Programmers Guide

INDUSTRIAL CONTROL COMMUNICATIONS, INC.

XTPro Specification

January 27, 2015
Version 2

TABLE OF CONTENTS

1. Revision History

Version	Date	Notes
1	11.24.2014	Initial release
2	01.23.2015	Changed TCP port to 843

2. Abbreviations and Terms

Term	Description
Server	Device hosting data (responds to requests)
Client	Device initiating requests
TCP	Transport Control Protocol
IP	Internet Protocol
COV	Change of Value
XML	Extensible Markup Language

3. Introduction

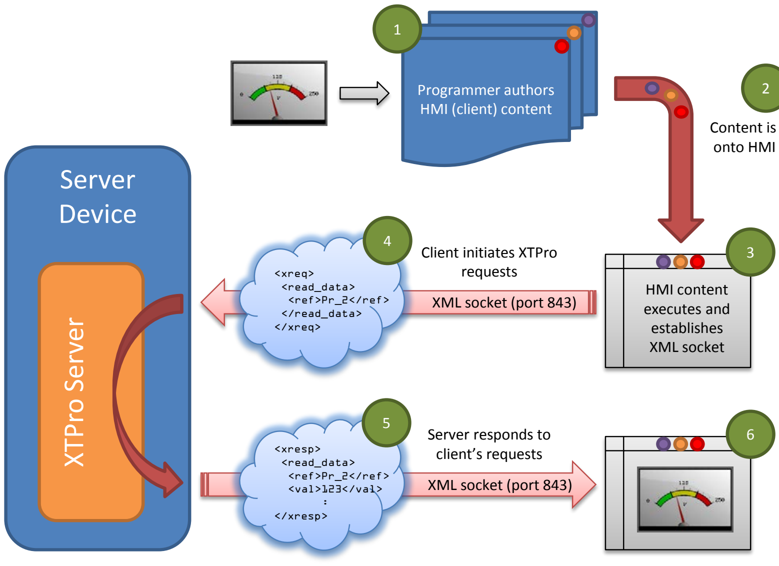
XTPro is an acronym for **X**ML **T**CP/IP **Pro**tocol. The XTPro specification is an application-layer (positioned at level 7 of the OSI model) messaging protocol that provides XML-based client/server communication via TCP port 843. Typically, XTPro is used for the implementation of graphical user interfaces (GUIs), such as advanced web servers or HMIs that have the ability to request information via XML sockets, and then manipulate and/or display the information in a rich application-specific manner.

XTPro is a request/response protocol that provides services specified by commands. This document will describe the format and contents of these commands and provide examples of transactions. For more information on the XML standard itself, please refer to the official reference at http://www.w3.org/TR/2004/REC-xml-20040204/.

Notes
In previous versions, communication may have been performed via TCP port 2000. To prevent firewall and port conflicts, the server device may allow configuration of the XTPro TCP port. Refer to the server device user's manual to confirm the TCP port.

3.1 HMI-Based Implementation

A representative implementation based upon a stand-alone HMI client is detailed in Figure 1. In this scenario, the client application is developed by using tools provided by the HMI manufacturer, and is hosted independently of the actual server device.


1
Programmer authors HMI (client) content
2
Content is loaded onto HMI device
3
HMI content executes and establishes XML socket
4
Client initiates XTPro requests
XML socket (port 843)
Server Device
XTPro Server
5
Server responds to client's requests
XML socket (port 843)
6

```
<xreq>
 <read_data>
  <ref>Pr_2</ref>
 </read_data>
</xreq>
```

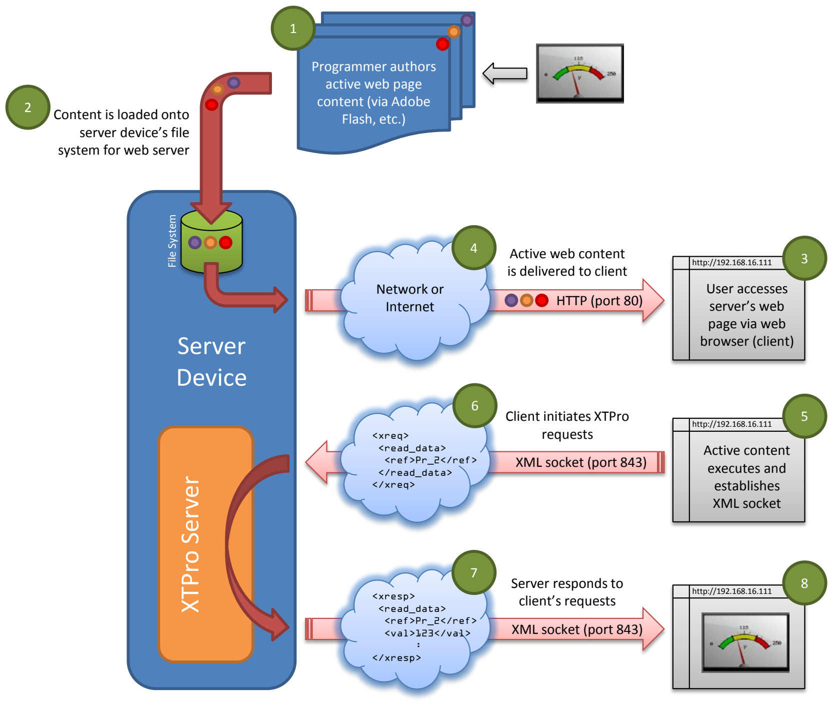
```
<xresp>
 <read_data>
  <ref>Pr_2</ref>
  <val>123</val>
     :
</xresp>
```

Figure 1: HMI-Based Implementation

<u>Notes</u>
In previous versions, communication may have been performed via TCP port 2000. To prevent firewall and port conflicts, the server device may allow configuration of the XTPro TCP port. Refer to the server device user's manual to confirm the TCP port.

3.2 Web Browser-Based Implementation

A representative implementation based upon using a web browser as the client is detailed in Figure 2. In this scenario, the client application is developed by using an active web server authoring tool (such as Adobe Flash®). The active content is then embedded into one or more HTML files and loaded onto the server device's file system (refer to the server device's Instruction Manual for detailed information regarding customization of the web server content). Accessing the device's web server via a standard web browser then loads the active content, which initiates communication with the server.


1
Programmer authors active web page content (via Adobe Flash, etc.)
2
Content is loaded onto server device's file system for web server
File System
Server Device
XTPro Server
4
Network or Internet
Active web content is delivered to client
HTTP (port 80)
3
http://192.168.16.111
User accesses server's web page via web browser (client)
6
Client initiates XTPro requests
XML socket (port 843)
5
http://192.168.16.111
Active content executes and establishes XML socket
7
Server responds to client's requests
XML socket (port 843)
8
http://192.168.16.111

```
<xreq>
 <read_data>
  <ref>Pr_2</ref>
 </read_data>
</xreq>
```

```
<xresp>
 <read_data>
  <ref>Pr_2</ref>
  <val>123</val>
      :
</xresp>
```

Figure 2: Web Browser-Based Implementation

Notes
In previous versions, communication may have been performed via TCP port 2000. To prevent firewall and port conflicts, the server device may allow configuration of the XTPro TCP port. Refer to the server device user's manual to confirm the TCP port.

4. Commands

Refer to Table 1 for a list of available commands. Any other commands not listed are invalid and will therefore result in error responses from the server.

Table 1: Commands

Command	Description	Page
noop	Sets connection to idle mode.	8
vzn	Get specification version.	9
id	Get product name and version.	10
read_data	Read data.	11
write_data	Write data.	12
load_file	Load an XML file from the file system.	13
store_file	Store and XML file to the file system.	14
reinit	Reinitialize the server.	15
auth	Verify authentication.	16
cov	Subscribe to change of value service.	17

Each XML transaction request must begin with an `<xreq>` start tag and may contain only one command. Server responses are encapsulated in `<xresp>` tags. With the exception of COV notification messages, server responses will echo the originally-requested command, and will also contain a status/error code.

The server response time may vary depending on the current network and server utilization/load: the client should allow sufficient time for server responses.

Certain characters may not be supported by all XML parsers. It is therefore recommended to avoid encoding any of special XML characters (predefined entities) listed in Table 2.

Table 2: Special XML Characters

Name	Character	Description
quot	“	Quotation mark
amp	&	Ampersand
apos	‘	Apostrophe
lt	<	Less-than sign
gt	>	Greater-than sign

4.1 noop

The ***noop*** command can be issued in order to stop COV notification messages (refer to section 4.10).

<u>Client Request Format</u>

```
<xreq>
   <noop/>
</xreq>
```

<u>Server Response Format</u>

```
<xresp>
   <noop/>
   <error>error_code</error>
</xresp>
```

<u>Notes</u>

- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.2 vzn

The ***vzn*** command can be issued in order to obtain the XTPro specification version supported by the server prior to any subsequent commands being issued. It is the responsibility of the client to ensure compatibility with the specification version supported by the server.

Client Request Format

```
<xreq>
   <vzn/>
</xreq>
```

Server Response Format

```
<xresp>
   <vzn>version_number</vzn>
   <error>error_code</error>
</xresp>
```

Notes

- *version_number*: Integer decimal value greater than or equal to 1.
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.3 id

The ***id*** command is used to request identification information about the specific device hosting the XTPro server.

Client Request Format

```
<xreq>
   <id/>
</xreq>
```

Server Response Format

```
<xresp>
   <id>
      <name>product_name</name>
      <vendor>vendor_name</vendor>
      <description>product_description</description>
      <vzn1>version_1</vzn1>
      <!-- Additional versions may be included here -->
      <vzn2>version_2</vzn2>
      :
      <vzn#>version_#</vzn#>
   </id>
   <error>error_code</error>
</xresp>
```

Notes

- *product_name*: Server device product name. Required tag.
- *vendor_name*: Vendor/manufacturer name. Optional tag.
- *product_description*: Product description. Optional tag.
- *version_1*: Required version number (typically indicates the server device's main application firmware version). Format is product-specific (refer to the server device's Instruction Manual for detailed information).
- *version_2* … *version_#*: Optional additional version numbers. If provided, format is product-specific (refer to the server device's Instruction Manual for detailed information). There is no explicit limit to the number of additional version numbers that may be provided, but each additional version number will be encapsulated within a unique tag comprised of the characters "`vzn`" plus a sequential numeric index (e.g. "`vzn2`", "`vzn3`", "`vzn4`", etc.)
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.4 read_data

The ***read_data*** command is used to read data from the server at a specified reference.

Client Request Format

```
<xreq>
   <read_data>
      <ref>reference</ref>
   </read_data>
</xreq>
```

Server Response Format

```
<xresp>
   <read_data>
      <ref>reference</ref>
      <val>data_value</val>
   </read_data>
   <error>error_code</error>
</xresp>
```

Notes

- *reference*: Reference targeting the desired data. Format is product-specific (refer to the server device's Instruction Manual for detailed information). The response *reference* field is always an echo of the request *reference* field.
- *data_value*: The reference's current data value. Format is product-specific (refer to the server device's Instruction Manual for detailed information).
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.5 write_data

The ***write_data*** command is used to write data to the server at a specified reference.

Client Request Format

```
<xreq>
   <write_data>
      <ref>reference</ref>
      <val>data_value</val>
   </write_data>
</xreq>
```

Server Response Format

```
<xresp>
   <write_data>
      <ref>reference</ref>
      <val>data_value</val>
   </write_data>
   <error>error_code</error>
</xresp>
```

Notes

- *reference*: Reference targeting the desired data to be written. Format is product-specific (refer to the server device's Instruction Manual for detailed information). The response *reference* field is always an echo of the request *reference* field.
- *data_value*: The data value to write to the reference. Format is product-specific (refer to the server device's Instruction Manual for detailed information). The response *data_value* field is always an echo of the request *data_value* field.
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.6 load_file

The ***load_file*** command is used to load an XML file from the server's file system. Because a nul termination ("0" byte) is used by the server to indicate the end-of-file condition, loading a non-XML file (containing "0" bytes) may result in undefined client behavior. Therefore, this command should only be used to load XML files.

Client Request Format

```
<xreq>
   <load_file>
      <file>file_path</file>
   </load_file>
</xreq>
```

Once the server has received the request, it will then retrieve the requested file from its file system and transmit it (with nul-termination) to the client. The server will then send the following response to indicate the end of the ***load_file*** transaction:

Server Response Format

```
<xresp>
   <load_file>
      <file>file_path</file>
   </load_file>
   <error>error_code</error>
</xresp>
```

Notes

- If the requested file is not found (e.g. invalid *file_path*), then no file is returned, and the only server response is the "server response format" outlined above. An error will also be indicated in the *error_code* field.
- *file_path*: The path to the desired file. Format is product-specific (refer to the server device's Instruction Manual for detailed information). The server response *file_path* field is always an echo of the request *file_path* field.
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.7 store_file

The ***store_file*** command is used to store an XML file to the server's file system. Because a nul termination ("0" byte) must be used by the client to indicate the end-of-file condition, storing a non-XML file (containing "0" bytes) may result in undefined server behavior. Therefore, this command should only be used to store XML files.

Client Request Format

```
<xreq>
   <store_file>
      <file>file_path</file>
   </store_file>
</xreq>
```

Once the client has sent the request, it may then begin transmitting the file (with nul-termination) to the server. When transmission is complete (nul-termination received), the server will then store the received file to its file system. The server will then send the following response to indicate the end of the ***store_file*** transaction:

Server Response Format

```
<xresp>
   <store_file>
      <file>file_path</file>
   </store_file>
   <error>error_code</error>
</xresp>
```

Notes

- *file_path*: The path for the server to use when storing the file. Format is product-specific (refer to the server device's Instruction Manual for detailed information). The server response *file_path* field is always an echo of the request *file_path* field.
- While the filename portion of the *file_path* field is arbitrary, the targeted folder structure must already exist, or the server will reject the store operation. Therefore, if a custom folder structure is to be used on the server's file system, that structure must first be created by alternate means (refer to the server device's Instruction Manual for detailed information on modifying the server's file system).
- If a file with the same filename as the received file currently exists at the targeted location, that file will be overwritten by the received file. Otherwise, the received file will be stored as a new file under the received filename.
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.8 reinit

The ***reinit*** command is used to reinitialize the server device. This command is typically issued by the client after storing a file (via the ***store_file*** command) that is used in some way for server device or driver configuration. In this case, although the configuration file has been stored onto the server device's file system, changes to this configuration file will not take effect until the server device is power cycled, or the ***reinit*** command is issued.

During reinitialization, the XML TCP socket used for XTPro communications will remain intact, but all other device drivers (control protocols, etc.) will be restarted. This may require several seconds to complete, after which time the server will issue the indicated response. Refer to the server device's Instruction Manual for specific information regarding the behavior of this command.

Request Format
```
<xreq>
   <reinit/>
</xreq>
```

Response Format
```
<xresp>
   <reinit/>
   <error>error_code</error>
</xresp>
```

Notes
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.9 auth

The ***auth*** command is used for validating authentication credentials with the server. Note that authentication is not a prerequisite for data communications via XTPro: it is available, however, to provide client programmers with the ability to restrict access to what they deem to be "administrative" features of their user interface if they desire.

For security purposes, the actual authentication credentials always remain stored on the server: the ***auth*** command issues a credential validation request that is either confirmed or rejected by the server. The mechanism of modifying the authentication credentials is not within the scope of XTPro: refer to the server device's Instruction Manual for specific information regarding the modification procedure.

Client Request Format

```
<xreq>
    <auth>
        <user>username</user>
        <pswd>password</pswd>
    </auth>
</xreq>
```

Server Response Format

```
<xresp>
    <auth>
        <user>username</user>
        <pswd>password</pswd>
    </auth>
    <error>error_code</error>
</xresp>
```

Notes

- *username*: The username to authenticate. Refer to the server device's Instruction Manual for detailed information. The server response *username* field is always an echo of the request *username* field.
- *password*: The password to authenticate. Refer to the server device's Instruction Manual for detailed information. The server response *password* field is always an echo of the request *password* field.
- *error_code*: Status/error code. Refer to Table 3 on page 19.

4.10 cov

The ***cov*** command is used for subscribing to the Change-of-Value (COV) notification service on the server. Once a ***cov*** command has been received and acknowledged by the server, it will begin to send unprompted COV notification messages to the client at periodic intervals. Refer to the server device's Instruction Manual for information on the specific notification interval. In comparison to continuously polling the data values of all references on the server with the ***read_data*** command, the COV notification service provides a more efficient mechanism for the client to detect changed data values.

COV notifications will continue to be sent by the server until any other command is issued by the client (including, but not limited to, the ***noop*** command). Once COV notifications have been terminated, the client must reissue a ***cov*** command if it wishes to once again subscribe to the COV notification service.

Client Request Format

```
<xreq>
   <cov/>
</xreq>
```

Server Response Format

```
<xresp>
   <cov/>
   <error>error_code</error>
</xresp>
```

Once the server has sent the COV response message, it will then begin to transmit COV notification messages to the client at a periodic rate. The specific format of these messages will vary depending upon whether or not any data values have changed since the previous COV notification message.

COV Notification Format (no COVs)

```
<xresp>
</xresp>
```

COV Notification Format (one or more COVs)

```
<xresp>
   <cov>
      <ref>reference</ref>
      <val>data_value</val>
   </cov>
   <!-- Additional COVs may be added here -->
</xresp>
```

Notes

- *reference*: The reference of the changed data. Format is product-specific (refer to the server device's Instruction Manual for detailed information).
- *data_value*: The new data value of the reference. Format is product-specific (refer to the server device's Instruction Manual for detailed information).
- *error_code*: Status/error code. Refer to Table 3 on page 19.
- The ***cov*** command may not be supported by all server devices: refer to the server device's Instruction Manual for detailed information.
- When more than one COV has occurred since the server's previous COV notification message, each COV will be included in the next notification message encapsulated within <cov>…</cov> tags.

5. Error Codes

Table 3: Status / Error Codes

Status / Error Code	Description
`none`	No error (success)
`invalid_reference`	Invalid reference
`invalid_value`	Invalid value
`invalid_command`	Unsupported command
`file_does_not_exist`	The file does not exist
`file_error`	File error
`resource_error`	Insufficient resources to complete the transaction. Try again at a later time.
`invalid_authentication`	Invalid authentication
`busy`	Server is busy and cannot complete the transaction at this time. Try again at a later time.
`error`	General/other error
`invalid_path`	Bad file path syntax
`invalid_directory`	The directory does not exist

INDUSTRIAL CONTROL COMMUNICATIONS, INC.

1600 Aspen Commons, Suite 210
Middleton, WI USA 53562-4720
Tel: [608] 831-1255   Fax: [608] 831-2045
http://www.iccdesigns.com