

MTConnect® Standard

SHDR Protocol Companion Specification

Version 1.1.0 – Draft

Prepared for: MTConnect Institute

Prepared by: William Sobel

Prepared on: September 29, 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 damages.

Table of Contents

1 SHDR Protocol 3

1.1 Changes Since Version 1.0 3

1.2 SHDR Overview 3

1.3 Conditions, Messages, and Alarm (Deprecated) 5

1.3.1 Condition 5

1.3.2 Message 6

1.3.3 Alarm (Deprecated) 6

1.4 Heartbeats 6

1.5 Many-To-Many Support (experimental) 7

1.6 Miscellaneous Data 7

Overview

The SHDR protocol is **NOT** part of the MTConnect standard, it was an implementation decision made to simplify the implementation of data gathering with external interfaces with a minimal amount of complexity and development effort. The MTConnect standard can be fully implemented without SHDR or an adapter, this is a design decision each implementer must make.

The reference implementation of the MTConnect Agent has made use of an intermediary process between the machine tool and the MTConnect restful HTTP server. This component is referred to as the *adapter.* The purpose of the adapter is to provide a very simple process that can work with almost any C or C++ application interface and provide the stream updates according to the requirements of the MTConnect specification.

The responsibilities of the adapter are as follows:

1. Provide a socket connection to which a process can connect and receive updates.
2. Only send data when it changes
3. Make sure all data that must conform to a controlled vocabulary, fixed set of values, only communicates those values. Examples are ControllerMode and Execution.
4. Provide recovery when client disconnects and support multiple connections for testing
5. Send all initial values to new clients when they connect
6. Format the data according to the SHDR specification
7. Perform any connection to the machine tool interfaces and gather data

The MTConnect Institute has provided an adapter framework that will take care of all the responsibilities except for the last one. This library is freely available and has example implementations that can be used as a basis for new adapters.

This document will provide a description of the SHDR protocol used by the adapter for those who would want to develop their own framework or would like a deep understanding of the workings of the adapter-agent protocol.

# SHDR Protocol

SHDR stood Simple H? Data Representation, I don’t remember what the H stood for, must have been something important. The data protocol has been enhanced in the latest version of the standard to support additional data types like the new condition and message as well as support for heartbeats and sending configuration data back to the agent.

## Changes Since Version 1.0

The following are the changes to the SHDR protocol from version 1.0 to 1.1. Since 1.1 is fully backward compatible with 1.0, the reference 1.1 agent supports a 1.0 adapter with no modifications.

1. Added the following items:
   1. Message
   2. Condition
2. Heartbeats
3. Sending of the following commands:
   1. uuid
   2. manufacturer
   3. station
   4. serialNumber
4. Alarm format is still supported for 1.0 compatibility, but will be deprecated in the next major release.
5. Added many-to-many support for sensor data collection

## SHDR Overview

The SHDR protocol was designed to be as simple as possible. It is a pipe ‘|’ delimited data stream that begins with a timestamp and then follows with a timestamp in ISO 8601 date time format with optional decimal places. The time is required to be in UTC and therefor must have a trailing Z to indicate that no timezone (+0) is being used.

For all data types except for Condition, Message, and the deprecated Alarm the format is a simple <Key>|<Value> pair. Here is a simple example of a line of SHDR:

2010-09-29T23:59:33.460470Z|Xact|1.4198908806

In the previous example we have a X actual position with the value 1.4198908806. The agent is responsible for converting to the MTConnect specified units assuming the nativeUnits are set correctly. The ISO 8601 format is in the format YYYY-MM-DDTHH:MM:SS.ffffZ, the number of decimal places after the seconds is left up to the implementation and the accuracy of the clock on the machine. If no timestamp is provided, the agent will provide one. For this case the line must begin with a pipe | symbol:

|Xact|1.4198908806

If more than one value as collected at the same time, such as a single API call returning all the positions at once, then they can be provided on the same line:

2010-09-29T23:59:33.460470Z|Xact|1.4198908806|Yact|-0.7842678428

The line must be terminated with a line feed (\n) which is ASCII character 10 or 0xA. The agent will strip off extraneous whitespace, but it is better not to output unneeded characters. For various events the same applies.

2010-09-29T23:59:29.458629Z|avail|**AVAILABLE**

The word AVAILABLE or UNAVAILABLE must be given since it is part of a restricted vocabulary required by MTConnect Standard. For a full list of this vocabulary, please refer to Part 3 of the MTConnect Standard. The current implementation of the reference agent will not validate the correctness of the word, so it is the responsibility of the adapter to make sure they are correct.

The key, in this case avail or in the previous examples Xact or Yact refer to the following items in the MTConnectDevices xml document. The name will match the first item it finds that has this key as its value in the following order: Source, name, id. Here are some examples

<DataItem category="EVENT" id="c1" name="mode" type="CONTROLLER\_MODE"/>

In this case the id is “c1” and the name is “mode”. The key must therefor be mode or c1. In the next example we have only an id.

<DataItem category="EVENT" id="c1" type="CONTROLLER\_MODE"/>

in this case the key must be c1. When more than one are provided, where the name and the id are given, the name is matched first:

<DataItem category="EVENT" id="mode" name="mode1" type="CONTROLLER\_MODE"/>

<DataItem category="EVENT" id="mode2" name="mode" type="CONTROLLER\_MODE"/>

In the previous contrived example, if the key is mode it will match the second line with the id mode2. This is because we always search name first before we search id. There is one additional place where the key can be specified, that is the source. Source was provided because some of the names were too long or complex for an attribute. Here’s an example:

<DataItem category="EVENT" id="c1" name="mode" type="CONTROLLER\_MODE">  
 <Source>channel.0.cnc@2-1</Source>  
</DataItem>

Here the key can be channel.0.cnc@2-1, mode, or c1 in that order. The search order is only important if there are conflicts. The reference implementation will try all possibilities and choose the first match. The names and sources are only unique within the device whereas the id is unique across all the devices in the XML document (an XML requirement.) An adapter will typically feed only one device (see exception below: Many-To-Many Support) and therefor the key will resolve only within the device the adapter is associated with.

## Conditions, Messages, and Alarm (Deprecated)

The following items have a single key with multiple values since they have a more complex structure. Since they require additional elements, they must be sent on a single line, no other key value pairs can be sent before or after a condition, message, or alarm.

### Condition

A condition reflects the state of the component it is associated with. For more information on the new condition semantics, please refer to Part 3 of the MTConnect standard. A condition consists of five parts. The first is the key similar to the previous items. The next four elements are as follows:

1. **Level** – This must be normal, warning, fault, or unavailable (case is not important). If this value is not one of the previous values, unavailable will be assumed.
2. **Native Code** **–** This is the number or short name of the alarm or warning passed through from the device.
3. **Native Severity** – The severity of the alarm as passed through from the device.
4. **Qualifier**  – This is a modifier which indicates which end of the range the condition applies or qualifies the condition. Examples are: HIGH, LOW, etc…
5. **Text** – The text describing the condition. This text is usually taken from the data source and passed through.

An example of a high oil temperature is as follows:

2010-09-29T23:59:33.460470Z|htemp|WARNING|HTEMP|1|HIGH|Oil Temperature High

This corresponds to the condition data item:

<DataItem category="CONDITION" id="htemp" type="TEMPERATURE"/>

And will be delivered as:

<Warning dataItemId="htemp" timestamp="2010-09-29T23:59:33.460470Z" sequence="399237840" type="TEMPERATURE" nativeSeverity="1" nativeCode="HTEMP" qualifier="HIGH" >Oil Temperature High</Warning>

If there is no message, as is usually the case when the operation is Normal, leave all the fields blank, but still provide the pipe delimiter for proper parsing as follows:

2010-09-29T23:59:33.460470Z|htemp|NORMAL||||

All of the fields are optional except for the Level. When a the condition is set to normal, it will reset all the active conditions, for more on this please refer the to MTConnect standard.

### Message

The message is a simple message that has no direct implication on the condition of the component or the device. The message does carry with it the nativeCode which is passed in after the key.

09-29T23:59:33.460470Z|message|CHG\_INSRT|Change Inserts

This message carries the native code CHG\_INSRT. The final XML will be formatted as follows:

<Message dataItemId="message" timestamp="2010-09-29T23:59:33.460470Z" sequence="399237840" nativeCode="CHG\_INSRT">Change Inserts</Message>

As with the condition, the message must appear on a separate line. If the native code is not available, it can be left blank and will not be delivered as follows:

2010-09-29T23:59:33.460470Z|message||Change Inserts

### Alarm (Deprecated)

Although it is highly discouraged, the 1.0 alarm is still supported in SHDR. It would be best if all the Alarms can be converted to the newer conditions and messages. As with all values it begins with a key and since it has multiple fields, it must appear on a line by itself. The fields are as follows:

1. **Code –** The legacy codes allowable for alarms.
2. **Native Code –** Same as the legacy code in the condition.
3. **Severity** – One of the allowable severities: CRITICAL, ERROR, WARNING, INFORMATION.
4. **State** – The activation state of the alarm: ACTIVE or CLEARED (where applicable).

The format is similar to the previous two, each of the fields are provided and the code and severity are required:

2010-09-29T23:59:33.460470Z|alarm|OIL\_TEMP\_HIGH|HTEMP|WARNING|ACTIVE|Oil Temperature High

Since this is deprecated, we will not go into further detail regarding the format of this item.

## Heartbeats

The new MTConnect agent supports heartbeats to detect when the adapter is no longer connected. This functionality was added since the adapter could become silent simply because the machine is not active and data is not changing. This would appear the same as a process that was either frozen or a machine was turned off without closing the socket properly.

The adapter may choose the heartbeat interval and the agent will respect it. The agent and the adapter should consider the other side dead if it does not receive a response or a heartbeat within 2 \* interval. For example, if the adapter responds to the heartbeat request with 10000 meaning 10000 ms or 10 seconds, the agent will consider the adapter dead if it does not respond within 20 seconds. Similarly, the adapter will consider the agent dead if it does not heartbeat in 20 seconds.

The protocol is rather simple, the agent sends the adapter a request on a single line as follows:

\* PING

Any line that begins with a \* is considered a command and is not parsed as a key/value pair stream. When the PING is received the adapter responds with a PONG as follows:

\* PONG 10000

The number after the pong is the heartbeat interval in milliseconds, in this case 10 seconds. If the adapter does not respond with a PONG the agent will consider the adapter a legacy adapter and will disable heartbeats and timeouts. The current implementation will have the server send pings every x seconds and disconnect if it does not receive any data in 20 seconds.

The adapter should check the read side of the socket after every full attempt to get data from the device and if it receives a PING it should immediately write a PONG on the socket. If a multithreaded adapter is used, the write must be mutexed to avoid interspersing the data.

After the agent disconnects or times out, the adapter should not exit, but should wait for another connection from the agent. As well, the agent will attempt to reconnect when a connection to the adapter disconnects or times out.

## Many-To-Many Support (experimental)

In order to support wireless sensors and similar devices that can aggregate data from multiple devices and send it to multiple agents, we have added an extra capability to the adapter and the agent to support the many-to-many connectivity. A single agent can have multiple adapters sending data to each device.

In the previous version we only supported a many to one relationship where each adapter could only supply data for a single device. This was fine for the cases of CNC controllers and sensors on a single machine. When we started using wireless sensors, the sensor receiver was collecting data for multiple devices and aggregating it to one adapter. This adapter would in turn need to supply this data to multiple agents.

To support this configuration, we added an optional : in the key. If the key contains a : it will be split into <device>:<key> and treated as it was previously in the key value pair matching referenced above. This is currently experimental. If the device portion is not specified, the default device for this adapter will be used and the key resolved in the normal manor.

## Miscellaneous Data

MTConnect now supports the following items that can be passed from the adapter to the Agent to allow for dynamic configuration. Currently the following items are supported.

1. uuid
2. manufacturer
3. station
4. serialNumber

These items will modify the fields in the XML returned from the probe request made to the agent. They are passed forward in a similar way to the PING with a \* in front as follows:

\* uuid: VMC1-1314421  
\* serialNumber: 1314421  
\* manufacturer: Big Machine Tool Corp

These items will replace the fields in the Device and Description elements of the MTConnectDevices XML document. Please read the Part 2 of the standard to see where these attributes appear. More items can be added to this list if desired, this was an initial set to test the functionality.