

MTConnect® Standard

Part 3 – Streams, Events, Samples, and Conditions

Version 1.1.0 – Draft F

Prepared for: MTConnect Institute

Prepared by: William Sobel

Prepared on: April 2, 2010

MTConnect Specification

AMT - The Association For Manufacturing Technology (“AMT”) owns the copyright in this MTConnect Specification. AMT grants to you a non-exclusive, non- transferable, revocable, non-sublicensable, fully-paid-up copyright license to reproduce, copy and redistribute the MTConnect Specification, provided that you may only copy or redistribute the MTConnect Specification in the form in which you received it, without modifications, and with all copyright notices and other notices and disclaimers contained in the MTConnect Specification.

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

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

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

Table of Contents

1 Overview 1

1.1 MTConnect® Document Structure 1

2 Purpose of This Document 2

2.1 Terminology 2

2.2 Terminology and Conventions 4

3 Streams, Samples and Events 5

3.1 Structure 5

3.2 Streams 6

3.3 DeviceStream 7

3.3.1 DeviceStream Attributes 8

3.3.2 DeviceStream Elements 8

3.4 ComponentStream 9

3.4.1 ComponentStream Attributes 9

3.4.2 ComponentStream Elements 10

3.5 Samples and Events 10

3.6 Samples 10

3.7 Sample 10

3.7.1 Sample attributes: 11

3.7.2 Sample Element Tag Names 11

3.7.3 Extensibility 13

3.8 Events 13

3.9 Event 13

3.9.1 Event Element Tag Names 14

3.10 Condition 18

3.10.1 Types of Condition 18

3.10.2 Attributes 19

3.10.3 Condition Contents - CDATA 20

3.10.4 Condition Types 20

3.10.5 Condition Examples 21

~~3.11~~ ~~Alarms~~  DEPRECATED: See Condition instead 23

4 Annotated XML Examples 25

4.1 Example of a current Request 25

Appendices 27

A. Bibliography 27

Table of Figures

Figure 1: Streams Example Structure 5

Figure 2: Streams Schema Diagram 7

Figure 3: DeviceStream Schema 8

Figure 4: ComponentStream Schema 9

Figure 5: Condition Schema 19

# 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.

## MTConnect® Document Structure

The MTConnect® specification is subdivided using the following scheme:

Part 1: Overview and Protocol – Version 1.1.0, Draft F

Part 2: Components and Data Items – Version 1.1.0, Draft F

Part 3: Streams, Events, Samples, and Conditions – Version 1.1.0, Draft F

Extensions to the standard will be made according to this scheme and new sections will be added as new areas are addressed. Documents will be named as follows: MTC\_Part\_<Number>\_<Description>.doc. All documents will be developed in Microsoft® Word format and released in Adobe® PDF format. For example, this document is MTC\_Part\_1\_Overview.doc.

# Purpose of This Document

This document is intended to:

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

Part 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. To reduce the amount of redundant information being transmitted and the resulting impact on the communications network, the descriptive information about a data item and its actual value are separated into different communication requests.

The information is broken into three types – Events, Samples, and Conditions. An Event represents the state of a data item or a message. Samples represent the point in time value of a continuously changing data item like axis position. Conditions represent the health of a device or component. This section also covers the vocabulary and format for each piece of data that can be retrieved from a machine.

## Terminology

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Service** An application that provides necessary functionality.

**Tag** Used to reference an instance of an XML element.

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

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

**UUID** Universally unique identifier.

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

**XML** Extensible Markup Language. <http://www.w3.org/XML/>

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

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

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

## Terminology and Conventions

Please refer to Part 1 “Overview and Protocol” Section 2 for XML Terminology and Documentation conventions.

# 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 in *Part 1*.) All the data are collected into streams and organized by device and then by component. A component stream has three parts: Samples and Events. Samples are point-in-time readings from a component reporting what the value is at that instant.

For an example, refer to the Device in Figure 2 below.

An Event changes state to a limited set of values or represents a message. 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, 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.

## Structure

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

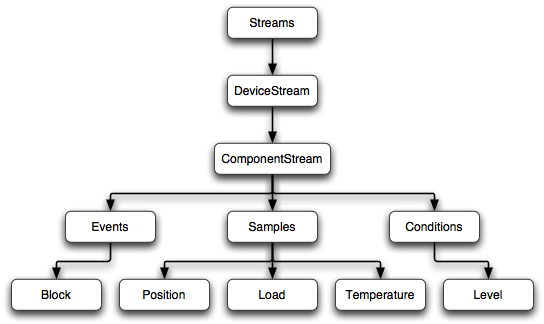


Figure 1: Streams Example Structure

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 will be delivered for this request.

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.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <MTConnectStreams ...>
3. <Header .../>
4. <Streams>
5. <DeviceStream uuid="1" name="mill-1">
6. <ComponentStream componentId="2" name="power" component="Power">
7. <Events>
8. <PowerState name="power" dataItemId="9" sequence="30055111" timestamp="2008-07-07T14:27:59.591">ON</PowerState>
9. </Events>
10. </ComponentStream>
11. </DeviceStream>
12. <DeviceStream uuid="2" name="mill-2">
13. <ComponentStream componentId="3" name="power" component="Power">
14. <Events>
15. <PowerState name="power" dataItemId="10" sequence="52162" timestamp="2008-06-11T10:17:33.291">ON</PowerState>
16. </Events>
17. </ComponentStream>
18. </DeviceStream>
19. </Streams>
20. </MTConnectStreams>

## Sequence Number and Protocol

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

## Streams

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



Figure : Streams Schema Diagram

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

## 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.

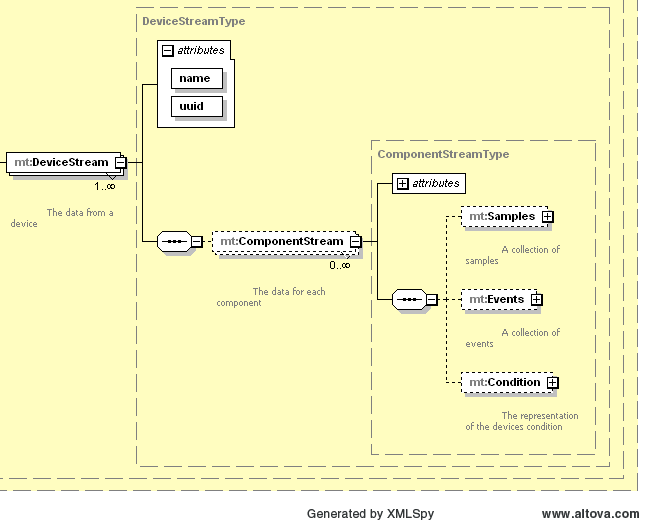


Figure : DeviceStream Schema

### DeviceStream Attributes

| **Attributes** | **Description** | **Occurrence** |
| --- | --- | --- |
| name | The device’s name. An NMTOKEN XML type. | 1 |
| uuid | The device’s unique identifier | 1 |

### DeviceStream Elements

| **Element** | **Description** | **Occurrence** |
| --- | --- | --- |
| ComponentStream | One component’s stream for each component with data | 0..INF |

## ComponentStream

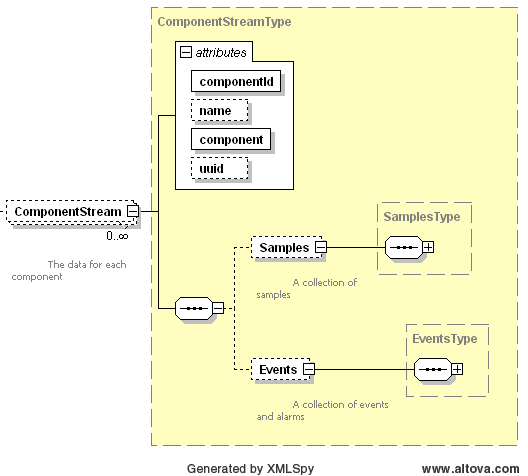


Figure : ComponentStream Schema

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.

### ComponentStream Attributes

| **Attribute** | **Description** | **Occurrence** |
| --- | --- | --- |
| name | This component’s name within the device. An NMTOKEN XML type. | 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 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 Elements of the ComponentStream classify the data into Events, samples, and Conditions. *(The classification is discussed below)*. The ComponentStream **MUST NOT** be empty. It **MUST** include an Events and/or a Samples element.

### 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 |

## Samples and Events

All sample and event 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.

## Samples

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

| **Element** | **Description** | **Occurrence** |
| --- | --- | --- |
| Sample | The subtype of Sample for this component stream | 1..INF |

## Sample

A Sample is an abstract type. This means there will never be an actual element called Sample, but any 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 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).*

### 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. | 1 |
| sequence | The sequence number of this event. Values from 1 to 2^63-1 must be supported. | 1 |
| timestamp | The timestamp of the sample. | 1 |
| dataItemID | The id attribute of the corresponding data retrieved in the probe request. | 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.

### Sample Element Tag Names

The following is a list of all the 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.

**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 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[[1]](#footnote-1). 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.

**Displacement** The displacement as measured from zero to peak. The displacement **MUST** have a value reported in MILLIMETER.

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

**Load** The load on a component. The load **MUST** always be reported in NEWTON or PERCENT 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[[2]](#footnote-2). 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.

**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.

**~~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

**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.

**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 [[3]](#footnote-3). 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.

**Torque** The torque of the component **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.

**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.

### 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.

## Events

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

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

## Event

A Event is an abstract type. This means there will never be an actual element called Event, but any element that is a sub-type of Event can be used in place of Sample. Examples of event sub-types are Block, Execution, and Line. Events types have values in any format.

| **Attribute** | **Description** | **Occurrence** |
| --- | --- | --- |
| name | The name **MUST** match the name of the event's associated DataItem. An NMTOKEN XML type. | 1 |
| sequence | The sequence number of this event. This value **MUST** have a maximum value of 2^63-1 and **MUST** be stored in a signed 64 bit integer. | 1 |
| timestamp | The time-stamp of the event | 1 |
| dataItemID | The id attribute of the corresponding data retrieved in the probe request. | 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 state 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.

### Event Element Tag Names

The Event 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.

**Availabilty** Represents the components 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 the 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.

**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, single block, or single step 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. |

**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 …>Y2</ CoupledAxes >

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

| **Value** | **Description** |
| --- | --- |
| CLOCKWISE | The component is rotating in a clockwise fashion using the right hand rule. |
| COUNTER\_CLOCKWISE | The component is rotating in a counter clockwise fashion using the right hand rule. |

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

| **Value** | **Description** |
| --- | --- |
| OPEN | The door is opened |
| CLOSED | The door is closed. |

**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 or the program has paused execution and is waiting to be continued. |
| STOPPED | The controller has been stopped. |

**EmergencyStop** The emergency stop state of the machine. The CDATA **MUST** be one of the following:

| **Value** | **Description** |
| --- | --- |
| ENABLED | The circuit is complete and the device is operating. |
| DISABLED | The circuit is open and the device must cease operation. |

**Line** The current line number of the program being executed. The CDATA **MUST** be an integer numeric value from 0 to 2^31.

**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 path mode is provided for devices that are controlling multiple sets of axes using one program. When PathMode is not provided it **MUST** be assumed to be INDEPENDENT.

| **Value** | **Description** |
| --- | --- |
| INDEPENDENT | A set of axes are operating independently and without the influence of another set of axes. |
| SYNCHRONOUS | The sets of axes are operating synchronously. |
| MIRROR | The sets of axes are mirroring each other. |

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

| **~~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.

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

| **Value** | **Description** |
| --- | --- |
| SPINDLE | The axis is operating like a spindle and spinning. |
| INDEX | The axis is indexing to a position. |
| CONTOUR | The axes is indexing and rotating at a programmed velocity. |

**ToolId** 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.

**WorkholdingId** This is a reference to an identifier for the current workholding. 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.

## Condition

Condition items provide a channel 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.

### Types of Condition

* **Normal**The item being monitored is operating normally and no action is required. Normal also indicates a Fault has been cleared if the item was previously identified with Fault.
* **Warning**The item being monitored is moving into the abnormal range and should be observed. No action is required at this time.
* **Fault**The item has failed and intervention is required to return to a normal condition. Transition to a normal condition indicates that the Fault has been cleared. A fault is something that always needs to be acknowledged before operation can continue. Faults are sometimes noted as an alarm.
* **Unavailable**The condition 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.

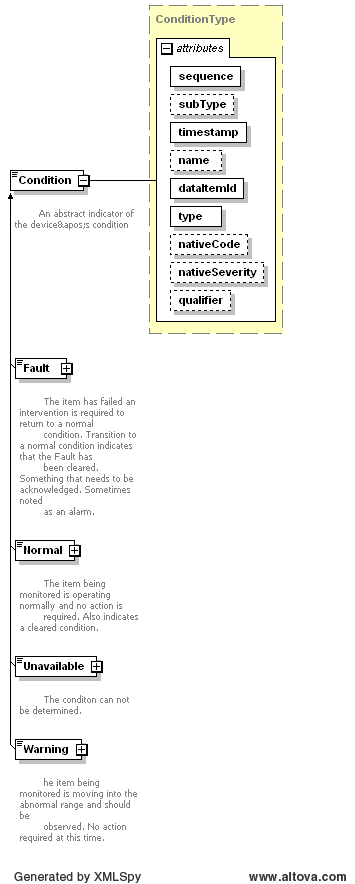
****

Figure : Condition Schema

### Attributes

| **Attribute** | **Description** | **Occurrence** |
| --- | --- | --- |
| sequence | The sequence number of this event. Values from 1 to 2^63-1 must be supported. | 1 |
| timestamp | The timestamp of the sample. | 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 data item type this condition refers to. | 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 alarm is represented on the component. | 0..1 |
| nativeSeverity | The pass thru severity from the device manufacturer. | 0..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 |

### 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.

### Condition Types

All existing Data Item 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 devices health. The following are the types specifically added for the Condition.

| **Data Item type/ qualifier** | **Description** |
| --- | --- |
| **AMPERAGE** | Indicates the electric current of a component is within operating limits. |
| HIGH | The amperage is too high. |
| LOW | The amperage is too low. |
| **COMMUNICATIONS** | A communications failure indicator. |
| **HARDWARE** | The operational conditions of the hardware subsystem of the component. |
| **LEVEL** | Indicates the level of a component is within operating limits. |
| HIGH | The level is too high. |
| LOW | The level is too low. |
| **LOAD** | Indicates the load of a component is within operating limits. |
| HIGH | The load is too high. |
| LOW | The load is too low. |
| **LOGIC\_PROGRAM** | An error occurred in the logic program or PLC (programmable logic controller). |
| **MOTION\_PROGRAM** | An error occurred in the motion program. |
| **PH** | Indicates the pH of a component is within operating limits. |
| HIGH | The pH is too high. |
| LOW | The pH is too low. |
| **PRESSURE** | Indicates the pressure of a component is within operating limits. |
| HIGH | The pressure is too high. |
| LOW | The pressure is too low. |
| **POSITION** | The component’s position is within operational limits. |
| **SYSTEM** | A condition representing something that is not the operator, program, or hardware. This is often used for operating system issues. |
| HIGH |  |
| LOW |  |
| **TEMPERATURE** | Indicates the temperature of a component is within operating limits. |
| HIGH | The temperature is too high. |
| LOW | The temperature is too low. |
| **VELOCITY** | Indicates the velocity of a component is within operating limits. |
| HIGH | The velocity is too high. |
| LOW | The velocity is too low. |
| **VOLTAGE** | Indicates the voltage of a component is within operating limits. |
| HIGH | The voltage is too high. |
| LOW | The voltage is too low. |
|  |  |

### Condition Examples

The following are abbreviated examples of the use of the Condition elements in XML. The condition has additional restrictions which are different form 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\_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 conditions 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 and the controller has a few conditions 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>**

**<Unavailable type="TEMPERATURE" id="ytmp" sequence="25" timestamp="..."/>**

**< Unavailable type="LOAD" id="ylc" sequence="26" timestamp="..."/>**

**< Unavailable type="POSITION" id="ypc" sequence="26" timestamp="..."/>**

**</Condition>**

</ComponentStream>

</DeviceStream>

<ComponentStream component="Controller" name="cont" componentId="cont">

<Events>

...

</Events>

**<Condition>**

**< Unavailable type="MOTION\_PROGRAM" id="cc2" sequence="25" timestamp="..."/>**

**< Unavailable type="COMMUNICATIONS" id="cc1" sequence="26" timestamp="..."/>**

**< Unavailable type="LOGIC\_PROGRAM" id="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 conditions that are normal, but there is a communications fault 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" id="ytmp" sequence="25" timestamp="..."/>**

**<Normal type="LOAD" id="ylc" sequence="26" timestamp="..."/>**

**<Normal type="POSITION" id="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" id="cc2" sequence="25" nativeCode="PR1123" timestamp="...">Syntax error on line 107</Fault>**

**<Fault type="MOTION\_PROGRAM" id="cc2" sequence="28" nativeCode="PR1123" timestamp="...">Syntax error on line 112</Fault>**

**<Fault type="MOTION\_PROGRAM" id="cc2" sequence="30" nativeCode="PR1123" timestamp="...">Syntax error on line 122</Fault>**

<Normal type="COMMUNICATIONS" id="cc1" sequence="26" timestamp="..."/> <Normal type="LOGIC\_PROGRAM" id="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 **MUST** treat condition items the same way it does Events, Samples, and Conditions and only return those that are in the current select window.

## ~~Alarms~~ DEPRECATED: See Condition instead

~~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~~ |
| ~~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.~~

# Annotated XML Examples

## Example of a current Request

The sample was generated with the following request:

http://10.1.23.5/LinuxCNC/sample?path=//Controller|//Power

The response is as follows:

1. <MTConnectStreams xmlns:m="urn:mtconnect.com:MTConnectStreams:1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:mtconnect.com:MTConnectStreams:1.1" xsi:schemaLocation="urn:mtconnect.com:MTConnectStreams:1.1” http://www.mtconnect.org/schemas/MTConnectStreams.xsd">
2. <Header sender="10.1.23.5" bufferSize="100000" creationTime="2008-07-07T23:22:40-07:00" nextSequence="31088439" version="1.1" instanceId="1214527986"/>
3. <Streams>

Events are grouped by equipment:

1. <DeviceStream uuid="linux-01" name="LinuxCNC">

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:

1. <ComponentStream componentId="2" name="power" component="Power">
2. <Events>
3. <PowerState name="power" dataItemId="9" sequence="30055111" timestamp="2008-07-10T10:27:59.591">ON</PowerState>
4. </Events>
5. </ComponentStream>

The control execution is now idle:

1. <ComponentStream componentId="8" name="Controller" component="Controller">
2. <Events>
3. <Execution name="execution" dataItemId="22" sequence="38148653" timestamp="2008-07-10T12:34:00.615">IDLE</Execution>

The execution unit is now running:

1. <Execution name="execution" dataItemId="22" sequence="38148753" timestamp="2008-07-10T12:35:00.615">EXECUTING </Execution>
2. </Events>
3. </ComponentStream>
4. </DeviceStream>
5. </Streams>
6. </MTConnectStreams>

Appendices

1. Bibliography
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.
9. 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.
10. 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.
11. H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New York, 1984.
12. International Organization for Standardization. *ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature.* Geneva, Switzerland, 2001.
13. *ASME B5.59-2 Version 9c: Data Specification for Properties of Machine Tools for Milling and Turning. 2005.*
14. *ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers. 2005.*
15. OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.*

1. From ASME B5.54 - 2005 [↑](#footnote-ref-1)
2. From ASME B5.54 - 2005 [↑](#footnote-ref-2)
3. From ASME B5.54 - 2005 [↑](#footnote-ref-3)