point	
					PRESSURE	pressure	Pressure		PASCAL	floating point	
					TEMPERATURE	temperature	Temperature		CELSIUS	floating point	
					VIBRATION	vibration	Vibration		HERTZ	floating point	
					OTHER		Other				

* The convention to be used for multiple linear, rotary, and spindle axes having the same axis designation is to index the letter with a number. For this standard the number shall start at 2 (i.e. X, X2, X3, etc. or S, S2, S3, etc.). This convention is in compliance with ISO 841:2001.

12.2. Events

EVENT Category

Device	Component	Component name	Sub-Component	Sub-Component name	DataItem type	DataItem name	Stream Element	CDATA
	Power	power			POWER_STATUS	power	PowerStatus	ON, OFF
	Controller	control			BLOCK	block	Block	text
					CODE	code	Code	text
					EXECUTION	execution	Execution	IDLE, EXECUTING, PAUSED
					LINE	line	Line	integer
					PROGRAM	program	Program	text
					CONTROLLER_MODE	controllermode	ControllerMode	AUTOMATIC, MANUAL, MANUAL_DATA_ENTRY
	Axes	axes	Spindle	S, S2, etc.	DIRECTION	Sdir, S2dir, etc.	Direction	CLOCKWISE, COUNTER_CLOCKWISE
	Device				TRANSFER	transfer	Transfer	IN, OUT
					TICK	tick	Tick	text
	All Components				ALARM	alarm	Alarm	text

13. 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.