NIST Technical Note 1754

Factory Equipment Network Testing Framework: Universal Client Application, Application Programming Interface

Jim Gilsinn
Kang Lee
John Michaloski
Fred Proctor
Eugene Song

http://dx.doi.org/10.6028/NIST.TN.1754

National Institute of Standards and Technology
U.S. Department of Commerce

NIST Technical Note 1754

Factory Equipment Network Testing Framework: Universal Client Application, Application Programming Interface

Jim Gilsinn
Kang Lee
Fred Proctor
John Michaloski
Yuyin Song
Engineering Laboratory (EL)

http://dx.doi.org/10.6028/NIST.TN.1754

September 2012

U.S. Department of Commerce
Rebecca M. Blank, Acting Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director

National Institute of Standards and Technology Technical Note 1754
Natl. Inst. Stand. Technol. Tech. Note 1754, 13 Pages (September 2012)
CODEN: NTNOEF

Table of Contents

Acronyms

API	Application programming interface
CIP	Common industrial protocol
DUT	Device under test
EL	Engineering laboratory
FENT	Factory equipment network equipment
ID	Identifier
IP	Internet protocol
NIST	U.S. National Institute of Standards & Technology
PM	Personality module
TCP	Transmission control protocol
UCA	Universal client application
UDP	User datagram protocol
XML	Extensible markup language
XSD	XML schema definition

1 Introduction

This document describes the application programming interface (API) used by the Factory Equipment Network Testing (FENT) Framework to communicate between the main Universal Client Application (UCA) and the Personality Module (PM). This API abstracts a small set of messages that are common to many different types of industrial devices, allowing application and code modularity. For more information about how the UCA-API fits into the overall FENT Framework architecture, see [1].

2 UCA-API Overview

The UCA-API specifies a common set of simple messages that can be passed between the UCA and the PM in the FENT Framework. This specification provides code modularity for the UCA and PMs by allowing the internal connection parameters and maintenance functions to be handled by the driver application inside the PM without the UCA needing to know the internal mechanisms or details. This approach means that the UCA can be designed to more easily work with multiple protocols without the need to build-in large portions of protocol stack code in the main framework testing application. Smaller, more dedicated applications, services, or functions are used as PM driver applications that conduct the actual protocol handling. Figure 1 shows a simplified version of the full FENT Framework architecture (see [1]) that concentrates on the UCA-API and its interactions with the UCA, PM, and device under test (DUT).

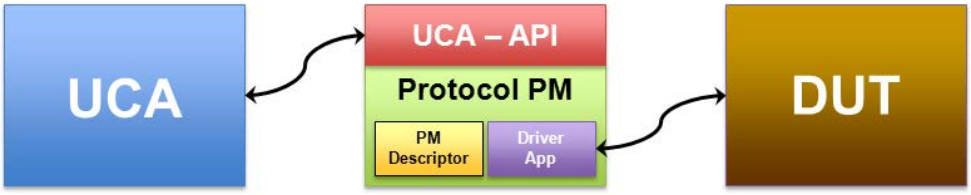

Figure 1 – Simplified Architecture Diagram Showing UCA-API

3 Message Structure

The UCA-API message structure is defined using an extensible markup language (XML) schema definition (XSD) to provide a well-known structure with both attributes and properties for each message. XML was chosen due to its wide adoption, text-based structure, ease-of-use, and human-readability. Going along with the FENT Framework's requirement for code modularity, PMs can be designed using a variety of programming languages and run on a variety of platforms. Using XML and XSD files satisfied this requirement since software libraries exist for many different programming languages and platforms. XML and XSD files also provide a way to encode messages in a common text-based method that can be passed between applications and devices using command-line arguments or sockets.

Figure 2 shows a graphical depiction of the XSD file defining a UCA-API message. A UCA-API message consists of attributes that classify the message and properties to provide additional information related to the message. Attributes apply to most or all messages while properties generally apply to a single type of message, with the exception of the `MessageData` field. Detailed information about each of the attributes and properties can be found in Sections 4 and 5 of this document respectively.

Some of the attributes and all of the properties for the message are shown as optional in the XSD file. This classification is due to the lack of control available in the XSD language to specify relational optionality for the attributes and properties. Be mindful of the Usage field shown for each of the attributes and properties shown in Sections 4 and 5 when building and validating messages using this schema.

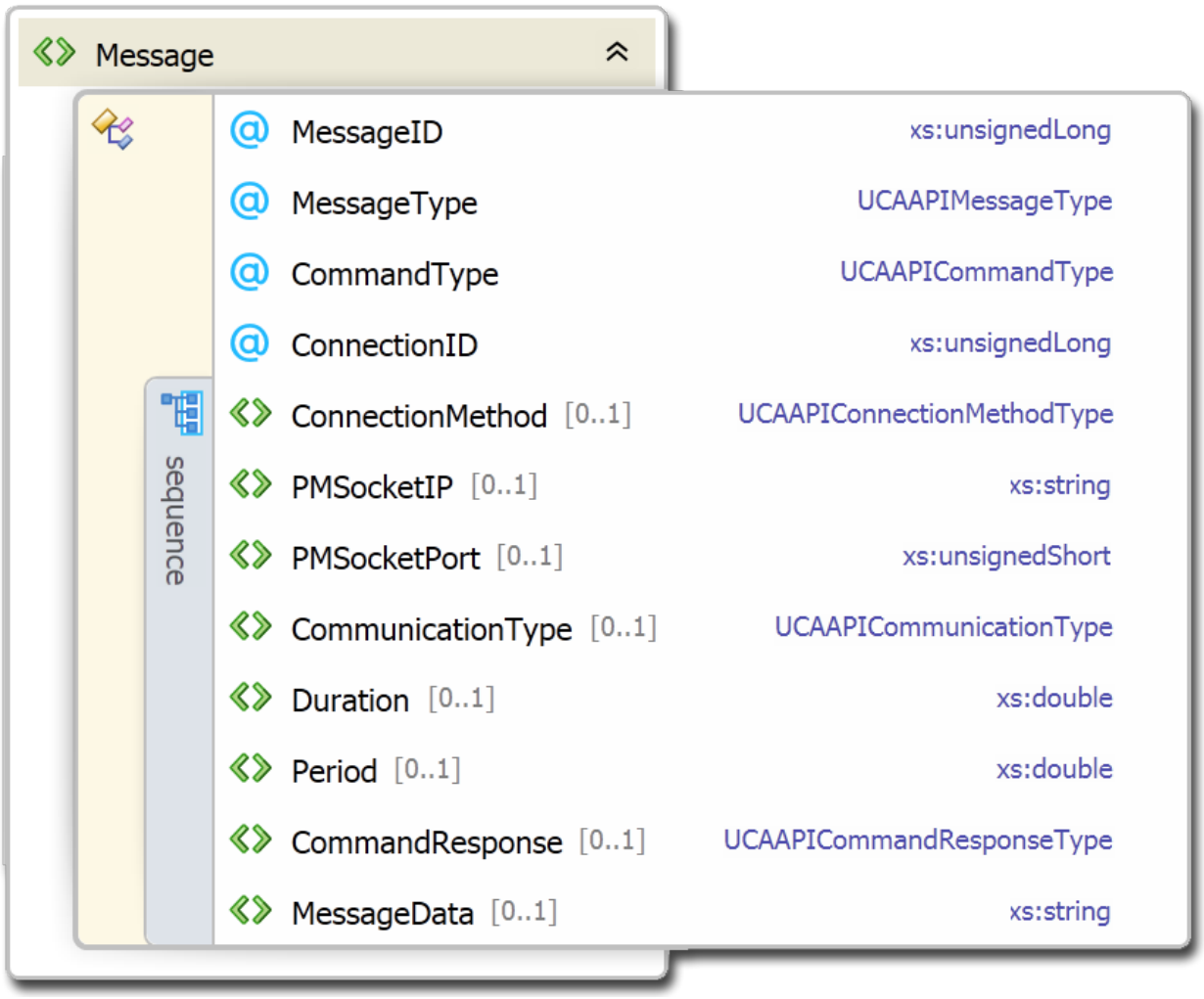

Figure 2 – Graphical Depiction of UCA-API Message XML Schema

4 Message Attributes

4.1 MessageID

Description	A message identifier (ID) to help distinguish individual messages. When a device sends a command to another device, the response from the subordinate device should use the same `MessageID` in its response as the one sent by the commanding device in its request.
Format	`unsignedLong` (32-bit unsigned integer)
Usage	**REQUIRED** – All messages

4.2 MessageType

Description Type of message being sent.

Format `UCAAPIMessageType` (see Table 1)

Usage **REQUIRED** – All messages

Table 1 – String Enumeration for `UCAAPIMessageType`

Enumerated Value	Description
`OpenConnection`	Open a connection to a device.
`CloseConnection`	Close a particular active connection to a device.
`StartCommunication`	Start the desired communication with a device. Used for communication streams that have to be initiated separately from opening a connection.
`StopCommunication`	Stop a particular communication stream with a device. Used for cases where communication streams can be closed separately from the connection to the device.
`Status`	A status message from a device not associated with a particular request message.
`Error`	An error message from a device not associated with a particular request message.

4.3 CommandType

Description Type of command. Indicates whether a particular message is a command request or a response to a request.

Format `UCAAPICommandType` (see Table 2)

Usage **REQUIRED** – All messages except `MessageType="Status"` or `MessageType="Error"`

Table 2 – String Enumeration for `UCAAPICommandType`

Enumerated Value	Description
`Request`	A request by one device to another device to perform some action.
`Response`	A response by one device to a request by another device.

4.4 ConnectionID

Description A connection identifier. Used to distinguish multiple connections between the PM and the DUT. Allows individual connections to the DUT to be opened, closed, started, stopped, and monitored.

CAUTION – Some protocols use connection IDs embedded inside for a similar reason, to distinguish between multiple connections with a particular device. The FENT `ConnectionID` may or may not use the same value as the one used by the PM and the DUT. PMs should be designed to be capable of internally mapping the two connection ID values or changing their FENT `ConnectionID` to match, should the user wish to do so for tracking purposes.

Format `unsignedLong` (32-bit unsigned integer)

Usage **OPTIONAL** – All messages

Only valid for PMs and protocols that support multiple connections to the DUT.

5 Message Properties

5.1 ConectionMethod

Description Method used by the FENT UCA application to communicate with the PM.

Format `UCAAPIConnectionMethodType` (see Table 3)

Usage **REQUIRED** – Only for `MessageType="OpenConnection"`

Table 3 – String Enumeration for `UCAAPIConnectionMethodType`

Enumerated Value	Description
`CommandLine`	The connection is established using an application executable (*.exe) that accepts command line arguments.
`Socket`	The connection is established using a communication socket between the main FENT UCA application and the PM.
`Service`	The connection is established using a service running on the local computer. *(NOT CURRENTLY IMPLEMENTED)*
`FunctionCall`	The connection and messaging between the FENT UCA application and the PM happens through function calls. *(NOT CURRENTLY IMPLEMENTED)*

5.2 PMSocketIP

Description String representing the Internet Protocol (IP) address used to connect to the PM socket.

Format	`string` IP version 4 addresses should be in dotted decimal format, similar to `192.168.0.2`. IP version 6 addresses should be in colon separated 16-bit hexadecimal format, similar to `FE80::202:B3FF:FE1E:8329`. Double colon notation is acceptable. IP version 4 addresses in IP version 6 notation (`::192.168.0.2`) will be interpreted and used as an IP version 6 address.
Usage	**REQUIRED** – Only for `MessageType="OpenConnection"` and `ConnectionMethod="Socket"`

5.3 PMSocketPort

Description	The Transmission Control Protocol (TCP) / User Datagram Protocol (UDP) port number used to connect to the PM socket.
Format	`unsignedShort` (16-bit unsigned integer)
Usage	**REQUIRED** – Only for `MessageType="OpenConnection"` and `ConnectionMethod="Socket"`

5.4 CommunicationType

Description	Communication type used for the connection
Format	`UCAAPICommunicationType` (see Table 4)
Usage	**REQUIRED** – Only for `MessageType="OpenConnection"`

Table 4 – String Enumeration for `UCAAPICommunicationType`

Enumerated Value	Description
`Polled`	Polled or master/slave type communication consisting of one device sending a request to another device that responds to that request.
`Published`	Publish/subscribe type communications where a device publishes information at a particular rate and another device subscribes to that information.
`Triggered`	Triggered type of communications where a device only sends messages after a certain set of conditions are met. This may be the change of state of some measured value (for example a 24 digital signal going from low to high) or some internal condition (for example a watchdog timer being triggered).

5.5 Duration

Description The time duration (in seconds) that one device should communicate with another.

Used for tests where a device should communicate for a particular amount of time and then stop. This duration may occur before the end of the test to measure the system response after the communication stops.

Format `double` (seconds)

Usage **OPTIONAL** – Only for `MessageType="OpenConnection"` or `MessageType="StartCommunication"`

5.6 Period

Description Cyclic period (in seconds) for the PM to communicate with the DUT.

For periodic connections, this value is the cyclic period used by the DUT to publish its information.

For polled connections, this value is the cyclic period used by the PM to poll the DUT.

Format `double` (seconds)

Usage **REQUIRED** – Only for `MessageType="OpenConnection"` and `CommunicationType="Published"`

OPTIONAL – Only for `MessageType="OpenConnection"` and `CommunicationType="Polled"`

5.7 CommandResponse

Description Response from a subordinate device indicating the outcome of a particular command request.

Format `UCAAPICommandResponseType` (see Table 5)

Usage **REQUIRED** – Only for `CommandType="Response"`

Table 5 – String Enumeration for `UCAAPIConnectionMethodType`

Enumerated Value	Description
`Success`	Command was successfully processed. `MessageData` should include any response from the device to successful completion.
`Failure`	Command failed for some reason. `MessageData` should include any relevant error response indicating the reason for failure.

5.8 MessageData

Description	Any data or information that goes along with a particular message. When sending a status or error message, `MessageData` contains the actual status or error message text. When opening a connection to a device, `MessageData` contains any relevant connection or runtime parameters necessary to properly establish the connection.
Format	`string`
Usage	**REQUIRED** – Only for `MessageType="Status"` and `MessageType="Error"` **OPTIONAL** – All other message types.

6 Example Messages

6.1 OpenConnection Example 1 – Request Using Command Line

Open a 10ms cyclic published connection to a device at IP address 192.168.210.20 using a command line PM. The PM would have to accept the command line argument as written in the `MessageData` field. This protocol does not support multiple connections.

```
<Message MessageID="145" MessageType="OpenConnection" CommandType="Request">
  <ConnectionMethod>CommandLine</ConnectionMethod>
  <CommunicationType>Published</CommunicationType>
  <Period>0.010</Period>
  <MessageData>dut_ipaddr=192.168.210.20</MessageData>
</Message>
```

6.2 OpenConnection Example 2 – Request Using Socket

Tell the PM at IP address 192.168.210.20 and port number 14510 to open a polled connection through a socket to IP address 192.168.210.35 and port number 14515. The message should include the series of hex bytes to configure the DUT properly. This protocol allows multiple connections, so it includes a `ConnectionID`.

```
<Message MessageID="37509" MessageType="OpenConnection" CommandType="Request"
ConnectionID="1313428308">
  <ConnectionMethod>Socket</ConnectionMethod>
  <PMSocketIP>192.168.210.20</PMSocketIP>
  <PMSocketPort>14510</PMSocketPort>
  <CommunicationType>Polled</CommunicationType>
  <MessageData>dut_ipaddr=192.168.210.35 dut_socket=14515
dut_config_data=00235E7C48A1</MessageData>
</Message>
```

6.3 OpenConnection Example 3 – Response to OpenConnection Example 2

Example 3 shows a response message to the open connection request in Example 2 indicating that the connection was opened successfully. No additional information was sent by the DUT about the open connection command.

```
<Message MessageID="37509" MessageType="OpenConnection" CommandType="Response"
ConnectionID="1313428308">
  <CommandResponse>Success</CommandResponse>
</Message>
```

6.4 StartCommunication Example

The `OpenConnection` message in Example 2 could represent a device that starts communicating immediately after it receives a connection or for a device that needs to be put into `RUN` mode. This example shows the `StartCommunication` message required to put the device in `RUN` mode for 30 seconds during a particular test.

```
<Message MessageID="37514" MessageType="StartCommunication" ConnectionID="1313428308">
  <Duration>30</Duration>
</Message>
```

6.5 Unsolicited Error Example

The DUT sends an error to the PM which gets reported to the UCA as an unsolicited error message. In this case, this is an EtherNet/IP™ "General Status Code" `0x2B` which refers to an “Unknown Modbus Error”. The interpretation of that error is “A [Common Industrial Protocol (CIP)] to Modbus translator received an unknown a Modbus Exception Code.”

```
<Message MessageID="67328" MessageType="Error">
  <MessageData>STATUS CODE 0x2B</MessageData>
</Message>
```

7 References

[1] *Factory Equipment Network Testing Framework: Concept, Requirements, and Architecture*, NIST Technical Note 1755, September 2012.