MTConnect® Standard

Part 3.1 – Interfaces

Version 1.3.0

Prepared for: MTConnect Institute
Prepared by: William Sobel
Prepared on: September 30, 2014

MTConnect® Specification and Materials

Table of Contents

Table of Figures

1 Overview

MTConnect® is a standard based on an open protocol for data integration. MTConnect® is not intended to replace the functionality of existing products, but it strives to enhance the data acquisition capabilities of devices and applications and move toward a plug-and-play environment to reduce the cost of integration.

MTConnect® is built upon the most prevalent standards in the manufacturing and software industries, maximizing the number of tools available for its implementation and providing the highest level of interoperability with other standards and tools in these industries.

To facilitate this level of interoperability, a number of objectives are being met. Foremost is the ability to transfer data via a standard protocol which includes:

- A device identity (i.e. model number, serial number, calibration data, etc.).

- The identity of all the independent components of the device.

- Possibly a device's design characteristics (i.e. axis length, maximum speeds, device thresholds, etc.).

- Most importantly, data captured in real or near-real-time (i.e. current speed, position data, temperature data, program block, etc.) by a device that can be utilized by other devices or applications (e.g. utilized by maintenance diagnostic systems, management production information systems, CAM products, etc.).

The types of data that may need to be addressed in MTConnect® could include:

- Physical and actual device design data

- Measurement or calibration data

- Near-real-time data from the device

To accommodate the vast amount of different types of devices and information that may come into play, MTConnect® will provide a common high-level vocabulary and structure.

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

1.1 MTConnect® Document Structure

The MTConnect® specification is subdivided using the following scheme:

Part 1: Overview and Protocol

Part 2: Components and Data Items

Part 3: Streams, Events, Samples, and Condition

Part 3.1: Interfaces

Part 4: Assets

Part 4.1: Cutting Tools

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

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

MTC_Part_<Number>_<Description>.doc.

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

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

2 Purpose of This Document

The four base MTConnect® documents are intended to:

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

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

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

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

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

2.1 Terminology

Adapter	An optional software component that connects the Agent to the Device.
Agent	A process that implements the MTConnect® HTTP protocol, XML generation, and MTConnect protocol.
Alarm	An alarm indicates an event that requires attention and indicates a deviation from normal operation. Alarms are reported in MTConnect as Condition.
Application	A process or set of processes that access the MTConnect® *Agent* to perform some task.
Attribute	A part of an XML element that provides additional information about that XML element. For example, the name XML element of the Device is given as <Device name="mill-1">...</Device>

CDATA The text in a simple content element. For example, `This is some text`, in `<Message ...>This is some text</Message>`.

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

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

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

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

Device A piece of equipment capable of performing an operation. A device may be composed of a set of components that provide data to the application. The device is a separate entity with at least one component or data item providing information about the device.

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

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

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

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

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

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

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

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

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

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

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

Stream A collection of `Events, Samples,` and `Condition` organized by devices and components.

Service An application that provides necessary functionality.

Tag Used to reference an instance of an XML element.

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

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

UUID Universally unique identifier.

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

XML Extensible Markup Language. http://www.w3.org/XML/

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

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

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

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

2.2 Terminology and Conventions

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

3 Interfaces

Interfaces are special types of components since they are a representation of a physical or logical connection between two devices. The interface components have data items of the event category with types specific to their function. Each interface data item type **MUST** have one of the following two subtypes: `REQUEST` or `RESPONSE`. The requestor is the side of the interface that asks the response side if a task or action can be performed and at the same time is indicating that it is waiting for that action to be completed before it can proceed. For example, if a robot wants a machine tool to open a door, the robot is the request and the machine tool is the response side. On the other hand, if the machine tool wants material to be loaded, the machine tool is the requestor and the response is the robot.

The interface model is based on read-only communications between two devices. This means there is no direct way to change the state of the associated device nor to cause an action to occur. Each device **MUST** present their current state and the associated device **MUST** respond to the changes in state when and if they are capable. The requests are akin to asking for a task to be completed and any device capable is able to respond in kind. The implementation details of how the device completes the requested task is up to the implementor, but the requestor merely concerned with the results (completion) of the task.

Instead of setting a variable on a remote device or raising the voltage on a wire, MTConnect has each device announce the request to any application or device monitoring its state. In effect, the request interface is a simple state machine that goes from `NOT_READY` to `READY`. And when the interface would like the activity performed, it transitions to `ACTIVE`.

In turn, the responder announces when it is `ACTIVE`ly performing the action and then changes to the `COMPLETE` state when it has completed the task. This pattern provides the basis for the coordination of actions between devices. Currently this scheme has been tested between two devices paired directly. In the future enhancements section of this document, we will discuss some ideas to extend this paradigm to multiple interconnected devices supporting cells or even swarms of devices in a mobile environment.

Messaging at this higher semantic level provides the basis for a lighter weight protocol, has simplified failure recovery scenarios and reduced coupling of inter-related devices. Decoupling of devices allows us to more readily replace parts of the system, for example, one Robot with another Robot that can perform the required activity. The interface layer stay the same since any device capable of handling material is capable of doing the work.

Both devices **MUST** have an `Interfaces` top-level component and as subcomponents there will be various interface component types as sub-components. The four components currently specified in the standard are `BarFeederInterface`, `MaterialHandlerInterface`, `DoorInterface`, and `ChuckInterface`. One of these components **MAY** be used to implement the corresponding interface. Additional interfaces can be defined by extending the standard.

Each interface component has data items that define the actions the interface can perform. For example, a `DoorInterface` will have two actions `OPEN_DOOR` and `CLOSE_DOOR` as the data item types for the two `EVENT`s. There are two subtypes for each action, `REQUEST` and

RESPONSE. These subtypes **MUST** be specified to define the role the interface will be playing in the interaction between the devices.

These actions are built using the EVENT category of the MTConnect data item since there is no special behavior required in addition to the EVENT functionality. Action Sub-Types

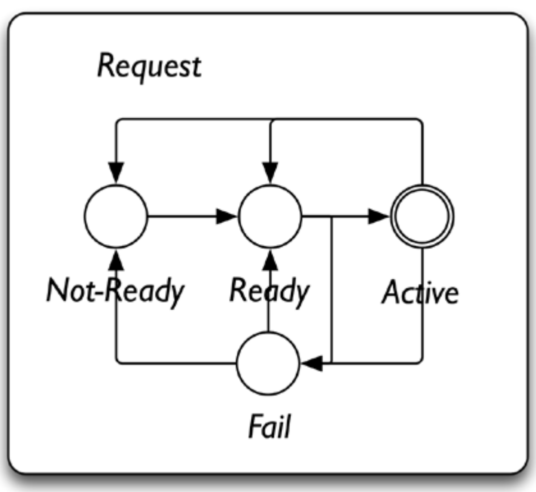
The REQUEST and RESPONSE roles are defined in the subType of the data item. There are four states for the REQUEST and five states for the RESPONSE since the RESPONSE can also COMPLETE the action it is performing. These states determine the interaction of the two participants and provide the basis of communication.

3.1 Request States

The request side is the simpler side; there are four request states, NOT_READY, READY, ACTIVE, and FAIL. The definition of the four states are as follows:

State	Description
NOT_READY	The requestor is not ready to make a request. This means that the response does not need to monitor the device state until the interface becomes READY.
READY	The requestor is in an idle state and will transition to active when it needs the interface action to be completed.
ACTIVE	The requestor is waiting for the action to be started and completed by the responder.
FAIL	The requestor has detected a failure condition and is stopping the action before it completes. Fail **SHOULD** occur after the request is active, but **MAY** occur after it is READY if the action fails before it can transition to ACTIVE.

The following diagram shows the possible state transitions for the request:

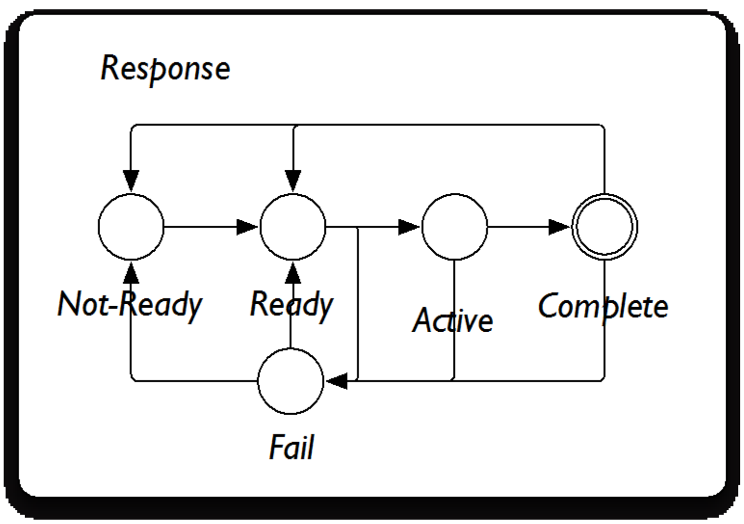


3.2 Response States

There are five response states, `NOT_READY`, `READY`, `ACTIVE`, `COMPLETE`, and `FAIL`. The definition of the five states are as follows:

State	Description
`NOT_READY`	The response is not ready to perform the action. The request state **SHOULD** not become `ACTIVE` when the response state is `NOT_READY`.
`READY`	The response is in an idle state and will transition to `ACTIVE` when it detects the request becomes `ACTIVE`.
`ACTIVE`	The response is actively performing the action.
`FAIL`	The response has failed to perform the action. The failure **SHOULD** wait for the request to acknowledge the FAIL state and transition back to `READY` or `NOT_READY`.
`COMPLETE`	The action is now completed. The response **MUST** wait for the requestor to transition from `ACTIVE` before it transitions back to `READY` or `NOT_READY`.

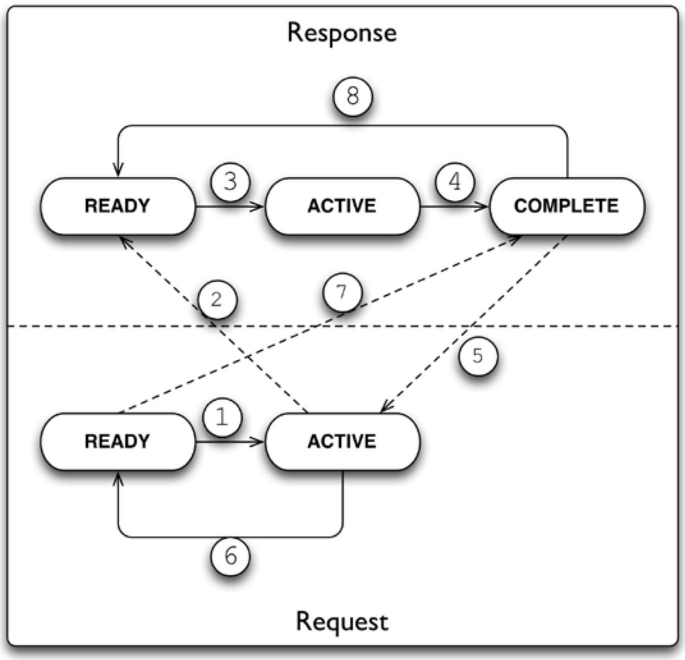
The following diagram shows the possible state transitions for the response:



3.3 Request/Response Interaction

Each request/response pair is meant to operate in tandem. The following is a description of the states and their transitions. In the following diagram shows the state transitions.

NOTE: The `FAIL` and `NOT_READY` states are not specified in this diagram, they have been removed to reduce clutter:



The initial state of both data items is the READY state. The dotted lines indicate observations when the change is detected.

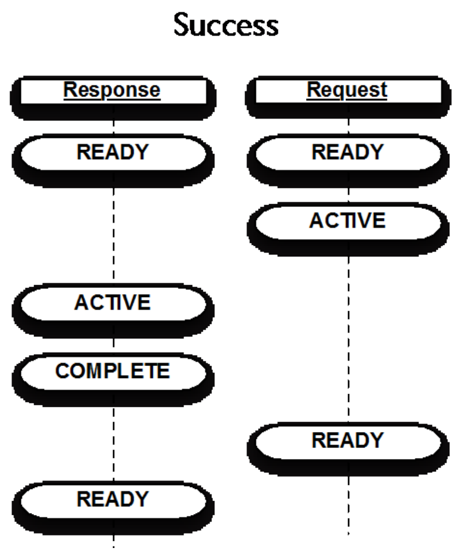
Step	Description
1	The request transitions from READY to ACTIVE signaling it request the action be performed.
2	The response detects the transition of the request.
3	The response transitions from READY to ACTIVE indicating that it is performing the action.
4	Once the action has been performed, it transitions to COMPLETE.
5	The request detects the action is COMPLETE.
6	The request transitions back to READY indicating it acknowledges the action.
7	The response detects the request is now READY.
8	In recognition of the acknowledgement, the response transitions back to READY.

After this action, both devices are in the READY state and can perform another action. Either side can transition to the NOT_READY state after an action is performed. This is a common instance in the case where the action changes the state of the device. An example of this is the DoorInterface and the OPEN_DOOR/CLOSE_DOOR data item. If the DOOR_STATE is OPEN, the OPEN_DOOR data item will have the value NOT_READY since you cannot open a door when it is already opened.

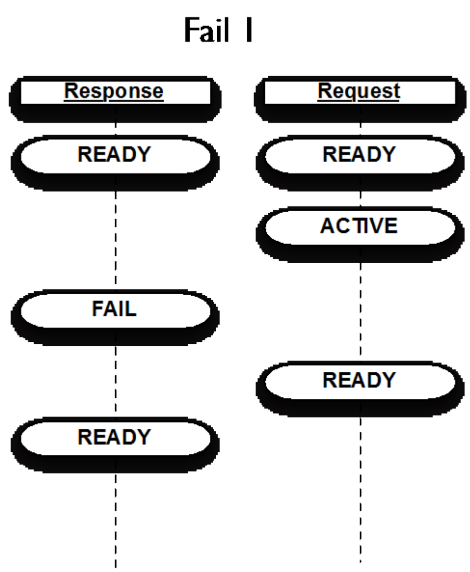
As stated before, there is no direct messaging. The request and response are watching their counterpart's transitions and making appropriate changes to their state in response to what they observe. The response is responsible for transitioning their state and making the necessary internal changes to the device to fulfill the requirements of the request.

3.4 Failure Handling

The first scenario will show the transitions in the successful scenario to provide the basis for the subsequent discussions.

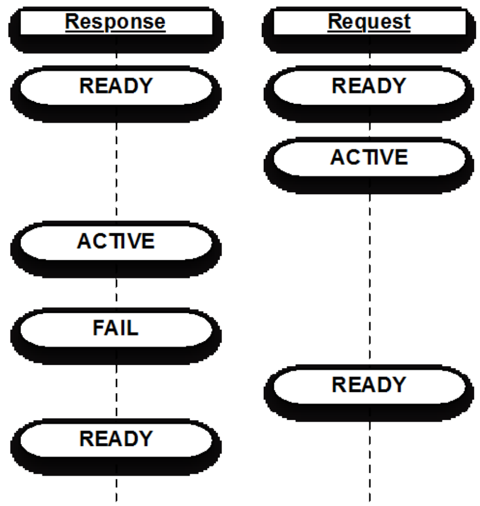


In the first simplest failure scenario, the request transitions to `ACTIVE` and the response immediately fails, meaning it tries to start and immediately stop before even making it to `ACTIVE`. When this occurs the request **MUST** transition back to `READY`.



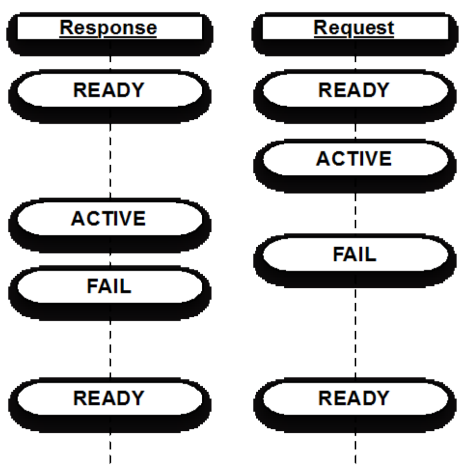
In the second, and most common scenario, the response will begin the action and then fail once it attempts to perform the action. The behavior is almost identical to the previous example where the request transactions back to ready once it detects the response failure.

Fail 2



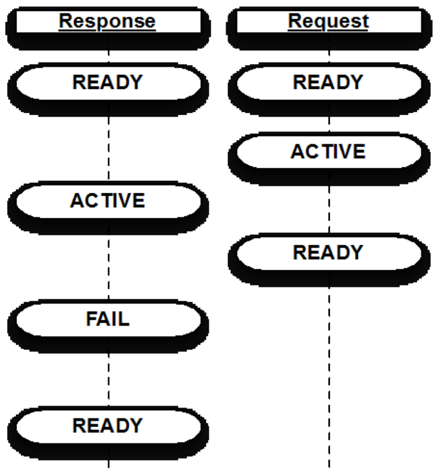
In the case where the requests detects a failure while the responder is performing the action, it will transition itself to `FAIL` and the response will fail as well. A `FAIL` can transition back to a `READY` when the counterpart is also in a FAIL state. It can also transition back to `READY` after a period of time to avoid both sides being stuck in `FAIL`.

Fail 3



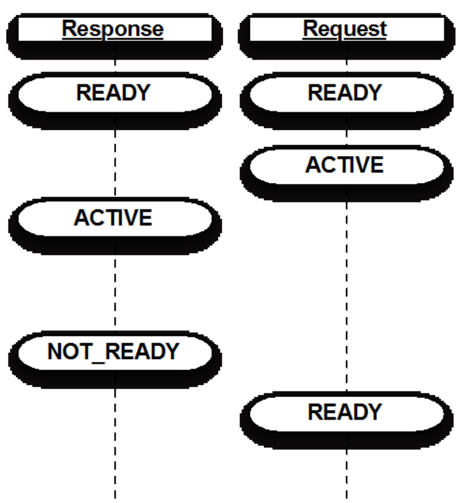
In some cases the `FAIL` state will not be available on a device. Once `ACTIVE`, a transition out of the `ACTIVE` state back to `READY` or `NOT_READY` before the response has completed will have the same effect as a `FAIL` state. This is to provide a solution where the device requesting or responding has a simple logic control and is not capable of determining the difference between stopping an action and failing an action.

Fail 4



The same will happen if the response transitions back to `READY` or `NOT_READY` before going to `COMPLETE`. The response **MUST** `COMPLETE` the action before it can be considered successful. In this case the response transitions back to `READY` before `COMPLETE`. This **MUST** be handled in the same manor as if the response had transitioned to `FAIL` first.

Fail 5



The final case is if any of these devices become unavailable or do not send a heartbeat within the desired amount of time. Upon reconnection, the state of the counterpart is ascertained and an appropriate action can be taken. Sometimes human intervention will be required to restore the devices to an operational state.

3.5 Additional State

The request or response **MAY** use any additional information within the event stream to validate the operation as well as provide contextual information regarding the task in progress. For example, the current PartId can be used to determine specific part handling between devices and communicate to the receiving party what to expect. This information can cascade between devices and allows for completely read-only data flow where the devices communicate their knowledge and the responder uses it to modify its behavior and pass the information along to subsequent devices.

Using the new `References` capability in the standard, critical data items required for proper operation can be associated with the `Interface` as a more efficient way of requesting the interface specific information using a more economical syntax. This will also allow for more efficient near-time information flows between devices and faster failure detection.

3.6 Future Enhancements

The next step in the inter-device specification will be supporting multiple connected devices with the same interface. For example, a cell with a robot, machine tool, and a CMM will each have material handling interface and the robot will load and unload each machine. As we move into more complex scenarios, the standard will be enhanced to accommodate the increased level of interaction.

Appendices

A. Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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