A Sierra Monitor Company

FieldServer White Paper

XML As an Industrial Data Communications Protocol

Abstract

With the ubiquity of high speed data networks (10, 100 and 1000 Mbits/sec) the bandwidth 'cost' of transferring bloated English text based data messages has become minimal. XML is a meta Language – a language for describing other languages. SGML is the Standard Generalized Markup Language the international standard for defining descriptions of the structure of different types of electronic document. XML is a subset of SGML. As such it is perfect for defining a data communications protocol where the queries, responses and data are self describing and self documenting. A protocol whose messages do not require a reader to have access to the protocol specification to make sense of them but which still complies with a standard in terms of formatting and parsability has been a popular goal. This paper outlines a method for achieving this.

XML – A Short Primer

XML is an acronym for "Extensible Markup Language."

XML is a project of the World Wide Web Consortium (W3C), and the development of the specification is being supervised by their XML Working Group. A Special Interest Group of co-opted contributors and experts from various fields contributed comments and reviews by email. XML is a public format: it is not a proprietary development of any company. The v1.0 specification was accepted by the W3C as Recommendation on Feb 10, 1998.

XML was developed for the internet as a replacement for HTML.

XML is a markup specification language with which you can design ways of describing information (text or data), usually for storage, transmission, or processing by a program: it says nothing about what you should do with the data (although your choice of element names may hint at what they are for):

As a meta language you must understand that the XML elements do not in themselves describe information or information transmission but the XML elements can be used to define one or more structures of information or information transmissions.

Thus one person may use XML to define a conversation as follows

```
<conversation>
  <greeting>Hello, world!</greeting>
```

```
  <response>Hi!</response>
</conversation>
```

And another person may use the following structure.

```
<conversation>
  <greeting>
    <spoken_by>Speaker1</spoken_by>
    <tone>normal></tone>
    <spoken_text>Hello!</spoken_text>
  </greeting>
  <response>
    <spoken_by>Speaker2</spoken_by>
    <tone>normal></tone>
    <spoken_text>Hi</spoken_text>
  </response>
</conversation>
```

A rough analogy is the use of the Ethernet to connect devices. Many vendors describe their systems as open because they have an Ethernet port on their device. Purchasers discover that they need to use the same protocols to communicate. Thus XML may be used by more than one vendor to define a data transmission protocol but this does not mean that all XML protocols are alike.

So what does XML gives us ? A structured way of defining a data communications protocol. The structure defines the following

- Rules for XML tag names
- Rules for dealing with white space
- Rules for case sensitivity
- Rules for character use
- Rules for Document Type Definitions
- Rules for Well-Formedness
- Etc.

XML – An Example from the Chemical Industry

The Chemical Markup Language (CML) uses the structure of XML to define a method for defining molecules, the attributes and behaviors in such a way that an 'average' chemist can understand the definitions. Read more at http://xml.coverpages.org/cml.html

The fragment below is extracted from "caffeine.xml" on the site http://www.xml-cml.org/

```
  <?xml version="1.0" ?>
- <!-- <?xml-stylesheet type="text/xsl" href="generic.xsl" ?>
  -->
- <document>
- <!--  CML document - caffeine - karne - 7/8/00
  -->
- <!--  file converted from: MDL .mol
  -->
- <cml title="caffeine" id="cml_caffeine_karne" xmlns="x-schema:cml_schema_ie_02.xml">
- <molecule title="caffeine" id="mol_caffeine_karne" convention="mol">
  <formula>C8 H10 N4 O2</formula>
  <string title="CAS">58-08-2</string>
  <string title="ACX">I1001269</string>
  <string title="DOT">UN 1544</string>
  <string title="RTECS">EV6475000</string>
  <float title="molecule weight">194.19</float>
  <float title="melting point" units="degC">238</float>
  <float title="specific gravity">1.23</float>
  <string title="water solubility" units="g/100 mL" convention="g per 100 mL at 23 degC">1-5</string>
  <string title="comments">White powder or white glistening needles usually melted together. LIGHT
SENSITIVE</string>
- <list title="alternate names">
  <string title="name">1,3,7-Trimethylxanthine</string>
  <string title="name">3,7-dihydro-1,3,7-trimethyl-1H-Purine-2,6-dione</string>
  <string title="name">1,3,7-Trimethyl-2,6-dioxopurine</string>
  <string title="name">eldiatric c</string>
  <string title="name">organex</string>
  <string title="name">1,3,7-trimethyl-2,6-dioxo-1,2,3,6-tetrahydropurine</string>
  <string title="name">caffenium</string>
  </list>
- <atomArray>
- <atom id="caffeine_karne_a_1" convention="mol">
  <float builtin="x3" units="A">-2.8709</float>
  <float builtin="y3" units="A">-1.0499</float>
  <float builtin="z3" units="A">0.1718</float>
  <string builtin="elementType">C</string>
  </atom>
- <atom id="caffeine_karne_a_2" convention="mol">
  <float builtin="x3" units="A">-2.9099</float>
  <float builtin="y3" units="A">0.2747</float>
```

An XML Protocol for Industrial Data Communications

The XML protocol was designed, by FieldServer, as a generic communications protocol between devices utilizing XML-like tagging. The benefits of an XML protocol are its open and flexible design, as well as the ease in which it can be read. All data is passed as ASCII text organized within a user defined tagging structure. An XML parser can interpret messages of different structures provided it can recognize the tagging design. This means that devices can send different message contents, in a different order and understand each other if they use the same XML message tagging design. ASCII messages are an open and portable means of transferring data which is both human and machine readable. Without prior knowledge of the specific protocol, the contents of a message can be understood because the data is self describing.

A generic XML protocol was developed for use with devices typically used in conjunction with the Field Server.

The protocol is described below. The XML drivers implement a subset of this protocol. A device which implements this protocol subset can be used in conjunction with these drivers.

Three Implementations of the FieldServer XML Data Communications Protocol

Ethernet TCP

- The XML TCP driver allows the FieldServer to transfer data to and from devices over an Ethernet connection.
- The FieldServer can emulate either a Server or Client. The driver is an active client driver. This means that it initiates read polls with a device which is expected to provide responses. Server functionality is provided by the driver too.
- The driver communicates data between devices via data tables (arrays) of a single data type. As the tables typically contains more than one data element, the retrieved data is stored in a number of consecutive data array locations in the FieldServer.
- The driver can be configured to read a specific variable from an XML device and store its value using optional scaling in a configurable location in a FieldServer data array.
- The FieldServer can emulate a large number of virtual nodes when configured as a server. When configured a s a client the FieldServer can poll any reachable IP address.

Ethernet UDP

The FieldServer can be configured as a client only. It is a producer type driver.

- This means that the driver sends messages to remote devices but does not expect and cannot processes responses.

- This means the driver cannot report whether these messages were received by the remote devices.
- It also means that the driver cannot read data from remote devices.

The driver is intended to 'broadcast' changed data which can be consumed by a remote device(s). Broadcast in this context means send the data messages to all IP addresses on the network or it can mean send a message to each specifically configured IP address.

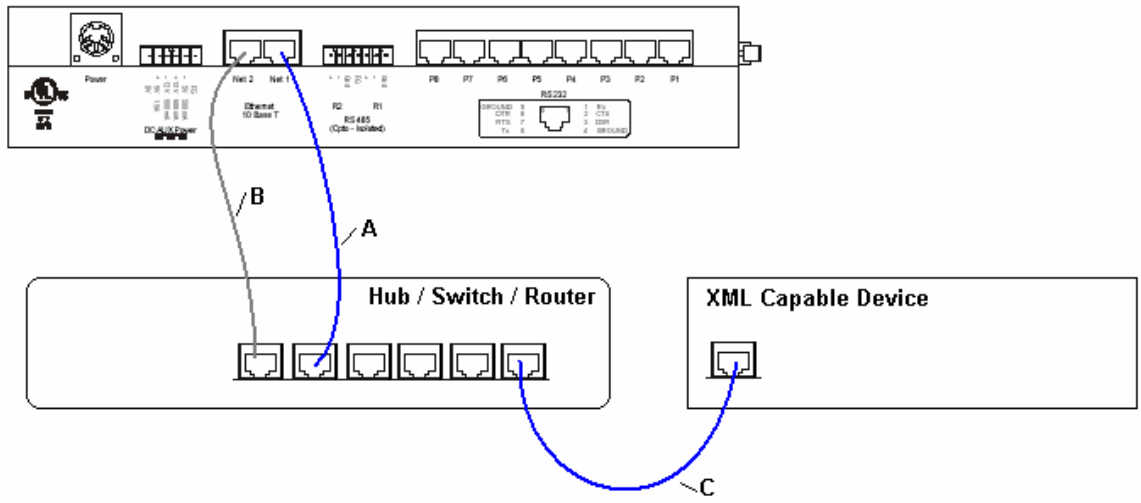
The driver can send messages to remote devices on a periodic basis or when data changes.

Server functionality is provided to support FST's Quality Assurance system. It is not documented but could be extended and documented if required by a customer. If required please discuss this issue with FST's sales group.

Serial

Provides the same functionality as the XML TCP driver but service is provided on RS232 or RS485. These lower speed networks are not well suited to the bandwidth requirements of an XML style protocol.

Typical Connection configurations



Segment A - 'Off the shelf' Ethernet patch cable with RJ45 connectors
Segment B - Alternate to 'A' – Adapters N1 or N2 may be used.
Segment C - 'Off the shelf' Ethernet patch cable with RJ45 connectors

FieldServer XML Data Communications Protocol Specification

In general, when the Field Server is acting as a Client request, messages are always sent in the form of a poll which contains a single query for a read or write action. If the Client is requesting a write, then data is appended to the message as data objects encapsulated within a data block.

As a Server, the Field Server will return messages in the form of a response. The response will contain the information from the original query, an appropriate response type, and a data block if the response contains data values.

Due to the parsing method used for this driver, the data elements must occur in the sequence demonstrated in the following two sample polls and responses.

XML Protocol Elements
<poll>
<response>
<error>
<messageTiming>
<messageStats>
<sequenceNumber>
<scaling>
<dataFormat>
<data>
<dtaOb>
<objectID>
<value>
<query>
<action>
<tablename>
<address>
<length>
<responseRequirement>
<source>
<destination>
<error>

XML Protocol Example

The following is a theoretical example of an XML message construction for a poll, response, broadcast or produced message.

```
<poll> or <response> or <broadcast> or <produced>

        // All message will contain error
        <error></

        // message timing is optional.
        <messageTiming></

        //  message stats are mandatory
        <messageStats></

        // If responseRequirement is omitted then assume 'none'
        <responseRequirement></

        // Source is optional for poll and response but mandatory for broacast
           and produced
        <source>

        // dest is optional for all but produced
        <destination></destination>
        <destination></destination>
        <destination></destination>

        // Query is mandatory in polls and optional in all other messages

        <query></


</poll> or </response> or </broadcast> or </produced>
```

Error Block:

The error block contains both an error number and a description of the error in ASCII separated by a newline. This is a mandatory block for both Client polls and Server responses, and is always the first block after the start of any message.

```
<error>
        //      Error                           Codes
        //      0                               No Error or Ack
        //      positive numbers                Error
        //      Negative  numbers               Warnings
        //      1                               Nak
        //      2                               Nak, no table
        //      3                               Nak, no offset
        //      4                               Nak, offset past end of table
        //      5                               Nak ....        INT16

        ! Add a description field.
</error>
```

Message Timing:

This tag identifies the timestamp of the message with respect to a known standard.

```
<messageTiming>

                // All fields are optional, some are prefered

                // Formats
                //
                //   'UTC'
                //   'GMT'        (toGMTstring like)
                //   'GMT as long int since 1970'
                //   'Local Time as long int since 1970'


                // Only when message is a <poll>
                <querySent format='UTC'></query_sent>


                // Only when message is a <response>
                <queryRcvd format='Secs since 1970'></query_rcvd>

                // Only when message is a <response>
                <responseSent format='UTC'><!response_sent>

                // Makes no sense
                <responseRcvd format='Secs since 1970'></response_rcvd>

</messageTiming>
```

MessageStats:

This field is used in both Client polls and Server responses to identify messages. The sequence number is an incrementing integer counter.

```
<messageStats>
      // message sequence number. Must be unique. Mandatory
      <sequenceNumber>UINT32</>
</messageStats>
```

Response Requirement:

The response requirement is implemented in both Client and Server messages just following the error and messageStats block. It contains the information describing the type of response message expected from the receiver. In the case of a poll for data, data would be expected. For a poll meant for writing data, only an acknowledgement is required.

```
<responseRequirement> none | ackAndData | ackOnly | DataOnly
</responseRequirement>
```

Source Block / Destination Block:

These blocks were not implemented within the serial XML driver. They identify the source and destination addresses of the message including the physical node name, identification, and the Field Server name or unique id.

```
<source>

        <nodeId></node>
        <nodeName>
        <fieldserver></fieldserver>

</source>



<destination>
        <nodeId></node>
        <nodeName>
        <fieldserver></fieldserver>
</destination>
```

Query Block:

The query block contains information pertain to a poll and is included with every Client message and every Server response to echo the original request. For the XML driver, scaling was not implemented and only a single data block outside of a query block was implemented. The data block always follows a query block in any message where appropriate.

```
<query>
        // all fields mandatory except scaling

        <action>read | write</action>
        <tableName></tablename>
        <address>UINT32 (zero referenced)</address>
        <length>UINT16 </length>

        // optional dataFormat

        <dataFormat> UINT16 | char* | long </dataFormat>

        // scaling is optional
        <scaling></scaling>

        // Data is mandatory in a poll when the query action = write
        // Data is optional in a poll when the query action = read
        // Data is optional in a response when the response is to a query where
           the action = write
        // Data is mandatory in a response when the response is to a query
            where the action = read
        // Data is mandatory in <broadcasts> and <produced>

        <data></data>
        <data></data>
        <data></data>
        <data></data>


</query>
```

Data Block:

A data block contains the tablename identifier, as well as one or more data objects. For the XML driver, tablename was not implemented here, but was included in all query blocks. (see below)

```
<data>
        // One data block each time the table or array name changes.

        // -----------

        // mandatory  table name,
        <tableName></
        // 1 or more objects
        <dtaOb></dtaOb>
        ...
        <dtaOb></dtaOb>
</data>
```

Data Object:

The data object is the lowest level descriptor of a single data value. For a poll, a data object contains a unique id (sequence) for the particular message, the address from which the value is read – or written to, and the value. For a response, the data object contains the same elements as the poll with the addition of age and quality elements.

```
<dtaOb>

   // One dtaOb for every 'value' being transmitted

  adr=uint16,val=float,scale_m=float,scale_c=float,qual=int16,id=uint16,age=?


        // id=uint16
        //             Sequence number, zero referenced.
        //             Start at zero, each dtaOb in
        //              a <data></data> block gets the next id.
        // id is optional


        // scaling is optional
        //     scaling is done by y=mx+c
        //     specifiy m with scale_m
        //     specifiy c with scale_c


        // address is a term equivalent to :-
        //                      an index into an array
        //                      an address location
        //                      an offset into a table
        //         is mandatory



        // value, mandatory


        // Quality optional
        //    will be used to describe the value as
        //  normal / forced / obsolete / waiting initialization / pending ...


        // Data age, optional.
        // If age information is available then send this information in some
           kind of absolute time.
```

Scaling:

This tag contains information for scaling data values which is additional to the scaling provided internally with the Field Server through the configuration files.

```
<scaling>
        // All fields are mandatory

        <applyWhen>before response | before store | never | before broadcast</

        // Scaling applied to the data array

        <dataArrayLowScale > -32767 to 32767, default 0 </dataArrayLowScale>
        <dataArrayHighScale >-32767 to 32767, default 100 </dataArrayHighScale>

        // scaling applied to value in the message

        <msgLowScale>-32767 to 32767, default 0 </nodeLowScale>
        <msgHighScale>>-32767 to 32767, default 100 </nodeHighScale>

        <units></>

</scaling>
```

Data Format: (not implement in XML driver)

This tag identifies the datatype being sent within a message. The XML driver treats all data as a FLOAT and stores values in that manner within the Field Server data arrays.

```
<dataFormat>

        BIT | BYTE | UINT16 | INT16 | UINT32 | INT32 | FLOAT | DOUBLE | UINT |
        INT | LONG | ULONG | CHAR

</dataFormat>
```

Examples

```
Sample 1 - Reads 3 elements from an array called DA_AI01 stored at offset 4,5,6

<poll>
        <error>
                0
                NO ERROR
        </error>
```

```
        <messageStats>
                <sequenceNumber>1</sequenceNumber>
        </messageStats>
        <responseRequirement> DataOnly <responseRequirement>

        <query>
                <action> read </action>
                <tableName> DA_AI01</tablename>
                <address> 4 </address>
                <length> 3 </length>
        </query>

</poll>
```

Sample 2 - Response to example 1

```
<response>
        <error>
                0
                NO ERROR
         </error>
        <messageStats>
                <sequenceNumber>1</sequenceNumber>
        </messageStats>
        <responseRequirement> None <responseRequirement>
        <query>
                <action> read </action>
                <tableName> DA_AI01</tablename>
                <address> 4 </address>
                <length> 3 </length>
        </query>
        <data>
                <tableName> DA_AI01</tablename>
                  <dtaOb>
                  adr=0,val=199.0000
                  </dtaOb>
                  <dtaOb>
                  adr=1,val=71.0000
                  </dtaOb>
                  <dtaOb>
                  adr=2,val=71.0000
                  </dtaOb>
        </data>

</response>
```

Revision History

Date	Document Revision	Resp	Comment
11Dec03	0	PMC	Draft. Issued for Internal Review Only.
12Dec03	1	PMC	Corrected minor types. Issued for Release.