MTConnect® Standard
Part 3 – Streams, Events, Samples, and Condition
Version 1.3.0

Prepared for: MTConnect Institute
Prepared by: John Turner
Prepared on: September 30, 2014

MTConnect® Specification and Materials

Table of Contents

1 Overview 1

1.1 MTConnect® Document Structure 2

2 Purpose of This Document 3

2.1 Terminology 3
2.2 Terminology and Conventions 5

3 Streams, Samples, Events, and Condition 6

3.1 Streams Response Header 6
3.2 Streams Structure 7
3.3 `DeviceStream` 9
3.3.1 `DeviceStream` Attributes 10
3.3.2 `DeviceStream` Elements 10
3.4 `ComponentStream` 10
3.4.1 `ComponentStream` Attributes 11
3.4.2 `ComponentStream` Elements 11
3.5 Types and Subtypes of Data Items 11
3.6 Samples and Events 13
3.7 `Samples` 13
3.8 `Sample` 13
3.8.1 Sample attributes: 14
3.8.2 `Time Series` 15
3.8.3 `Time Series` attributes: 16
3.8.4 `Sample` XML Element Tag Names 16
3.8.5 Extensibility 20
3.9 `Events` 20
3.10 `Event` 20
3.10.1 Event attributes: 21
3.10.2 Discrete Events 21
3.10.3 Event Element Tag Names 21
3.10.4 Interface Event Element Tag Names 29
3.11 `Condition` 29
3.11.1 Types of Condition 30
3.11.2 `Condition` Attributes 31
3.11.3 `Condition` Contents - CDATA 31
3.11.4 `Condition` Types 32
3.11.5 `Condition` Examples 33
~~3.12 Alarms~~ DEPRECATED: See `Condition` 35

Appendices 37

A. Bibliography 37

B. Annotated XML Examples 39

B.1. Example of a `current` Request 39

Table of Figures

Figure 1: Header Schema Diagram for `MTConnectStreams` 6
Figure 2: Streams Schema Diagram 7
Figure 3: `Streams` Example Structure 8
Figure 4: `DeviceStream` Schema 9
Figure 5: `ComponentStream` Schema 10
Figure 6: `Sample` Schema 14
Figure 7: `Time Series` Schema 16
Figure 8: `Event` Schema 20
Figure 9: `Condition` Schema 30

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 industries, 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® focused on a limited set of the characteristics that were selected based on the fact that they could have an immediate effect on the efficiency of operations. Subsequent versions of the standard have and will continue to add additional functionality to more completely define the manufacturing environment.

1.1 MTConnect® Document Structure

The MTConnect® specification is subdivided using the following scheme:

Part 1: Overview and Protocol

Part 2: Components and Data Items

Part 3: Streams, Events, Samples, and Condition

Part 4: Assets

These four documents are considered the basis of the MTConnect Standard. Information applicable to basic machine and device types will be included in these documents. Additional parts to the standard will be added to provide information and extensions to the standard focused on specific devices, components, or technologies considered requiring separate emphasis. All information specific to the topic of each additional part **MUST** be included within that document even when it is subject matter of one of the base parts of the standard.

Documents will be named (file name convention) as follows:

MTC_Part_<Number>_<Description>.doc.

For example, the file name for Part 2 of the standard is MTC_Part_2_Components.doc.

All documents will be developed in Microsoft® Word format and released in Adobe® PDF format.

2 Purpose of This Document

The four base MTConnect® documents are intended to:

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

Part 1 of the MTConnect Standard provides an overview of the MTConnect Architecture and the Protocol; including communications, fault tolerance, connectivity, and error handling requirements.

Part 2 of the MTConnect® standard focuses on the data model and description of the information that is available from the device. The descriptive data defines how a piece of equipment should be modeled, the structure of the component hierarchy, the names for each component (if restricted), and allowable data items for each of the components.

Part 3 of the MTConnect standard focuses on the data returned from a `current` or `sample` request (for more information on these requests, see Part 1). This section covers the data representing the state of the machine.

Part 4 of the MTConnect® standard provides a semantic model for entities that are used in the manufacturing process, but are not considered to be a device nor a component. These entities are defined as MTConnect® Assets. These assets may be removed from a device without detriment to the function of the device, and can be associated with other devices during their lifecycle. The data associated with these assets will be retrieved from multiple sources that are responsible for providing their knowledge of the asset. The first type of asset to be addressed is Tooling.

2.1 Terminology

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

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

Alarm An alarm indicates an event that requires attention and indicates a deviation from normal operation. Alarms are reported in MTConnect as `Condition`.

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

Attribute A part of an XML element that provides additional information about that XML element. For example, the `name` XML 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 `<Message ...>This is some text</Message>`.

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 may be composed of a set of components that provide data to the application. The device is a separate entity with at least one component or data item providing information about the device.

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*.

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

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

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

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

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

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

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

Results	A general term for the `Samples`, `Events`, and `Condition` contained in a `ComponentStream` as a response from a `sample` or `current` request.
Sample	A sample is a data point from within a continuous series of data points. An example of a Sample is the position of an axis.
Socket	When used concerning inter-process 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, Samples,` and `Condition` 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 inter-process communication. It is based on the IP stack (Internet Protocol) and provides the flow-control and reliable transmission layer on top of the IP routing infrastructure.
URI	Universal Resource Identifier. This is the official name for a web address as seen in the address bar of a browser.
UUID	Universally unique identifier.
XPath	XPath is a language for addressing parts of an XML Document. See the XPath specification for more information. http://www.w3.org/TR/xpath
XML	Extensible Markup Language. http://www.w3.org/XML/
XML Schema	The definition of the XML structure and vocabularies used in the XML Document.
XML Document	An instance of an XML Schema which has a single root XML element and conforms to the XML specification and schema.
XML Element	An element is the central building block of any XML Document. For example, in MTConnect® the Device XML element is specified as **`<Device>...</Device>`**
XML `NMTOKEN`	The data type for XML identifiers. It **MUST** start with a letter, an underscore "_" or a colon ":" and then it **MUST** be followed by a letter, a number, or one of the following ".", "-", "_", ":". An `NMTOKEN` cannot have any spaces or special characters.

2.2 Terminology and Conventions

Please refer to Section 2 of *Part 1, Overview and Protocol* for XML Terminology and Documentation conventions.

3 Streams, Samples, Events, and Condition

The MTConnect *Agent* collects data from various sources and delivers it to applications in response to `Sample` or `Current` requests. (See *Protocol* section in *Part 1*.) All the data is collected into streams and organized by device and then by component. A component stream has three parts: `Samples`, `Events`, and `Condition`.

`Samples` are point-in-time readings from a component reporting what the value is at that instant.

`Events` change state to a limited set of values or represent a message. It is assumed that an `event` remains at a state until the next occurrence of the `event` occurs; it cannot have any intermediate values between the reported values. The following are examples of `Events`: `Block`, `Execution`, `Message` etc.

A `Condition` communicates the device's health and ability to function. It can be one of `UNAVAILABLE`, `NORMAL`, `WARNING`, or `FAULT` and there can be multiple active conditions at one time; whereas a `sample` or `event` can only have a single value at one point in time.

3.1 Streams Response Header

Every MTConnect® response **MUST** contain a header as the first XML element below the root element of any MTConnect® XML Document sent back to an application. (See *Header* in *Part 1, Section 4.5* for details on the Header structure)

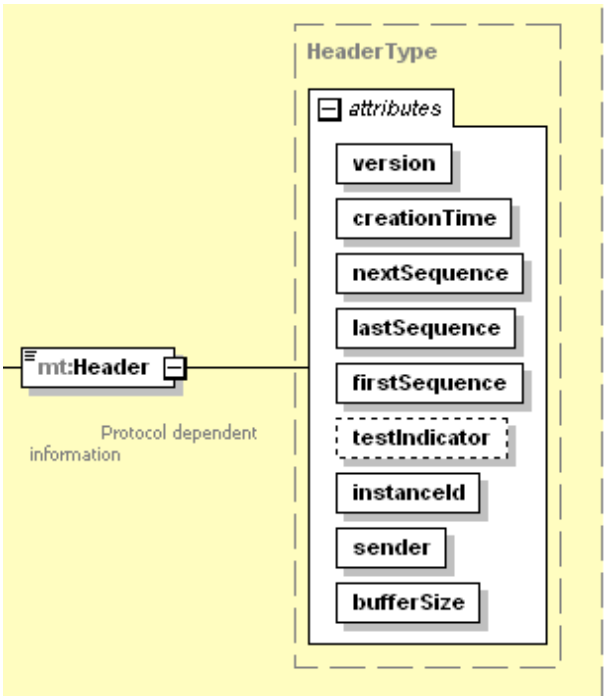

Figure 1: Header Schema Diagram for `MTConnectStreams`

3.2 Streams Structure

A `Streams` XML element is the high level container for all device streams. Its function is to contain `DeviceStream` sub-elements. There **MUST** be no attributes or other type XML elements within the `Streams` element.

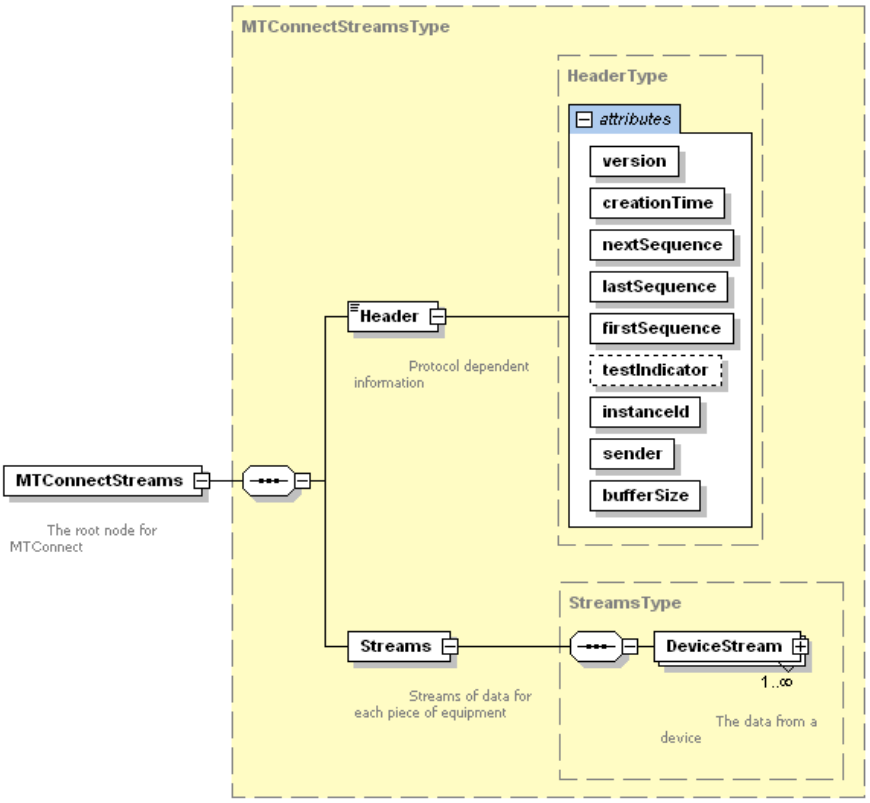

Figure 2: Streams Schema Diagram

Elements	Description	Occurrence
`DeviceStream`	The stream of `Samples`, `Events`, and `Condition` for each device.	1..INF

`Streams` **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 will be delivered for this request.

The following diagram illustrates the structure of the `Streams` with some `Samples`, `Events`, and `Condition` at the lowest level:

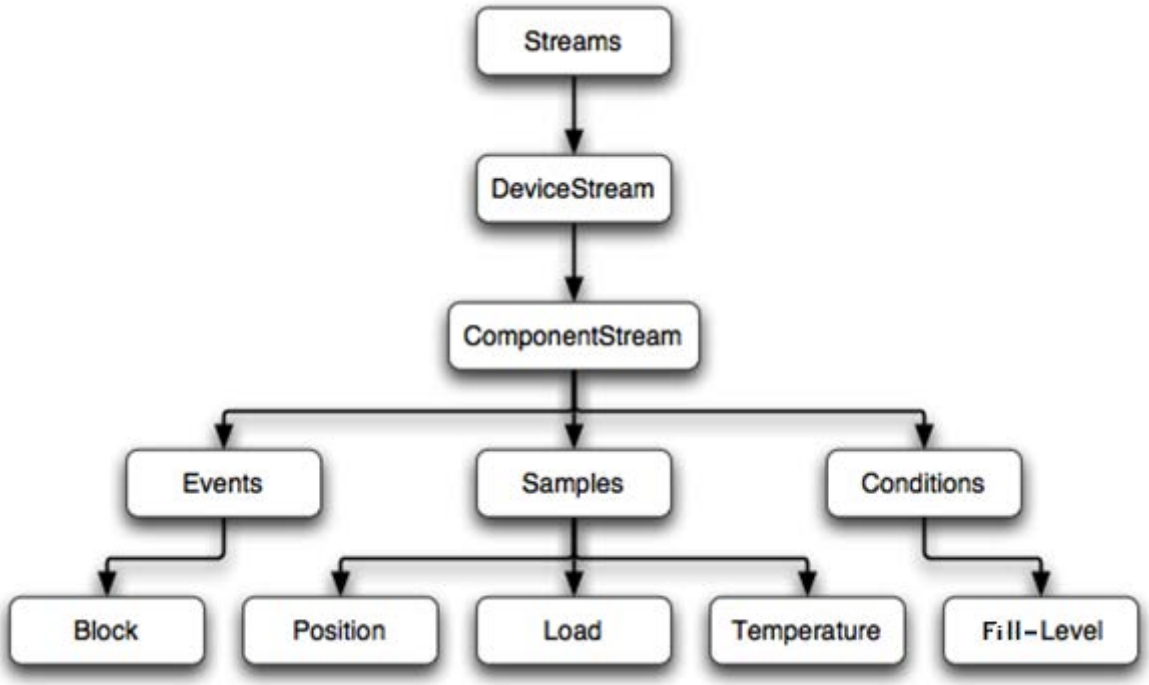

Figure 3: `Streams` Example Structure

Below is an example XML Document response for an *Agent* with two devices, mill-1 and mill-2. The data is reported in two separate device streams.

```
<MTConnectStreams …>
  <Header … />
  <Streams>
    <DeviceStream name="mill-1" uuid="1">
      <ComponentStream component="Device" name="mill-1" componentId="d1">
        <Events>
          <Availability dataItemId="avail1" name=="avail" sequence="5"
              timestamp="2010-04-06T06:19:35.153141">AVAILABLE</Availability>
        </Events>
      </ComponentStream>
    </DeviceStream>
    <DeviceStream name="mill-2" uuid="2">
      <ComponentStream component="Device" name="mill-2" componentId="d2">
        <Events>
          <Availability dataItemId="avail2" name="avail" sequence="15"
              timestamp="2010-04-06T06:19:35.153141">AVAILABLE</Availability>
        </Events>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```

The sequence numbers are unique across the two devices in the example above. The applications **MUST NOT** assume that the event and sample sequence numbers are strictly in sequence. All sequence numbers **MAY NOT** be included. An example of this case would occur when a `Path` argument is provided and all the `Samples`, `Events`, and `Condition` are not selected or when the *Agent* is supporting more than one device and data from only one device is requested. Refer to *MTConnect® Part 1, Overview and Protocol, Section 5: Protocol* for more information.

3.3 `DeviceStream`

A `DeviceStream` is created to hold the device-specific information like the name of the device and its UUID, 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.

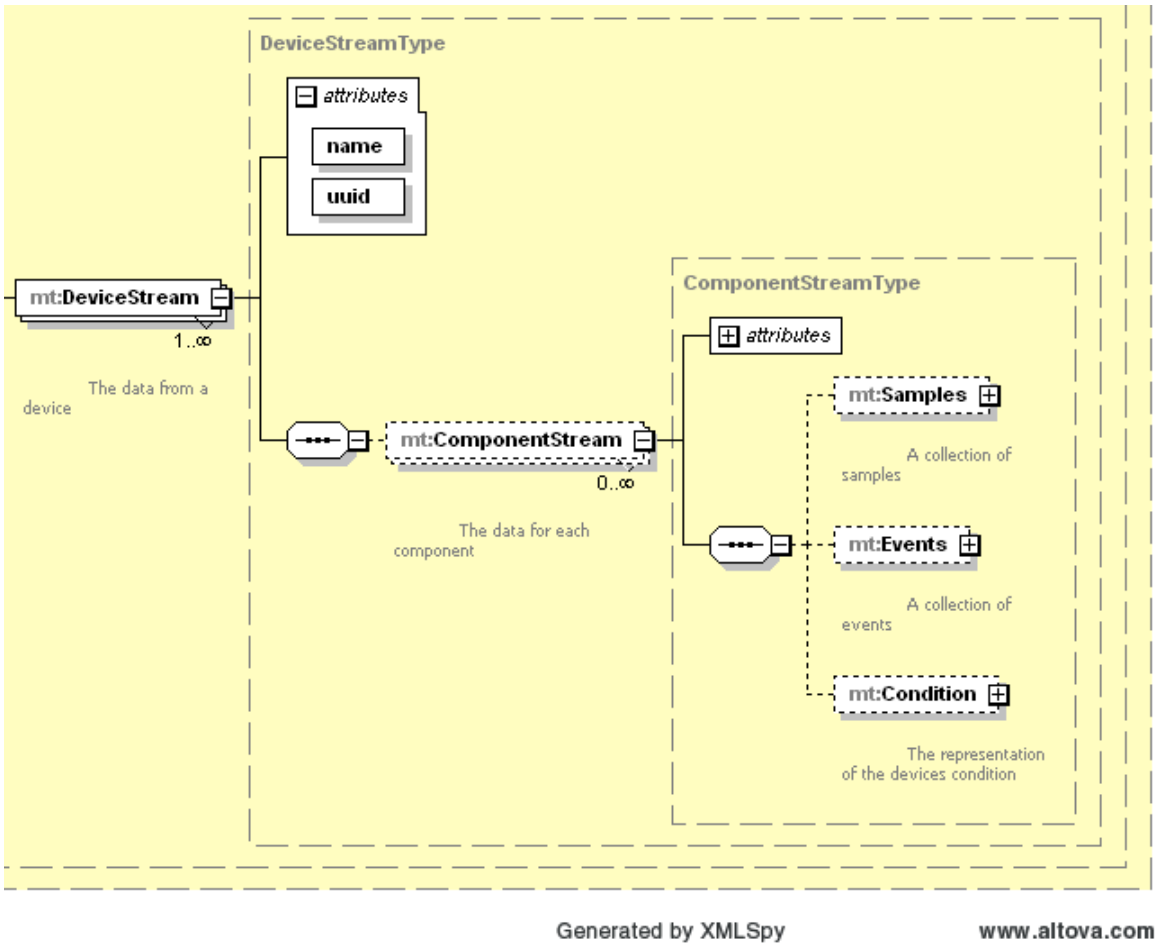

Figure 4: `DeviceStream` Schema

3.3.1 `DeviceStream` Attributes

Attributes	Description	Occurrence
`name`	The device's name. An NMTOKEN XML type.	1
`uuid`	The device's unique identifier	1

3.3.2 `DeviceStream` Elements

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

3.4 `ComponentStream`

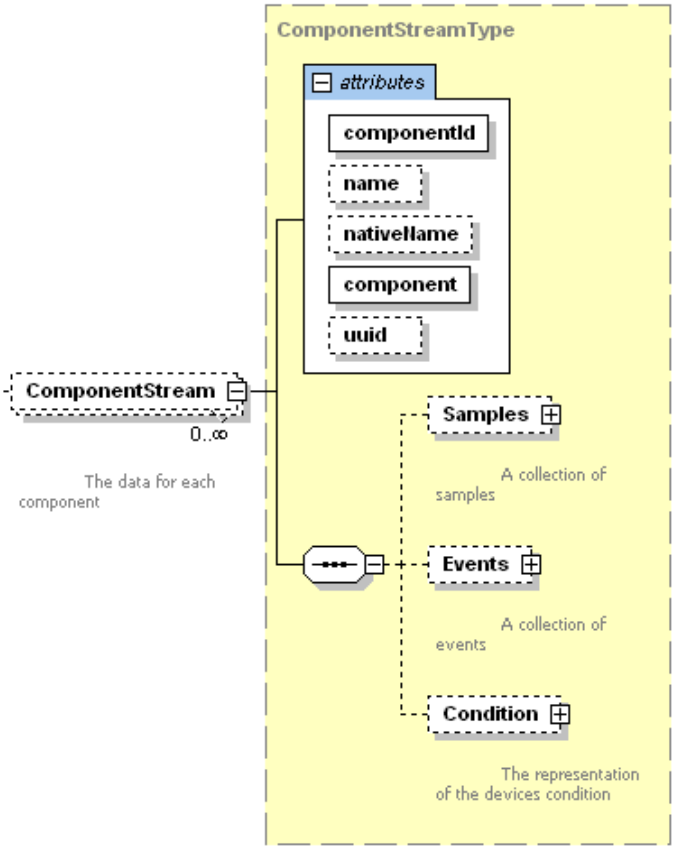

Figure 5: `ComponentStream` Schema

A `ComponentStream` is similar to the `DeviceStream`. It contains the information specific to the component within the `Device`. The `uuid` only needs to be specified if the Component has a `uuid` assigned.

3.4.1 `ComponentStream` Attributes

Attribute	Description	Occurrence
`name`	This component's name within the device. An NMTOKEN XML type.	0..1
`nativeName`	The name the device manufacturer assigned to the component. If the native name is not provided it **MUST** be the `name`.	0..1
`component`	The XLM element name for the component	1
`uuid`	The component's unique identifier	0..1
`componentId`	Corresponds to the `id` attribute of the component in the probe request (Refer to Probe in Part 1).	1

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

3.4.2 `ComponentStream` Elements

Element	Description	Occurrence
`Events`	The events for this component stream	0..1
`Samples`	The samples for this component	0..1
`Condition`	The condition of the device.	0..1

3.5 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, Event, or Condition` with the following rules:

- The type name will be all in capitals with an underscore (_) between words.
- The XML 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 `DOOR_STATE` is `DoorState`, `POSITION` is `Position`, and `ROTARY_VELOCITY` is `RotaryVelocity.`

The following example shows the transformation between the `DataItem` name as returned in a `Probe` request and the corresponding structured data returned in a `Stream` XML element returned from a `Current` or `Sample` request. In the Probe request, each `DataItem` defines its `DataItem` `type`, `category`, and (if applicable) the `subType`.

The probe request will return the response below.

```
<Path name="path" id="p1">
   <DataItems>
      <DataItem type="PATH_POSITION" category="EVENT" id="p2"
         subType="ACTUAL" name="Zact"/>
      <DataItem type="CONTROLLER_MODE" category="EVENT" id="p3"
         name="mode" />
      <DataItem type="PROGRAM" category="EVENT" id="p4" name="program" />
      <DataItem type="EXECUTION" category="EVENT" id="p5"
         name="execution" />
      <DataItem type="BLOCK" category="EVENT" id="p6" name="block" />
   </DataItems>
</Path>
```

The transformation from the Probe *(as defined in Part 1 of the standard)* to the Current or Sample will occur per the example below. This example also illustrates how the subType is placed in the ComponentStream. In the Current and Sample request, data items will be returned in the ComponentStream grouped into their respective categories. Also note how the CONTROLLER_MODE was changed to ControllerMode in the current request below.

```
<ComponentStream componentId="p1" component="Path"
    name="path">
   <Events>
      <PathPosition dataItemId="p2" timestamp="2009-03-
         04T19:45:50.458305" subType="ACTUAL" name="Zact"
         sequence="150651130">7.02</PathPosition>
      <Block dataItemId="p6" timestamp="2009-03-04T19:45:50.458305"
         name="block" sequence="150651134">x0.371524 y-0.483808</Block>
      <ControllerMode dataItemId="p3" timestamp="2009-02-
         26T02:02:35.716224" name="mode"
         sequence="182">AUTOMATIC</ControllerMode>
   </Events>
</ComponentStream>
```

3.6 Samples and Events

All Samples and Events values **MUST** be able to provide UNAVAILABLE as a valid value when the data source is not connected or the data source is unable to retrieve information. The UNAVAILABLE value will persist until the connection is restored and a new value can be retrieved. This state does not imply the device is no longer operational, it only implies that the state cannot be determined.

3.7 Samples

The Samples XML element **MUST** contain at least one Sample element. The Samples MXL element acts only as a container for all the Sample XML elements to provide a logical structure to the XML Document.

Element	Description	Occurrence
Sample	The sub-element of Samples for this component stream	1..INF

3.8 Sample

A Sample is an abstract type. This means there will never be an actual element called Sample, but any XML element that is a sub-type of Sample can be used as a sub-element of Samples. Examples of sample sub-types are Position, Load, and Angle. Sample types **MUST** have numeric values.

If two adjacent samples for the same Component and DataItem 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.

For DataItems containing an attribute for duration, the timestamp associated with the sample references the time the sample value was reported or the statistics were computed, NOT the time the interval began. The time the interval began can be computed by subtracting the duration from the timestamp. Two samples can have overlapping intervals as in the case where statistics are computed at various frequencies.

For example, a one minute average and a five minute average can both have the same start time (Lets say 05:10:00), but their timestamps will be 05:11:00 with a duration of 60 seconds for the one minute average and a timestamp of 05:15:00 with a duration of 300 seconds for the five minute average. This allows for varying statistical methods to be applied with different interval lengths without having duplicate timestamps and durations. If a statistical data item does not report for a period greater than the previous duration, it can be assumed the computed value has not changed since the last value.

The same concepts are used for time-series samples as well where the timestamp of the series is set to the time the last value was recorded and the timestamp minus the duration is the time the

first sample was recorded. See *Part 3, Section 3.8.2* for more information on Time Series samples.

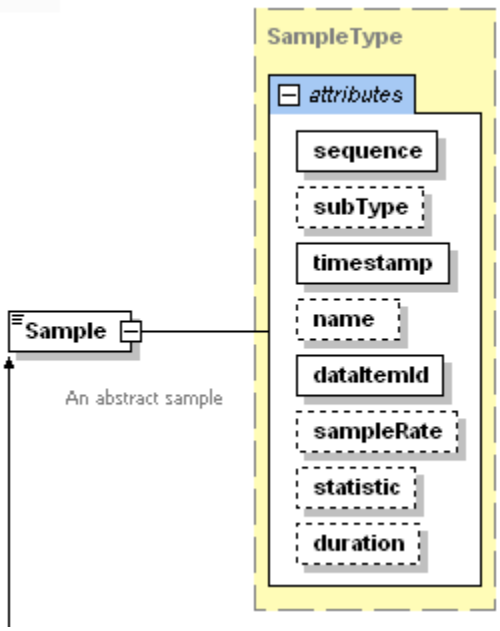

Figure 6: `Sample` Schema

3.8.1 Sample attributes:

Attribute	Description	Occurrence
`name`	The name **MUST** match the `name` of the `DataItem` this `Sample` is associated with. It **MUST** be an NMTOKEN XML type.	0..1
`sequence`	The sequence number of this event. The value **MUST** be represented as an unsigned 64 bit with valid values from 1 to 2^64-1.	1
`timestamp`	The time the sample value was reported or the statistics were computed. The timestamp **MUST** always represent the end of the collection interval when a `duration` or a *TimeSeries* is provided. The most accurate time available to the device **MUST** be used for the timestamp.	1
`dataItemID`	The `id` attribute of the corresponding data retrieved in the probe request.	1
`subType`	The sub-type of the `DataItem`	0..1

Attribute	Description	Occurrence
sampleRate	The rate at which successive samples of a DataItem are recorded. Sample rate is expressed in terms of samples per second. If the sample rate is smaller than one, the number can be represented as a floating point number. For example, a rate 1 per 10 seconds would be 0.1 The sampleRate attribute **MUST** be included in the *TimeSeries* streams element if it is not constant OR if it is not in the DataItem. If the sampleRate is constant it **MAY** be placed in the DataItem and does not need to be repeated in the streams element.	0..1
statistic	The type of statistical calculation specified for the DataItem	0..1
duration	The time elapsed since the statistic calculation was last reset	0..1

A Sample **MUST** contain CDATA as the content between the element tags. A position is formatted like this:

```
1. <Position sequence="112" timestamp="2007-08-09T12:32:45.1232" name="Xabs"
      dataItemId="10">123.3333</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.

3.8.2 Time Series

A Time Series is a Sample which includes multiple readings of a DataItem taken at a specified sample rate. A time series can be used for collecting high frequency samples of a DataItem and then providing the series of samples to an application as a single DataItem. A time series contains the same attributes as a Sample, plus one additional attribute sampleCount. For a Time Series, sampleRate defines the time period (frequency) for the collection of each reading of the DataItem and sampleCount defines the total number of readings being transmitted. The CDATA **MUST** be a series of floating point numbers. The number of readings **MUST** match the sampleCount. The units for a Time Series **MUST** be the same as specified for the DataItem.

The XML element of the Sample for a DataItem with an attribute of representation will be the transformation of the DataItem type by capitalizing the first character of each word and then removing the underscore and adding the representation type. For example, ANGULAR_VELOCITY with representation defined as TimeSeries **MUST** be AngularVelocityTimeSeries. If representation is not defined or it is VALUE, then the transformation **MUST** be AngularVelocity.

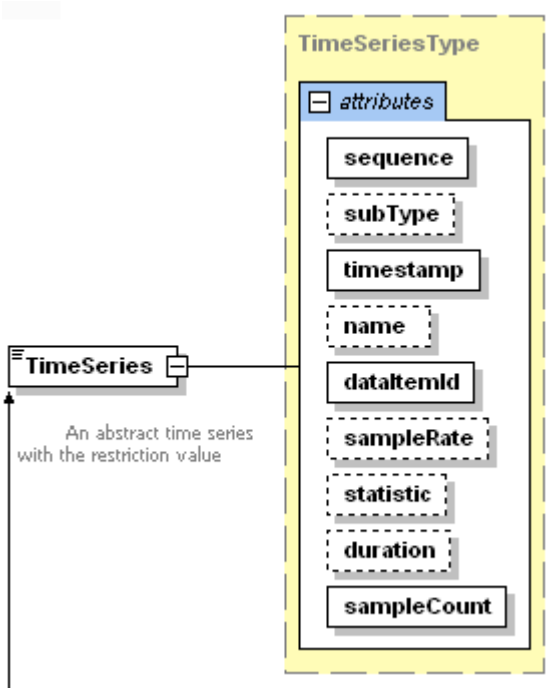

Figure 7: `Time Series` Schema

3.8.3 `Time Series` attributes:

Attribute	Description	Occurrence
`sampleCount`	The number of readings of a `DataItem` provided in a Time Series.	1

3.8.4 `Sample` XML Element Tag Names

The following is a list of all the XML elements that can be placed in the `Samples` section of the `ComponentStream`. All `Samples` have a numeric value as the CDATA or `UNAVAILABLE` if the data is in an indeterminate state.

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

`AccumulatedTime` The accumulated time associated with a component. The `AccumulatedTime` **MUST** have a numeric value and **MUST** be reported in `SECOND`.

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

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 An angular velocity represents the rate of change in angle. An `AngularVelocity` **MUST** always be reported in `DEGREE/SECOND` 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 linear axis of the tool relative to the workpiece[1]. An `AxisFeedrate` **MUST** always be reported in `MILLIMETER/SECOND` ~~or `PERCENT` for `OVERRIDE`~~ and **MUST** always have a numeric CDATA value as a floating point number.

ClockTime The reading of a timing device at a specific point in time. The time **MUST** have a value reported in W3C ISO 8601 format of `YYYY-MM-DDThh:mm:ss.ffff`

Concentration Percentage of one component within a mixture of components. The `Concentration` **MUST** have a value reported in `PERCENT`.

Conductivity The ability of a material to conduct electricity. The `Conductivity` **MUST** have a value reported in `SIEMENS/METER.`

Displacement The displacement measured as the change in position of an object. The `Displacement` **MUST** have a value reported in `MILLIMETER`.

ElectricalEnergy The measurement of electrical energy consumed by a component. `ElectricalEnergy` **MUST** have a value reported in `WATT_SECOND`.

Flow The rate of flow of a fluid. The `Flow` **MUST** have a value reported in `LITER/SECOND`.

Frequency The rate measurement of the number of occurrences of a repeating event per unit time. The `Frequency` **MUST** have a numeric value and **MUST** be reported in `HERTZ`.

FillLevel The measurement of the amount of a substance remaining compared to the planned maximum amount of that substance. The `FillLevel` **MUST** be reported in PERCENT.

Length The length of an object, usually a piece of material or stock. The Length **MUST** be report in MILLIMETER.

[1] From ASME B5.54 - 2005

LinearForce The measurement of the amount of push or pull introduced by an actuator or exerted on an object. The `LinearForce` **MUST** be reported in NEWTON.

Load The measurement of the percent of the standard rating of a device. The `Load` **MUST** always be reported in `PERCENT` and **MUST** always have a numeric CDATA value as a floating point number.

Mass The measurement of the mass of an object(s) or an amount of material. The `Mass` **MUST** be reported in KILOGRAM.

PathFeedrate Path Feedrate is defined as the rate of motion of the feed path of the tool relative to the workpiece[2]. A `PathFeedrate` **MUST** always be reported in `MILLIMETER/SECOND` **MUST** always have a numeric CDATA value as a floating point number.

PathPosition The program position as given in 3 dimensional space. This position **MUST** default to `WORK` coordinates, if the `WORK` coordinates are defined, and **MUST** be given as a space delimited vector of floating point numbers given in `MILLIMETER_3D` units. The `PathPosition` will be given in the following format and **MUST** be listed in order X, Y, and Z:
`<PathPosition …>10.123 55.232 100.981</PathPosition>`
Where X = 10.123, Y = 55.232, and Z=100.981.

PH The measure of acidity or akalinity. The `PH` **MUST** be a numeric value and **MUST** be provided in `PH`.

~~**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.~~ DEPRECATED in Release 1.1

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. The default coordinate system for `Position` **MUST** be `MACHINE_COORDINATES`.

PowerFactor The measurement of the ratio of real power flowing to a load to the apparent power in that AC circuit. The `PowerFactor` **MUST** be a numeric value and **MUST** be provided in `PERCENT`.

Pressure The force per unit area exerted by a gas or liquid. `Pressure` **MUST** be a numeric value and **MUST** be provided in `PASCAL`.

[2] From ASME B5.54 - 2005

Resistance The measure of the degree to which an object opposes an electrical current through it. The Resistance **MUST** be a numeric value and **MUST** be provided in OHM.

RotaryVelocity The rate of rotation of a rotary axis. A RotaryVelocity speed **MUST** always be reported in REVOLUTION/MINUTE.

SoundLevel The measure of acoustic sound level or sound pressure level. The SoundLevel **MUST** be provided in DECIBEL.

~~**SpindleSpeed** The rate of rotation of a machine spindle [3]. A spindle speed **MUST** always be reported in REVOLUTION/MINUTE and **MUST** always have a numeric CDATA value as a floating point number.~~ DEPRICATED in Release 1.2. See RotaryVelocity.

Strain The measured amount of deformation per unit length of an object. Strain **MUST** be reported as PERCENT.

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

Tilt The measured amount of angular displacement of an object. Tilt **MUST** be reported as MICRO_RADIAN.

Torque The turning force exerted on an object or by an object. Torque **MUST** be reported in units of NEWTON_METER and **MUST** have a numeric CDATA value as a floating point number.

Velocity A velocity represents the rate of change in position along one or more linear 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.

Viscosity The measurement of a fluid's resistance to flow. Viscosity **MUST** be reported as PASCAL_SECOND.

Voltage The measurement of electrical potential between two points. The Voltage **MUST** have a numeric value and **MUST** be reported in VOLT.

VoltAmpere The measurement of apparent power in an electrical circuit, equal to the product of the RMS voltage and RMS current. The VoltAmpere **MUST** have a numeric value and **MUST** be reported in VOLT_AMPERE.

VoltAmpereReactive The measurement of reactive power in an AC electrical circuit. The VoltAmpereReactive **MUST** have a numeric value and **MUST** be reported in VOLT_AMPERE_REACTIVE.

[3] From ASME B5.54 - 2005

Wattage The electrical power (volt-amps) consumed or dissipated by an electrical circuit or device. The Wattage **MUST** have a numeric value and **MUST** be reported in WATT.

3.8.5 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.

3.9 Events

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

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

3.10 Event

An Event is an abstract type. This means there will never be an actual element called Event, but any XML element that is a sub-type of Event can be used in place of Event. Examples of event sub-types are Block, Execution, and Line. Event types **MAY** have values defined by a controlled vocabulary as specified in *Section 3.10.2* or **MAY** contain a character string representing data provided by the device.

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 Availability transitions from UNAVAILABLE to AVAILABLE, there is no intermediate state that can be inferred. Therefore, most Events have a controlled vocabulary as their content.

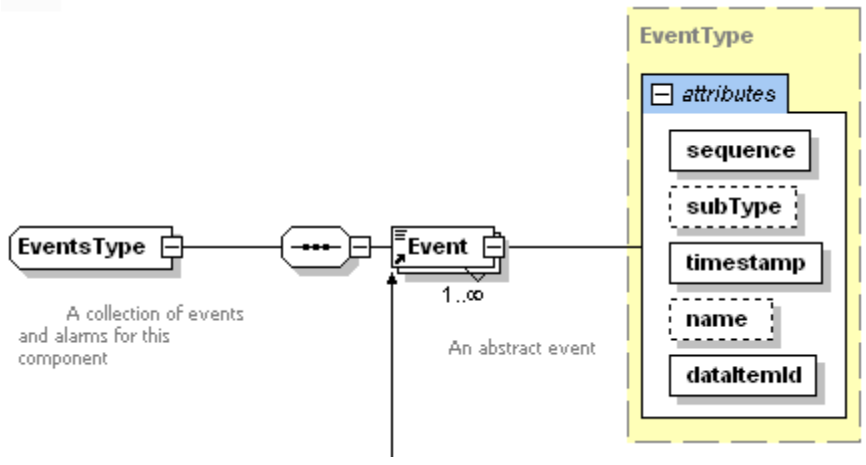

Figure 8: Event Schema

3.10.1 **Event attributes:**

Attribute	Description	Occurrence
`name`	The name **MUST** match the `name` of the `DataItem` this sample is associated with. It **MUST** be an NMTOKEN XML type.	0..1
`sequence`	The sequence number of this event. The value **MUST** be represented as an unsigned 64 bit with valid values from 1 to 2^64-1.	1
`timestamp`	The timestamp of the sample. The most accurate time available to the device **MUST** be used for the `timestamp`	1
`dataItemID`	The `id` attribute of the corresponding data retrieved in the probe request.	1
`subType`	The sub-type of the `dataItem`	0..1

3.10.2 **Discrete Events**

If the `representation` of the `DataItem` is specified as `DISCRETE` it indicates that each occurrence of the data may have the same value as the previous occurrence of the data. There is no reported state change between occurrences of the data.

In this case, duplicate occurrences of the same data value **SHOULD NOT** be suppressed.

Examples of a `DISCRETE` data type would be a Parts Counter that reports the completion of each part, versus the accumulation of parts. Also, `Message` does not typically have a reset state and may re-occur each time an event specific message is triggered.

The XML element of the `Event` for a `DataItem` with an attribute of `representation` will be the transformation of the `DataItem` type by capitalizing the first character of each word and then removing the underscore and adding the representation type. For example, `PART_COUNT` with `representation` defined as `DISCRETE` **MUST** be `PartCountDiscrete`. If `representation` is not defined or it is VALUE, then the transformation **MUST** be `PartCount`.

3.10.3 **Event Element Tag Names**

The `Event` XML elements represent the state of various device attributes. The following is a list of all the event elements that may be placed within the `Events` section of the `ComponentStream`.

`ActiveAxes` The set of axes being controlled by a `Path`. The value **MUST** be a space delimited set of axes names. For example:
`<ActiveAxes …>X Y Z C</ActiveAxes>`
If this is not provided, it **MUST** assumed the `Path` is controlling all the axes.

`ActuatorState` An actuator state represents a device for moving or controlling a mechanism or system. The CDATA **MUST** be as follows:

Value	Description
ACTIVE	The actuator is operating or active
INACTIVE	The actuator is not operating or inactive

AxisFeedrateOverride A percentage override to adjust the feed rate of a linear axis. The CDATA **MUST** be a numeric value, either an integer or decimal floating point number. The subType specifies the specific feedrate that has been overridden.

AxisState An indicator of the current axis homed position and motion. The CDATA **MUST** be as follows:

Value	Description
HOME	The axis is in its home position.
TRAVEL	The axis is in motion
STOPPED	The axis is stopped

Availability Represents the component's ability to communicate its availability. This **MUST** be provided for the device and **MAY** be provided for all other components.

Value	Description
AVAILABLE	The component is available.
UNAVAILABLE	The component is not available.

AxisCoupling Describes the way the axes will be associated to each other. This is used in conjunction with COUPLED_AXES to indicate the way they are interacting**.**

Value	Description
TANDEM	The axes are physically connected to each other and **MUST** operate as a single unit.
SYNCHRONOUS	The axes are coupled and are operating together in lockstep.
MASTER	The axis is the master of the CoupledAxes
SLAVE	The axis is a slave of the CoupledAxes

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.~~ **DEPRECATED in Rel. 1.1.** Duplicates Block.

ChuckInterlock An interlock that prevents the chuck from being operated. The subType specifies the operation that is currently blocked. The CDATA **MUST** be as follows:

Value	Description
ACTIVE	The chuck cannot be operated.
INACTIVE	The chuck can be operated.

ChuckState The open closed state of the chuck. The CDATA **MUST** be as follows:

Value	Description
OPEN	The chuck is open to the point of a positive confirmation
CLOSED	The chuck is closed to the point of a positive confirmation
UNLATCHED	The chuck is not closed to the point of a positive confirmation and not open to the point of a positive confirmation

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

Value	Description
AUTOMATIC	The controller is configured to automatically execute a program.
SEMI_AUTOMATIC	The controller is operating in a single cycle mode.
MANUAL	The controller is under manual control by the operator.
MANUAL_DATA_INPUT	The operator can enter operations for the controller to perform. There is no current program being executed.
EDIT	The controller is currently editing a program in the foreground.

Value	Description
SINGLE_BLOCK	The machine is executing single block or instruction.

CoupledAxes As a Linear or Rotary axis data item, refers to the set of associated axes to be used in conjunction with AxisCoupling. The value will be a space delimited set of axes names. For example:
<CoupledAxes …>Y1 Y2</CoupledAxes >

Direction A Direction indicates the direction of rotation. The CDATA MUST be as follows:

Value	Description
CLOCKWISE	The rotary component is rotating in a clockwise fashion using the right hand rule.
COUNTER_CLOCKWISE	The rotary component is rotating in a counter clockwise fashion using the right hand rule.
POSITIVE	A linear component moving in the direction of increasing position value
NEGATIVE	A linear component moving in the direction of decreasing position value

DoorState A DoorState represents an opening that can be opened or closed. The CDATA **MUST** be as follows:

Value	Description
OPEN	The door is open to the point of a positive confirmation
CLOSED	The door is closed to the point of a positive confirmation
UNLATCHED	The door is not closed to the point of a positive confirmation and not open to the point of a positive confirmation

EndOfBar A YES/NO value indicating that the end of bar has been reached.

Value	Description
YES	The end of bar has been reached
NO	The end of bar has not been reached

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

Value	Description
READY	The controller is ready to execute. It is currently idle.
ACTIVE	The controller is actively executing an instruction.
INTERRUPTED	The operator has paused execution of the controller and the program is waiting to be continued.
STOPPED	The program has been stopped without completion and cannot be resumed. The stop was not the intent of the programmer and was initiated by the machine or operator.
FEED_HOLD	The controller has is in a feed hold and motion has been stopped.
PROGRAM_STOPPED	The program has been stopped.
PROGRAM_COMPLETED	The program has completed execution.
PROGRAM_OPTIONAL_STOP	The program has been intentionally optionally stopped using an M01 or similar code.

EmergencyStop The emergency stop state of the machine, device, or controller path. The CDATA MUST be one of the following:

Value	Description
ARMED	The circuit is complete and the device is operating.
TRIGGERED	The circuit is open and the device **MUST** cease operation.

~~**FeedrateOverride** A percentage override to adjust the surface feed rate of the tool for this path a linear axis. The CDATA **MUST** be a numeric value, either an integer or decimal floating point number. The subType specifies the specific feedrate that has been overridden.~~

FunctionalMode The overall status of the device or component for production. The CDATA MUST be one of the following:

Value	Description

Value	Description
PRODUCTION	The device is ready for automated production.
SETUP	The device is being setup for a new part type.
TEARDOWN	The current setup is being removed from the device.
MAINTENANCE	The device is being repaired or routine service is being performed.
PROCESS_DEVELOPMENT	The device is being used to prove-out a new process.

InterfaceState The connection state of the interface. This indicates if the entire component is active. The CDATA **MUST** be as follows:

Value	Description
ENABLED	The interface is currently operational.
DISABLED	The interface is not operational.

Line This event refers to the optional program line number. For example in RS274/NGC, the line number begins with an N and is followed by 1 to 5 digits (0 – 99999). If there is not an assigned line number in the programming system as in RS274, the line number will refer to the position in the executing program. The line number **MUST** be an alpha-numeric value.

Message A text notification. Format **MAY** be any valid text string.

OperatorId An identifier of the person operating the device.

PalletId This is a reference to an identifier for the current pallet available at the device.

PartCount The number of parts produced. This will not be counted by the agent and **MUST** only be supplied if the controller provides the count.

PartId This is a reference to an identifier for the current part being machined. It is a placeholder for now and can be used at the discretion of the implementation.

PathMode The `PathMode` is provided for devices that are controlling multiple motion paths and their associated axes. When `PathMode` is not provided, it **MUST** be assumed to be `INDEPENDENT`.

Value	Description

Value	Description
INDEPENDENT	The path is operating independently and without the influence of another path.
MASTER	The path provides the reference motion from which a Synchronous or Mirror Path will follow
SYNCHRONOUS	The path and its associated axes are operating synchronously with the Master path.
MIRROR	The path and its associated axes are mirroring the Master path.

PathFeedrateOverride A percentage override to adjust the feed rate of this path. The CDATA **MUST** be a numeric value, either an integer or decimal floating point number. The subType specifies the specific feedrate that has been overridden.

~~**PowerStatus** Power status **MUST** be either ON or OFF.~~ DEPRECATED in Rel. 1.1

~~Value~~	~~Description~~
~~ON~~	~~The power to the component is ON.~~
~~OFF~~	~~The power to the component is OFF.~~

PowerState Power state of a device or component. DEPRECATION WARNING: **MAY** be deprecated in the future.

Value	Description
ON	The power to the component is ON.
OFF	The power to the component is OFF.

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

ProgramEdit This refers to the controls program editing states. On many controls, a program can be edited while another program is currently being executed. The CDATA **MUST** be as follows:

Value	Description
ACTIVE	A program is currently being edited.

Value	Description
READY	The controller is capable of editing a program, but it is not currently editing a program.
NOT_READY	The controller is in a state where it cannot edit a program.

ProgramEditName The name of the program being edited. This is used in conjunction with PROGRAM_EDIT when in ACTIVE state.

ProgramComment The latest active comment in the executing program. A comment is non-executable code within the instruction stream.

ProgramHeader The header section of the current executing program. The content **SHOULD** be limited to 512 bytes.

RotaryMode The mode in which the rotary axis is currently operating. The CDATA **MUST** be one of the following:

Value	Description
SPINDLE	The axis is as a spindle.
INDEX	The axis configured for indexing to a position.
CONTOUR	The axis is interpolating its position as part of the path position defined by the controller.

RotaryVelocityOverride The percentage override to adjust the programmed spindle speed. The CDATA **MUST** be a numeric value, either an integer or decimal floating point number. ~~The subType specifies the specific feedrate that has been overridden.~~

SpindleInterlock An indicator of the spindle lockout when power has been removed and it is free to rotate. The CDATA **MUST** be as follows:

Value	Description
ACTIVE	The spindle cannot be operated.
INACTIVE	The spindle can be operated.

~~**ToolId**~~ **Deprecated in Rel. 1.2. See ToolAssetID.** ~~This is a reference to an identifier for the current tool in use by the Path. It is a placeholder for now and can be used at the discretion of the implementation. Once mobile assets have been defined, this will refer to the corresponding asset.~~

ToolAssetId This is a reference to an identifier for the current tool in use by the Path.

WorkholdingId This is a reference to an identifier for the current work holding or part clamp available to the device.

3.10.4 Interface Event Element Tag Names

An interface event is the same as all other events, except they all share the same set of controlled values and the behavior signifies a specific interaction between devices. There is a detailed discussion of the interaction of the interfaces in *Part 3.1: Interfaces.* This document will list the available defined interfaces. The interfaces CDATA **MUST** be limited to the following values:

VALUE	Description
NOT_READY	The request or response is not ready to perform the action
READY	The request or response is in an idle.
ACTIVE	The request or response is actively performing the action.
FAIL	The request or response has failed to perform the action
COMPLETE	The response is now completed.

The following are the currently defined set of interfaces:

MaterialFeed Requests material is fed into a device from a feeder.

MaterialChange Requests the device change the type of material being loaded or fed.

MaterialRetract Requests the material be removed from the device by retraction.

PartChange Requests that the type of part being made be changed. Coupled with PART_ID to indicate the part.

MaterialLoad A request for material to be loaded into the device.

MaterialUnload A request for material to be unloaded from the device.

Open A request for the device to open the target of the interface.

Close A request for the device to close the target of the interface.

3.11 Condition

Condition provides a method by which the machine can communicate its health and ability to function. A condition can be one of Normal, Warning, Fault, or Unavailable. A Component **MAY** have multiple active conditions at one time whereas a Sample or Event can only have a single value at a point in time.

3.11.1 Types of Condition

- **`Normal`**
  The item being monitored is operating normally and no action is required. `Normal` also indicates a `Fault` or `Warning` condition has been cleared if the item was previously identified with `Fault` or `Warning`.

- **`Warning`**
  The item being monitored is moving into the abnormal range and should be observed. No action is required at this time. Transition to a `Normal` condition indicates that the `Warning` condition has been cleared.

- **`Fault`**
  The item has failed and intervention is required to return to a `Normal` condition. Transition to a `Normal` condition indicates that the `Fault` condition has been cleared. A `Fault` condition is something that always needs to be acknowledged before operation can continue. Faults are sometimes noted as an alarm.

- **`Unavailable`**
  The value of the item is in an indeterminate state since the data source is no longer providing data. This will also be the initial state of the `Condition` before a connection is established with the data source. The `Condition` **MUST** be `Unavailable` when the value is unknown.

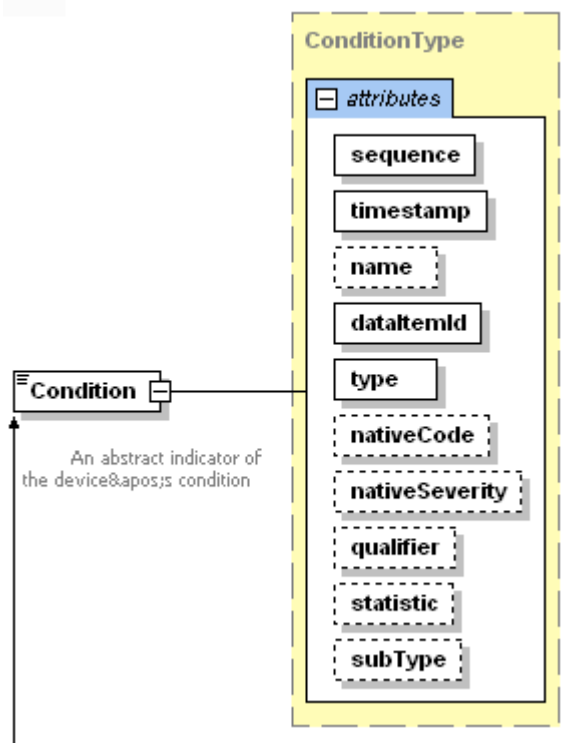

Figure 9: `Condition` Schema

3.11.2 `Condition` Attributes

Attribute	Description	Occurrence
`sequence`	The sequence number of this event. The value **MUST** be represented as an unsigned 64 bit with valid values from 1 to 2^64-1.	1
`timestamp`	The timestamp of the `Sample`. The most accurate time available to the device **MUST** be used for the timestamp	1
`dataItemID`	The `id` attribute of the corresponding data retrieved in the `Probe` request.	1
`name`	The `name` **MUST** match the name of the event's associated `DataItem`. An NMTOKEN XML type.	0..1
`type`	The `DataItem` type this `Condition` refers to.	1
`sub-type`	The sub-type of the `DataItem` this `Condition` refers to.	0..1
`qualifier`	Qualifies the `Condition` and adds context or additional clarification. This optional attribute can be used to convey information like `HIGH`, `LOW`, …	0..1
`nativeCode`	The native code for the piece of equipment. This is the way the `Condition` is represented by the component.	0..1
`nativeSeverity`	The pass thru severity from the device manufacturer.	0..1
`statistic`	The type of statistical calculation specified for the `DataItem`	0..1
`xs:lang`	An optional attribute that specifies language of the alarm or condition text. Refer to IETF RFC 4646 (http://www.ietf.org/rfc/rfc4646.txt) or successor for a full definition of the values for this attribute. Does not appear in the Header schema diagrams	0..1

3.11.3 `Condition` Contents - CDATA

The contents are the optional text from the data source in the un-interpreted form. The text is provided for informational purpose only for interpretation by the application or other client software.

3.11.4 `Condition` Types

All existing `DataItem` types **MAY** be used as types for the `Condition` types. There are some additional types that have been added that represent logical parts of the device architecture and allow for better association and representation of the device's health. The following are the types specifically added for the `Condition`.

Data Item type/ qualifier	Description
ACTUATOR	A `condition` with the motion drive, servo, or actuator.
COMMUNICATIONS	A communications failure indicator.
HARDWARE	The operational condition of the hardware subsystem of the component.
LOGIC_PROGRAM	An error occurred in the logic program or PLC (programmable logic controller).
MOTION_PROGRAM	An error occurred in the motion program.
SYSTEM	A condition representing something that is not the operator, program, or hardware. This is often used to represent operating system issues.

3.11.5 `Condition` Examples

The following are abbreviated examples of the use of the `Condition` elements in XML. The condition has additional restrictions which are different from the `Event` and `Sample`. The following will demonstrate the differences and usage of the `Condition`.

```
...
<Linear id="y" name="Y">
  <DataItems>
    <DataItem type="POSITION" subType="ACTUAL" id="yp" category="SAMPLE"
         name="Yact" units="MILLIMETER" nativeUnits="MILLIMETER"
         coordinateSystem="MACHINE"/>
    <DataItem type="POSITION" id="ylc" category="CONDITION"/>
    <DataItem type="LOAD" id="ylc" category="CONDITION"/>
    <DataItem type="TEMPERATURE" id="ytc" category="CONDITION"/>
  </DataItems>
</Linear>
...

<Controller id="cont" name="controller">
  <DataItems>
    <DataItem type="PROGRAM" id="pgm" category="EVENT" name="program"/>
    <DataItem type="BLOCK" id="blk" category="EVENT" name="block"/>
    <DataItem type="LINE" id="ln" category="EVENT" name="line"/>
    <DataItem type="PATH_FEEDRATE" id="pf" category="SAMPLE" name="Fact"
       units="MILLIMETER/SECOND" nativeUnits="FOOT/MINUTE" subType="ACTUAL"
       coordinateSystem="WORK"/>
    <DataItem type="PATH_FEEDRATE" id="pfo" category="SAMPLE" name="Fovr"
       units="PERCENT" nativeUnits="PERCENT" subType="OVERRIDE"/>
    <DataItem type="PATH_POSITION" id="pp" category="SAMPLE" name="Ppos"
       units="MILLIMETER" nativeUnits="MILLIMETER" coordinateSystem="WORK"/>
    <DataItem type="TOOL_ASSET_ID" id="tid" category="EVENT" name="Tid"/>
    <DataItem type="PART_ID" id="pid" category="EVENT" name="Pid"/>
    <DataItem type="EXECUTION" id="exec" category="EVENT" name="execution"/>
    <DataItem type="CONTROLLER_MODE" id="cm" category="EVENT" name="mode"/>

    <DataItem type="COMMUNICATIONS" id="cc1" category="CONDITION"/>
    <DataItem type="MOTION_PROGRAM" id="cc2" category="CONDITION"/>
    <DataItem type="LOGIC_PROGRAM" id="cc3" category="CONDITION"/>
  </DataItems>
</Controller >
```

In the previous example we have focused on two components, a Linear Y axis and a controller. They both have `Condition` associated with them. The axis has a temperature sensor and a load sensor that will alert when the temperature or load goes out of range. The controller also has a few `Condition` associated with the `Program` and `Communications`.

When everything is working properly, a `Current` request will deliver the following XML:

```
<DeviceStream uuid="HM1" name="HMC_3Axis">
  <ComponentStream component="Linear" name="Y" componentId="y">
    <Samples>
      <Position dataItemId="yp" name="Yact" subType="ACTUAL" sequence="23"
         timestamp="2009-11-13T08:00:00">213.1232</Position>
    </Samples>
    <Condition>
      <Normal type="TEMPERATURE" dataItemId="ytmp" sequence="25"
         timestamp="..."/>
      <Normal type="LOAD" dataItemId ="ylc" sequence="26" timestamp="..."/>
      <Normal type="POSITION" dataItemId ="ypc" sequence="26"
          timestamp="..."/>
    </Condition>
  </ComponentStream>
</DeviceStream>
  <ComponentStream component="Controller" name="cont" componentId="cont">
    <Events>
       ...
    </Events>
    <Condition>
      <Normal type="MOTION_PROGRAM" dataItemId ="cc2" sequence="25"
         timestamp="..."/>
      <Normal type="COMMUNICATIONS" dataItemId ="cc1" sequence="26"
         timestamp="..."/>
      <Normal type="LOGIC_PROGRAM" dataItemId ="cc3" sequence="26"
         timestamp="..."/>
    </Condition>
  </ComponentStream>
</DeviceStream>
```

The example below shows all of the `Condition` items reporting that everything is normal for the linear axis Y and that the controller has two `Condition` that are normal, but there is a `Fault` of sub-type `Communications` on the device.

```
<DeviceStream uuid="HM1" name="HMC_3Axis">
  <ComponentStream component="Linear" name="Y" componentId="y">
    <Samples>
      <Position dataItemId="yp" name="Yact" subType="ACTUAL" sequence="23"
         timestamp="2009-11-13T08:00:00">213.1232</Position>
    </Samples>
    <Condition>
      <Normal type="TEMPERATURE" dataItemId="ytmp" sequence="25"
         timestamp="..."/>
      <Normal type="LOAD" dataItemId ="ylc" sequence="26" timestamp="..."/>
      <Normal type="POSITION" dataItemId ="ypc" sequence="26"
         timestamp="..."/>
    </Condition>
  </ComponentStream>
</DeviceStream>
  <ComponentStream component="Controller" name="cont" componentId="cont">
    <Events>
       ...
    </Events>
    <Condition>
```

```
        <Normal type="MOTION_PROGRAM" id="cc2" sequence="25" timestamp="..."/>
        <Fault type="COMMUNICATIONS" id="cc1" sequence="26" nativeCode="IO1231"
            timestamp="...">Communications error</Fault>
        <Normal type="LOGIC_PROGRAM" id="cc3" sequence="26" timestamp="..."/>
      </Condition>
    </ComponentStream>
</DeviceStream>
```

When a failure occurs the item **MUST** be reported as a `Fault`. This indicates that intervention is required to fix the problem and reset the state of the machine. In the following example, we show how multiple `Faults` on the same `Condition` can exist.

```
</DeviceStream>
  <ComponentStream component="Controller" name="cont" componentId="cont">
    <Events>
       ...
    </Events>
    <Condition>
      <Fault type="MOTION_PROGRAM" dataItemId="cc2" sequence="25"
          nativeCode="PR1123" timestamp="...">Syntax error on line
          107</Fault>
      <Fault type="MOTION_PROGRAM" dataItemId ="cc2" sequence="28"
          nativeCode="PR1123" timestamp="...">Syntax error on line
          112</Fault>
      <Fault type="MOTION_PROGRAM" dataItemId ="cc2" sequence="30"
          nativeCode="PR1123" timestamp="...">Syntax error on line
          122</Fault>
      <Normal type="COMMUNICATIONS" dataItemId ="cc1" sequence="26"
          timestamp="..."/>
      <Normal type="LOGIC_PROGRAM" dataItemId="cc3" sequence="26"
          timestamp="..."/>
    </Condition>
  </ComponentStream>
</DeviceStream>
```

In this case a bad motion program was loaded and multiple errors were reported. When this occurs all errors **MUST** be provided and classified accordingly. The only exception to having multiple values per `Condition` is `Normal`. If the `Condition` is `Normal`, there **MUST** only be one `Condition` with that type present. There **MUST NOT** be more than one `Normal` and a `Normal` **MUST NOT** occur with a `Fault` or `Warning` of the same type.

A `Sample` request **MUST** treat `Condition` items the same way it does `Events` and `Samples` and only return those that are in the current selection window.

~~3.12 Alarms~~ DEPRECATED: See `Condition`

~~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~~

Attribute	Description	Occurrence
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
lang	An optional attribute that specifies language of the alarm text. Refer to IETF RFC 4646 (http://www.ietf.org/rfc/rfc4646.txt) or successor for a full definition of the values for this attribute.	0..1

The code can have one of the following values:

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

Appendices

A. Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

B. Annotated XML Examples

B.1. Example of a `current` Request

```
<?xml version="1.0" encoding="UTF-8"?>
<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mtconnect.org:MTConnectStreams:1.1"
xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.1
http://www.mtconnect.org/schemas/MTConnectStreams_1.1.xsd">
  <Header creationTime="2010-04-16T21:19:35+00:00" sender="localhost"
      instanceId="1267747762" bufferSize="131072" version="1.1"
      nextSequence="739103692" firstSequence="738972620"
      lastSequence="739103691" />

```

The above is a standard header. The buffer size is 131072 entries. The first sequence number is `738972620` and the last sequence number is `739103691`, if you subtract and add one, gives 131072 entries; this means the buffer is full. For the next streaming request, you would request with *from* set to `739103692`.

```
  <Streams>
    <DeviceStream name="VMC-3Axis" uuid="000">
      <ComponentStream component="Path" name="path" componentId="pth">
        <Samples>
          <PathFeedrate dataItemId="Fovr" sequence="738968517"
             timestamp=
             "2010-04-16T21:09:58.356100">100.0000000000</PathFeedrate>
          <PathFeedrate dataItemId="Frt" sequence="739103685"
             timestamp="2010-04-16T21:19:07.019367">0</PathFeedrate>
        </Samples>
        <Events>
          <Block dataItemId="cn2" name="block" sequence="739103493"
             timestamp="2010-04-16T21:19:05.751294">G0Z1</Block>
          <ControllerMode dataItemId="cn3" name="mode" sequence="738968515"
             timestamp=
             "2010-04-16T21:09:58.356100">AUTOMATIC</ControllerMode>
          <Line dataItemId="cn4" name="line" sequence="739103687"
             timestamp="2010-04-16T21:19:07.051368">0</Line>
          <Program dataItemId="cn5" name="program" sequence="738968514"
             timestamp="2010-04-16T21:09:58.356100">FLANGE_CAM.NGC</Program>
          <Execution dataItemId="cn6" name="execution" sequence="739103689"
             timestamp="2010-04-16T21:19:07.063369">READY</Execution>
        </Events>
      </ComponentStream>

```

The `Path` component has both `Samples` and `Events`. The information regarding the path feedrate and feedrate override are considered sampled information in the `Path`. The events are related to the execution of the `Program` for this `Path`.

```
<ComponentStream component="Rotary" name="C" componentId="c1">
  <Samples>
    <RotaryVelocity dataItemId="c2" name="Sspeed" sequence="739103691"
        subType="ACTUAL" timestamp=
        "2010-04-16T21:19:07.063369">0.0000000000</RotaryVelocity>
    <RotaryVelocity dataItemId="c3" name="Sovr" sequence="738968518"
       subType="OVERRIDE" timestamp=
       "2010-04-16T21:09:58.356100">100.0000000000</RotaryVelocity>
  </Samples>
  <Events>
    <RotaryMode dataItemId="cm" name="Cmode" sequence="2"
         timestamp="2010-03-05T00:09:22.457383">SPINDLE</RotaryMode>
  </Events>
  <Condition>
    <Normal dataItemId="Cload" sequence="738968524" timestamp=
        "2010-04-16T21:09:58.356100" type="LOAD" />
  </Condition>
</ComponentStream>
```

The rotary C axis is the spindle and can be seen by checking the RotaryMode. In this case, it is constrained to the value SPINDLE and will probably have a native name of “S”. There is also a `Condition` which is monitoring the spindle load and is currently `Normal`.

```
<ComponentStream component="Linear" name="X" componentId="x1">
  <Samples>
    <Position dataItemId="x2" name="Xact" sequence="739103504"
       subType="ACTUAL" timestamp=
       "2010-04-16T21:19:05.795297">0.0019900000</Position>
    <Position dataItemId="x3" name="Xcom" sequence="739103489"
        subType="COMMANDED" timestamp=
        "2010-04-16T21:19:05.751294">0.0019900000</Position>
  </Samples>
  <Condition>
    <Normal dataItemId="Xload" sequence="738968525" timestamp=
        "2010-04-16T21:09:58.356100" type="LOAD" />
  </Condition>
</ComponentStream>
```

Each of the linear axes has an actual and commanded position that is represented as `Samples` as well as a `Condition` monitoring the load. This is the same pattern for all the linear axes.

```
      <ComponentStream component="Linear" name="Y" componentId="y1">
        <Samples>
          <Position dataItemId="y2" name="Yact" sequence="739103500"
             subType="ACTUAL" timestamp=
             "2010-04-16T21:19:05.783296">0.0002004431</Position>
          <Position dataItemId="y3" name="Ycom" sequence="739103490"
             subType="COMMANDED" timestamp=
             "2010-04-16T21:19:05.751294">0.0002000000</Position>
        </Samples>
        <Condition>
          <Normal dataItemId="Yload" sequence="738968526" timestamp=
             "2010-04-16T21:09:58.356100" type="LOAD"/>
        </Condition>
      </ComponentStream>
      <ComponentStream component="Linear" name="Z" componentId="z1">
        <Samples>
          <Position dataItemId="z2" name="Zact" sequence="739103690"
              subType="ACTUAL" timestamp=
              "2010-04-16T21:19:07.063369">1.0000000000</Position>
          <Position dataItemId="z3" name="Zcom" sequence="739103684"
             subType="COMMANDED" timestamp=
             "2010-04-16T21:19:07.019367">1.0000000000</Position>
        </Samples>
        <Condition>
          <Normal dataItemId="Zload" sequence="738968527" timestamp=
             "2010-04-16T21:09:58.356100" type="LOAD"/>
        </Condition>
      </ComponentStream>
      <ComponentStream component="Controller" name="controller"
          componentId="cn1">
        <Events>
          <EmergencyStop dataItemId="estop" sequence="738968519"
              timestamp="2010-04-16T21:09:58.356100">RESET</EmergencyStop>
        </Events>
        <Condition>
          <Normal dataItemId="clp" sequence="738968528" timestamp=
             "2010-04-16T21:09:58.356100" type="LOGIC_PROGRAM"/>
        </Condition>
      </ComponentStream>
```

Since the Path has included the Execution and Program state, the Controller now contains mainly Condition about the hardware and the state of the device.

```
      <ComponentStream component="Device" name="VMC-3Axis" componentId="dev">
        <Events>
          <Availability dataItemId="avail" sequence="9" timestamp=
              "2010-03-05T00:09:22.457383">AVAILABLE</Message>
          <Message dataItemId="msg" sequence="29" timestamp=
              "2010-03-05T00:09:22.457383">UNAVAILABLE</Message>
        </Events>
      </ComponentStream>
```

Availability is the one required Events for the device and it is currently AVAILABLE. If the machine is powered off then this will become UNAVAILABLE. There have been no messages on this machine, so the message state is currently UNAVAILABLE.

```
      <ComponentStream component="Coolant" name="coolant" componentId="cool">
        <Condition>
          <Normal dataItemId="clow" sequence="738968520" timestamp=
              "2010-04-16T21:09:58.356100" type="FILL_LEVEL"/>
        </Condition>
      </ComponentStream>
      <ComponentStream component="Hydraulic" name="hydraulic"
           componentId="hsys">
        <Condition>
          <Normal dataItemId="hlow" sequence="738968521" timestamp=
              "2010-04-16T21:09:58.356100" type="FILL_LEVEL"/>
          <Normal dataItemId="hpres" sequence="738968522" timestamp=
              "2010-04-16T21:09:58.356100" type="PRESSURE"/>
          <Normal dataItemId="htemp" nativeCode="HTEMP" qualifier="HIGH"
              sequence="739051314" timestamp="2010-04-16T21:15:42.835731"
              type="TEMPERATURE"/>
        </Condition>
      </ComponentStream>
```

The previous two components are Systems. Systems will usually report on the Condition of the components, as can be seen here it is reporting on the Temperature and the Pressure in the Hydraulic (system) and the FillLevel of the Coolant (system).

```
    </DeviceStream>
  </Streams>
</MTConnectStreams>
```