



MTConnect[®] Standard

Part 5.0 – Interfaces

Version 1.4.0

Prepared for: MTConnect Institute

Prepared on: March 31, 2018

MTConnect[®] Specification and Materials

AMT - The Association For Manufacturing Technology (“AMT”) owns the copyright in this MTConnect[®] Specification or Material. AMT grants to you a non-exclusive, non-transferable, revocable, non-sublicensable, fully-paid-up copyright license to reproduce, copy and redistribute this MTConnect Specification or Material, provided that you may only copy or redistribute the MTConnect Specification or Material 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 or Material.

If you intend to adopt or implement an MTConnect Specification or Material in a product, whether hardware, software or firmware, which complies with an 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 info@MTConnect.org.

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.

This Material and all MTConnect Specifications and Materials are 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 Specifications or Materials in any product, including, without limitation, any expressed or implied warranty of non-infringement, 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 MTConnect Specifications or Materials 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.

1 Table of Content

2	1	Purpose of This Document	3
3	2	Terminology and Conventions.....	4
4	3	Interfaces Overview.....	5
5	3.1	Interfaces Architecture	5
6	3.2	Request and Response Information Exchange	6
7	4	Interfaces for Devices and Streams Information Models.....	10
8	4.1	Interfaces	11
9	4.2	Interface.....	11
10	4.2.1	XML Schema Structure for Interface.....	12
11	4.2.2	Interface Types	13
12	4.2.3	Data for Interface	14
13	4.2.3.1	References for Interface.....	14
14	4.2.4	Data Items for Interface.....	14
15	4.2.4.1	INTERFACE_STATE for Interface.....	15
16	4.2.4.2	Specific Data Items for the Interaction Model for Interface	16
17	4.2.4.3	Event States for Interfaces.....	18
18	5	Operation and Error Recovery	22
19	5.1	Request/Response Failure Handling and Recovery.....	22
20		Appendices	30
21	A.	Bibliography.....	30
22			
23			

24 **Table of Figures**

25	Figure 1: Data Flow Architecture for <i>Interfaces</i>	6
26	Figure 2: <i>Request</i> and <i>Response</i> Overview.....	8
27	Figure 3: <i>Interfaces</i> as a <i>Structural Element</i>	10
28	Figure 4: <i>Interface</i> Schema.....	12
29	Figure 5: <i>Request</i> State Diagram	19
30	Figure 6: <i>Response</i> State Diagram.....	21
31	Figure 7: Success Scenario	22
32	Figure 8: <i>Responder</i> – Immediate Failure	23
33	Figure 9: <i>Responder</i> Fails While Providing a Service	24
34	Figure 10: Requester Fails During a Service Request	25
35	Figure 11: <i>Requester</i> Makes Unexpected State Change	26
36	Figure 12: <i>Responder</i> Makes Unexpected State Change	27
37	Figure 13: <i>Requester/Responder</i> Communication Failures	28
38		

39 **1 Purpose of This Document**

40 This document, *Part 5.0 – Interfaces* of the MTConnect® Standard, defines a structured data
41 model used to organize information required to coordinate inter-operations between pieces of
42 equipment.

43 This data model is based on an *Interaction Model* that defines the exchange of information
44 between pieces of equipment and is organized in the MTConnect Standard as the XML element
45 *Interfaces*.

46 *Interfaces* is modeled as an extension to the *MTConnectDevices* and *MTConnectStreams*
47 XML documents. *Interfaces* leverages similar rules and terminology as those used to
48 describe a component in the *MTConnectDevices* XML document. *Interfaces* also uses
49 similar methods for reporting data to those used in the *MTConnectStreams* XML document.

50 As defined in *Part 2.0 – Devices Information Model*, *Interfaces* is modeled as a *Top Level*
51 component in the *MTConnectDevices* document (see *Figure 3* below). Each individual
52 *Interface* XML element is modeled as a *Lower Level* component of *Interfaces*. The
53 data associated with each *Interface* is modeled within each *Lower Level* component.

54

55 Note: See *Part 2.0 – Device Information Model* and *Part 3.0 - Streams Information Model* of
56 the MTConnect Standard for information on how *Interfaces* is structured in the
57 XML documents which are returned from an *MTConnect Agent* in response to a
58 *Probe*, *Sample*, or *Current* request.

59

60 **2 Terminology and Conventions**

61 Refer to *Section 5 of Part 1.0 - Overview and Functionality* for a dictionary of terms, reserved
62 language, and document conventions used in the MTConnect® Standard.

63 **3 Interfaces Overview**

64 In many manufacturing processes, multiple pieces of equipment must work together to perform a
 65 task. The traditional method for coordinating the activities between individual pieces of
 66 equipment is to connect them together using a series of signal wires to communicate equipment
 67 states and demands for action. These interactions are usually accomplished by using simple
 68 binary ON/OFF signals.

69 In the MTConnect[®] Standard, *Interfaces* provides a means to replace this traditional method for
 70 interconnecting pieces of equipment with a structured *Interaction Model* that provides a rich set
 71 of information used to coordinate the actions between pieces of equipment. Implementers may
 72 utilize the information provided by this data model to (1) realize the interaction between pieces
 73 of equipment and (2) to extend the functionality of the equipment to improve the overall
 74 performance of the manufacturing process.

75 The *Interaction Model* used to implement *Interfaces* provides a lightweight and efficient
 76 protocol, simplifies failure recovery scenarios, and defines a structure for implementing a Plug-
 77 And-Play relationship between pieces of equipment. By standardizing the information exchange
 78 using this higher level semantic information model, an implementer may more readily replace a
 79 piece of equipment in a manufacturing system with any other piece of equipment capable of
 80 providing similar *Interaction Model* functions.

81 Two primary functions are required to implement the *Interaction Model* for *Interfaces* and
 82 manage the flow of information between pieces of equipment. Each piece of equipment needs to
 83 have:

- 84 • An *MTConnect Agent* which provides:
 - 85 - The data required to implement the *Interaction Model*.
 - 86 - Any other data from a piece of equipment needed to implement the *Interface* –
 - 87 operating states of the equipment, position information, execution modes, process
 - 88 information, etc.
- 89 • A client software application that enables the piece of equipment to acquire and interpret
 90 information from another piece of equipment.

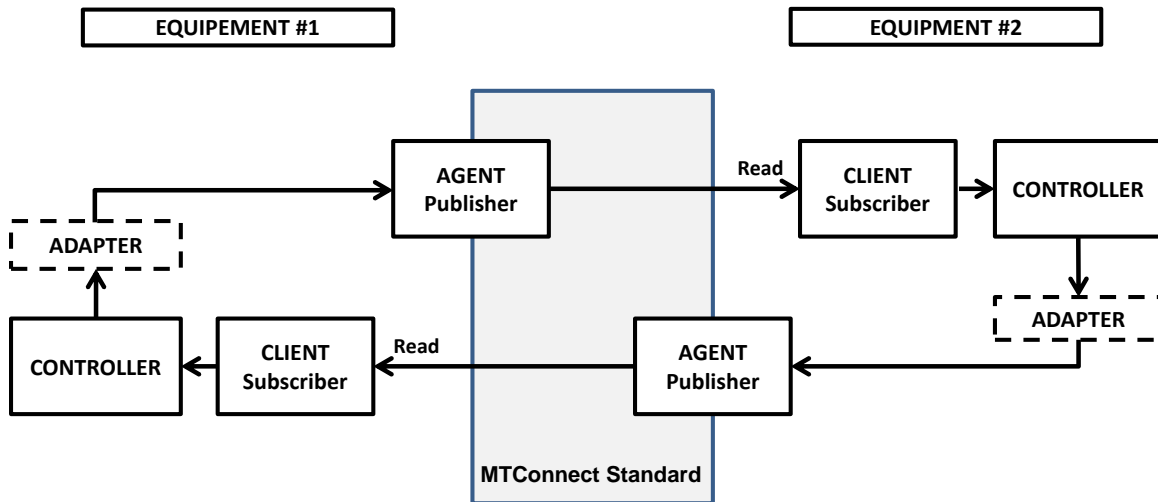
91 **3.1 Interfaces Architecture**

92 MTConnect Standard is based on a communications method that provides no direct way for one
 93 piece of equipment to change the state of, or cause an action to occur by, another piece of
 94 equipment. The *Interaction Model* used to implement *Interfaces* is based on a *Publish/Subscribe*
 95 type of communications as described in *Part 1 – Overview and Functionality* and utilizes a
 96 *Request* and *Response* information exchange mechanism. For *Interfaces*, pieces of equipment
 97 must perform both the publish (*MTConnect Agent*) and subscribe (client) functions.

98 Note: The current definition of *Interfaces* addresses the interaction between two pieces of
 99 equipment. Future releases of the MTConnect Standard may address the interaction
 100 between multiple (more than two) pieces of equipment.

101 The diagram below provides a high-level overview of a typical system architecture used to
 102 implement *Interfaces*.

103



104

105 **Figure 1: Data Flow Architecture for *Interfaces***

106

107 Note: The data flow architecture illustrated in *Figure 1* above was historically referred to in
 108 the MTConnect Standard as a read-read concept.

109 In the implementation of the *Interaction Model for Interfaces*, two pieces of equipment can
 110 exchange information in the following manner. One piece of equipment indicates a *Request* for
 111 service by publishing a type of *Request* using a data item provided through an *MTConnect Agent*
 112 as defined in *Section 4* below. The client associated with a second piece of equipment, which is
 113 subscribing to data from the first machine, detects and interprets that *Request*. If the second
 114 machine chooses to take an action to fulfill this *Request*, it can indicate its acceptance by
 115 publishing a *Response* using a data item provided through its *MTConnect Agent*. The client on
 116 the first piece of equipment will continue to monitor information from the second piece of
 117 equipment until it detects an indication that the *Response* to the *Request* has been completed or
 118 has failed.

119 An example of this type of interaction between pieces of equipment can be represented by a
 120 machine tool that wants material to be loaded by a robot. In this example, the machine tool is the
 121 *Requester* and the robot is the *Responder*. On the other hand, if the robot wants the machine tool
 122 to open a door, the robot becomes the *Requester* and the machine tool the *Responder*.

123 **3.2 Request and Response Information Exchange**

124 The concept of a *Request* and *Response* information exchange is not unique to MTConnect
 125 *Interfaces*. This style of communication is used in many different types of environments and
 126 technologies.

127

128 An early version of a *Request* and *Response* information exchange was used by early sailors.
129 When it was necessary to communicate between two ships before radio communications were
130 available, or when secrecy was required, a sailor on each ship could communicate with the other
131 using flags as a signaling device to request information or actions. The responding ship could
132 acknowledge those requests for action and identify when the requested actions were completed.

133 The same basic *Request* and *Response* concept is implemented by MTCConnect *Interfaces* using
134 the `EVENT` data items defined in *Section 4*.

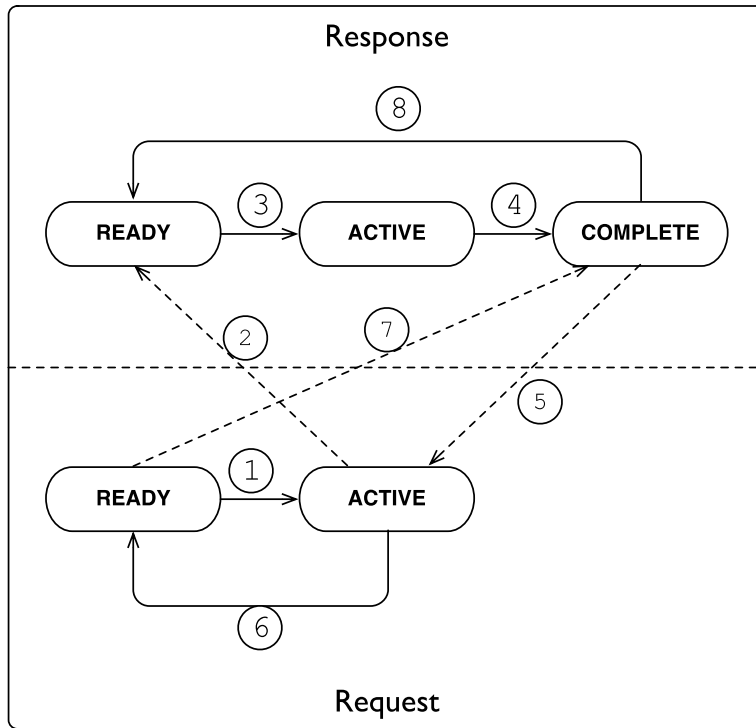
135 The `DataItem` elements defined by the *Interaction Model* each have a `Request` and
136 `Response` subtype. These subtypes identify if the data item represents a *Request* or a
137 *Response*. Using these data items, a piece of equipment changes the state of its *Request* or
138 *Response* to indicate information that can be read by the other piece of equipment. To aid in
139 understanding how the *Interaction Model* functions, one can view this *Interaction Model* as a
140 simple state machine.

141 The interaction between two pieces of equipment can be described as follows. When the
142 *Requester* wants an activity to be performed, it transitions its *Request* state from a `READY` state
143 to an `ACTIVE` state. In turn, when the client on the *Responder* reads this information and
144 interprets the *Request*, the *Responder* announces that it is performing the requested task by
145 changing its response state to `ACTIVE`. When the action is finished, the *Responder* changes its
146 response state to `COMPLETE`. This pattern of *Request* and *Response* provides the basis for the
147 coordination of actions between pieces of equipment. These actions are implemented using
148 `EVENT` category data items. (See *Section 4* for details on the `Event` type data items defined for
149 *Interfaces*.)

150 Note: The implementation details of how the *Responder* piece of equipment reacts to the
151 *Request* and then completes the requested task are up to the implementer.

152

153 The diagram below provides an example of the *Request* and *Response* state machine:



154

155

156

Figure 2: *Request* and *Response* Overview

157 The initial condition of both the *Request* and *Response* states on both pieces of equipment is
 158 READY. The dotted lines indicate the on-going communications that occur to monitor the
 159 progress of the interactions between the pieces of equipment.

160

161 The interaction between the pieces of equipment as illustrated in *Figure 2* progresses through the
 162 following sequence:

Step	Description
1	The <i>Request</i> transitions from READY to ACTIVE signaling that a service is needed.
2	The <i>Response</i> detects the transition of the <i>Request</i> .
3	The <i>Response</i> transitions from READY to ACTIVE indicating that it is performing the action.
4	Once the action has been performed, the <i>Response</i> transitions to COMPLETE.
5	The <i>Request</i> detects the action is COMPLETE.
6	The <i>Request</i> transitions back to READY acknowledging that the service has been performed.
7	The <i>Response</i> detects the <i>Request</i> has returned to READY.
8	In recognition of this acknowledgement, the <i>Response</i> transitions back to READY.

163

164 After the final action has been completed, both pieces of equipment are back in the READY state
 165 indicating that they are able to perform another action.

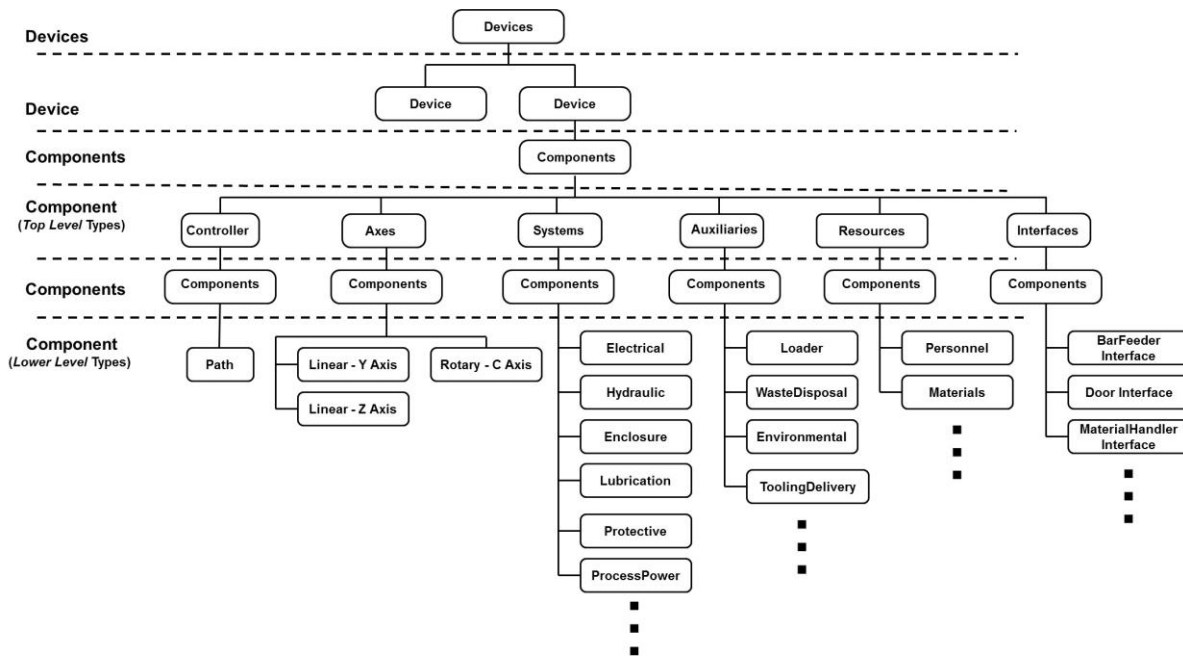
166 4 Interfaces for Devices and Streams Information Models

167 The *Interaction Model* for implementing *Interfaces* is defined in the MTConnect® Standard as an
 168 extension to the MTConnectDevices and MTConnectStreams XML documents.

169 A piece of equipment **MAY** support multiple different *Interfaces*. Each piece of equipment
 170 supporting *Interfaces* **MUST** organize the information associated with each *Interface* in a *Top*
 171 *Level* component called *Interfaces*. Each individual *Interface* is modeled as a *Lower Level*
 172 component called *Interface*. *Interface* is an abstract type XML element and will be
 173 replaced in the XML documents by specific *Interface* types defined below. The data
 174 associated with each *Interface* is modeled as data items within each of these *Lower Level*
 175 *Interface* components.

176 The following XML tree illustrates where *Interfaces* is modeled in the *Device Information*
 177 *Model* for a piece of equipment.

178



179

180 **Figure 3: Interfaces as a Structural Element**

181

182

183 4.1 Interfaces

184 `Interfaces` is an XML *Structural Element* in the `MTConnectDevices` XML document.
185 `Interfaces` is a container type XML element. `Interfaces` is used to group information
186 describing *Lower Level Interface* XML elements, which each provide information for an
187 individual *Interface*.

188 If the `Interfaces` container appears in the XML document, it **MUST** contain one or more
189 `Interface` type XML elements.

190 4.2 Interface

191 `Interface` is the next level of *Structural Element* in the `MTConnectDevices` XML
192 document. As an abstract type XML element, `Interface` will be replaced in the XML
193 documents by specific `Interface` types defined below.

194 Each `Interface` is also a container type element. As a container, the `Interface` XML
195 element is used to organize information required to implement the *Interaction Model* for an
196 *Interface*. It also provides structure for describing the *Lower Level Structural Elements*
197 associated with the `Interface`. Each `Interface` contains *Data Entities* available from the
198 piece of equipment that may be needed to coordinate activities with associated pieces of
199 equipment.

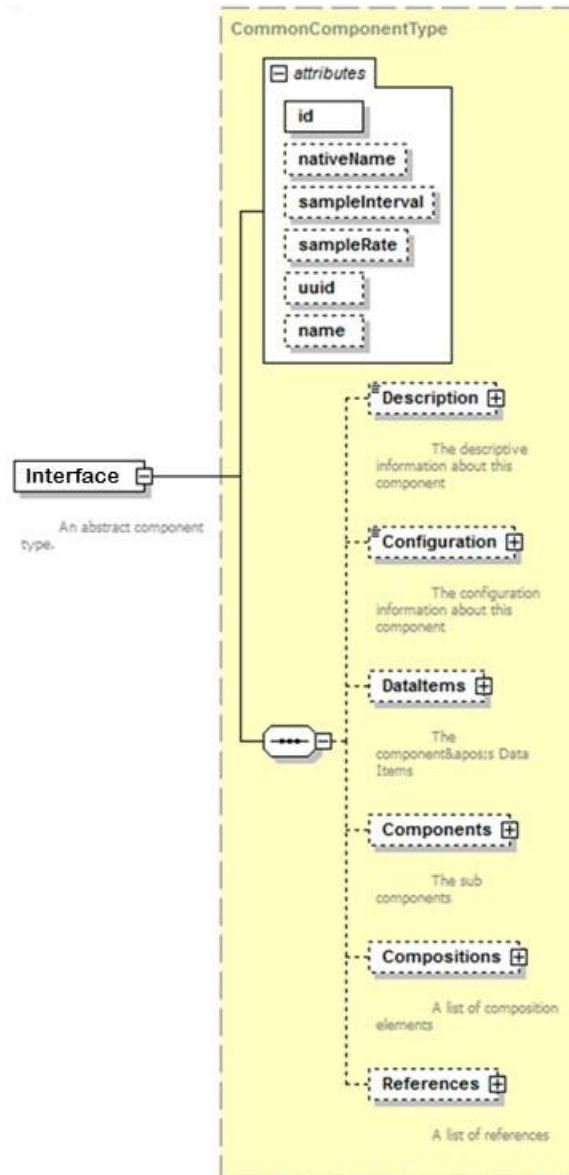
200 The information provided by a piece of equipment for each *Interface* is returned in a
201 `ComponentStream` container of an `MTConnectStreams` document in the same manner as
202 all other types of components.

203

204 **4.2.1 XML Schema Structure for Interface**

205 The following XML schema represents the structure of an Interface XML element.

206 The schema for an Interface element is the same as defined for Component elements
 207 described in *Section 4.4 in Part 2.0 – Devices Information Model* of the MTCConnect Standard.
 208 The figure below shows the attributes defined for Interface and the elements that may be
 209 associated with Interface.



210
 211 **Figure 4: Interface Schema**
 212

213 Refer to *Part 2.0 – Devices Information Model, Section 4.4* for complete descriptions of the
 214 attributes and elements that are illustrated above for Interface.

215 **4.2.2 Interface Types**

216 As an abstract type XML element, *Interface* is replaced in the *MTConnectDevices*
 217 document with a XML element representing a specific type of *Interface*. An initial list of
 218 *Interface* types is defined below.

Interface	Description
BarFeederInterface	<p>BarFeederInterface provides the set of information used to coordinate the operations between a Bar Feeder and another piece of equipment.</p> <p>Bar Feeder is a piece of equipment that pushes bar stock (i.e., long pieces of material of various shapes) into an associated piece of equipment – most typically a lathe or turning center.</p>
MaterialHandlerInterface	<p>MaterialHandlerInterface provides the set of information used to coordinate the operations between a piece of equipment and another associated piece of equipment used to automatically handle various types of materials or services associated with the original piece of equipment.</p> <p>A material handler is a piece of equipment capable of providing any one, or more, of a variety of support services for another piece of equipment or a process:</p> <ul style="list-style-type: none"> Loading/unloading material or tooling Part inspection Testing Cleaning Etc. <p>A robot is a common example of a material handler.</p>
DoorInterface	<p>DoorInterface provides the set of information used to coordinate the operations between two pieces of equipment, one of which controls the operation of a door.</p> <p>The piece of equipment that is controlling the door MUST provide the data item <i>Door_State</i> as part of the set of information provided.</p>
ChuckInterface	<p>ChuckInterface provides the set of information used to coordinate the operations between two pieces of equipment, one of which controls the operation of a chuck.</p> <p>The piece of equipment that is controlling the chuck MUST provide the data item <i>Chuck_State</i> as part of the set of information provided.</p>

219

220 Note: Additional *Interface* types may be defined in future releases of the *MTConnect*
 221 Standard.

222 In order to implement the *Interaction Model* for *Interfaces*, each piece of equipment associated
 223 with an *Interface* **MUST** provide an *Interface* XML element for that type of *Interface*. A
 224 piece of equipment **MAY** support any number of unique *Interfaces*.

225 **4.2.3 Data for Interface**

226 Each *Interface* **MUST** provide (1) the data associated with the specific *Interface* to implement
 227 the *Interaction Model* and (2) any additional data that may be needed by another piece of
 228 equipment to understand the operating states and conditions of the first piece of equipment as it
 229 applies to the *Interface*.

230 Details on data items specific to the *Interaction Model* for each type of *Interface* are provided in
 231 *Section 4.2.4*.

232 An implementer may choose any other data available from a piece of equipment to describe the
 233 operating states and other information needed to support an *Interface*.

234 **4.2.3.1 References for Interface**

235 Some of the data items needed to support a specific *Interface* may already be defined elsewhere
 236 in the XML document for a piece of equipment. However, the implementer may not be able to
 237 directly associate this data with the *Interface* since the MTConnect Standard does not permit
 238 multiple occurrences of a piece of data to be configured in a XML document. *References*
 239 provides a mechanism for associating information defined elsewhere in the *Information Model*
 240 for a piece of equipment with a specific *Interface*.

241 *References* is an XML container that organizes pointers to information defined elsewhere in
 242 the XML document for a piece of equipment. *References* **MAY** contain one or more
 243 *Reference* XML elements.

244 *Reference* is an XML element that provides an individual pointer to information that is
 245 associated with another *Structural Element* or *Data Entity* defined elsewhere in the XML
 246 document that is also required for an *Interface*.

247 *References* is an economical syntax for providing interface specific information without
 248 directly duplicating the occurrence of the data. It provides an efficient, near-time, information
 249 flow between pieces of equipment.

250 For more information on the definition for *References* and *Reference*, see *Section 4.7* and
 251 *4.8* of *Part 2.0 - Devices Information Model*.

252 **4.2.4 Data Items for Interface**

253 Each *Interface* XML element contains data items which are used to communicate
 254 information required to execute the *Interface*. When these data items are read by another piece
 255 of equipment, that piece of equipment can then determine the actions that it may take based upon
 256 that data.

257 Some data items **MAY** be directly associated with the *Interface* element and others will be
 258 organized in a *Lower Level* *References* XML element.

259 It is up to an implementer to determine which additional data items are required for a particular
 260 *Interface*.

261 The data items that have been specifically defined to support the implementation of an *Interface*
 262 are provided below.

263 **4.2.4.1 INTERFACE_STATE for Interface**

264 INTERFACE_STATE is a data item specifically defined for *Interfaces*. It defines the
 265 operational state of the *Interface*. This is an indicator identifying whether the *Interface* is
 266 functioning or not.

267 An INTERFACE_STATE data item **MUST** be defined for every Interface XML element.

268 INTERFACE_STATE is reported in the MTConnectStreams XML document as
 269 InterfaceState. InterfaceState reports one of two states – ENABLED or
 270 DISABLED, which are provided in the CDATA for InterfaceState.

271 The table below shows both the INTERFACE_STATE data item as defined in the
 272 MTConnectDevices document and the corresponding *Element Name* that **MUST** be reported
 273 in the MTConnectStreams document.

EVENT Data Item Type	Event <i>Element Name</i>	Description and Valid Data Values
INTERFACE_STATE	InterfaceState	The current functional or operational state of an <i>Interface</i> type element indicating whether the <i>Interface</i> is active or not currently functioning. Valid Data Values: - ENABLED: The <i>Interface</i> is currently operational and performing as expected. - DISABLED: The <i>Interface</i> is currently not operational. When the INTERFACE_STATE is DISABLED, the state of all data items that are specific for the <i>Interaction Model</i> associated with that <i>Interface</i> MUST be set to NOT_READY.

274

275

276 4.2.4.2 Specific Data Items for the *Interaction Model for Interface*

277 A special set of data items have been defined to be used in conjunction with *Interface* type
 278 elements. When modeled in the *MTConnectDevices* document, these data items are all *Data*
 279 *Entities* in the *EVENT* category (See *Part 3.0 – Streams Information Model* for details on how
 280 the corresponding data items are reported in the *MTConnectStreams* document). They
 281 provide information from a piece of equipment to *Request* a service to be performed by another
 282 associated piece of equipment; and for the associated piece of equipment to indicate its progress
 283 in performing its *Response* to the *Request* for service.

284 Many of the data items describing the services associated with an *Interface* are paired to describe
 285 two distinct actions – one to *Request* an action to be performed and a second to reverse the action
 286 or to return to an original state. For example, a *DoorInterface* will have two actions
 287 *OPEN_DOOR* and *CLOSE_DOOR*. An example of an implementation of this would be a robot
 288 that indicates to a machine that it would like to have a door opened so that the robot could extract
 289 a part from the machine and then asks the machine to close that door once the part has been
 290 removed.

291 When these data items are used to describe a service associated with an *Interface*, they **MUST**
 292 have one of the following two *subType* elements: *REQUEST* or *RESPONSE*. These *subType*
 293 elements **MUST** be specified to define whether the piece of equipment is functioning as the
 294 *Requester* or *Responder* for the service to be performed. The *Requester* **MUST** specify the
 295 *REQUEST* *subType* for the data item and the *Responder* **MUST** specify a corresponding
 296 *RESPONSE* *subType* for the data item to enable the coordination between the two pieces of
 297 equipment.

298 These data items and their associated *subType* provide the basic structure for implementing the
 299 *Interaction Model* for an *Interface*.

300 The table below provides a list of the data items that have been defined to identify the services to
 301 be performed for or by a piece of equipment associated with an *Interface*.

302

303 The table also provides the corresponding transformed *Element Name* for each data item that
 304 **MAY** be returned by an *MTCConnect Agent* as an Event type XML Data Entity in the
 305 MTCConnectStreams XML document. The *Controlled Vocabulary* for each of these data
 306 items are defined below in *Section 4.2.4.3*.

307

EVENT Data Item Type	Event <i>Element Name</i>	Description
MATERIAL_FEED	MaterialFeed	Service to advance material or feed product to a piece of equipment from a continuous or bulk source.
MATERIAL_CHANGE	MaterialChange	Service to change the type of material or product being loaded or fed to a piece of equipment.
MATERIAL_RETRACT	MaterialRetract	Service to remove or retract material or product.
PART_CHANGE	PartChange	Service to change the part or product associated with a piece of equipment to a different part or product.
MATERIAL_LOAD	MaterialLoad	Service to load a piece of material or product.
MATERIAL_UNLOAD	MaterialUnload	Service to unload a piece of material or product.
OPEN_DOOR	OpenDoor	Service to open a door.
CLOSE_DOOR	CloseDoor	Service to close a door.
OPEN_CHUCK	OpenChuck	Service to open a chuck.
CLOSE_CHUCK	CloseChuck	Service to close a chuck

308

309

310 **4.2.4.3 Event States for Interfaces**

311 For each of the data items above, the *Valid Data Values* for the CDATA that is returned for these
 312 data items in the MTConnectStreams document is defined by a *Controlled Vocabulary*. This
 313 *Controlled Vocabulary* represents the state information to be communicated by a piece of
 314 equipment for the data items defined in the table above.

315 The *Request* portion of the *Interaction Model* for *Interfaces* has four states as defined in the table
 316 below:

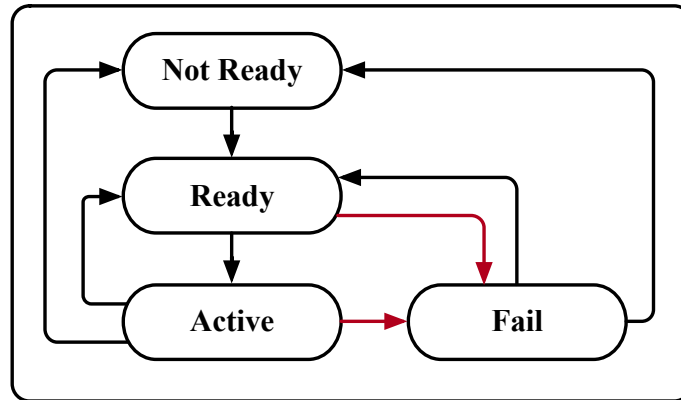
317

Request State	Description
NOT_READY	The <i>Requester</i> is not ready to make a <i>Request</i> .
READY	The <i>Requester</i> is prepared to make a <i>Request</i> , but no <i>Request</i> for service is required. The <i>Requester</i> will transition to ACTIVE when it needs a service to be performed.
ACTIVE	The <i>Requester</i> has initiated a <i>Request</i> for a service and the service has not yet been completed by the <i>Responder</i> .
FAIL	<p>CONDITION 1: When the <i>Requester</i> has detected a failure condition, it indicates to the <i>Responder</i> to either not initiate an action or stop its action before it completes by changing its state to FAIL.</p> <p>CONDITION 2: If the <i>Responder</i> changes its state to FAIL, the <i>Requester</i> MUST change its state to FAIL.</p> <p>ACTIONS: After detecting a failure, the <i>Requester</i> SHOULD NOT change its state to any other value until the <i>Responder</i> has acknowledged the FAIL state by changing its state to FAIL.</p> <p>Once the FAIL state has been acknowledged by the <i>Responder</i>, the <i>Requester</i> may attempt to clear its FAIL state.</p> <p>As part of the attempt to clear the FAIL state, the <i>Requester</i> MUST reset any partial actions that were initiated and attempt to return to a condition where it is again ready to perform a service. If the recovery is successful, the <i>Requester</i> changes its <i>Request</i> state from FAIL to READY. If for some reason the <i>Requester</i> is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.</p>

318

319

320 The following diagram shows a graphical representation of the possible state transitions for a
321 *Request*:



322
323 **Figure 5: Request State Diagram**
324
325

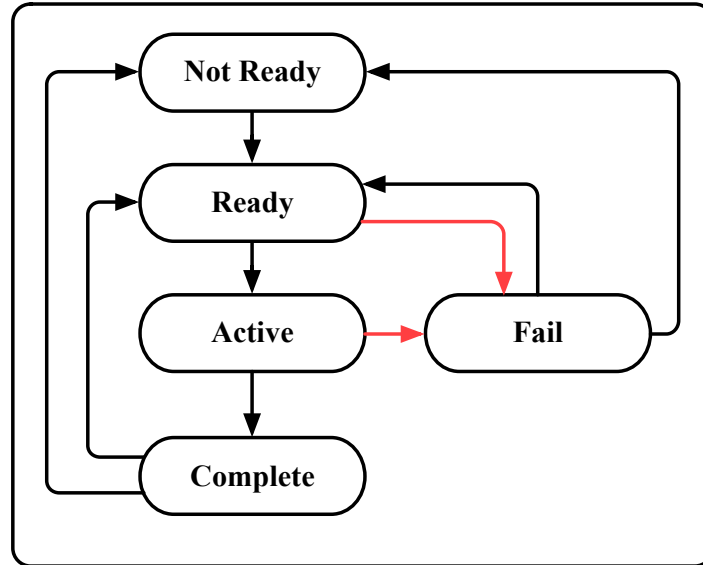
326 The *Response* portion of the *Interaction Model for Interfaces* has five states as defined in the
 327 table below:

Response State	Description
NOT_READY	The <i>Responder</i> is not ready to perform a service.
READY	<p>The <i>Responder</i> is prepared to react to a <i>Request</i>, but no <i>Request</i> for service has been detected.</p> <p>The <i>Responder</i> MUST transition to ACTIVE to inform the <i>Requester</i> that it has detected and accepted the <i>Request</i> and is in the process of performing the requested service.</p> <p>If the <i>Responder</i> is not ready to perform a <i>Request</i>, it MUST transition to a NOT_READY state.</p>
ACTIVE	<p>The <i>Responder</i> has detected and accepted a <i>Request</i> for a service and is in the process of performing the service, but the service has not yet been completed.</p> <p>In normal operation, the <i>Responder</i> MUST NOT change its state to ACTIVE unless the <i>Requester</i> state is ACTIVE.</p>
FAIL	<p>CONDITION 1:</p> <p>The <i>Responder</i> has failed while executing the actions required to perform a service and the service has not yet been completed or the <i>Responder</i> has detected that the <i>Requestor</i> has unexpectedly changed state.</p> <p>CONDITION 2:</p> <p>If the <i>Requester</i> changes its state to FAIL, the <i>Responder</i> MUST change its state to FAIL.</p> <p>ACTIONS:</p> <p>After entering a FAIL state, the <i>Responder</i> SHOULD NOT change its state to any other value until the <i>Requester</i> has acknowledged the FAIL state by changing its state to FAIL.</p> <p>Once the FAIL state has been acknowledged by the <i>Requester</i>, the <i>Responder</i> may attempt to clear its FAIL state.</p> <p>As part of the attempt to clear the FAIL state, the <i>Responder</i> MUST reset any partial actions that were initiated and attempt to return to a condition where it is again ready to perform a service. If the recovery is successful, the <i>Responder</i> changes its <i>Response</i> state from FAIL to READY. If for some reason the <i>Responder</i> is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.</p>
COMPLETE	<p>The <i>Responder</i> has completed the actions required to perform the service.</p> <p>The <i>Responder</i> MUST remain in the COMPLETE state until the <i>Requester</i> acknowledges that the service is complete by changing its state to READY.</p> <p>At that point, the <i>Responder</i> MUST change its state to either READY if it is again prepared to perform a service or NOT_READY if it is not prepared to perform a service.</p>

328

329 The state values described in the above tables **MUST** be provided in the CDATA for each of the
330 *Interface* specific data items provided in the MTConnectStreams document.

331 The following diagram shows a graphical representation of the possible state transitions for a
332 *Response*:

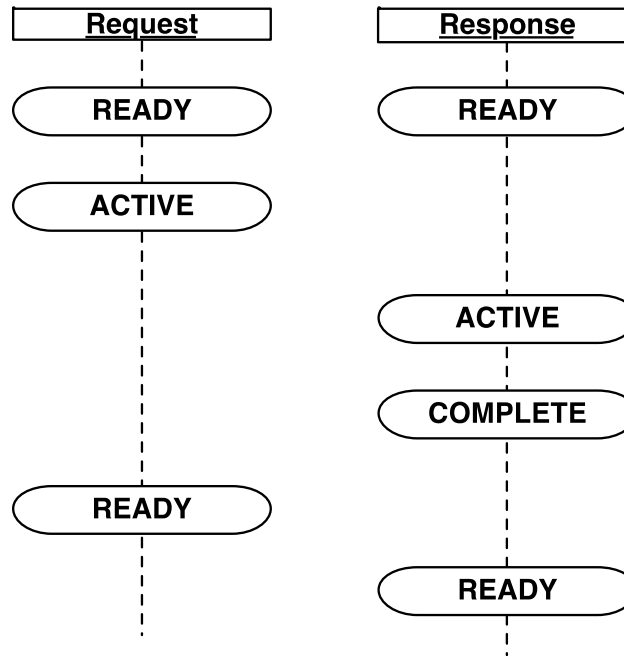


333
334 **Figure 6: Response State Diagram**
335

336 5 Operation and Error Recovery

337 The *Request/Response* state model implemented for *Interfaces* may also be represented by a
 338 graphical model. The following scenario demonstrates the state transitions that occur during a
 339 successful *Request* for service and the resulting *Response* to fulfill that service *Request*.

340



341

342

Figure 7: Success Scenario

343

344 5.1 *Request/Response* Failure Handling and Recovery

345 A significant feature of the *Request/Response Interaction Model* is the ability for either piece of
 346 equipment to detect a failure associated with either the *Request* or *Response* actions. When
 347 either a failure or unexpected action occurs, the *Request* and the *Response* portion of the
 348 *Interaction Model* can announce a FAIL state upon detecting a problem. The following are
 349 graphical models describing multiple scenarios where either the *Requester* or *Responder* detects
 350 and reacts to a failure. In these examples, either the *Requester* or *Responder* announces the
 351 detection of a failure by setting either the *Request* or the *Response* state to FAIL.

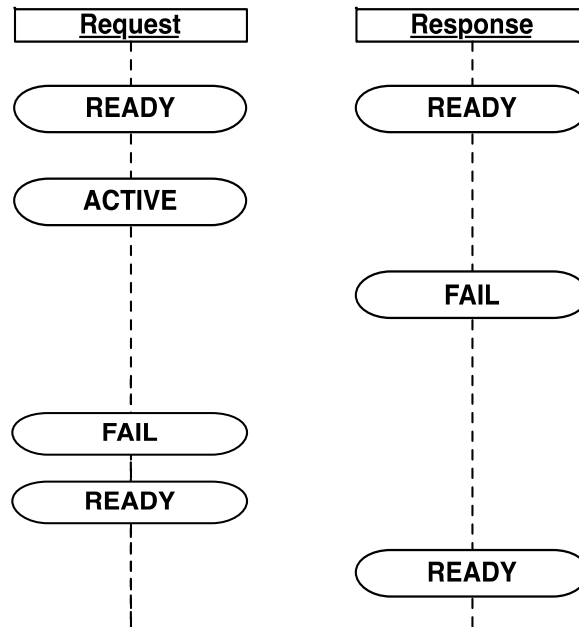
352 Once a failure is detected, the *Interaction Model* provides information from each piece of
 353 equipment as they attempt to recover from a failure, reset all of their functions associated with
 354 the *Interface* to their original state, and return to normal operation.

355

356 The following are scenarios that describe how pieces of equipment may react to different types
 357 of failures and how they indicate when they are again ready to request a service or respond to a
 358 request for service after recovering from those failures:

359 Scenario #1 – Responder Fails Immediately

360 In this scenario, a failure is detected by the *Responder* immediately after a *Request* for service
 361 has been initiated by the *Requester*.



362
 363 **Figure 8: Responder – Immediate Failure**
 364

365 In this case, the *Request* transitions to *ACTIVE* and the *Responder* immediately detects a
 366 failure before it can transition the *Response* state to *ACTIVE*. When this occurs, the *Responder*
 367 transitions the *Response* state to *FAIL*.

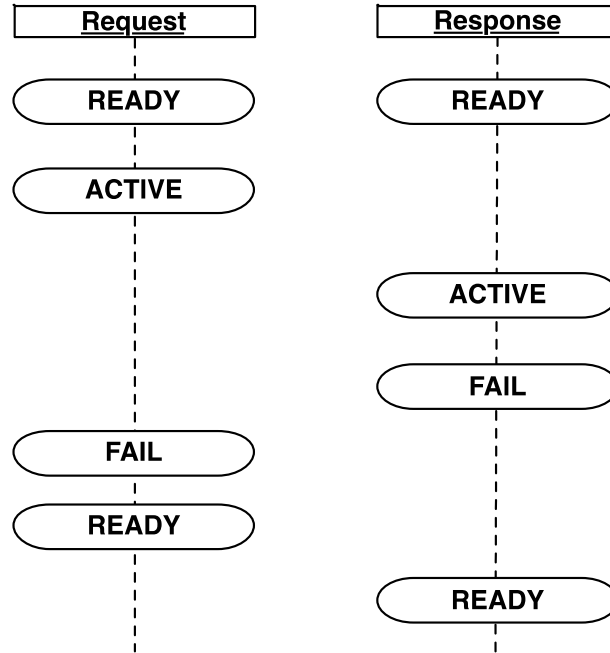
368 After detecting that the *Responder* has transitioned its state to *FAIL*, the *Requester* **MUST**
 369 change its state to *FAIL*.

370 The *Requester*, as part of clearing a failure, resets any partial actions that were initiated and
 371 attempts to return to a condition where it is again ready to request a service. If the recovery is
 372 successful, the *Requester* changes its state from *FAIL* to *READY*. If for some reason the
 373 *Requester* cannot return to a condition where it is again ready to request a service, it transitions
 374 its state from *FAIL* to *NOT_READY*.

375 The *Responder*, as part of clearing a failure, resets any partial actions that were initiated and
 376 attempts to return to a condition where it is again ready to perform a service. If the recovery is
 377 successful, the *Responder* changes its *Response* state from *FAIL* to *READY*. If for some reason
 378 the *Responder* is not again prepared to perform a service, it transitions its state from *FAIL* to
 379 *NOT_READY*.

380 Scenario #2 – Responder Fails While Providing a Service

381 This is the most common failure scenario. In this case, the *Responder* will begin the actions
 382 required to provide a service. During these actions, the *Responder* detects a failure and
 383 transitions its *Response* state to FAIL.



384

385 **Figure 9: Responder Fails While Providing a Service**

386

387 When a *Requester* detects a failure of a *Responder*, it transitions its state from ACTIVE to
 388 FAIL.

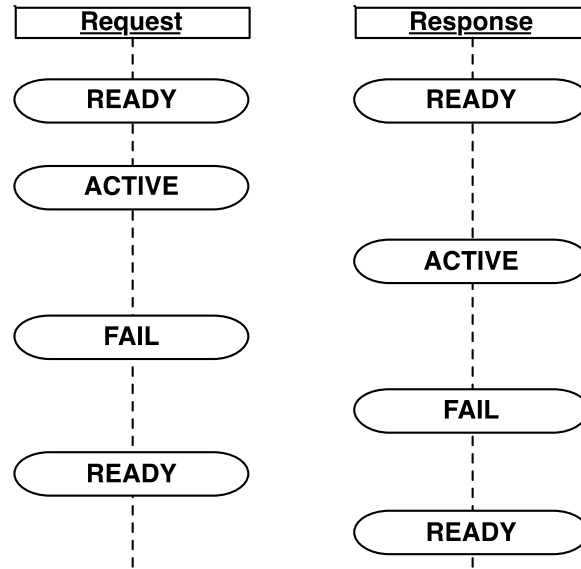
389 The *Requester* resets any partial actions that were initiated and attempts to return to a condition
 390 where it is again ready to request a service. If the recovery is successful, the *Requester* changes
 391 its state from FAIL to READY if the failure has been cleared and it is again prepared to request
 392 another service. If for some reason the *Requester* cannot return to a condition where it is again
 393 ready to request a service, it transitions its state from FAIL to NOT_READY.

394 The *Responder*, as part of clearing a failure, resets any partial actions that were initiated and
 395 attempts to return to a condition where it is again ready to perform a service. If the recovery is
 396 successful, the *Responder* changes its *Response* state from FAIL to READY if it is again
 397 prepared to perform a service. If for some reason the *Responder* is not again prepared to
 398 perform a service, it transitions its state from FAIL to NOT_READY.

399

400 Scenario #3 – Requester Failure During a Service Request

401 In this scenario, the *Responder* will begin the actions required to provide a service. During
 402 these actions, the *Requester* detects a failure and transitions its *Request* state to FAIL.



403

404 **Figure 10: Requester Fails During a Service Request**

405

406 When the *Responder* detects that the *Requester* has transitioned its *Request* state to FAIL, the
 407 *Responder* also transitions its *Response* state to FAIL.

408 The *Requester*, as part of clearing a failure, resets any partial actions that were initiated and
 409 attempts to return to a condition where it is again ready to request a service. If the recovery is
 410 successful, the *Requester* changes its state from FAIL to READY. If for some reason the
 411 *Requester* cannot return to a condition where it is again ready to request a service, it transitions
 412 its state from FAIL to NOT_READY.

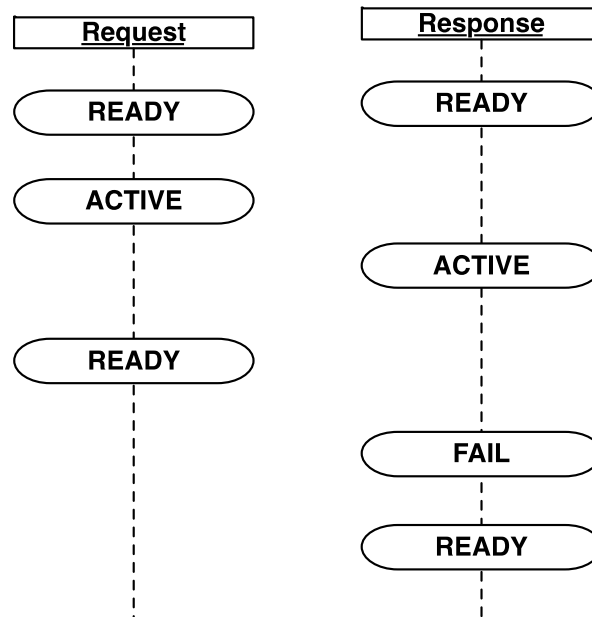
413 The *Responder*, as part of clearing a failure, resets any partial actions that were initiated and
 414 attempts to return to a condition where it is again ready to perform a service. If the recovery is
 415 successful, the *Responder* changes its *Response* state from FAIL to READY. If for some reason
 416 the *Responder* is not again prepared to perform a service, it transitions its state from FAIL to
 417 NOT_READY.

418

419 Scenario #4 – Requester Changes to an Unexpected State While Responder is Providing a
 420 Service

421 In some cases, a *Requester* may transition to an unexpected state after it has initiated a *Request*
 422 for service.

423 As demonstrated below, the *Requester* has initiated a *Request* for service and its *Request* state
 424 has been changed to ACTIVE. The *Responder* begins the actions required to provide the
 425 service. During these actions, the *Requester* transitions its *Request* state back to READY before
 426 the *Responder* can complete its actions. This **SHOULD** be regarded as a failure of the
 427 *Requester*.



428
 429 **Figure 11: Requester Makes Unexpected State Change**
 430

431 In this case, the *Responder* reacts to this change of state of the *Requester* in the same way as
 432 though the *Requester* had transitioned its *Request* state to FAIL (i.e., the same as in Scenario
 433 #3 above).

434 At this point, the *Responder* then transitions its *Response* state to FAIL.

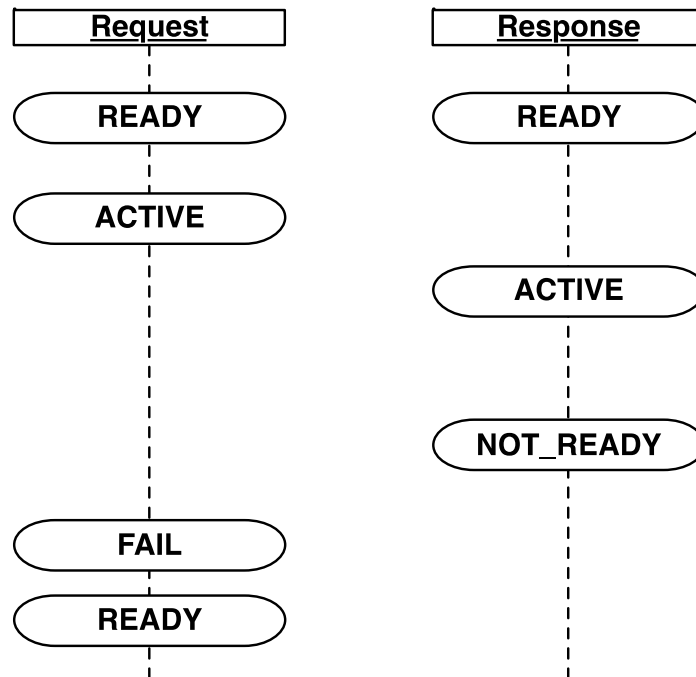
435 The *Responder* resets any partial actions that were initiated and attempts to return to its original
 436 condition where it is again ready to perform a service. If the recovery is successful, the
 437 *Responder* changes its *Response* state from FAIL to READY. If for some reason the *Responder*
 438 is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.

439 Note: The same scenario exists if the *Requester* transitions its *Request* state to NOT_READY.
 440 However, in this case, the *Requester* then transitions its *Request* state to READY after it
 441 resets all of its functions back to a condition where it is again prepared to make a
 442 *Request* for service.

443 Scenario #5 – Responder Changes to an Unexpected State While Providing a Service

444 Similar to Scenario #5, a *Responder* may transition to an unexpected state while providing a
 445 service.

446 As demonstrated below, the *Responder* is performing the actions to provide a service and the
 447 *Response* state is ACTIVE. During these actions, the *Responder* transitions its state to
 448 NOT_READY before completing its actions. This should be regarded as a failure of the
 449 *Responder*.



450

451 **Figure 12: Responder Makes Unexpected State Change**

452

453 Upon detecting an unexpected state change of the *Responder*, the *Requester* transitions its state
 454 to FAIL.

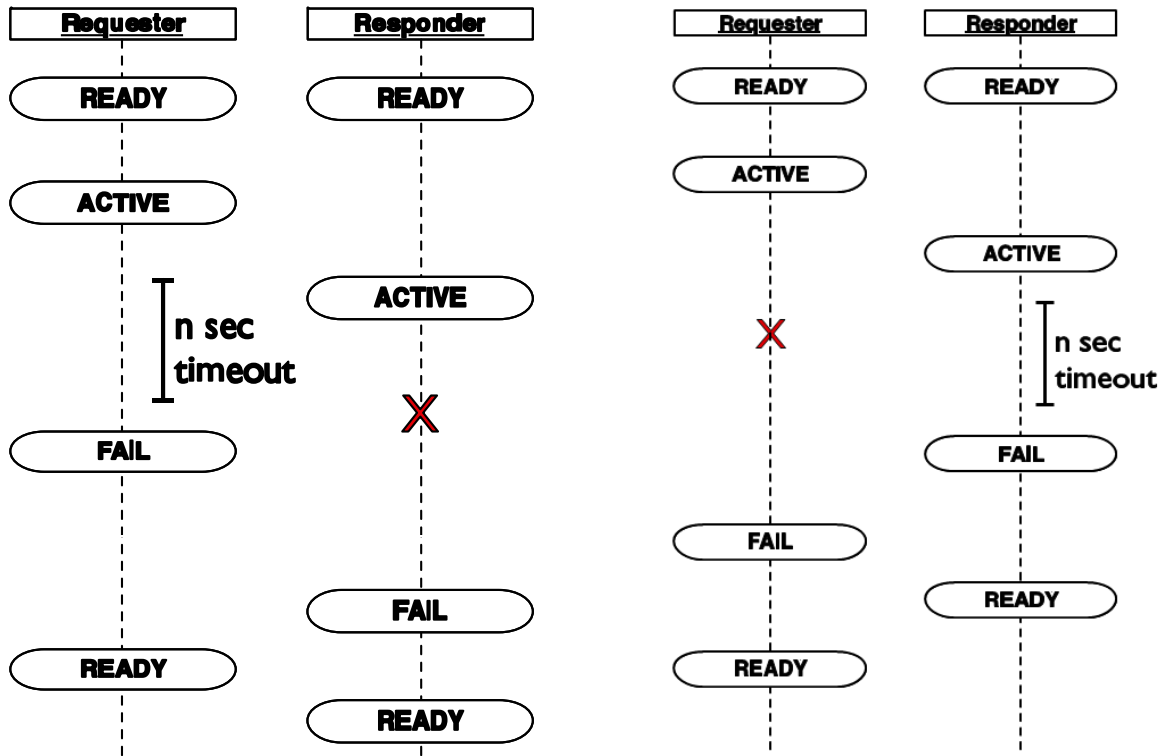
455 The *Requester* resets any partial actions that were initiated and attempts to return to a condition
 456 where it is again ready to request a service. If the recovery is successful, the *Requester* changes
 457 its state from FAIL to READY. If for some reason the *Requester* cannot return to a condition
 458 where it is again ready to request a service, it transitions its state from FAIL to NOT_READY.

459 Since the *Responder* has failed to an invalid state, the condition of the *Responder* is unknown.
 460 Where possible, the *Responder* should try to reset to an initial state.

461 The *Responder*, as part of clearing the cause for the change to the unexpected state, should
 462 attempt to reset any partial actions that were initiated and then return to a condition where it is
 463 again ready to perform a service. If the recovery is successful, the *Responder* changes its
 464 *Response* state from the unexpected state to READY. If for some reason the *Responder* is not
 465 again prepared to perform a service, it maintains its state as NOT_READY.

466 Scenario #6 – Responder or Requester Become UNAVAILABLE or Experience a Loss of
 467 Communications

468 In this scenario, a failure occurs in the communications connection between the *Responder* and
 469 *Requester*. This failure may result from the `InterfaceState` from either piece of
 470 equipment returning a value of `UNAVAILABLE` or one of the pieces of equipment does not
 471 provide a heartbeat within the desired amount of time (See *Part 1.0 - Overview and*
 472 *Functionality* for details on heartbeat).



473
 474 **Figure 13: Requester/Responder Communication Failures**

475
 476 When one of these situations occurs, each piece of equipment assumes that there has been a
 477 failure of the other piece of equipment.

478 When normal communications are re-established, neither piece of equipment should assume
 479 that the *Request/Response* state of the other piece of equipment remains valid. Both pieces of
 480 equipment should set their state to `FAIL`.

481 The *Requester*, as part of clearing its `FAIL` state, resets any partial actions that were initiated
 482 and attempts to return to a condition where it is again ready to request a service. If the recovery
 483 is successful, the *Requester* changes its state from `FAIL` to `READY`. If for some reason the
 484 *Requester* cannot return to a condition where it is again ready to request a service, it transitions
 485 its state from `FAIL` to `NOT_READY`.

486

487 The *Responder*, as part of clearing its `FAIL` state, resets any partial actions that were initiated
488 and attempts to return to a condition where it is again ready to perform a service. If the
489 recovery is successful, the *Responder* changes its *Response* state from `FAIL` to `READY`. If for
490 some reason the *Responder* is not again prepared to perform a service, it transitions its state
491 from `FAIL` to `NOT_READY`.

492

Appendices

493 A. Bibliography

- 494 1. Engineering Industries Association. *EIA Standard - EIA-274-D*, Interchangeable Variable,
495 Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically
496 Controlled Machines. Washington, D.C. 1979.
- 497 2. ISO TC 184/SC4/WG3 N1089. *ISO/DIS 10303-238*: Industrial automation systems and
498 integration Product data representation and exchange Part 238: Application Protocols:
499 Application interpreted model for computerized numerical controllers. Geneva,
500 Switzerland, 2004.
- 501 3. International Organization for Standardization. *ISO 14649*: Industrial automation systems
502 and integration – Physical device control – Data model for computerized numerical
503 controllers – Part 10: General process data. Geneva, Switzerland, 2004.
- 504 4. International Organization for Standardization. *ISO 14649*: Industrial automation systems
505 and integration – Physical device control – Data model for computerized numerical
506 controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.
- 507 5. International Organization for Standardization. *ISO 6983/1* – Numerical Control of
508 machines – Program format and definition of address words – Part 1: Data format for
509 positioning, line and contouring control systems. Geneva, Switzerland, 1982.
- 510 6. Electronic Industries Association. *ANSI/EIA-494-B-1992*, 32 Bit Binary CL (BCL) and 7
511 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines.
512 Washington, D.C. 1992.
- 513 7. National Aerospace Standard. *Uniform Cutting Tests* - NAS Series: Metal Cutting
514 Equipment Specifications. Washington, D.C. 1969.
- 515 8. International Organization for Standardization. *ISO 10303-11*: 1994, Industrial
516 automation systems and integration Product data representation and exchange Part 11:
517 Description methods: The EXPRESS language reference manual. Geneva, Switzerland,
518 1994.
- 519 9. International Organization for Standardization. *ISO 10303-21*: 1996, Industrial
520 automation systems and integration -- Product data representation and exchange -- Part
521 21: Implementation methods: Clear text encoding of the exchange structure. Geneva,
522 Switzerland, 1996.
- 523 10. H.L. Horton, F.D. Jones, and E. Oberg. *Machinery's handbook*. Industrial Press, Inc. New
524 York, 1984.
- 525 11. International Organization for Standardization. *ISO 841-2001: Industrial automation*
526 *systems and integration - Numerical control of machines - Coordinate systems and*
527 *motion nomenclature*. Geneva, Switzerland, 2001.

- 528 12. ASME B5.57: *Methods for Performance Evaluation of Computer Numerically Controlled*
529 *Lathes and Turning Centers*, 1998
- 530 13. ASME/ANSI B5.54: *Methods for Performance Evaluation of Computer Numerically*
531 *Controlled Machining Centers*. 2005.
- 532 14. OPC Foundation. *OPC Unified Architecture Specification, Part 1: Concepts Version 1.00.*
533 *July 28, 2006.*
- 534 15. IEEE STD 1451.0-2007, *Standard for a Smart Transducer Interface for Sensors and*
535 *Actuators – Common Functions, Communication Protocols, and Transducer Electronic*
536 *Data Sheet (TEDS) Formats*, IEEE Instrumentation and Measurement Society, TC-9, The
537 *Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH99684,*
538 *October 5, 2007.*
- 539 16. IEEE STD 1451.4-1994, *Standard for a Smart Transducer Interface for Sensors and*
540 *Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data*
541 *Sheet (TEDS) Formats*, IEEE Instrumentation and Measurement Society, TC-9, The
542 *Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH95225,*
543 *December 15, 2004.*