TRIANGLE RESEARCH INTERNATIONAL

Fx2424

With floating point support

Ladder+BASIC
Super PLC

USER'S MANUAL

Revision 1.2

Copyright Notice & Disclaimer

All rights reserved. With the exception of legitimate TRi PLC users, who may print or make copies of this manual for reference purposes, no parts of this manual may be reproduced in any form without the express written permission of TRi.

Triangle Research International, Inc. (TRi) makes no representations or warranties with respect to the contents hereof. In addition, information contained herein are subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, TRi assumes no responsibility for errors or omissions or any damages resulting from the use of the information contained in this publication.

Windows is a trademark of Microsoft Inc.
MODBUS is a trademark of Mobdus.org
All other trademarks belong to their respective owners.

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	2/15/2014	First Release
Rev. 1.0	10/30/2014	Revised section 2.10 on Web Control Added reference to SmartTILE solution for OEMs
Rev. 1.1	12/01/2015	Further revised section 2.10 on Web Control Added reference to SmartTILE solution for OEMs
Rev. 1.2	2/18/2016	Updated specs regarding the number of simultaneous Fserver/Modbus TCP connections

Conditions of Sale and Product Warranty

Triangle Research International Inc. (TRi) and the Buyer agree to the following terms and conditions of Sale and Purchase:

1. The TRi Products are guaranteed against defects in materials or workmanship for a period of one year from the date of registered purchase. Any unit found during this time to have manufacturing defect/s will, at the discretion of TRi, be repaired or replaced.

2. TRi will not be responsible for the repair or replacement of any unit damaged by user modification, negligence, abuse, improper installation, or mishandling.

3. TRi is not responsible to the Buyer for any special, consequential, indirect or similar damages, including any lost profits or loss data arising from the use or the loss of use of TRi products. In no case shall TRi's liability exceed the purchase price for the products.

4. TRi products are NOT certified to be FAILSAFE and hence must NOT be used in applications where failure of the product could lead to physical harm or loss of human life. The Buyer is responsible to conduct his/her own tests to meet the safety regulation of the respective industry.

5. Products distributed, but not manufactured by TRi, carry the full original manufacturers warranty. Such products include, but are not limited to: power supplies, sensors, I/O modules and battery backed RAM.

6. TRi reserves the right to alter any feature or specification at any time with each product revision release.

Notes to Buyer: If you disagree with any of the above terms or conditions you should promptly return the unit to the manufacturer or distributor within 30 days from date of purchase for a full refund.

TABLE OF CONTENTS

Page #

1 INSTALLATION GUIDE FOR FX2424 PLC 1-1

1.1 Overview 1-1

1.2 Physical Mounting & Wiring 1-3
1.2.1 Analog I/O Ports: 1-3
1.2.2 Digital I/O Ports: 1-4
1.2.3 Output Ground Terminals: 1-4

1.3 Power Supply 1-4

1.4 Digital Input Circuits 1-5

1.5 Digital Output Circuits 1-6
1.5.1 Electrical Specifications: 1-6
1.5.2 Digital Output Wiring Diagram 1-6
1.5.3 Inductive Load 1-7

1.6 LCD Display Port 1-7
1.6.1 Wiring Instruction of LCD216 and LCD420 Backlight. 1-8
1.6.2 Programming The LCD Display 1-9

1.7 Program and Data Memory 1-10
1.7.1 Program Memory 1-10
1.7.2 Non Volatile FRAM Memory 1-10
1.7.3 Volatile Data Memory 1-10
1.7.4 Extended Data File Storage 1-11

1.8 DIP SWITCHES 1-12
1.8.1 Usefulness of SW1-4 1-12

1.9 Real Time Clock Battery 1-12

1.10 CPU Status Indicators 1-13
1.10.1 RTC Error (Green LED) 1-13
1.10.2 Pause (Red LED) 1-13
1.10.3 Run Error (Red LED) 1-13

1.11 Create a Custom PLC with SmartTILE 1-14

2 ETHERNET PORT 2-1

2.1 Configuring The Ethernet Port 2-1
2.1.1 IP Address 2-2
2.1.2 GateWay IP Addr 2-3
2.1.3 SMTP Server IP Address 2-3
2.1.4 DNS Server IP Address 2-3
2.1.5 No of Connections (FServer/ Modbus TCP) 2-4
2.1.6 FServer Port No. 2-4
2.1.7 Modbus/TCP Secondary Port No. 2-5
2.1.8 LAN Speed 2-5
2.1.9 Node Name 2-5
2.1.10 Username and Password (FServer only) 2-5
2.1.11 Use Username/Password (Yes/No)? 2-5

2.1.12 Access Level 2-5
2.1.13 Advanced Configuration 2-5
2.1.14 Standalone Version of Ethernet Configuration Software 2-6

2.2 Connecting Ethernet to the PLC 2-7
2.2.1 Connecting the PLC to a Local Area Network 2-7
2.2.2 Setting up Ethernet Communication Directly Between a PC and a F-series PLC 2-9

2.3 On-line Monitoring/Programming via FServer 2-15

2.4 Using F-series PLC "Network Services" Commands 2-16
2.4.1 Get our IP Address 2-17
2.4.2 DNS command: Resolving Domain Name into IP Address 2-17
2.4.3 Send Email 2-18
2.4.4 Open Connection to Remote FServer or TLServer to Use NETCMD$ 2-20
2.4.5 Remote File Services 2-21
2.4.6 Other Network Services Tags 2-21

2.5 MODBUS/TCP Server and Client Connection 2-22
2.5.1 Connecting To The PLC's MODBUS/TCP Server 2-22
2.5.1.1 Bit Address Mapping 2-22
2.5.1.2 Word Address Mapping 2-24
2.5.2 MODBUS/TCP Access Security 2-24
2.5.3 Making a Modbus/TCP Client Connection to Other Modbus/TCP Server 2-26

2.6 Getting data from Internet: Connecting to The Internet Time Server 2-28

2.7 Web Service: Accessing PLC's data from MS Excel 2-29

2.8 Accessing The PLC from Internet 2-31
2.8.1 Small Local Area Network Using Consumer Grade Network Router 2-31
2.8.2 Large Corporate Local Area Network 2-32

2.9 Installing a Control Web Page Into the Fx PLC 2-33
2.9.1 Installing the FileZilla program 2-33
2.9.2 Configure FileZilla to Communicate with the Fx2424 2-33
2.9.3 Transferring and Retrieving Files from the PLC Web Server 2-36
2.9.4 Troubleshooting FileZilla File Transfer Problems 2-37

2.10 Accessing and Customizing the HTML Web Interface for Control and Monitoring 2-39
2.10.1 Next Generation of Super PLC Web Control 2-39

3 PROGRAMMING I/O AND INTERNAL RELAYS 3-1

3.1 Introduction 3-1

3.2 Programming DIO with Ladder Logic 3-1
3.2.1 For Physical I/O 3-1
3.2.2 For Internal Relays (Non-Latching) 3-1
3.2.3 For Internal Relays (Latching) 3-1
3.2.4 Programming Examples: 3-1
3.2.4.1 Example 1 – Editing Label Names 3-1
3.2.4.2 Example 2 – Creating a Simple Ladder Logic Circuit 3-2
3.2.4.3 Example 3 – Creating a Latching Relay Circuit 3-2

3.3 Programming DIO in a Custom Function 3-2
3.3.1 Editing Label Names: 3-3
3.3.2 Controlling I/O from Custom Functions: 3-3
3.3.3 Example 1 – Turn on/off an Output 3-4
3.3.4 Example 2 – Toggle an Output 3-4
3.3.5 Example 3 – Test the Status of an Output 3-4

4 TIMERS, COUNTERS AND SEQUENCERS 4-1

4.1 Introduction 4-1
4.1.1 Timer Coils 4-1
4.1.2 Counter Coils 4-1
4.1.3 Sequencers 4-1

4.2 Programming timers and counters on Ladder Logic 4-2
4.2.1 For Timers 4-2
4.2.2 For Counters 4-2
4.2.3 Example 1 – Creating a Simple Timer Circuit in Ladder Logic 4-2
4.2.4 Example 2 – Creating a Simple Counter Circuit in Ladder Logic 4-3

4.3 Programming timers and counters in Custom Function 4-4
4.3.1 Timers and Counters Present Values 4-4
4.3.2 Inputs, Outputs, Relays, Timers and Counters Contacts 4-4
4.3.3 Changing The Timer and Counter Set Values in a Custom Function 4-4
4.3.4 Controlling a Timer or Counter in a Custom Function 4-4

4.4 Programming Sequencers on Ladder Logic 4-5
4.4.1 Introduction 4-5
4.4.2 Advance Sequencer - [AVseq] 4-6
4.4.3 Resetting Sequencer - [RSseq] 4-6
4.4.4 Setting Sequencer to Step N - [StepN] 4-6
4.4.5 Reversing a Sequencer 4-6
4.4.6 Program Example 4-6

4.5 Programming Sequencers in Custom Function 4-7

5 ANALOG INPUTS AND OUTPUTS 5-1

5.1 Analog Power Supply 5-1

5.2 Analog Inputs 5-1
5.2.1 A/D #1 to 4 5-2
5.2.2 A/D #5 to 8 5-2
5.2.3 Interfacing to two-wire 4-20mA sensors 5-3
5.2.4 Using Potentiometer to Set Parameters 5-4
5.2.5 Reading Analog Input Data 5-4
5.2.6 Moving Average 5-5
5.2.7 Scaling of Analog Data 5-5

5.3 Temperature Measurement Using Analog Inputs 5-6
5.3.1 Thermistor Temperature Sensors 5-6
5.3.2 Using LM34 Semiconductor Sensor 5-7
5.3.3 Using Thermocouple 5-7
5.3.4 Using PT100 Temperature Sensor 5-7

5.4 Analog Outputs 5-8
5.4.1 Programming The Analog Output 5-8

5.5 Calibration of ADC and DAC & Moving Average Definition 5-9
5.5.1 ADC Calib. 5-9
5.5.2 ADC Zero Offset 5-10
5.5.3 DAC Calib. 5-10
5.5.4 DAC Zero Offset 5-11
5.5.5 A/D Moving Avg 5-11

6 SPECIAL DIGITAL I/OS 6-1

7 HIGH SPEED COUNTERS 7-1

7.1 Introduction 7-1

7.2 Enhanced Quadrature Decoding 7-2

7.3 Interfacing to 5V type Quadrature Encoder 7-2

1.1.1.1 F-Series PLC 1

8 FREQUENCY / SPEED MEASUREMENT 8-1

8.1 Programming of PM Input 8-1

8.2 Applications 8-2

8.2.1 Measuring RPM Of A Motor 8-2

8.2.2 Measuring Transducer with VCO Outputs 8-2

8.2.3 Measuring Transducer with PWM Outputs 8-2

8.3 Frequency Measurement on High Speed Counter Inputs 8-3

9 INTERRUPTS 9-1

9.1 Input Interrupts 9-1

9.2 Periodic Timer Interrupt (PTI) 9-2

9.3 Power Failure Interrupt (PFI) 9-2

10 STEPPER MOTOR CONTROL 10-1

10.1 Technical Specifications: 10-1

10.2 Stepper Motor Driver 10-1

10.2.1 Stepper Motor Wiring Types 10-1

10.2.2 Damping Circuits (Inductive Bypass) 10-2

10.3 Stepper Motor Controller 10-3

10.3.1 Interfacing to 5V Stepper Motor Driver Inputs 10-4

10.4 Programming Stepper Control Channel 10-5

10.4.1 Introduction 10-5

10.4.2 Full-Step Driver Mode: 10-5

10.4.3 Half-Step Driver Mode: 10-5

10.4.4 Setting the Acceleration Properties 10-5

10.4.5 Using the STEPMOVE Command 10-6

10.4.5.1 Example: PLC as a Motor Controller 10-6

10.4.5.2 Example: PLC as a Full-Step Motor Driver 10-7

10.4.5.3 Example: PLC as a Half-Step Motor Driver 10-7

10.4.6 Using the STEPMOVEABS Command 10-7

10.4.6.1 Example: PLC as a Motor Controller 10-8

10.4.6.2 Example: PLC as a Full-Step Motor Driver 10-8

10.4.6.3 Example: PLC as a Half-Step Motor Driver 10-8

10.4.7 Demo Program for Stepper Motor Control 10-8

11 PULSE WIDTH MODULATED OUTPUTS 11-1

11.1 Introduction 11-1

11.2 Fx2424 PLC PWM Outputs 11-1

11.3 Increasing Output Drive Current (Opto-Isolated) 11-2

11.4 Position Control Of RC Servo Motor 11-3
11.4.1 Using Fx2424 PWM Output To Control RC Servo (Non-Isolated) 11-4
11.4.2 Using Fx2424 PWM Output To Control RC Servo (Opto-Isolated) 11-5
11.4.3 RC Servo Positioning Resolution 11-5

12 REAL TIME CLOCK 12-1

12.1 Introduction 12-1

12.2 TBASIC variables Used for Real Time Clock 12-1

12.3 RTC Error Status On Ladder Logic 12-1

12.4 Setting the RTC Using TRiLOGI Software 12-2

12.5 Setting the RTC Using TBASIC 12-2

12.6 Setting the RTC from Internet Time Server 12-3

12.7 Setting up an Alarm Event in TBASIC 12-3

12.8 RTC Calibration 12-3

12.9 Troubleshooting the RTC 12-4

13 LCD DISPLAY PROGRAMMING 13-1

13.1 SETLCD Command 13-1

13.2 Special Commands For LCD Display 13-1

13.3 Displaying Numeric Variable With Multiple Digits 13-1

13.4 Displaying Floating-point 13-2

14 SERIAL COMMUNICATIONS 14-1

14.1 Introduction: 14-1

14.2 COMM1: RS232C Port with Female DB9 Connector 14-1

14.3 COMM2 / COMM3: Two-wire RS485 Port 14-2
14.3.1 PROGRAMMING AND MONITORING 14-2
14.3.2 Accessing 3rd Party RS485-based Devices 14-3
14.3.3 Interfacing Other Devices to Modbus Host or to the Internet 14-3
14.3.4 Distributed Control 14-3

14.4 Changing Baud Rate and Communication Formats: Use of the SETBAUD Statement 14-3

14.5 Support of Multiple Communication Protocols 14-5

14.6 Accessing the COMM Ports from within TBASIC 14-6

14.7 Using The PLC As a Modbus / Omron Slave – SCADA, HMI Applications 14-8
14.7.1 MODBUS ASCII Protocol Support 14-8
14.7.1.1 BIT ADDRESS MAPPING 14-10
14.7.1.2 WORD ADDRESS MAPPING 14-11
14.7.2 MODBUS RTU Protocol Support 14-11
14.7.3 OMRON Host Link Command Support 14-11
14.7.4 Application Example: Interfacing to SCADA Software 14-12

14.8 Using The PLC As a Modbus Master – Getting Data From Power or Flow Meters **..... 14-13**
14.8.1 Fx2424 PLC As MODBUS RTU Master 14-13

14.9 Using Modem to Remotely Program/Monitor The PLC 14-14
14.9.1 Wiring 14-14
14.9.2 Programming 14-14

15 HOST LINK PROTOCOL INTRODUCTION 15-1

15.1 Multiple Communication Protocols 15-1

15.2 Native Mode Communication Protocols 15-1

15.3 Point-To-Point Communication Format 15-1
15.3.1 Command/Response Frame Format (Point to Point) 15-2
15.3.2 Error Response Format 15-2

15.4 MULTI-POINT COMMUNICATION SYSTEM 15-3
15.4.1 Command/Response Frame Format (Multi-point) 15-3
15.4.2 Calculation of FCS 15-4
15.4.3 Communication Procedure 15-4
15.4.4 Framing Errors 15-5
15.4.5 Command Errors 15-5
15.4.6 SHOULD YOU USE POINT-TO-POINT OR MULTI-POINT PROTOCOL? 15-5

15.5 RS485 Primer 15-6
15.5.1 RS485 Network Interface Hardware 15-6
15.5.2 Protection of RS485 Interface 15-6
15.5.3 Single Master RS485 Networking Fundamentals 15-7
15.5.4 Multi-Master RS485 Networking Fundamentals 15-8
15.5.4.1 Multiple Access with Collision Detection 15-8
15.5.4.2 Token Awarding Scheme 15-9
15.5.4.3 Rotating Master Signal 15-9
15.5.5 TROUBLE-SHOOTING AN RS485 NETWORK 15-9

16 HOST LINK PROTOCOL FORMAT 16-1

16.1 Device ID Read 16-1

16.2 Device ID Write 16-1

16.3 Read Digital Input Channels 16-1
16.3.1 Definition of Input Channels 16-2

16.4 Read Digital Output Channels 16-2

16.5 Read Internal Relay Channels 16-3
16.5.1 Definition of Internal Relay Channel Numbers 16-3

16.6 Read Timer Contacts 16-3
16.6.1 Definition of Timer-Contact Channel Numbers 16-4

16.7 Read Counter Contacts 16-4
16.7.1 Definition of Counter-Contact Channel Numbers: 16-4

16.8 Read Timer Present Value (P.V.) 16-4

16.9 Read Timer Set Value (S.V.) 16-5

16.10 Read Counter Present Value (P.V.) 16-5

16.11 Read Counter Set Value (S.V.) 16-5

16.12 Read Variable - Integers (A to Z) 16-6

16.13 Read Variable - Strings (A$ to Z$) 16-6

16.14 Read Variable – Floating Point (A# to Z#) 16-6
16.15 Read Variable – Single Data Memory (DM[1] to DM[4000]) 16-7
16.16 Read Variable – Multiple Data Memory (DM[1] to DM[4000]) 16-7
16.17 Read Variable – Single Floating Point Array FP[1] to FP[1000] 16-8
16.18 Read Variable – Multiple Floating Point Data FP[1] to FP[1000] 16-8
16.19 Read Variable - System Variables 16-8
16.20 Read Variable - High Speed Counter HSCPV[ ] 16-9
16.21 Write Inputs 16-10
16.22 Write Outputs 16-10
16.23 Write Relays 16-10
16.24 Write Timer-contacts 16-10
16.25 Write Counter-contacts 16-11
16.26 Write Timer Present Value (P.V.) 16-11
16.27 Write Timer Set Value (S.V.) 16-11
16.28 Write Counter Present Value (P.V.) 16-12
16.29 Write Counter Set Value (S.V.) 16-12
16.30 Write Variable - Integers (A to Z) 16-12
16.31 Write Variable - Strings (A$ to Z$) 16-13
16.32 Write Variable – Floating-point (A# to Z#) 16-13
16.33 Write Variable – Singe Data Memory (DM[1] to DM[4000]) 16-13
16.34 Write Variable – Multiple Data Memory (DM[1] to DM[4000]) 16-14
16.35 Write Variable – Singe Float Array (FP[1] to FP[1000]) 16-14
16.36 Write Variable – Multiple Float Array (FP[1] to FP[1000]) 16-14
16.37 Write Variable - System Variables 16-15
16.38 Write Variable - High Speed Counter HSCPV[ ] 16-15
16.39 Halting the PLC 16-16
16.40 Resume PLC Operation 16-16
16.41 Read Analog Input 16-16
16.42 Read EEPROM Integer Data 16-17
16.43 Read EEPROM String Data 16-17
16.44 Write Analog Output 16-18
16.45 Write EEPROM Integer Data 16-18
16.46 WRITE EEPROM String Data 16-18
16.47 Force Set/Clear Single I/O Bit 16-19
16.48 Using OMRON Host Link Commands 16-19
16.48.1 Read IR Registers 16-20
16.48.2 WRITE IR Registers 16-20

16.48.3 Read Data Memory DM[1] to DM[4000] 16-21
16.48.4 WRITE Data Memory DM[1] to DM[4000] 16-21

16.49 Testing of Host Link Commands 16-22

16.50 Visual Basic Sample Program 16-23

16.51 Inter-PLC Networking Using NETCMD$ Command 16-23

16.52 Inter PLC Networking Using MODBUS Protocols 16-24

17 IR REMOTE CONTROL 17-1

17.1 Introduction 17-1

17.2 Applications of IR Remote Control 17-1

17.3 Decoding Sony IR Remote Control Signal 17-2
17.3.1 Relocating The IR Remote Sensor 17-3

17.4 Learning and Playing back IR Remote Control Signals 17-3
17.4.1 Recording IR Remote Control Data Set 17-3
17.4.2 Loading IR Remote Control Data Set 17-4
17.4.3 Playback of IR Remote Control Signal 17-4

17.5 SETSYSTEM 17, xxx Command Summary 17-5

18 LIGHT DIMMER CONTROL 18-1

18.1 Introduction 18-1

18.2 Control of 120V/240V AC TRIAC 18-2

18.3 Zero-crossing Detection 18-3

18.4 Programming Light Dimmer Control 18-3
18.4.1 Enable Zero-Crossing Interrupt 18-3
18.4.2 Programming Individual Dimmer Channel 18-3
18.4.3 Programming Example 18-4

19 ZIGBEE NETWORKING 19-2

19.1 Introduction 19-2

19.2 Installation of the XBEE Module 19-2

19.3 Using The XBEE's Transparent Mode 19-3
19.3.1 Connecting to Fx2424 Using XBEE 64-bit Addressing 19-3
19.3.2 Defining Node Identifier for XBEE Module 19-5
19.3.3 Wring TBASIC Program To Configure XBEE 19-7
19.3.4 Practical Applications of XBEE Modules In Transparent Mode 19-7

19.4 Using The XBEE's API Mode 19-8

20 EXTENDED FILE SYSTEM 20-2

20.1 Introduction 20-2
20.1.1 Advantage of Using Data Uploading 20-3

20.2 File Structure and File Naming of The Extended File System 20-3

20.3 Transferring Files To The PLC's Web Server 20-4

20.4 Accessing The Extended Data Files Using TBASIC 20-4
20.4.1 Open A File For Writing New Data 20-4
20.4.2 Open A File For Appending Data To The End Of The File 20-5
20.4.3 Delete A File 20-5
20.4.4 Open A File For Reading 20-5

20.5 Setting Up The FileZilla FTP Server 20-7
20.5.1 Download and Setup FTP Server 20-7
20.5.2 Testing Connection To The FTP Server Using Telnet 20-9

20.6 Uploading File From PLC to FileZilla FTP Server Directory 20-12
20.6.1 Overview of The FTP Protocol 20-12
20.6.2 PLC FTP Upload Procedure 20-12
20.6.3 Monitoring The FTP Upload Progress 20-13

20.7 Setting Up A FTP Server Behind a Windows Firewall. 20-13

20.8 Uploading File From PLC to FileZilla FTP Server Directory 20-15
20.8.1 Overview of The FTP Protocol 20-15
20.8.2 PLC FTP Upload Procedure 20-16
20.8.3 Monitoring The FTP Upload Progress 20-16

20.9 Setting Up A FTP Server Behind a Windows Firewall. 20-18

21 USING PLC AS A MODBUS/TCP GATEWAY 21-1

21.1 Introduction 21-1

21.2 Application Ideas for Modbus/TCP Gateway 21-1

21.3 Configuring The PLC As Modbus/TCP Gateway 21-1

21.4 Fine-Tuning The Modbus/TCP Gateway Function 21-2

21.5 Modbus/TCP Gateway Sample Program 21-3

Chapter 1 Installation Guide For Fx2424 PLC

1 INSTALLATION GUIDE FOR FX2424 PLC

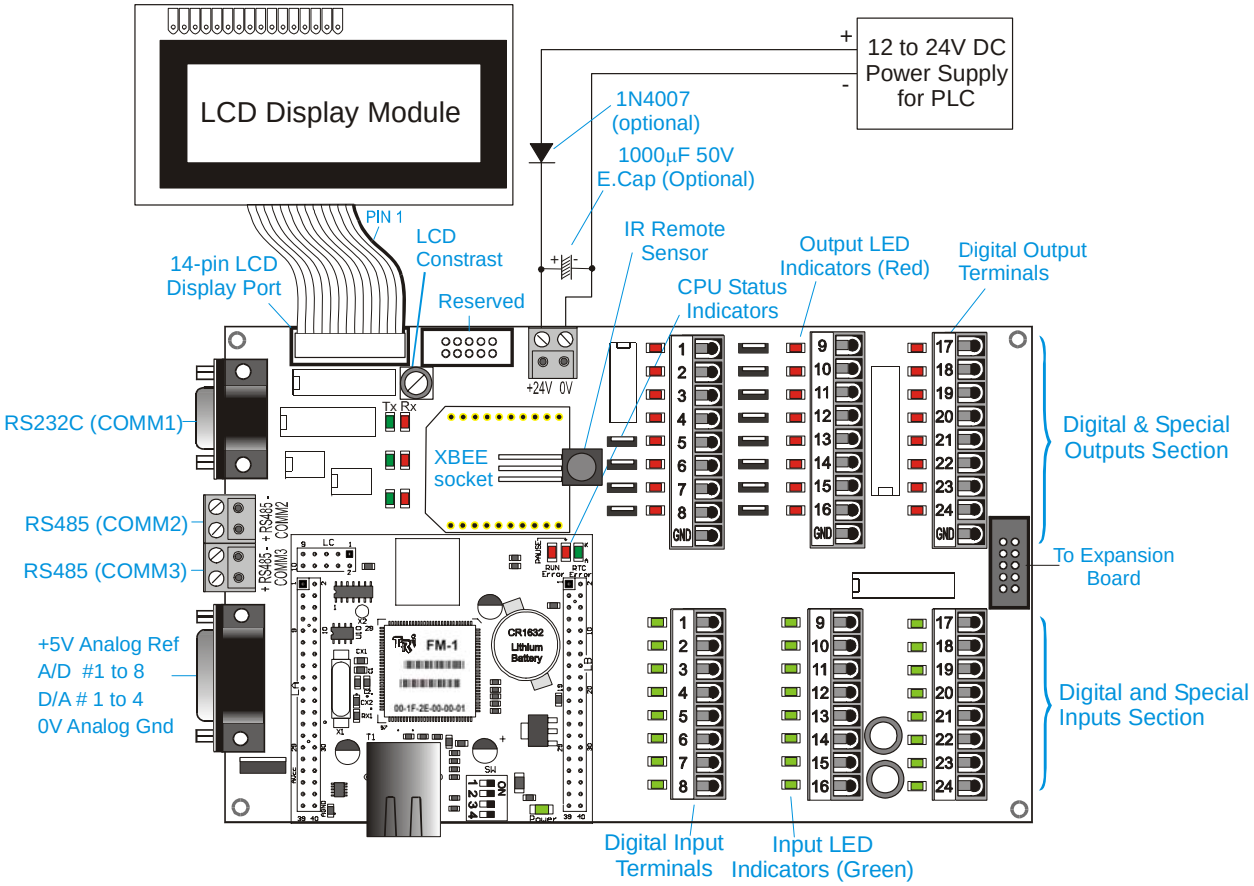


Figure 1.1

1.1 Overview

The Fx2424 PLC is a new generation of F-series PLC and designed to replace the F2424 model. Fx2424 is electrically identical to the F2424. The main difference is that the Fx2424 supports all the floating-point TBASIC commands available in I-TRiLOGI version 7 (TL7), whereas the original F2424 only support the integer math available in I-TRiLOGI version 6 (TL6).

Fx2424 is designed to be 100% backward compatible to F2424, hence it can also be programmed by TL6 as well as TL7. This means in most cases the F2424 users can simply transfer their ".PC6" file to the Fx2424 unmodified.

For existing users of F2424 upgrading to Fx2424, the following summarizes the main difference between the two PLC models:

	Fx2424	F2424
Program memory	23.5K words	16K words
Programming software	TL7 and TL6	TL6 Only
Floating point math	32-bit Single precision IEEE754	-
Floating point variables	A# to Z#, FP[1] to FP[1000]	-
Modbus support for floating-point vars	Mobdus memory mapping for FP[1] to FP[1000]	-
Hostlink commands for floating-point vars	New commands added "RVF" and "RV#A" to "RV#Z"	-
DIP Switch #3 ON	• Default IP = 192.168.1.15:9080 • Bypass username/password and trusted IP	• No change in IP address. • Bypass username/password and trusted IP
Support multiple DM Read command	New modified RVD command.	Only via OMRON command.
DM volatility	DM[1] to DM[4000] can be non-volatile	Only DM[300] to DM[4000] can be non-volatile
READMODBUS READMB2 WRITEMODBUS WRITEMB2	SETSYSTEM 6, n (n = function #) Support Function # 01, 02, 03, 04 Support Function # 01, 02, 03, 04 Support Function # 05, 06, 16 Support Function # 15, 16	SETSYSTEM 6, n Support Function # 03, 04 Support Function # 03, 04 Support Function # 06, 16 Support Function # 16
DHCP IP address.	Will request for IP address from DHCP server if IP = 0.0.0.0	Not supported.

The Fx2424 PLC features an Ethernet port that can be connected directly to a network router, switch, or hub for access to the LAN or to the Internet. The Ethernet port supports the FServer (for remote programming or monitoring) and a Modbus/TCP server (for access by third party devices) with up to 10 simultaneous connections. The user's program can also easily connect to another PLC or Modbus/TCP slave devices via the Internet, email real time data to any email address(es), or connect to the Internet Time Server to get the most accurate real time information! The Ethernet port also supports "Web Services", allowing enterprise software, such as a database program or MS Excel, to query for information from multiple PLCs instantaneously.

The basic Fx2424 unit comprises 8 analog inputs (12-bit, 0-5VDC), 4 analog outputs (12-bit, 0-5V DC), 24 digital Inputs, and 24 digital outputs. 12 of the digital outputs (5 to 16) can each sink up to 8A peak and 4A continuous, 24VDC current from the load and can be used to drive up to 3 small stepper motors directly – saving you the expense of buying separate stepper motor drivers! In addition, 6 of the digital inputs can be used for decoding and measuring pulses received from up to 3 digital encoders, allowing you to measure position and velocity of a moving object in real time. 10 of the digital inputs can also be defined as interrupt inputs, allowing fast events to be handled in the shortest possible time and to not be constrained by the program scan time.

The Fx2424 is expandable up to a total of 128 digital inputs and 128 digital outputs using optional expansion modules EXP1616R and EXP4040. It has one RS232 port and two RS485 ports; all of them are conversant in MODBUS protocol. The built-in LCD port allows for a simple interface to industry-standard LCD modules from 8 characters to 80 characters.

1.2 Physical Mounting & Wiring

The Fx2424 PLC comprises two PCBs that are stacked together in a very compact form factor. There is a Fx-CPU board that carries the RJ45 Ethernet port socket, the DIP switch, and the status LEDs. There is also a battery compartment that takes one CR1632 primary lithium cell, which powers the battery-backed real time clock to the CPU.

The second PCB is the "F-2424 carrier board" that carries the power elements and all the I/O interfaces and connectors including the power connector, the RS232 and RS485 terminals, the digital input and output drivers, and the expansion connector. There is even an Infra Red (IR) remote control sensor that lets you use a Sony TV or VCR remote control to control your PLC!

The Fx-CPU board is plugged into the F-2424 carrier board via two 40-pin IDC headers and a one 10-pin IDC header. **IMPORTANT**: At no time must the CPU board be unplugged from the carrier board when the PLC is powered ON. Doing so can cause serious damage to either or both PCBs and is not covered by warranty. We discourage unplugging the CPU board from the carrier board except for repair or upgrade purpose as doing so may expose the CPU header pins to high voltages or electrostatic discharge that can damage sensitive components on the CPU board.

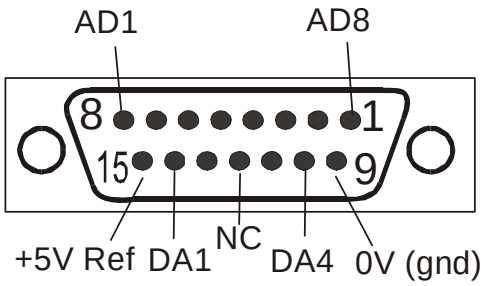
The assembled Fx2424 PLC can be easily installed in many kinds of plastic or metal enclosures. You need to use 5 PCB standoffs (or screws and nuts) to support the controller and fasten it to a console box. Alternatively, you can mount it on the optional "DIN-KIT 2" and clip it onto the standard DIN rail.

The following subsections show some hardware details of the I/Os that are available on the Fx2424 PLC model. Separate chapters in this manual will be devoted to discussing the programming methods for this hardware.

1.2.1 Analog I/O Ports:

The 8 channels of analog inputs and 4 separate channel of analog outputs are available via a DB15 connector along the left edge of the Fx2424 PLC. The Fx2424 PLC also supplies a +5V analog reference-voltage output and the analog ground on the female DB15 connector, as shown below:

Signal	Pin #
A/D #1	8
A/D #2	7
A/D #3	6
A/D #4	5
A/D #5	4
A/D #6	3
A/D #7	2
A/D #8	1
D/A #1	14
D/A #2	13
D/A #3	11
D/A #4	10
Analog Ref. AV_{CC}	15
Analog ground AV_{SS}	9



*NC - No Connection for pin 12

Figure 1.2

The specifications and programming methods for the analog I/Os are detailed in Chapter 5 of this manual.

1.2.2 Digital I/O Ports:

Detachable screw terminals are provided for quick connection to all digital inputs, outputs and power supply wires. Each block of screw terminals can easily be detached from the controller body, enabling easy replacement of the controller board when necessary. Since the terminal block for the digital I/Os is inserted vertically to the board surface, you would need to remove the terminal block before you can start wiring. Insert a small flat-head screwdriver under the terminal block and apply even pressure to raise the terminal block until it becomes loosened from the connecting-pin strip, as shown below:

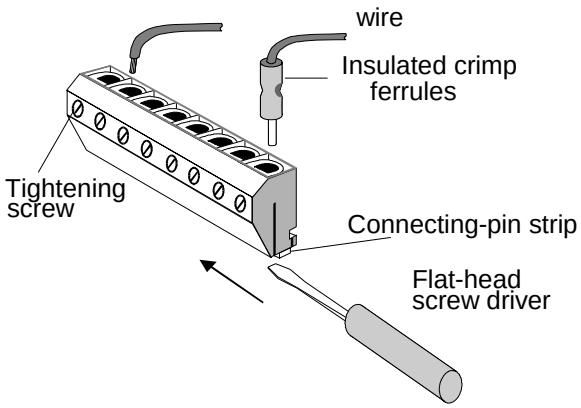


Figure 1.3 - Removing Screw Terminal block

Although wires may be connected directly to the screw terminal, insulated crimp ferrules should be used to provide a good end termination to multi-stranded wires. Use of ferrules reduces the possibility of stray wire strands short-circuiting adjacent terminals and their use is, therefore, highly recommended.

1.2.3 Output Ground Terminals:

Notice that the output screw terminal block has an additional GND (or COM) terminal that is not found on the Input terminal block. This is a common ground terminal for all the 8 outputs on the same terminal block. This terminal should be connected to the 0V of the PLC's power supply to provide a ground return path for the 8 outputs on the same terminal block.

1.3 Power Supply

The Fx2424 PLC requires a single regulated, 12 to 24V (+/- 5% ripple) DC power supply for the CPU. The PLC typically consumes less than 100mA and thus you may also use the same power supply to power the CPU and a small output load, as long as the total load current is within the power supply maximum limit. It is recommended that whenever possible, use a higher voltage power supply since the voltage difference between ON and OFF state is wider for operation at higher voltage.

If the loads require higher current (such as driving stepper motors) then it is recommended that a separate power supply be used for the output load. Note that if you choose to use two separate power supplies, their 0V terminals should be tied together to provide a common ground reference for the two power supplies.

The two power supplies are also recommended to be of the same output voltage. Otherwise, if the voltage of the load power is lower than the voltage of the CPU power, the output LED will light up even when the output driver is not energized. This is because the current could flow out of the PLC's output into the (lower voltage) load power supply and therefore light up the LED. This can be very confusing to users even though it should not damage the LED. Should this happen, you will need to add a series diode to the output terminal to block the output LED currents from flowing through the load and back to the load power.

Please use only an industrial grade linear or switching regulated power supply from established manufacturers. Using a poorly made switching power supply can give rise to a lot of problems if the noisy high frequency switching signals are not filtered properly.

Note: If your application demands very stable analog I/Os you should choose a linear power supply instead of a switching power source for the CPU.

Always place the power supply as close to the PLC as possible and use a separate pair of wires to connect the power to the PLC. Keep the power supply wires as short as possible and avoid running them along side high current cables in the same cable conduit. The Fx2424 PLC will be reset when the power supply voltage dips below 9V. It is a good idea to connect a 470μF to 1000μF, 50V electrolytic capacitor near the power supply connector to suppress any undesirable voltage glitches from conducting into the PLC. If other high current devices, such as a frequency inverter, were to affect the operation of the PLC, you should then also connect a diode before the capacitor to prevent reverse current which might flow back to the power supply, as shown in the following diagram:

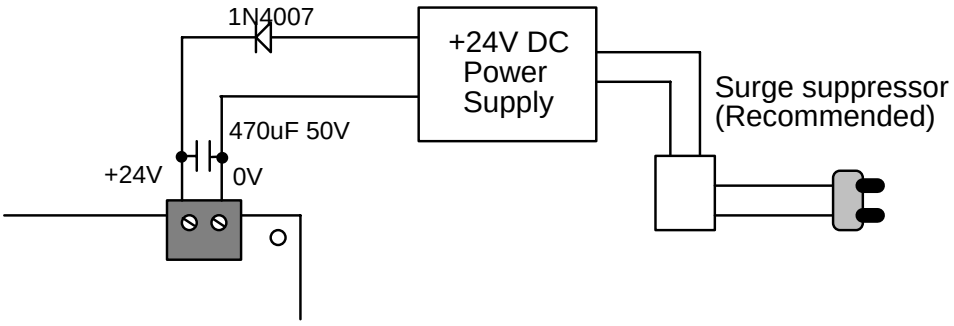


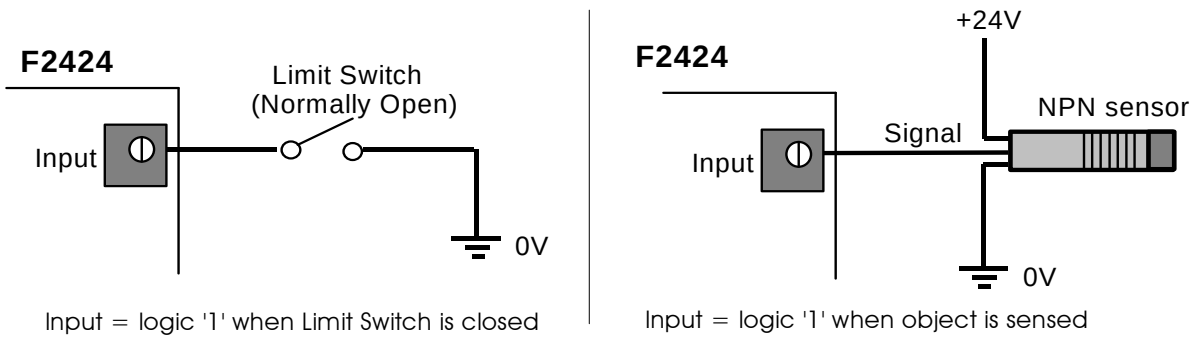
Figure 1.4

If the AC main is affected by nearby machines drawing large amounts of current (such as large three-phase motors), you should use a surge-suppressor to prevent any unwanted noise voltage from being coupled into the Fx2424 power supply. The required current rating for the power supply depends mainly on the total output current, taking into consideration the peak current demand and the duty cycle of the operation.

1.4 Digital Input Circuits

All inputs have green colored LED indicators. Every 8 inputs are grouped together into a single strip of detachable screw terminal. All inputs are NPN type, meaning that to turn ON an input, you should connect it to the low-voltage rail (0V terminal) of the power supply as shown in the following diagram. The input numbers are marked on their screw terminals as well as on the PCB alongside the strip pin.

All digital inputs are directly programmable in Ladder Logic, as well as in TBASIC custom functions. Some **programming examples** are detailed in “Chapter 3 –Digital I/Os and Internal Relays”



Input Voltage for **Logic 0:**	Open Circuit or +10V to +24VDC
Input Voltage for **Logic 1:**	0V to +2.5V DC

Figure 1.5 –Interfacing to Limit Switch and NPN Sensor

1.5 Digital Output Circuits

1.5.1 Electrical Specifications:

	Output #1 to 4	**Output #5 to #16**	**Output #17 to #24**
Output Driver type	NPN Darlington Transistors	N-Channel power MOSFET with low r_{DS} = 0.05 Ω	N-Channel power MOSFET array
Maximum Breakdown Voltage	50V	55V	50V
Maximum Output Current:	1A	8A	1A
Continuous Output Current	350mA	4A	250mA
Output Voltage when OFF	Resistor pulled up to 24V power rail		
Output Voltage when ON:	1.2V @1A	0.2V @4A	0.2V @1A
Inductive Back EMF Bypass	Yes (Built-in diode)	Yes (Intrinsic Zener)	Yes (Intrinsic Zener)

All outputs have red colored LED indicators. The Fx2424 PLC employs “sinking” (NPN) type power transistors or MOSFET outputs that turn ON by sinking current from the load to the 0V terminal. Every 8 outputs are grouped together into a single strip of detachable screw terminal. Figure 1.6 shows the wiring diagram of the digital outputs.

Note:

Output #5 to #16 can each sink up to **8A peak or 4A continuous current** @24VDC**.**

Output #5 to #8 can also be configured as **PWM outputs** to drive heating elements or proportional valves using the SETPWM command – please see Chapter 6 for more information.

1.5.2 Digital Output Wiring Diagram

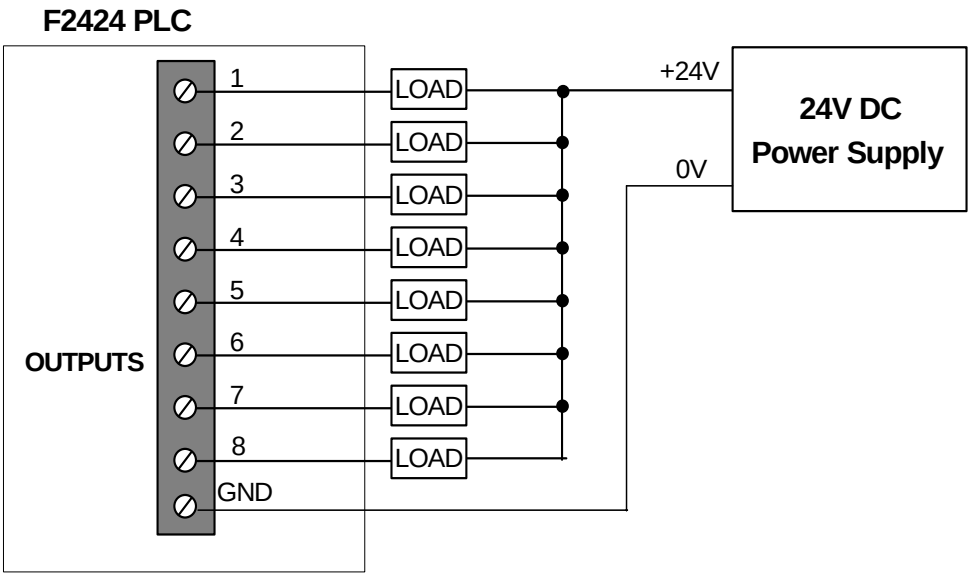


Figure 1.6 - Transistor Output Interfacing to Load

All digital outputs are directly programmable in Ladder Logic as well as in TBASIC custom functions. Some **programming examples** are detailed in "Chapter 3 –Digital I/Os and Internal Relays"

1.5.3 Inductive Load

When switching inductive loads such as a solenoid or a motor, always ensure that a bypass diode is connected to absorb inductive kicks, which will occur whenever the output driver is turned OFF. Although all the PLC digital outputs already incorporate either internal diodes or intrinsic Zener bypass diodes to protect the driver, some may only activate when the inductive kick voltage rises above 100V DC. This can result in a large dose of noise being introduced into the system and may have undesirable effects. We recommend using a fast recovery diode such as UF4001 to UF4007 connected as shown in the following diagram to absorb the inductive noise:

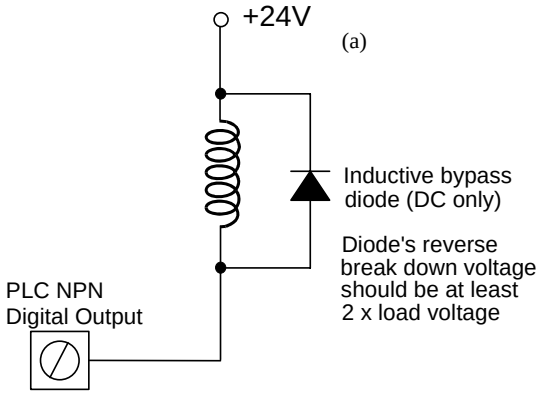


Figure 1.7 – Protective Solid State Output Against Inductive Kick.

1.6 LCD Display Port

The built-in 14-pin **LCD DISPLAY PORT** on the Fx2424 is compatible with the Hitachi HD44780 display controller, which is a *de facto* industry standard. This allows low cost, easily available LCD modules

from third-party manufacturers to be connected directly to the PLC to implement a very economical man-machine interface. You simply connect the LCD module to the Fx2424 LCD port using a 14-way IDC ribbon cable and header. Make sure that the pin number on the PLC matches that of the LCD module. Wrong cable connections can destroy the LCD port or LCD module.

There are quite a number of 14-pin LCD modules available on the market with many different display sizes or character formats to choose from. e.g., 1x8, 1x16, 1x20, 2x16, 2x20, 1x40, 2x40 and 4x20. Triangle Research International (**TRi**) currently supplies two models of backlit LCD display modules for use with the Fx2424 PLC. Each model is supplied with a 0.5m long, 14-pin ribbon cable for connection to the PLC's LCD port. The LCD216 is a 2 line x 16 characters per line module and the LCD420 is a 4 line x 20 characters per line module. For more electrical and mechanical information on these two LCD models, please visit our website at:

http://www.tri-plc.com/lcd.htm

The contrast of the LCD can be adjusted using the preset potentiometer VR1, marked with the word "contrast", just below the LCD connector. If you find that the LCD display is too dark or too dim, please adjust VR1 with a small screwdriver to obtain the preferred contrast from the regular viewing angle.

1.6.1 Wiring Instruction of LCD216 and LCD420 Backlight.

To use the LCD, simply plug the 14-pin connector into the LCD port header; making sure that pin #1 (the colored wire) of the ribbon cable aligns with pin #1 of the port header. (Pin 1 is the pin with a square solder pad and the pin number is also printed on the PCB.)

Both LCD models provide yellow-green LED backlight, which can be powered by the same DC power supply as the PLC with the simple addition of a current-limiting resistor. Wiring of the backlight differs slightly for the two models, as shown below. It is assumed that the unit will be connected to the 24V power supply of the Fx2424 PLC. If you are using other voltages, then compute the values of the current limiting resistors to obtain the same backlight current, taking note that the LCD backlight LED has a forward voltage drop of about 4V.

IMPORTANT!

Please ensure that the current limiting resistor is in place and wired properly before turning on the power supply. Connecting the LED backlight to the 24V power supply without the current limiting resistor will definitely destroy the backlight unit. Also, please ensure that the polarity is connected correctly, because reverse connection is likely to damage the backlight unit as well.

For the LCD420, the backlight is brought to a screw terminal via a small PCB. Use a multi-meter to check which terminal is connected to pin 15 (Anode) and 16 (Cathode-) if you are unsure of the polarity.

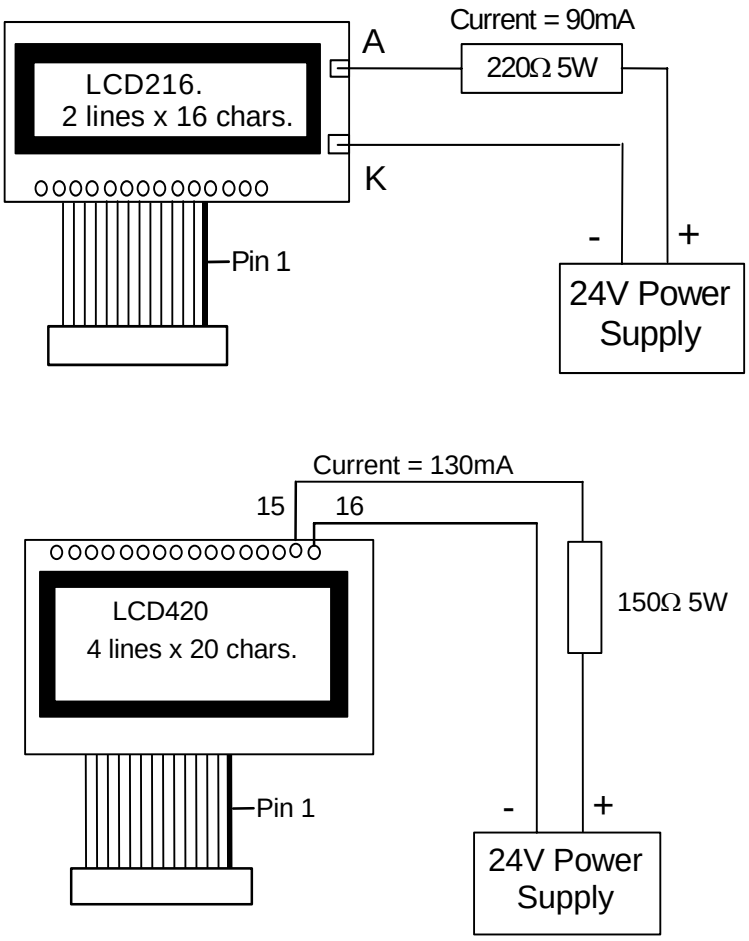


Figure 1.8 – Wiring of LCD216 and LCD420 backlight.

If you are using voltages other than 24V DC to power the Fx2424, then compute the resistor value using the following formula: R = (V – 4)/I

E.g. For LCD420 at 12V DC,
R = (12 - 4)/0.13 = 62 ohms. Power dissipation = I^2 x R = 0.13^2 x 62 > 1W.

The resistor value is only approximate. You may increase or decrease the resistance value to alter the brightness of the backlight.

If you wish to adjust the brightness of the LCD backlight, you could wire the backlight to one of the PLC's PWM output. The resistor value selected should provide the current required for full brightness when the PWM duty cycle = 100%. By varying the duty cycle of the PWM output you can adjust the brightness of the LCD backlight. E.g. you can create the effect of fade-in/fade-out or flashing LCD backlight to get the operator's attention.

1.6.2 Programming The LCD Display

Some programming examples for the LCD display are presented in Chapter 13.

1.7 Program and Data Memory

1.7.1 Program Memory

The Fx2424 has by default 23.5K words (16-bit) of program memory stored in the CPU Flash memory area vs a default of only 16K on F2424 (some later version of F2424 do support up to 23.5K but requires setting DIP switch #2)

Each ladder logic element (contacts or coils) takes up 1 word of memory. A TBASIC statement or function takes up half a word to four or five words, depending on the number of parameters the statement or function has.

The program memory can be erased and reprogrammed more than one hundred thousand times, which is a limit that you are unlikely to ever reach.

1.7.2 Non Volatile FRAM Memory

Users of the M-Series PLCs (T100MD+ and T100MX+ PLCs) may be familiar with the PLCs' EEPROM memory as well as some of its limitations.

The Fx2424 PLC does not use the EEPROM memory anymore. Instead, the Fx2424 employs 6K words of a new, high-tech memory product called "Ferromagnetic RAM" (FRAM). FRAM is non-volatile memory just like the EEPROM that it replaced; yet unlike the EEPROM, which can only be erased and reprogrammed a limited number of times, FRAM has no practical limit on how often it can be overwritten. In fact, since they are really RAM memory in nature, there is no "erase" and "program" time penalty and you can read and write to them at full speed. Also, unlike the case of T100M+ EEPROM. accessing the FRAM frequently would not have any adverse effect on serial communications.

Since the FRAM are in fact a replacement of the EEPROM data memory, you can use the same TBASIC commands (LOAD_EEP( ), LOAD_EEP$, SAVE_EEP, and SAVE_EEP$) in TRiLOGI version 6 or 7 to read and write to the FRAM memory. Just take note that FRAM has all the advantages of EEPROM but none of its disadvantages.

1.7.3 Volatile Data Memory

All the TBASIC variables used in the Fx2424 PLC: A to Z, A# to Z#, FP[1] to FP[1000], DM[1] to DM[4000] and string A$ to Z$, EMINT[1] to EMINT[16] and EMLINT[1] to EMLINT[16] are in the category of "volatile data memory" - meaning when you turn off power to the PLC the memory content will be lost and they will be reset to zero when the PLC is powered up again.

However, the memory array DM[1] to DM[4000]* as well as FP[1] to FP[1000] can be made non-volatile by turning on DIP switch #1. This is because internally, DM[1] to DM[4000] as well as FP[1] to FP[1000] are actually implemented in FRAM rather than regular RAM (however, they are not part of the 6K words available for use mentioned in Section 1.7.2). You can therefore use these DMs as "safe" memory locations whose content will be preserved in the case of a power failure.

Of course you can also use the SAVE_EEP and LOAD_EEP to achieve the same result as using DM[1] to DM[4000]. It may be just a little more "natural" to use DM in your program instead of sprinkling LOAD_EEP and SAVE_EEP commands throughout your program code.

* In the original F2424 only memory array DM[301] to DM[4000] can be made non-volatile.

1.7.4 Extended Data File Storage

Fx2424 supports extended data file storage for storing web pages and some are directly accessible from within the PLC program. For more information please refer to Chapter 20 of this manual.

1.8 DIP SWITCHES

DIP Switch	**OFF**	**ON**
SW1-1	All outputs, relays, timers and counter values are non-retentive.	DM[1] to DM[4000] data are non volatile and will not be reset.
SW1-2	-	-
SW1-3	-	• Disable the use of username/password and Trusted IP for FServer and Modbus/TCP Server. • PLC IP address defaults to 192.168.1.15:9080
SW1-4	Normal Run mode	Suspends execution of the ladder logic program. But host communication remains active.

1.8.1 Usefulness of SW1-4

We have taken every effort to ensure that the host communication is always available even when the user-program ends up in a dead-loop. This allows the user to re-transfer a new program to the PLC and overwrite the bad program. However, you may still encounter a situation whereby after transferring a new program to the PLC, you keep encountering communication errors and you are unable to erase the bad program. This is especially common if you have been experimenting with the communication commands such as `SETBAUD`, `SETPROTOCOL`, `PRINT` or `OUTCOMM`. These commands may modify the communication baud rate, format, or protocol or set the PLC to send data out of a COMM port that conflicts with i-TRiLOGI. In such cases, you can turn ON DIP Switch SW1-4 and perform a power-on reset for the PLC. The PLC will not execute the bad program that causes communication problems and you can then transfer a new program into the PLC to clear up the problem.

Note that when the PLC has been power-reset with DIP Switch #4 set to ON, all 3 serial port will boot up with default baud rate and communication format of 38,400, 8,n,1. (This differs from the T100M+ PLC.)

1.9 Real Time Clock Battery

All Fx2424 PLCs have a built-in, battery-backed Real-Time clock (RTC). On the Fx2424 CPU board there is a battery holder that takes a single CR1632 lithium primary battery. Due to certain international aviation safety restrictions on shipping of products that contain lithium cells by airfreight or air courier, your PLC may or may not be supplied with the battery. If the battery is not included, you can purchase the battery from a local electronics component supplier, such as www.futureelectronics.com (for US or Canadian users), and simply press it into the battery holder.

The PLC's RTC consumes about 1μA during power down. A Ranata CR1632 lithium battery has a charge capacity of 125 mAh. This means that theoretically the battery should be able to supply the RTC for 125*1000 hours, or about 5200 days (> 10 years). Since this Lithium battery has a shelf life of 10 years, providing for a safety factor of 2 would mean that the single Ranata CR1632 should be able to power the PLC's RTC for at least 5 years, and could be for as long as 10 years if the PLC is never powered down.

1.10 CPU Status Indicators

There are three LED indicators on Fx-CPU board with the markings shown on the right

All these indicators will light up for about 0.5 seconds during power-on. Thereafter they should go off and if any one of them remains lit, it represents the various operating status of the PLC as follows:

1.10.1 RTC Error (Green LED)

If this indicator is turned ON, it indicates the PLC's real-time clock (RTC) has lost track of date and time. The RTC.Err flag in the "Special Bit" menu will also be turned ON. This indicator will be turned OFF automatically after you have set the PLC's date and time using the "Set PLC's Real Time Clock" command in the "Controller" pull-down menu. For more information about using the RTC, please refer to Chapter 12.

1.10.2 Pause (Red LED)

This indicator will be turned ON if one of the following has occurred:

1. The PLC's program is corrupted.
2. A PAUSE statement has been executed
3. The user halts the PLC by pressing the <P> key during On-Line Monitoring.
4. DIP-Switch SW1-4 is turned ON, which halts the program.

If this light is ON, please connect the host computer running i-TRiLOGI to the PLC and run the "On-Line Monitoring" program. You will be informed of the reason that caused the PAUSE condition. Except for condition i) and iv), you can release the PLC from the PAUSE state by clicking on the "Pause" button or by pressing the <P> key during "On-Line Monitoring". If the PLC's program is corrupted, then you must re-transfer your program to the PLC.

1.10.3 Run Error (Red LED)

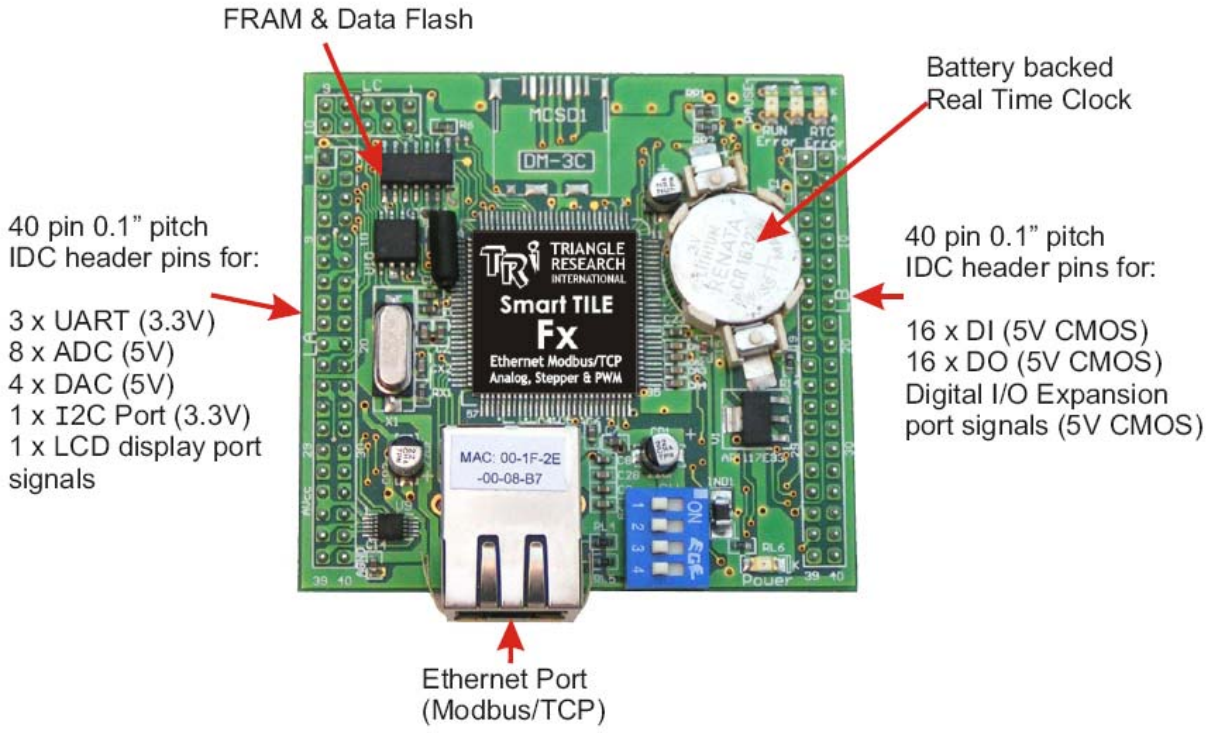
When this indicator turns ON, it shows that a run-time error has occurred during execution of a TBASIC command. The system will halt at the CusFn where the error took place. If the programmer now executes the "On-Line Monitoring" command in TRiLOGI, the cause of the run-time error and the CusFn where the error occurred will be reported on TRiLOGI screen. If you have installed an LCD display, the PLC will take over the LCD screen and display the cause of the runtime error and the function where it occurred.

The TBASIC simulator captures many possible run-time errors including out-of-range values, but in the Fx2424 PLC, only a few most important run-time errors are reported. The remaining errors are ignored. The following are the few run-time errors that will be reported in Fx2424 PLC:

1. Divide By Zero
2. FOR-NEXT loop with STEP = 0!
3. Call Stack Overflow! Circular CALL suspected!
4. Illegal Opcode - Please inform manufacturer!
5. System Variable Index out-of-range: This is normally caused by using an unavailable subscript. E.g. DM[0], INPUT[-1], DM[5000], etc. Check the subscript value, especially if it contains a variable (e.g. DM[X], if X=0 this will lead to a runtime error).

All run-time errors should be identified and corrected before you proceed any further.

1.11 Create a Custom PLC with SmartTILE



A **SmartTILE** (Smart TRi Integrated Logic Engine) is the main CPU control board that can be used by an OEM equipment maker to build its own custom-made, low cost and very powerful Ladder Logic + BASIC programmable logic controllers.

In fact the **SmartTILE-Fx** model is the CPU board used on the powerful F-series Super PLC families that have been incorporated in a variety of industrial OEM applications since 2009. Each F-series PLC is made up two components:

1) The **SmartTILE-Fx** CPU board, which is the brains of each F-series PLC. It is identical throughout the PLC family. The SmartTILE board is programmable by the Floating-point version of I-TRiLOGI Version 7.xx (TL7).

2) The carrier board, which is the electrical interface between the SmartTILE controller and more common industrial voltage level I/Os. It is unique to the PLC model. The SmartTILE plugs into the carrier board via IDC header pins and sockets. This manual covers the **Fx2424** PLC, which is the combination of a SmartTILE-Fx CPU board and carrier board.

The SmartTILE is now being supplied as a separate part so that any form of industrial automation user/developer/supplier (OEM, product/system designers, machine builders, integrators, etc.) can utilize the time-proven, powerful and flexible platform already realized in Triangle Research PLC products, such as the F-series, FMD series, and Nano-10 PLCs, and design their own carrier board to interface to the specific I/O and communication signals demanded by their application or industry. All the necessary signals with wiring examples are provided in this manual, so it is easy for any in-house or contracted design engineer to develop a carrier board that seamlessly integrates with the SmartTILE board and its inherent platform. For companies that would like to use the SmartTILE as a platform, but would prefer not to handle the carrier board design, Triangle Research is ready to support you in developing your carrier

board solution. Please contact info@triplc.com with your company info, application, time-to-market, design requirements, estimated usage, and any other relevant information.

The SmartTILE features an Ethernet port that can be connected directly to a network router, switch, or hub for access to the LAN or to the Internet. The Ethernet port supports the FServer (for remote programming or monitoring) and a Modbus/TCP server (for access by third party devices) with up to 6 simultaneous connections. The user's program can also easily connect to another PLC or Modbus/TCP slave devices via the Internet, email real time data to any email address(es), or connect to the Internet Time Server to get the most accurate real time information! The Ethernet port also supports "Web Services", allowing enterprise software, such as a database program or MS Excel, to query for information from multiple PLCs instantaneously.

Each unit comprises 8 analog inputs (12-bit, 0-5VDC), 4 analog outputs (12-bit, 0-5V DC), 16 digital Inputs (5V CMOS), and 16 digital outputs (5V CMOS) In addition, 6 of the digital inputs can be used for decoding and measuring pulses received from up to 3 digital encoders, allowing you to measure position and velocity of a moving object in real time. 10 of the digital inputs can also be defined as interrupt inputs, allowing fast events to be handled in the shortest possible time and to not be constrained by the program scan time.

A SmartTILE-based PLC can be designed with up to a total of 128 digital inputs and 128 digital outputs via the 5V CMOS expansion signals from the SmartTILE. It has 3 independent 3.3V UARTs and all of them are conversant in MODBUS protocol. The LCD port signal interface allows the carrier board designer to potentially interface to industry-standard LCD modules from 8 characters to 80 characters, as well as the standard LCD216 and LCD420 displays that Triangle Research carries.

More information can be obtained from the SmartTILE-Fx product page on the Triangle Research website:

http://triplc.com/smarttile.htm

Chapter 2 Ethernet Port

2 ETHERNET PORT

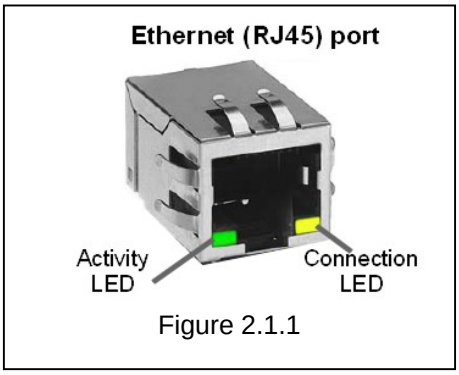


Figure 2.1.1

Every Fx2424 PLC has a single, built-in 10/100 Base-T Ethernet port that uses the standard RJ45 connector on its CPU board. You can easily connect the Fx2424 PLC to your network router, switch, or hub using the straight CAT-5 cable. When a connection is made, the yellow “Connection LED” on the RJ45 connector will light up, indicating that the PLC has been connected to the network router.

Once connected to the network, the TRiLOGI programming software and TRi-ExcelLink works instantaneously with the PLC, allowing remote programming and process monitoring over the LAN or the Internet. In addition, the Ethernet facility at this port can host web pages and Java applets so that users of the equipment can control/monitor their equipment using their web browser from anywhere.

Before connecting the PLC’s Ethernet port, you should configure it using the configuration tool that is now built into the 6.3x or higher version of i-TRiLOGI programming software. Previously, a standalone Ethernet Configuration program was required to setup the basic and advanced Ethernet, ADC, and RTC (Real Time Clock) options. This standalone version is still available and more information on this program, including where to download it, is available at 2.14 Standalone Version of Ethernet Configuration Software). Now these settings can be configured directly from the TRiLOGI programming software, as described next.

2.1 Configuring The Ethernet Port

This tool, which is located in the "Controller" Menu of TRiLOGI, allows you to configure the Ethernet Port and ADC/RTC calibration settings on a TRi's PLC with built-in Ethernet port, such as the Fx PLCs. When the “Ethernet & ADC Calibration” is selected from the “Controller” menu, you will see the following screen:

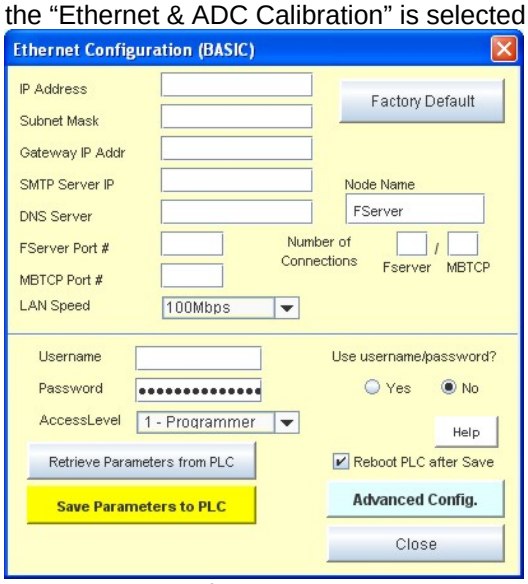


Figure 2.1.2

The configuration program communicates with the PLC the same way as i-TRiLOGI, which is through the serial TLServer or the Ethernet FServer. If your i-TRiLOGI is not yet connected to a TLServer software or directly to the PLC’s built-in hostlink command server (known as F-Server for all F-series PLCs), then you will need to login the standard way from the “Controller” menu.
(See the i-TRiLOGI Programmers Reference Manual for more information on this)

If you are communicating through TLServer, which is via the RS232 serial port, your PC should have an RS232 port. Otherwise, you should purchase a USB to RS232 converter (see http://www.tri-plc.com/USB-RS232.htm) in order to run this program.

Before you make any changes to the PLC's Ethernet configuration parameters, it is a good idea to retrieve the current settings. You can do this by clicking on the “Retrieve Parameters from PLC” button and answer "Yes" when prompted. If your i-TRiLOGI is not yet connected to a TLServer software or directly to the PLC’s F-Server, then you will see the standard login popup window.

After you have retrieved the existing parameters from the PLC, you will see that various fields in the configuration software screen are filled up. Note that the Fx PLC has two built-in "Server" programs that listen on a few different ports on the PLC for incoming TCP/IP request packets:

1) The FServer supports the TRi proprietary programs such as the i-TRiLOGI software and TRi-ExcelLink program. The FServer listens on the default port 9080, which is also the default port that i-TRiLOGI uses to connect to TLServer (a PC based server program that is needed for programming the PLCs via RS232 or RS485 serial ports).

2) A MODBUS/TCP server that listens on port #502 and supports the industry standard MODBUS/TCP protocols.

Note:

- These two servers share the same Ethernet port and therefore the same IP address and gateway addresses described in the following sections.

- Fx PLC host both the FServer and Modbus/TCP Server, each providing multiple simultaneous connections to external clients. This means that it is possible to connect multiple TRiLOGI, ExcelLink and Modbus/TCP clients to the PLC, all at the same time! Section 2.1.5 shows you how you can change the maximum number of connections for each server.

2.1.1 IP Address

One of the most important parameters that you must define here is the "IP Address" field. By default, every F-series PLC is shipped with the static IP address: "192.168.1.5". You will need to assign the PLC's with an IP address that is unique on your network and yet is accessible from your PC. If you are on a company network, then you must consult your company's system administrator to assign you a useable IP address.

However, if your PC is connected to a small local area network, then most likely you will have a LAN IP address of 192.168.XXX.YYY. For proper networking, you should set the PLC's IP address to "192.168.XXX.ZZZ". i.e. The first three numbers should match each other. The fourth number (ZZZ) should be an IP address that is not used by any other devices on your network.

Note that majority of small LANs are built using a network router that assigns dynamic IP addresses (called DHCP server) to the PC. You should enter the administrator page of the router and define the range of DHCP for use by the PCs and then you may assign the PLC with any IP address that is outside of the DHCP range. E.g. If you define the DHCP address range to be 192.168.1.100 to 192.168.1.150, then you may assign the PLC with any IP address between 192.168.1.2 to 192.168.99 (Usually the router itself would have the IP address 192.168.1.1 so that address is not available) and also between 192.168.151 to 192.168.1.254 (again making sure that 192.168.1.254 is not already used by your router).

DHCP IP Address *(* New on Fx PLC Only)*

For the Fx2424 PLC, if you set the IP address to 0.0.0.0 then the PLC will send DHCP request to the DHCP server to request for an IP address. The DHCP server is assumed to have the same IP address as the "Gateway" server.

You can find out the IP address assigned to the PLC by sending the command: PRINT #4 "<IP>" and then read the response using INPUT$(4)

(see example in "C:\TRiLOGI\TL7\usr\samples\Ethernet\TestEthernet.PC7" and the address can then be displayed on the PLC's LCD display using SETLCD command).

2.1.2 GateWay IP Addr

The Gateway IP address lets the Fx2424 communicate with other LAN segments or connect to the Internet. The gateway address is usually the local IP address of the router where the PLC is connected. For small local networks with no plan for connection to the Internet, the Gateway IP Address is not needed and can be set to 0.0.0.0. But if you plan to use the Fserver's email capability then you must fill in the correct Gateway IP Address. Ask your system administrator if you have any question about this.

2.1.3 SMTP Server IP Address

The SMTP (Simple Mail Transport Protocol) Server field lets you define the IP address of the email server that the PLC can use to send out emails from user's program (please see section 2.3.3 for more details on how to program the PLC to send emails). This is the same SMTP server that your normal email client software such as Thunderbird or MS Outlook uses to send out email. You can ask your Internet Service Provider (ISP) for the IP address of their SMTP server. The ISP usually provides the SMTP server in domain name form (such as "mail.sbcglobal.net"), but you should also be able to request the numerical IP address of the SMTP server from the ISP.

For Windows XP or Vista users, you can resolve the IP address as follows: First, launch the "Command Prompt" window. Then enter the command nslookup <smtpserver name>" to get the IP address. An example is shown below where the IP address of mail.sbcglobal.net is resolved to the IP address: "207.115.36.120":

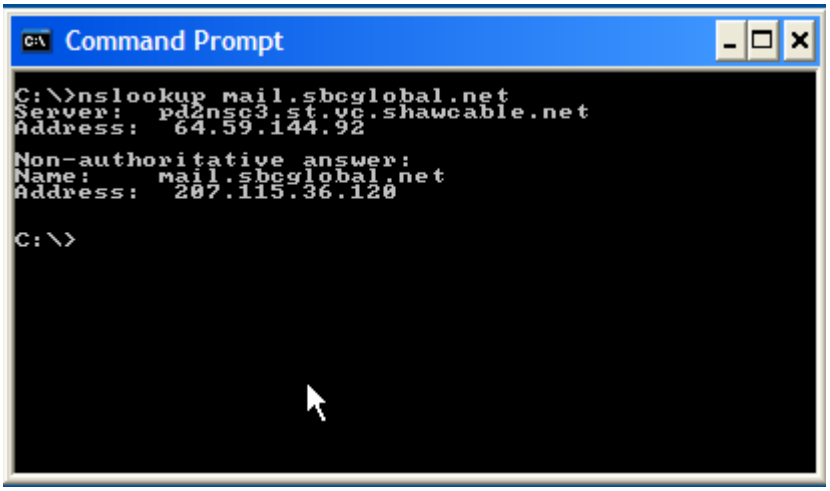


Figure 2.1.3

Windows users may also search the Internet for a free "host.exe" tool that lets you resolve the IP address from a given domain name (one host.exe tool that we found to work was downloaded from http://pigtail.net/LRP/dig/). For example, executing the command line: "host mail.sbcglobal.net" will resolve its IP address. (Of course you can only use this smtp server provided your ISP is SBC, almost no SMTP server will relay emails from a client that is not its one of its own subscribers).

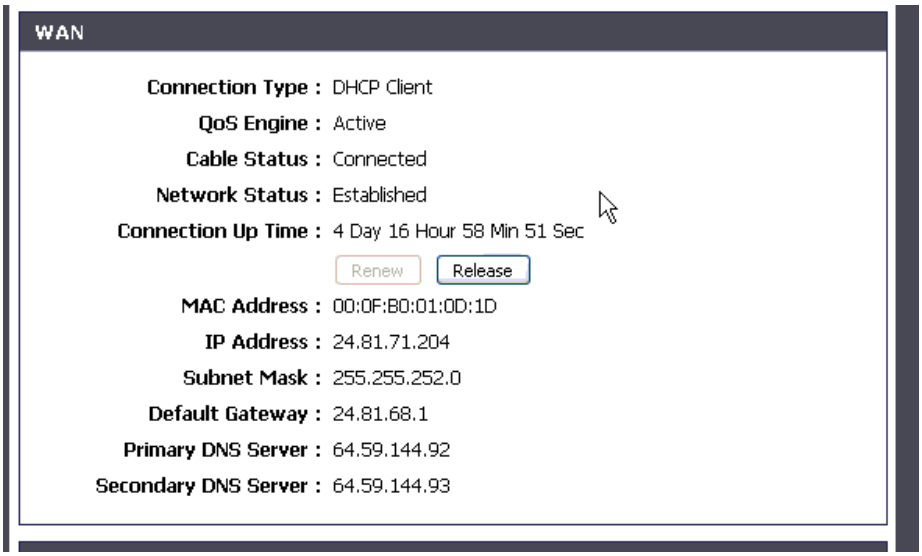
If you do not plan to use the FServer to send out emails yet, then you can leave the default SMTP Server IP Address = 0.0.0.0. You can change the settings anytime later when you need it.

2.1.4 DNS Server IP Address

DNS (Domain Name Server) allows the FServer to contact a remote server by means of domain name instead of IP Address. The DNS takes in the given domain name (such as yahoo.com) and returns the IP

address of the target server. You will need to fill in the DNS IP Address if you intend to ask the Fserver to resolve a domain name into an IP address, or you intend to contact a client directly by using domain name instead of using IP Address.

The DNS server IP address could be the same as the Gateway IP Addr described in section 2.1.3. But it will be more efficient to define the actual DNS server IP address that your ISP provides. You can usually obtain the DNS server IP address by going to the Administrator page of your router and pull up information on the WAN. E.g. The DNS server address you can see from the following screen is 64.59.144.92.



2.1.5 No of Connections (FServer/ Modbus TCP)

The F-series CPU assigns sufficient memory to support up to a maximum of 10 simultaneous TCP/IP connections to the FServer and Modbus/TCP server. By default, each server is assigned a maximum of 5 connections each. However, to improve flexibility, you can re-assign the mix of maximum connections between the two servers as long as the no. of Modbus/TCP connections does not exceed 5 and the no. of FServer connections does not exceed 8. This means that you can define 5 to 8 FServer connections and 2 to 5 Modbus/TCP connections. When you change the number in one box the other box will change automatically so that the total number of possible connections remains at 10.

2.1.6 FServer Port No.

The Port number is a 16-bit integer (range 0 to 65535) that needs to be specified on top of the IP address when accessing the FServer from across the network. The default value is 9080, which is the same default value used by the TLServer and TRiLOGI client software.

A second port with a value of the main port + 1 is automatically defined for a single FServer connection. Port 9081 is defined by default. The purpose is to allow for a dedicated access port in case the connections on the other port are used up

Please see the TRiLOGI programmer's manual for an explanation of the use of the port number. One reason why you may want to change the port number is to use the "port forwarding" capability of an NAT router so that different F-series PLCs may be accessible from the Internet using the same public IP address of the router but with different port numbers.

2.1.7 Modbus/TCP Secondary Port No.

According to MODBUS.ORG specifications, all Modbus/TCP servers must listen on port #502. However, Modbus.org also permits the device to be assigned a different secondary port number. As such, the Modbus/TCP server will always listen on port #502 for all of its connections by default. Should you choose to define a secondary port number, then the Modbus/TCP server will only listen on port 502 on one connection while the additional connections (1 to a maximum of 5) would be listening on the secondary port.

You may specify any port number between 1024 and 65535 (except for the port number already used by the FServer) to be the secondary port number. Please see the TRiLOGI programmer's manual for an explanation of the use of the port number. One reason why you may want to change the port number is to use the "port forwarding" capability of an NAT router so that different F-series PLCs may be accessible from the Internet using the same public IP address of the router but with different port numbers.

2.1.8 LAN Speed

You can set the LAN speed of the Fx2424 to be either 10 Mbps or 100 Mbps (default) per second using this menu option. Most modern Ethernet switch or router will automatically connect to the PLC using its default LAN speed and there is usually no need for user to configure this option except in very special application where the devices are connected via a 10Mbps Ethernet "hub", which demands all connected device to be set to the same 10Mbps LAN speed.

2.1.9 Node Name

You can assign up to 16 ASCII characters (any character) in naming a PLC. The node name is currently not used by the network router so it is merely a convenient name for user to identify a PLC.

2.1.10 Username and Password (FServer only).

You can use the username and password feature to prevent unauthorized access to the FServer. It adopts the same proprietary encryption scheme used in the TLServer and TRiLOGI software to encrypt the password transmission. However, unlike the TLServer that allows you to define unlimited number of usernames and passwords, the FServer only permits a single set of username/password and this is limited to a length of 16 characters. (Although you may define the passwords for a USER and a VISITOR privilege).

2.1.11 Use Username/Password (Yes/No)?

In applications where there is no danger of unauthorized access to the PLC via FServer, you can elect not to use the username/password. With the "No" option selected, the TRiLOGI client can log-in to the FServer using whatever username and password since FServer will bypass the username and password authentication and allow the client to log in.

2.1.12 Access Level

You can define the access level that the TRiLOGI client is permitted to operate under on the PLC. Three access levels are currently defined: 1 for Programmer, 2 for User and 3 for Guest. Please see the i-TRiLOGI Programmer's Reference manual for the definition of the access levels.

2.1.13 Advanced Configuration

The Advanced Configuration button lets you configure other more advanced (beyond the basic Ethernet configuration), but less often used features of the PLC. This includes definition of the "Trusted IP" addresses (see Section 2.4.2) as well as calibrations of the PLCs Analog I/Os (see Section 5.5).

2.1.14 Standalone Version of Ethernet Configuration Software

The standalone version of the Ethernet Configuration software, that was previously the only option for configuring the Ethernet port and advanced settings on the F-series, can still be downloaded from our website at the following URL:

http://www.tri-plc.com/download/FserverConfig/index.htm

Please download and run the "SetupFPLCConfig.exe" file from the above URL to install the "F-series Ethernet Configuration Utility" program.

After you have installed the configuration program, please click on the Windows "Start" button and open the "F-series PLC Configuration" group and select the "Ethernet Configuration Utility" program. The following screen should appear:

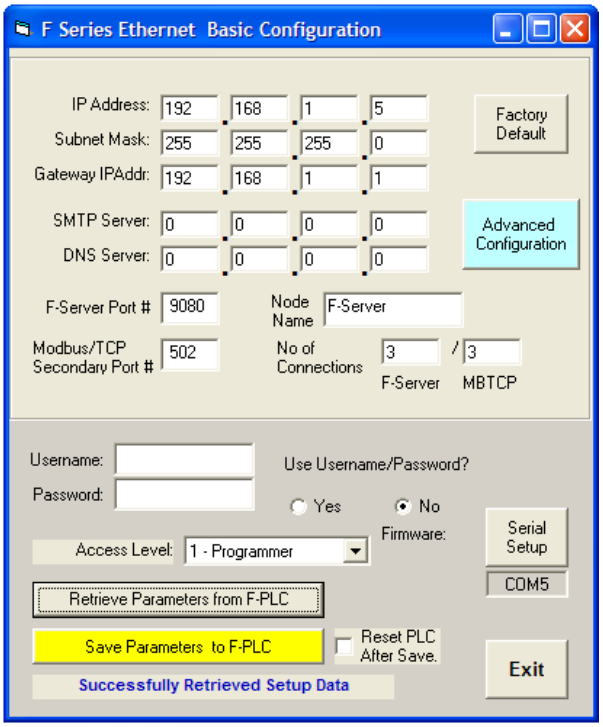


Figure 2.1.4

The configuration program communicates with the PLC via the RS232 serial port, so your PC should have a RS232 port. Otherwise, you should purchase a USB to RS232 converter.

First, click on the "Serial Setup" button and set the PC's COM port to the same settings as the PLC's RS232 port. (Default settings are 38,400bps, 8 data bit, 1 stop bit and no parity). It is important that you select a valid COM port on the PC, otherwise the program will crash and exit when it fails to open the COM port. The selected COM port number is shown in a small text box below the "Serial Setup" button so that you can see the currently selected COM port readily.

Next, click on the "Retrieve Parameters from F-PLC" so that you can capture a copy of the current configuration in the PLC. You can then selectively modify the parameters of interest.

The main difference between this standalone version of the F-series Ethernet Configuration software and the built-in version is that it communicates directly with the PLC via the serial port, whereas the built-in version requires TLServer or the FServer in order to create a connection. However, the standalone version can only communicate through the PLCs serial port, whereas the built-in version can connect to the PLC through a serial or Ethernet connection.

The standalone version can be useful for OEM designers who want to provide their customers with a way to configure the Ethernet settings, and/or calibrate the ADC, and RTC settings but don't want to provide access to the programming software.

2.2 Connecting Ethernet to the PLC

Introduction

All Fx2424 PLC can connect to the PC running TRiLOGI many ways as follows:

a) Wired connection to a router that the PC is also connected to (PC connection can be wired or wireless).
b) Direct connection to the PCs wired Ethernet port via crossover cable
c) Wireless connection to a wireless router if the PLC is connected to a wireless bridge (adapter)

Only the first two options, which are most common, will be described here.

2.2.1 Connecting the PLC to a Local Area Network

In a typical local area network (LAN) there would be one router (wired or both wired and wireless) that the network devices connect to, one modem that provides Internet to the router, and the devices connected to the router (such as the PLC and PC).

Before You Begin

The first thing you need to do is configure the network settings in the PLC to match those of the LAN. This is typically done as follows:

1) Find out what your routers gateway address is (typically 192.168.1.1 or 192.168.0.1) and what static IP addresses are free to use with your PLC.

   NOTE: if the routers gateway address is "192.168.1.1", the default PLC IP address (192.168.1.5) will most likely work unless it is already used by another device on the same network. If it is free to use, the next two steps can be skipped as the PLC will already be able to connect to the LAN.

2) Connect to the PLCs serial port from the PC with TRiLOGI and TLServer.

3) Edit the PLC network settings using the Ethernet & ADC Configuration tool from the "Controller" menu in TRiLOGI. Only the IP address is necessary to configure for basic connection to the LAN.

Please refer to the Quick Connection guide in appendix 1 of the i-TRiLOGI Programmers Reference Manual. to connect the PLC to your PC via serial port.

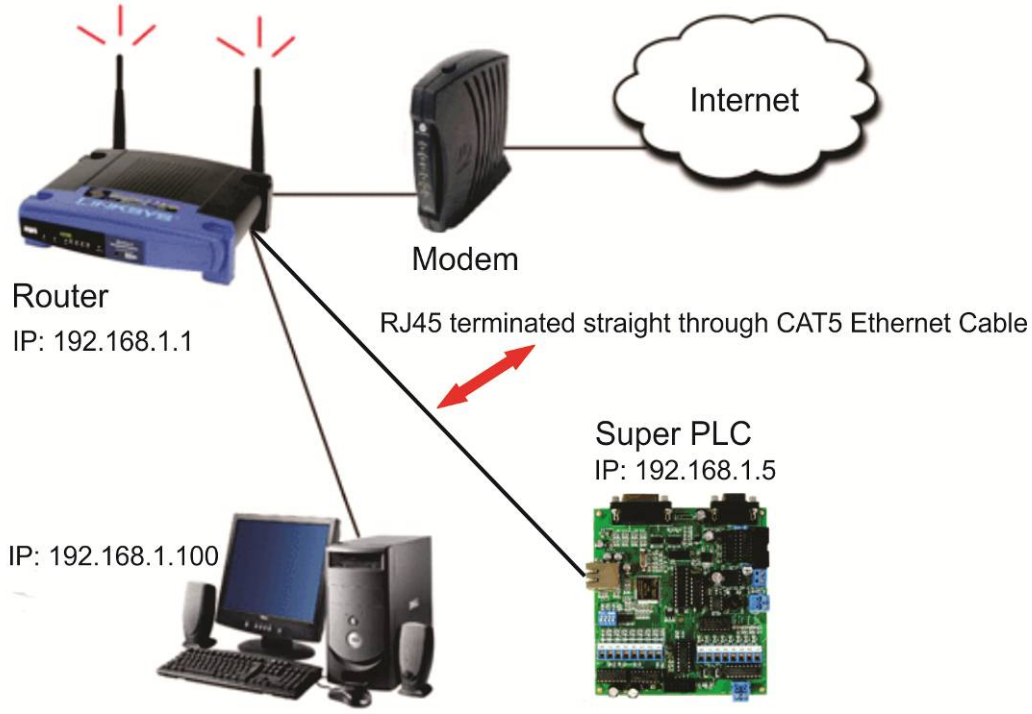
Please refer to section 2.1 of this manual for more detailed information on network configuration.

Network Wiring

You will likely already have a network available that consists of an Internet modem that provides Internet to a wired or wireless router, which has at least one PC connected to it.

In this case, all that needs to be done is connect the PLC to the router with standard Ethernet cable. The final network configuration will likely resemble the below simple network diagram.

If you are on a corporate network, then you will need to consult the IT administrator to get the PLC connected to the network.



2.2.2 Setting up Ethernet Communication Directly Between a PC and a F-series PLC

It is possible for your PC to communicate directly with a F-series PLC through the Ethernet port. This means that you can have a peer-to-peer connection between the F-series PLC and your PC without an Internet connection or any additional network equipment. The PC being used would need a spare Ethernet port in order to do this.

What You Will Need

1. A PC with a spare Ethernet port (RJ45 connection)
2. An Ethernet cable**
3. A F-series PLC

**There are two types of Ethernet cables: A standard straight through cable, which is used in most situations, and a crossover cable, which is required when there is no auto-switching hardware in either of the two Ethernet ports (PC and PLC). The F-series PLC does NOT have auto-switching capabilities so it will depend on the PCs Ethernet port whether or not a crossover cable is required. If you are not sure if your PC's Ethernet port is able to perform auto-switching (most modern PC has auto-switching built-in), then it is best to use a crossover cable. A crossover Ethernet cable will work in both situations, but a straight through Ethernet cable will only work if one or both of the Ethernet ports can perform auto-switching.

How to Set Up Your PC

Before you can communicate with the PLC, you will need to configure the Ethernet port on your PC to match the settings that you configured in the PLC. For more information on configuring the PLC Ethernet settings, see section 2.1 Configuring The Ethernet Port. Once you have configured your PLC's Ethernet settings, you will need to set your PCs Ethernet port to have a static IP address that is on the same subnet as the IP address you had set for your PLC. How you do this is a bit different depending on your operating system, so the following sections will show step-by-step instructions for configuring your PCs Ethernet settings on Windows XP and then on Windows Vista.

Windows XP Ethernet Configuration

1. **Open Network Connections**. The first thing you will need to do is open the Network Connections window from the "Start" menu by selecting "Show all connections" from the "Connect To" menu, as shown here:

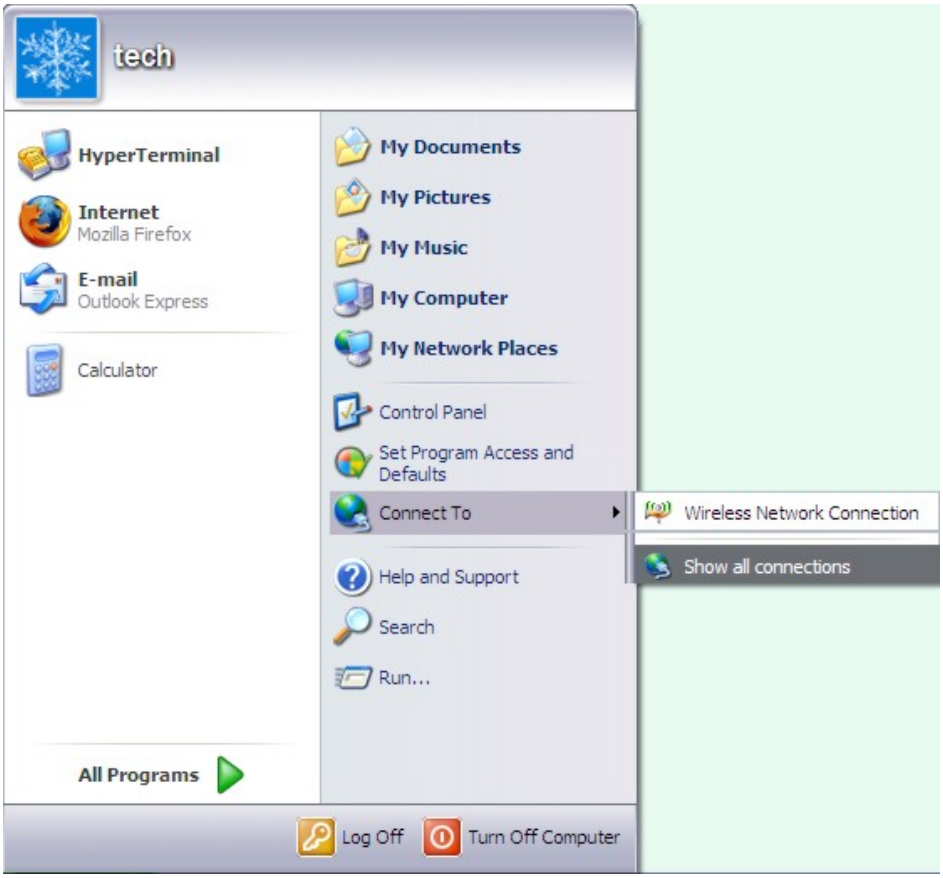


Figure 2.8.1

2. **Local Area Network Properties**. Next the Network Connections window should open, which will display all of your network connections.

Now you can either right click on the Local Area Connection and select "Properties" or highlight it and click on "Change settings of this connection" from the Network Tasks sidebar, as shown above. When you do this, the following window will open.

Figure 2.8.2

3. Edit TCP/IP Properties

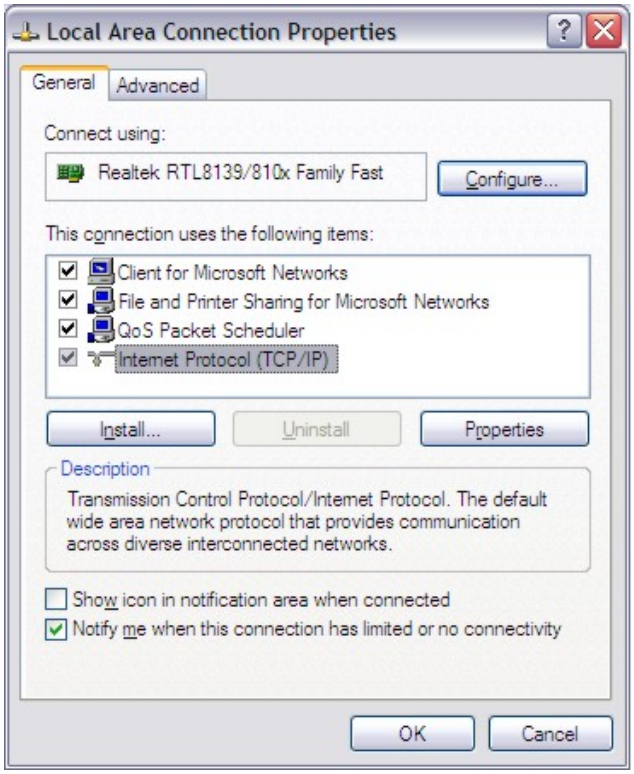


Figure 2.8.3

Now you want to highlight the Internet Protocol (TCP/IP) item and click on Properties. This will open up a new window that allows you to change the network settings for the Ethernet port that you selected. The Ethernet hardware you are configuring is shown under "Connect using:" in the current window.

4. Configure IP Address and Gateway

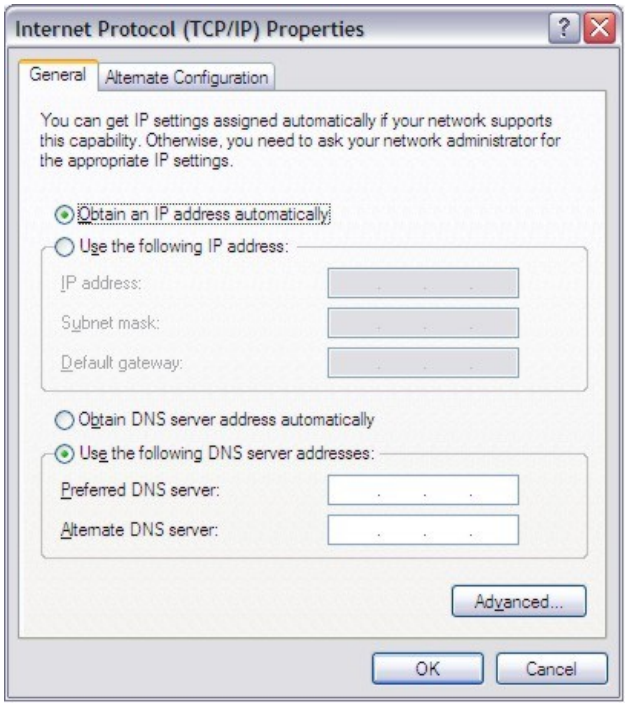


Figure 2.8.4

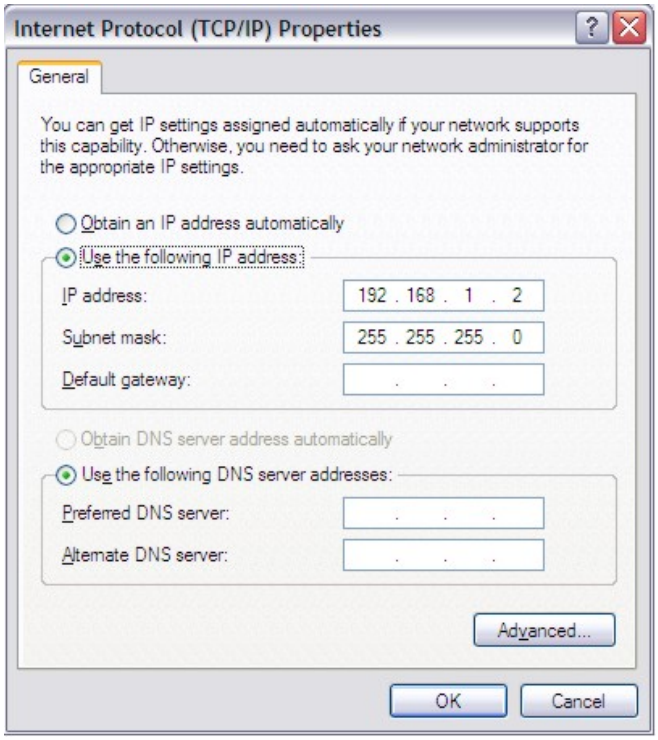
By default your Ethernet settings will be configured to be obtained automatically (in this case only the IP settings are set to be obtained automatically). This setting will not work since the F-series PLC does not have the ability to assign IP address to the PC's Ethernet port. So you need to manually set the IP address and the Subnet mask. The IP address should be on the same subnet as the IP address you have set for your PLC. For example, if your PLCs IP address is set to the default address of "192.168.1.5", then you should set your PCs IP address to be:

192.168.1.xxx

The xxx can be any number between 1 and 255, except for 5 (your PC can't have same IP address as your PLC)

The Subnet mask should be set to "255.255.255.0" no matter what and the other settings should be left blank because you won't be connecting this Ethernet port to the Internet.

5. Sample Configuration



Here is an example of a configuration that will work if your PLC has an IP address of 192.168.1.xxx, where xxx is anything from 1-255 except 2.

Once you have set the IP address and Subnet mask, you can click on OK and then click on OK again to close the "Local Area Connection Properties" window shown in Figure 2.8.5

Note: When using direct Ethernet connection you should leave the "Default gateway" field empty. This can prevent the PC from trying to access the Internet through its Ethernet port which can cause the PC to lose internet connection.

Figure 2.8.5

Now your Ethernet port has been configured and you can try to connect to your PLC from I-TRiLOGI using your PLCs IP address and port number.

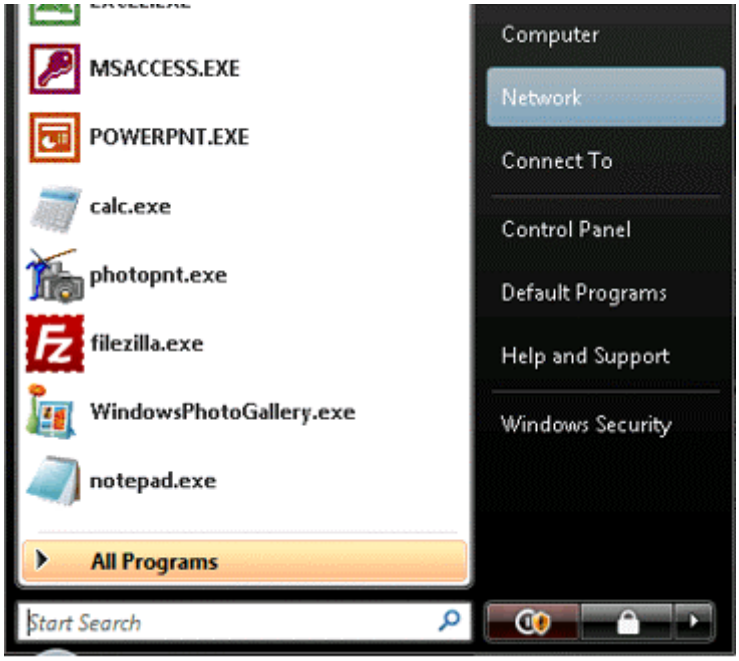
Note:

a. Connecting the PC's Ethernet port directly to the PLC only establishes a private network connection between the two devices. That is to say, the PLC connected this way is ONLY accessible to this PC. Even if the PC is connected to a LAN (e.g. via Wi-Fi connection), the PLC connected to this PC via direct Ethernet connection will not be accessible by any other devices on the same LAN. and the PLC will not be able to open network connection to any other devices beyond the PC that it is connect to this way.

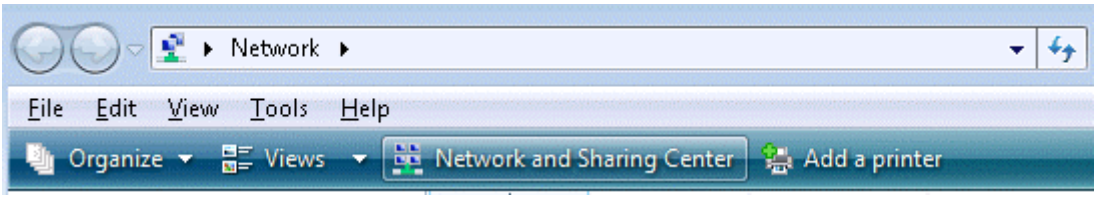
Sometimes the PC that is connected directly to the PLC via Direct Ethernet Connection may lose its Internet connection via Wi-Fi. This is because the O/S thinks that the direct Ethernet connection to the PLC is a real network connection and attempt to route TCP/IP packets through it. That will result in a connection failure. If this occurs and you need to access the Internet, then please temporarily remove the Ethernet cable from the PLC and the PC should then be able to access the Internet again.

Windows 7 Ethernet Configuration

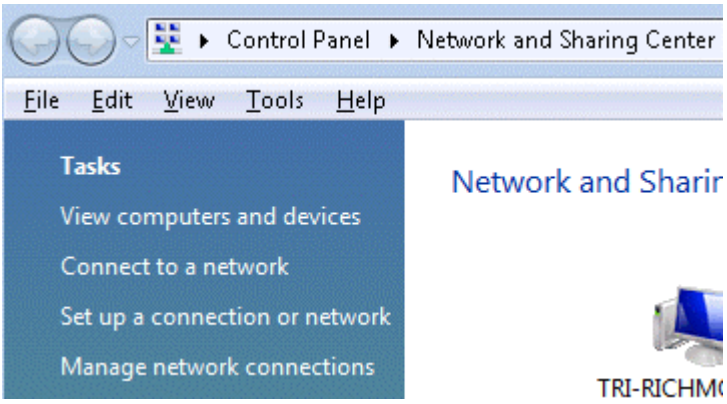
1. **Open the Network Window**. The first thing you will need to do is open the Network window from the "Start" menu by selecting "Network", as shown here:



2. **Open Network and Sharing Center**: In the Network window, click on the Network and Sharing Center option on the top pane as in the below picture.



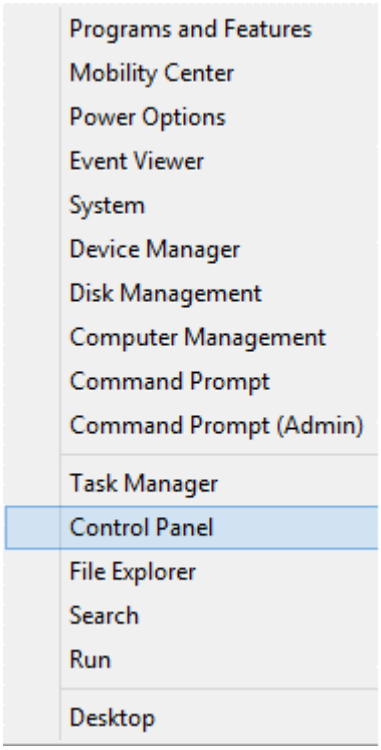
3. **Manage Network Connections**: In the Network and Sharing Center window, click on the Manage network connections option under the tasks pane on the left side of the window as in the below picture.

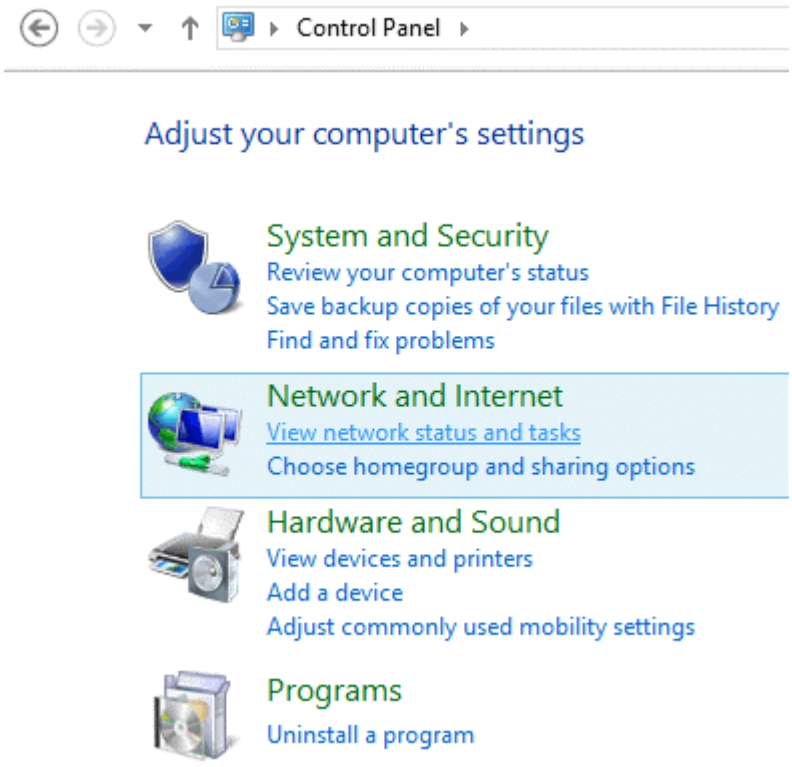


4. Follow steps 2-5 in the Windows XP Ethernet Configuration section above to complete the setup, as it is the same process.

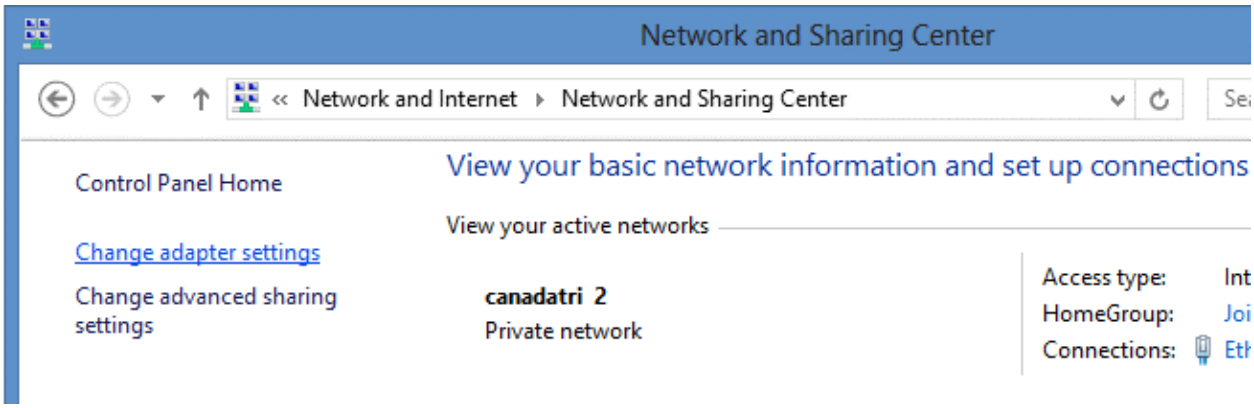
Windows 8 Ethernet Configuration

1. **Open the Network Window**. The first thing you will need to do is open the Control Panel from the "Start" menu by right-clicking in the bottom left corner of the screen where the Start Menu is accessed (below left screenshot):





2. **Open View Network Status and Tasks Window**. Next you need to click on the "View network status and tasks" link under the "Network and Internet" category (above-right screenshot), which will bring up the "Network and Sharing Center" window (below screenshot).



3. **Click on the "Change Adapter Settings" Link**. This will bring up the "Network Connections" window where you can configure the necessary network settings for the local adapter.

Follow steps 2-5 in the Windows XP Ethernet Configuration section above to complete the setup, as it is the same process.

2.3 On-line Monitoring/Programming via FServer

If you have used the I-TRiLOGI software to connect to TLServer previously, the procedure is identical. To test TRiLOGI communication with the F-series PLC, click on “Controller -> On-Line Monitoring”, or simply press <CTRL-M> keys.

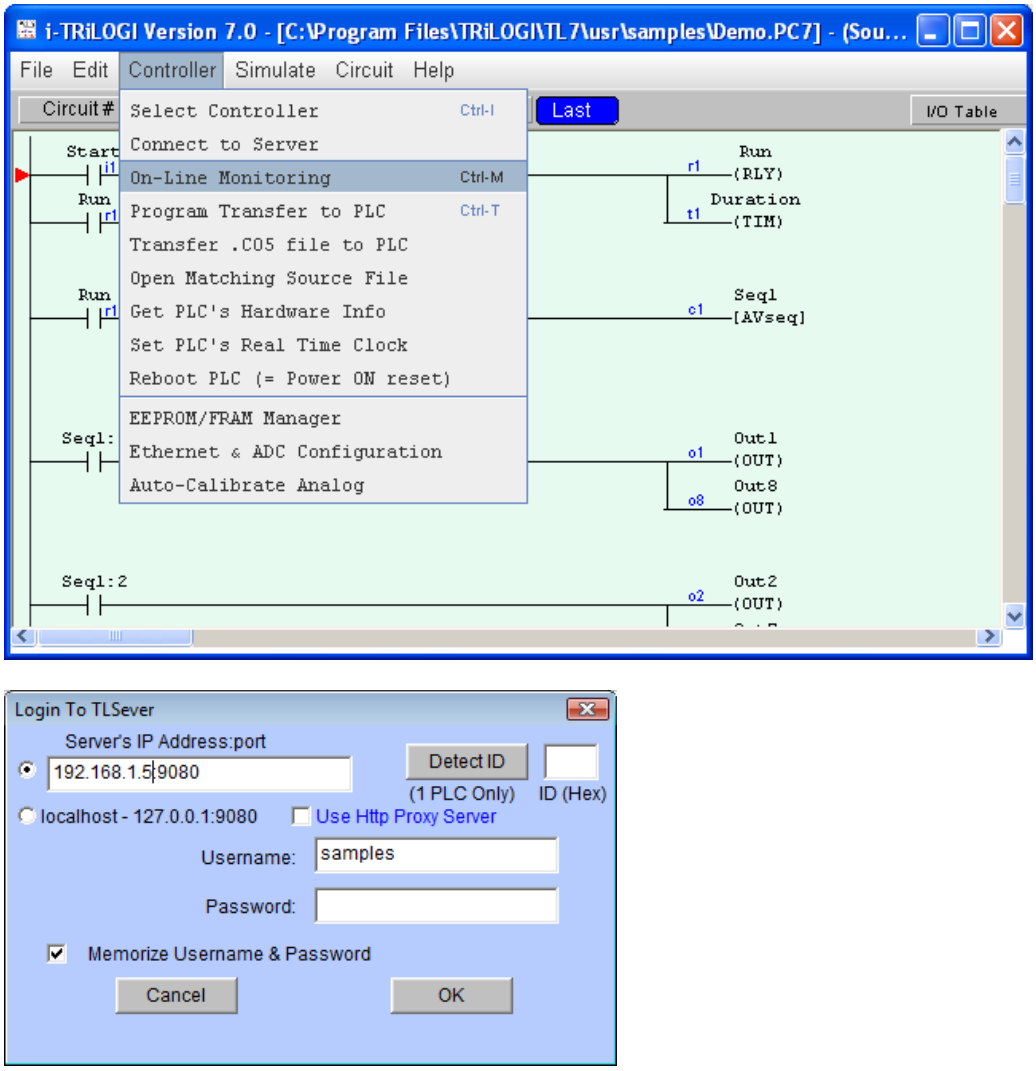


Figure 2.2.1

When the “Login To TLServer” screen pops up, enter the “IP address : port”, the username and the password that you have defined for the FServer earlier using the configuration software, and click on the “Detect ID” button to detect the PLC’s ID. If TRiLOGI is able to connect to the Fx2424 via the Ethernet network, then the PLC’s ID will appear in the ID box. When you click the “OK” button, the on-line monitoring screen should appear and you should see the “Activities LED” on the RJ45 connector blinking away. You have now successfully connected to the FServer and you can run all the commands under the “Controller” menu, including transferring your new program to the PLC or setting the PLC’s Real Time Clock, etc. For more details on using these commands, please refer to i-TRiLOGI Programmer’s Reference.

Likewise, to transfer your new program to the PLC, you can click on the "Controller" menu and select "Program Transfer to PLC" or press the <Ctrl-T> keys. If TRiLOGI is already connected to the FServer, the program transfer will begin immediately after you've confirmed your action. Otherwise the same "Login to TLServer" screen, as shown on Figure 2.2.1, will appear for you to complete the login sequence.

Note:

1) Unlike the TLServer, which allows unlimited connection time, the FServer on the PLC will disconnect the client if there is no activity for more than 10 minutes. The current version of the TRiLOGI program may not detect that the connection has been closed and it may instead think that the PLC is not present. When this happens you should click on "Controller" menu and select "Disconnect" to properly shut down the connection that has already been reset by the PLC.

2) If you are unable to connect to the PLC, then check that both the PLC and the PC running your TRiLOGI software are connected to the same network and are on the same subnet. Generally for a subnet mask of 255.255.255.0, if the PC's IP address is 192.168.1.xxx then the PLC should have an IP address of 192.168.1.yyy and it will not work if the PLC has IP address such as 192.168.0.yyy or 192.168.2.yyy, since this means that the two devices are on different subnets. Likewise, if your PC's IP address is "192.168.0.xxx", then please change your PLC's IP address to "192.168.0.yyy". Also ensure that the PLC's IP address is not already assigned to another device on the same network, otherwise a conflict would occur and communication is not possible.

2.4 Using F-series PLC "Network Services" Commands

The Fx2424 implements a list of "Network Services" commands similar to what you may have read in the User's Manual of the X-Server ("NS commands) and TLServer ("Files and Email Services").

These "Network Services" or **NS** in short, can be used to instruct the Fx2424's operating system to perform a number of network related functions client connection via the Ethernet port. These commands allow the PLC to connect remotely to another PLC in another building or another part of the world via the Internet! This allows peer-to-peer networking, or so-called "M2M" (machine to machine communication) to take place between the PLCs.

Notes:

1. In the case of the TLServer, the PLC typically communicates with these external hardware or software servers via its COMM1 serial port, and thus TBASIC statements and functions such as PRINT #1, INPUT$(1) and NETCMD$(1) are used since the NS commands are sent through the serial port #1.

2. Since the Ethernet is already built-in on the Fx2424, you do not need to send NS commands via any of its serial ports (this also means that no serial port is sacrificed in order to have access to Ethernet communication). However, to help users of the XServer and TLServer migrate easily to the Fx2424 Ethernet port, we implement the NS commands using similar command format as that on the XServer and TLServer. But instead of sending the commands through COMM1, you will interact with the O/S through COMM port #4.

   Of course, since the Fx2424 doesn't have 4 serial ports, COMM port #4 is therefore only a "virtual comm. port" and its creation is merely to simplify implementation of the NS commands.

3. The "TestEthernet.pc6" (download from: http://www.tri-plc.com/trilogi/FPLCsamples.zip)

includes all examples of how to use the NS commands via virtual comm. port #4, which serves as a good starting point for you to learn these simple but yet powerful methods for making a client connection over the LAN or the Internet.

4. The PLC reserves only a single client socket to implement the Network Services. If you use any of the NS commands listed below, please ensure that the command is completed (so that the client socket can be closed) before issuing a different NS command.

All NS commands begin with a string enclosed within the angle bracket called a "tag", e.g. "<EMAIL>", "<CONNECT>". Most NS commands end with a closing tag "</>" except the "<REMOTEFS>" tag, which ends with a "</REMOTEFS>" closing tag. Depending on the command type, the F-series CPU may return one or more response strings via virtual comm. port #4, from which the PLC can read to determine if the NS command has been executed properly.

The PLC can operate the Ethernet port by means of TBASIC INPUT$ and PRINT commands operating on COMM 4. It uses the PRINT #4 command to send out NS commands and the INPUT$(4) command to receive response data via the Ethernet port.

Notes:

1. Only PRINT #4 and INPUT$(4) are implemented on virtual COMM port #4. The Fx2424 currently DOES NOT support the INCOMM (4) and OUTCOMM 4 commands. Note that any non -zero ASCII data can be sent using the PRINT #4 command.

The following subsections describe the various Network Service commands available to the Fx2424.

2.4.1 Get our IP Address

Format: <IP>
Response: xxx.xxx.xxx.xxx:nnnn (IP address:port of FServer)

Example:
```
PRINT #4     "<IP>"
SETLCD 1,1,"Our IP="+INPUT$(4)
```

Note: This IP address is returned instantly, so there is no need to wait for INPUT$(4).

2.4.2 DNS command: Resolving Domain Name into IP Address

In order to use this command successfully, you must first correctly define the DNS Server IP address mentioned in Section 2.1.4.

Format: **`<DNS [domain name]>`**

Response:

xxx.xxx.xxx.xxx	IP address string returned by DNS server
ERR:07-DNS Unresolved	Either DNS server not properly defined or the domain name does not exist.

Response: xxx.xxx.xxx.xxx (IP address string returned by domain name server)

STATUS(3): This function returns 1 on success and 0 on failure.

Example:

```
PRINT #4  "<DNS tri-plc.com>"
FOR I = 1 to 10000
     A$ = INPUT$(4)
     IF LEN(A$) <> 0
          SETLCD 1,1," IP="+A$
     RETURN
     ENDIF
NEXT
```

Notes:

a) There is no need for the closing tag </> to end this command.

b) If your DNS server has been correctly defined, the above program should return the IP address as a string such as "130.94.216.144". You can then use this IP address string in all the other NS commands to be described in the following sub-sections.

c) The DNS server may take some time to resolve the domain name. If it is unable to resolve the domain name then it will return an error string, so your program should test to see if it receives the ERR07 error message to determine whether the returned string is useable.

d) Although it is possible to embed the domain name directly in the NS command in place of IP address, it is usually much more efficient to use the IP address directly if it is known in advance. This is because the DNS server may take some time to resolve the domain name into IP address each time it is called, and there is a possibility that the domain name server may be overloaded or down momentarily when it is needed, and hence complicating the attempt for the PLC to connect to a remote server. Therefore we recommend that you use the <DNS *domain* > tag to resolve the domain name into IP address first and then use the resulting IP address for all Network Services commands via the Internet.

2.4.3 Send Email

Format:

```
<EMAIL [recipient email address]>
        SENDER:   [sender email address]
SUBJECT: [whatever text string]
[body of the email line 1]
[body of the email line 2]
    .....
</>
```

Response:

<OK>	Email successfully sent
ERR:04-Not Connected	Failed to connect to SMTP server (Section 2.1.3)
ERR:06-Email Failure	Failed to complete email transmission.

STATUS(3): This function returns 1 on success and 0 on failure. Note that this function only returns the email status after the closing tag </> has been sent. If the function is polled before the last closing tag is sent, the status is indeterminate.

Description: You can use this command to send out an email for you at any time. The FServer uses the defined SMTP server and Gateway IP addresses (See Section 2.1) to perform this task. If it encounters any errors, it will send back an error string, which begins with the "ERR:" followed by the reason for the error. Although the sender's email address does not have to be a valid email address, it is good to at least use a valid domain name as the sender address. Otherwise the SMTP server may refuse to send the email because it may deduce that an email with an invalid domain name is likely to be a Spam mail.

Example: Please refer to the fnEmail function in the "TestEthernet.PC6" file.

2.4.4 Open Connection to Remote FServer or TLServer to Use NETCMD$

Format: <CONNECT [IP address:port of TLServer or XServer]>
[username string]
[password string]

Response:

<CONNECTED>	Successfully connected to remote FServer, TLServer or XServer at the IP address.
ERR:05-Prev Conn.ON	Another NS command has been executed and left the client socket opened but did not execute the PRINT #4 "</>" to close the client socket.
ERR:04-Not Connected	Failed to connect to remote Fserver or TLServer.

STATUS(3): This TBASIC function returns 1 if the connection is active and returns 0 if the connection has ended. You can test the connection status to determine if the connection is still alive.

Description: This service allows your PLC to log in to another TRi Super PLC such as another Fx, FMD or Nano-10 PLC connected via TLServer through the Internet.

You execute this command by first sending the string "**<CONNECT** xxx.xxx.xxx.xxx:9080**>"** using the PRINT #4 command, where xxx.xxx.xxx.xxx is the IP address of the remote FServer or TLServer, followed by sending the username and password needed to log in to the remote server. Each line should be terminated with a CR (carriage return) character. (The PRINT #4 command automatically appends the CR character).

Once a connection with the remote server is established, the CPU will return the response string **<CONNECTED>** to the user program, which can read it using the INPUT$(4) function. The STATUS(3) function can also be used to test if the connection is successful and alive. When the program gets the confirmation of connection, it can then use the TBASIC "NETCMD$(4, x$)" command to read or write data to the remote PLCs as if the remote PLC is locally connected to COMM4 port of this PLC, as shown in the following example:

```
A$ = NETCMD$(4, "@01RI00")
```

Multiple NETCMD$ commands can be executed as long as the connection is alive. You can test the connection status by checking the result of the STATUS(3) function.

Once all the command exchanges have been completed, you should send a **</>** tag to close the client connection to the remote server so that other NS commands can be executed in other parts of the program.

Example: Please refer to the "fnConnect" and "fnNetCmd" custom functions in the demo program: "TestEthernet.PC6".

2.4.5 Remote File Services

Format: <REMOTEFS [IP Address of remote TLServer 2.1 & above]>
[File Service tag for TLServer]
…..
</REMOTEFS>

Response: The response strings sent by the remote TLServer in response to the [File Service tag] sent by this PLC. Or,

ERR:04-Not Connected	Failed to connect to remote TLServer.

Example: Please refer to the "fnRFS1" and "fnRFS2" custom functions in the demo program: "TestEthernet.PC6".

Description: This commands allows the Fx2424 to connect to a remote TLServer to perform any of the "Files & Email Services" that a TLServer normally provides to PLCs that are connected to it. This includes creating text files on a remote TLServer and writing or appending data to it anytime. This makes it very convenient for the PLC to collect large amounts of data and save them to the easily accessible, virtually limitless hard disk storage space that is available in today's PCs.

For detailed descriptions of the available [File Service Tags] please refer to TRiLOGI programmer's reference manual under the chapter "File & Email Services".

All TLServer's "File & Email Services" tags, such as <Email>, <WRITE>,<APPEND>, <READ> and <READ RTC> are available to the F-series PLC through the use of the **<REMOTEFS>** tag. You simply have to wrap the abovementioned command tags between the **<REMOTEFS** IPAddr:port**>** and **</REMOTEFS>** tag, where "IPAddr:port" is the IP address and listening port of the remote TLServer. E.g. through the <READ RTC[]> tag, the PLC can synchronize its Real Time clock with a remote TLServer. (As you will later see, this feature is probably not very useful for the Fx2424 anymore since Fx2424 has the ability to connect to the NIST Time Server to update its real time clock to Atomic clock accurately!)

Note: Only **TLServer version 2.1 or above** can handle the <REMOTEFS> command tag sent by the Fx PLC.

2.4.6 Other Network Services Tags

We will describe two more Network Services commands: <TCPCONNECT> and <MBTCPCONNECT> in separate sections later in this manual.

2.5 MODBUS/TCP Server and Client Connection

The Fx2424 supports both the FServer and the industry standard **MODBUS/TCP** server simultaneously. This means that all F-series PLCs are ready to interface directly with many third party industrial control devices that support the MODBUS/TCP protocol. These include the SCADA software, HMI hardware, OPC Server, HVAC controllers and many other industrial control devices.

In addition, the Fx2424 can be used both as a MODBUS/TCP **SERVER** as well as a MODBUS/TCP **CLIENT** simultaneously**.** This means that the Fx2424 can readily read data from any device that has a MODBUS/TCP server, such as: flow meters, AC/DC drives, HVAC elements, RTUs, network sensors etc. It is also possible to perform **peer-to-peer** networking with other MODBUS/TCP controllers (e.g. another F-series PLC) over a LAN or over the Internet!

2.5.1 Connecting To The PLC's MODBUS/TCP Server

By default, the F-series CPU supports up to 3 simultaneous MODBUS/TCP connections. You can change the number of simultaneous MODBUS/TCP connections from 2 to 5 using the "F-series Ethernet Basic Configuration" program as described in Section 2.1.

The PLC will listen on the default, well-known MODBUS/TCP port #502 for one or all of the connections. However, it is also possible to define a secondary port number as described in Section 2.1.7 (Note that if you define a secondary port number, then only one of the MODBUS/TCP connection will listen on port #502 and the remaining connections will only be listening on the secondary port number.)

If you have a MODBUS/TCP client program (e.g. you can download a trial version of "Modbus Poll" from http://www.modbustools.com for testing), you simply specify the F-PLC's IP address and connect to it. Once connected, you will then be able to read from or write to most of the Fx2424's internal data from the MODBUS/TCP client. The PLC's I/O and internal variables are mapped to the MODBUS device space according to table 2.5.1.

2.5.1.1 Bit Address Mapping

All the F-series I/O bits are mapped identically to both the MODBUS "0x" and 1x space. The bit register offset is shown in the last column of Table 2.5.1. Although MODBUS names the "0x" address space as "Coil" (which means output bits) and the "1x" address space as "Input Status" (which means input bits only), the Fx2424 treats both spaces the same. Some MODBUS drivers only allow a "read" from 0x space and a "write" to 1x space but you still use the same offset shown on Table 2.5.1.

Example:

1. To map an element to the PLC Input 5, you select the MODBUS register address 0-0005. You can also map the element to the PLC's output #2. In that case, you should map it to MODBUS register address 0-0258.

2. To map an HMI toggle switch symbol to the PLCs input #5, if you are restricted to select only MODBUS 1x address space, then you will have to map the switch to 1-0005, and, likewise, you can map the switch to output #2 using the MOBDUS address 1-0258. However, if the driver allows the switch to be mapped to the 0x space then you can use MODBUS register space 1-0258 and 0-0258 for the output #2 mapping with identical result.

Table 2.5.1 Memory Mapping of F-series CPU Internal Data to MODBUS Register

Fx2424 PLC	I/O #	MODBUS Holding Register Mapping (Modicon convention)	MODBUS Bit Addr. Mapping
Input	**n**		**n**
	1 to 16	40001.1 to 40001.16	1 to16
	17 to 32	40002.1 to 40002.16	17 to 32
	33 to 48	40003.1 to 40003.16	33 to 48
	49 to 64	40004.1 to 40004.16	49 to 64
	65 to 80	40005.1 to 40005.16	65 to 80
	81 to 96	40006.1 to 40006.16	81 to 96
Output	**n**		**256 + n**
	1 to 16	40017.1 to 40017.16	257 to 272
	17 to 32	40018.1 to 40018.16	273 to 288
	33 to 48	40019.1 to 40019.16	289 to 304
	49 to 64	40020.1 to 40020.16	305 to 320
	65 to 80	40021.1 to 40021.16	321 to 336
	81 to 96	40022.1 to 40022.16	337 to 352
Timer	**n**		**512+n**
	1 to 16	40033.1 to 40033.16	513 to 528
	17 to 32	40034.1 to 40034.16	529 to 544
	33 to 48	40035.1 to 40035.16	545 to 560
	49 to 64	40036.1 to 40036.16	561 to 576
Counter	**n**		**768 + n**
	1 to 16	40049.1 to 40049.16	769 to 784
	17 to 32	40050.1 to 40050.16	785 to 800
	33 to 48	40051.1 to 40051.16	801 to 816
	49 to 64	40052.1 to 40052.16	817 to 832
Relay	**n**		**1024 + n**
	1 to 16	40065.1 to 40065.16	1025 to 1040
	17 to 32	40066.1 to 40066.16	1041 to 1056
	33 to 48	40067.1 to 40067.16	1057 to 1072
	49 to 64	40068.1 to 40068.16	1073 to 1088
	65 to 80	40069.1 to 40069.16	1089 to 1104
	81 to 96	40070.1 to 40070.16	1105 to 1120
	97 to 112	40071.1 to 40071.16	1121 to 1136
	113 to 128	40072.1 to 40072.16	1137 to 1152
	129 to 144	40073.1 to 40073.16	1153 to 1168
	145 to 160	40074.1 to 40074.16	1169 to 1184
	161 to 176	40075.1 to 40075.16	1185 to 1200
	177 to 192	40076.1 to 40076.16	1201 to 1216
	193 to 208	40077.1 to 40077.16	1217 to 1232
	209 to 224	40078.1 to 40078.16	1233 to 1248
	..	..	..
	497 to 512	40097.1 to 40097.16	1521 to 1536

Fx2424 PLC	Variables	Modbus Holding Register (Modicon convention)	Modbus Holding Register (Binary address)
Timer Present Values	1 to 64	40129 to 40192	128 to 191
Counter Present Values	1 to 64	40257 to 40320	256 to 319
Clock	TIME[1] TIME[2] TIME[3]	40513 40514 40515	512 513 514
Date	DATE[1] DATE[2] DATE[3] DATE[4]	40517 40518 40519 40520	516 517 518 519
Data Memory	DM[1] DM[2] …. DM[4000]	41001 41002 …. 45000	1000 1001 … 3999
Floating-point Array *(Note: available only on Fx model)*	FP[1] FP[2] … FP[501] ….. FP[1000]	45001 & 45002 45003 & 45004 ….. 46001 & 46002 …. 46999 & 47000	5000 & 5001 5002 & 5003 ….. 6000 & 6001 …. 6998 & 6999

2.5.1.2 Word Address Mapping

As shown in Table 2.5.1, to access the PLC's DM[1], you use MODBUS address space 4-1001 (Modicon, use 1000 for binary addressing) and so on. To access the Real Time Clock Hour data (TIME[1]), use 4-0513. The I/O channels can also be read or written as 16-bit words by using the addresses from 4-0001 to 4-0320.

Some MODBUS drivers (such as National Instruments "Lookout" software) even allow you to manipulate individual bits within a 16-bit word. So it is also possible to map individual I/O bits to the "4x" address space. E.g. Input bit #1 can be mapped to 4-0001.1 and output bit #2 is mapped to 4-0257.2, etc. This is how it is shown in Table 2.5.1. However, if you do not need to manipulate the individual bit, then you simply use the address 4-0001 to access the system variable INPUT[1] and address 4-0257 to access the system variable OUTPUT[1]. Note that INPUT[1] and OUTPUT[1] are TBASIC system variables and they each contain 16 bits that reflect the on/off status of the actual physical input and output bits #1 to #16.

2.5.2 MODBUS/TCP Access Security

If a Fx2424 is to be accessible only on the local area network, then the direct connections offered by MODBUS/TCP provide simplicity without time-consuming login sequences. As such Modbus/TCP

protocol is itself inherently insecure and not meant to be exposed directly to the Internet where malicious attacker can take control of the equipment.

Typically Modbus TCP devices are deployed in a local area network and protected by the network firewall. A SCADA or OPC server program is used to connect to the Modbus/TCP devices on LAN and the PC running the SCADA/OPC server software can then be the only device exposed to the Internet where secured access can be implemented by the server software. External HMI client software can then make connection to the SCADA/OPC server which will in turn access the Modbus/TCP devices such as the F-series PLCs.

However, if you only have a single Modbus/TCP device and need to expose it to the public Internet for external access, then you ought to consider the security issues associated with MODBUS/TCP connections.

Since a MODBUS/TCP connection does not require a username/password login sequence (unlike the FServer login), the only way to protect against unauthorized access is through the "Trusted IP" addresses defined using the F-series Ethernet Configuration software.

To define a list of "Trusted IP" addresses, please click on TL7's "Controller" menu and select "Ethernet & ADC Configuration". Next, click on the "Advanced" button on the "FServer Basic Configuration" and you should see the following Advanced Configuration screen.

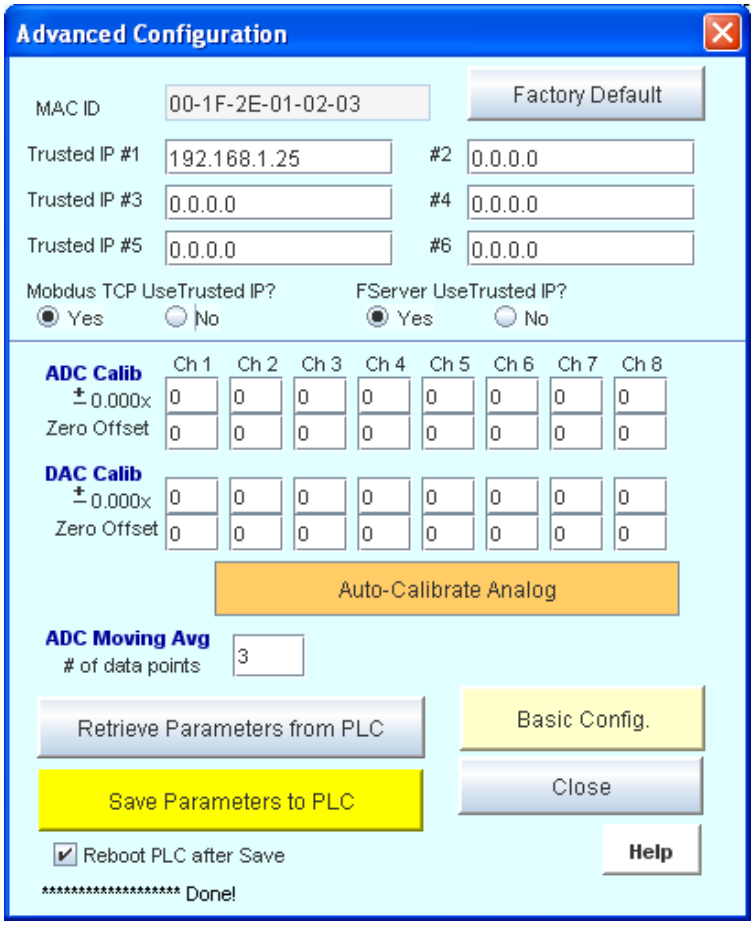


Figure 2.4.1

The first thing you should do is to click on the "Retrieve Parameters from PLC" so that you can capture a copy of the current configuration in the PLC and you can then modify selectively.

You can define a list of up to 6 "Trusted IP" addresses in this panel. To enable the Modbus/TCP Trusted IP, click on the "Yes" button next to the "Modbus/TCP Use Trusted IP".

Note: The FServer can also be enabled to only allow connections from devices that match one of the "Trusted IP" defined in this panel. This is on top of the username/password login sequence that can be enabled/disabled from the Basic Configuration screen. In other words, you can choose either security method to access the FServer or implement both security methods at the same time.

After you have defined the list of trusted IP addresses and checked the "Use Trusted IP" radio button, click on the "Save Parameters to FServer" to save your data to the PLC's non-volatile memory.

When “MODBUS/TCP Use Trusted IP” is enabled, it means that only TCP/IP packets that come from a client whose IP address matches one of the “Trusted IP” would be allowed connection to the MODBUS/TCP server.

2.5.3 Making a Modbus/TCP Client Connection to Other Modbus/TCP Server

By using the “Network Services” commands described in Section 2.4, it is unbelievably easy for the Fx2424 to be used as a MODBUS/TCP client to access any industrial control or HVAC device and sensors that support a MODBUS/TCP server. Best of all, you can do it without learning any specifics of TCP/IP programming!

To open a client socket and connect to a Modbus/TCP Server that is listening on port 502 (default Modbus/TCP port), you only need to send the command tags <MBTCPCONNECT xxx.xxx.xxx.xxx:502> to the CPU via virtual COMM port #4. E.g.

```
PRINT #4 "<MBTCPCONNECT 192.168.1.105:502>"
```

If connection is successful, the system will return the string “<CONNECTED>” on virtual comm. port #4, which you can check with the INPUT$(4) command.

Once the connection is successfully established, you can begin to use the built-in TBASIC commands: READMODBUS, WRITEMODBUS, READMB2 and WRITEMB2 operating on virtual comm. port #4 to send MODBUS commands and receive processed responses from a remote MODBUS/TCP Server!! This greatly simplifies your programming task, since it is very similar to communicating with a Modbus RTU slave that is connected to the serial port #1, 2, or 3. Although in this case, the Modbus/TCP device could be located in the other hemisphere and connected via the Internet!

The full syntax for the <MBTCPCONNECT> tag is described below:

Format: <MBTCPCONNECT [IP address:502] of another Modbus/TCP Server>

Response:

<CONNECTED>	Successfully connected to the Modbus/TCP server of the specified IP address.
ERR:05-Prev Conn.ON	Another NS command has been executed and left the client socket opened but did not execute the PRINT #4 “</>” to close the client socket.
ERR:04-Not Connected	Failed to connect to the targeted Modbus/TCP Server

STATUS(3): This TBASIC function returns 1 if the Modbus/TCP connection is live and returns 0 if the connection has ended. You can test the connection status to determine if the connection is still alive.

Description: This service allows your PLC to log in to any device that supports a Modbus/TCP server and is connected to the same LAN or to the Internet. Of course, you may also use it to connect to another F-series PLC on the Internet since every F-series PLC has a MODBUS/TCP server too.

Once the connection with the Modbus/TCP server is established, the CPU will return the response string **<CONNECTED>** to the users program, which can

read it using the INPUT$(4) function. The STATUS(3) function can also be used to determine if the connection is successful and alive.

When the program gets the confirmed connection, it can then use any one of the four TBASIC commands: READMODBUS, WRITEMODBUS, READMB2, WRITEMB2 to read or write data to the remote devices via the virtual comm. port #4, as if a Modbus slave device has been locally connected to a COMM4 port of this PLC. (You do not need to distinguish between Modbus ASCII and RTU in this case, simply use comm. port #4 in your all your commands).

Multiple Modbus master commands can be sent as long as the connection is live. You can test the connection status by checking the result of the STATUS(3) function at any time.

Once all the command exchanges have been completed, you should send a **</>** tag to close the client connection to the remote server so that other NS commands can be executed in other parts of the program.

Example: Please refer to the “fnMBTCP”, “fnRdMBTCP” and “fnWrtMBTCP” custom functions in the demo program: “TestEthernet.PC6”.

2.6 Getting data from Internet: Connecting to The Internet Time Server

The Fx2424 features a special NS command tag <TCPCONNECT xxx.xxx.xxx.xxx: portno> that allows you to connect to any server to download data. However, since the PLC does not have a lot of memory for storing incoming text data, it is not suitable for downloading information from a commercial website that sends many kilobytes of data in a single download. It can however, be very useful to connect to some servers that send small amounts of information. For example, there are many Internet Time Servers on the Internet that allow users to synchronize their computer clocks via the Internet. The service responds to time requests from any Internet client in several formats, including the DAYTIME, TIME, and NTP protocols. The simplest are those that send responses in ASCII data and you can extract the date and time information from the response ASCII string once you know the format.

You can search on the Internet for a suitable timeserver and use the TELNET program on your PC to access them to examine their display format. Most timeservers listen either on port 13 or port 123 so you need to specify the port number together with their IP address when sending the <TCPCONNECT> command.

Format: <TCPCONNECT [IP address:portno] of time server>

Response:

- none -	Successfully connected to the Modbus/TCP server of the specified IP address.
ERR:05-Prev Conn.ON	Another NS command has been executed and left the client socket opened but did not execute the PRINT #4 “</>” to close the client socket.
ERR:04-Not Connected	Failed to connect to the targeted server.

STATUS(3): This TBASIC function returns 1 if connected, or 0 if connection fails..

Description: Once a connection is made, you can then interact with the remote server using the PRINT #4 and INPUT$(4) command. You use the INPUT$(4) command to read CR-terminated text strings sent by the server. You can also send data to the remote server using the PRINT #4 command.

Example: Please refer to the “fnTCPconn1” custom function in the demo program: “TestEthernet.PC6” to see an example of how the PLC can connect to an NIST timer server and use the returned data to update the PLC’s real-time clock.

Note: Some NIST time servers have strict policy against abuse so you should avoid sending repeated request within a short period of time, otherwise further connections may be denied once you are considered to have violated their connection policy.

2.7 Web Service: Accessing PLC's data from MS Excel

The FServer provides an extremely useful feature called “Web Service”. You can actually use your web browser to access the Fx2424 internal data by specifying the following URL:

<IP Address: portno of FServer>/HOSTLINK/<Point-to-point hostlink command without “*”>

E.g. Please enter the following URL into your web browser URL address space:

192.168.1.5:9080/HOSTLINK/IR

You will see the following data appear on your browser screen:

IR01

“IR” is one of the many “host link commands” that allows a host computer to read or write to the PLC’s internal data space using ASCII strings. This particular command “IR” is for reading the PLC’s ID and in this case the PLC returns “01” by default. For more details on the list of host link commands, please refer to Chapter 15 of this manual.

Normally the host link commands are sent to the PLC via the serial port (as per all other PLC models produced by TRi). The Fserver, however, permits these host link commands to be sent using the HTTP protocol, which enables the Fx2424 to be easily accessible by enterprise software using what is known as “Web Query” methods. The enterprise software only needs to know the format of the host link command required to read the target data and then they can use their web query capability to query the PLC and extract the required data from the response string.

One example, which you can try immediately, is to use the Microsoft Excel 2000 (or later version) spreadsheet program. First, open a blank spreadsheet, then click on the “Data” menu and select “Get External Data” -> New Web Query, as shown below:

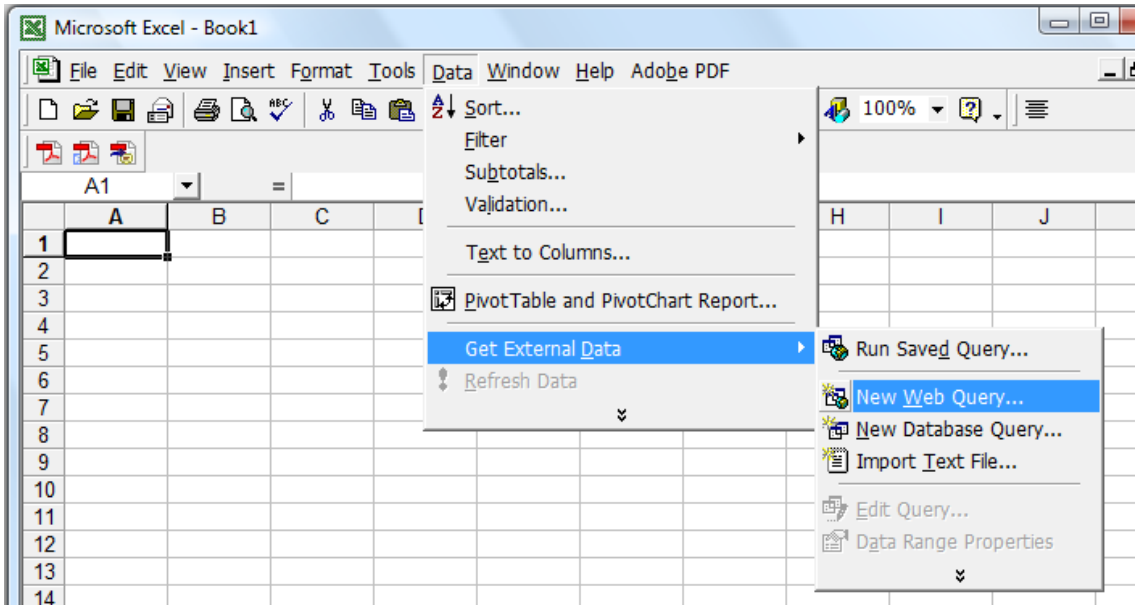


Figure 2.6.1

Next, please enter the text as shown in the following diagram and then click OK. This will command the Excel spreadsheet to send the web query string “RI00” to the Fx PLC that is connected to the network with IP address = 192.168.1.5 and port 9080. The query string “RI00” is for reading the status of 8-bit input channel #0 (which covers the logic states of input bit 1 to 8).

If the FServer is accessible by the PC from the network router, it will send the response data, which will be displayed on the selected spreadsheet cell where the New Web Query was defined earlier.

The response data shown on the cell could be RI00. The response data includes the command header “RI” as defined in the HostLink Command protocol described in Chapter 15. The data 00 indicates that none of the inputs 1 to 8 are currently turned ON.

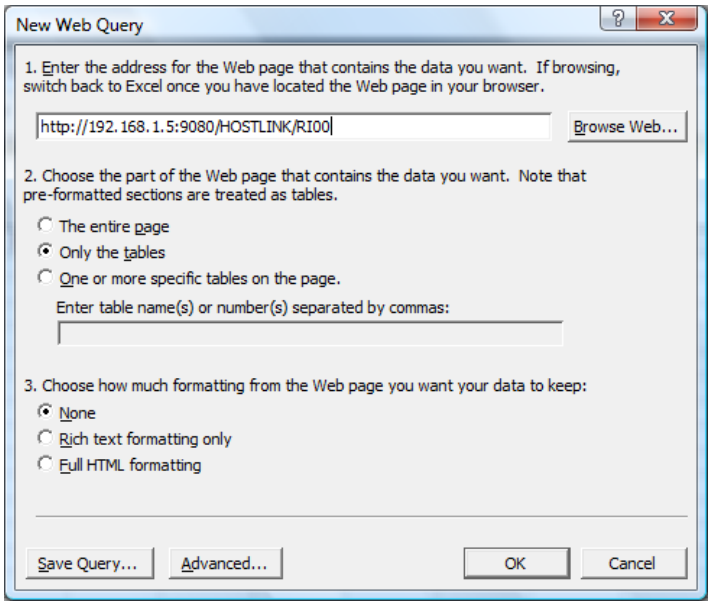


Figure 2.6.2

To see how the response data changes in response to the actual PLC’s input, please turn on some of PLC’s digital inputs 1 to 8, then right-click on the cell where the web query was defined and select te “Refresh Data” command. You should see a new “RIXX” string appear at the selected cell where “XX” is the hexadecimal representation of the 8 input bits 1-8. E.g. if only inputs 2 and 8 are turned ON, then the binary pattern is 1000 0010 which in hexadecimal form is 82 and the response string would therefore be “RI82”. You can then write an excel formula to extract the data “82” and use it for your other computation purpose. By using a different Host Link command, the Excel spreadsheet can read and write to the PLC’s internal data very easily.

Notes:

1. If you have enabled “Use Username/Password” for the FServer, you will be prompted by your Excel program to enter the Username and password before you can receive the response data.

2. We have provided a more complete Excel spreadsheet example “ExcelQuery.xls” which can be downloaded from: http://www.tri-plc.com/appnotes/F-series/ExcelQuery.xls The macro in this file converts the RIXX data it receives into ON/OFF indicators on the Excel Spreadsheet cells. Note that this spreadsheet file uses the “HEX2DEC” function that is not normally available when you first install the Excel program. But you can add it in by installing the “Analysis Toolpak”. Please search your Excel Help file for the specific method of adding in this toolpak as it may change from one version of Excel to another. On Excel 2000, you can click on the “Tools->Add Ins”, check the “Analysis Tookpak” check box and then click OK. MS Excel will automatically install the toolpak for you.

2.8 Accessing The PLC from Internet

2.8.1 Small Local Area Network Using Consumer Grade Network Router

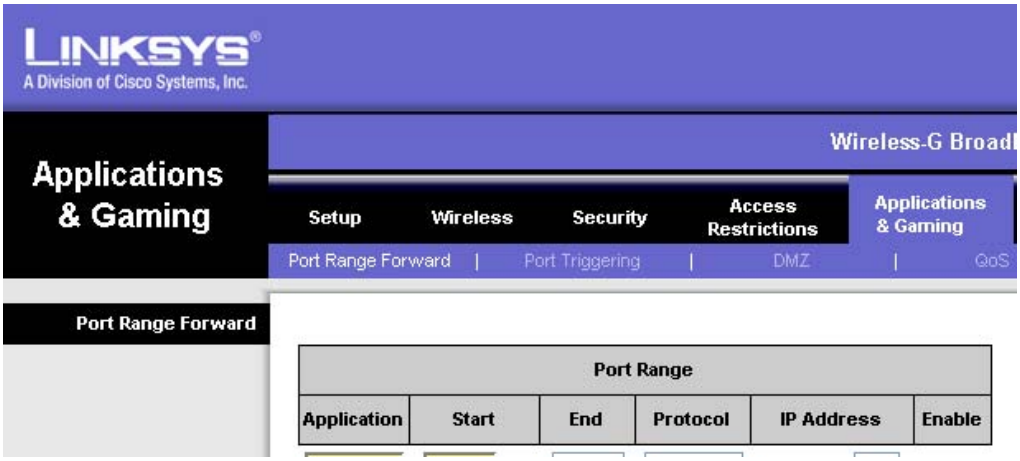
When you connect a F-series PLC to your home Ethernet router, the PLC would have joined a “private” local area network (LAN). It is accessible, through its private static IP address, by other devices on the same LAN as long as each device is on the same “subnet” (See section 2.1 for an explanation of subnet settings). The PLC is also able to access the Internet through the router because the router would translate a private TCP/IP packet sent from the PLC into a public TCP/IP packet out of the Internet and if there is any return data from the Internet meant for the PLC, the router would know that and automatically routes the return packet back to the PLC. The router performs what is known as “Network Address Translation (NAT)” and such routers are called NAT routers.

However, the same FServer and Modbus/TCP servers on the PLC are typically inaccessible from the public Internet. This is because the router has a built-in firewall that does not permit external TCP/IP packets from the public Internet to reach the devices on the private LAN. In other words, the NAT router allows the PLC outgoing access to the Internet but by default does not allow incoming access.

Most small NAT routers for home use such as those produced by Linksys, Netgear, D-Link or Belkin do allow you to configure the router to “open” and “forward” a specific port number to a specific device on the private network. For example, if your PLC static IP address is 192.168.1.5 and you wish to open its FServer port (9080) but not its Modbus/TCP port (502) to the public internet, you would configure your router such that it will forward the incoming TCP/IP packet destined for port number 9080 to the device at IP address 192.168.1.5. Once you have done that, you will then be able to access the FServer from the Internet using the router’s public IP address (this is typically assigned by the Internet Service Provider) and the port number 9080. However, the Modbus/TCP port is not accessible from the Internet since this port number is not opened and not mapped by the router.

You should read your router’s User’s Manual to find out how to configure the router to perform the “port forwarding” described above since each router model has a different user interface. For example, on the D-Link DI-624 router you configure the router by clicking on the “Advanced” tab and selecting “Virtual Server” from the router configuration page, as shown below:

On the Linksys WRT54G router, you would configure it from the “Applications & Gaming” menu under the “Port Range Forward” tab, as shown below:



2.8.2 Large Corporate Local Area Network

In the case of a medium to large corporate LAN, whether incoming and outgoing TCP/IP packets are allowed to go through the corporate firewall is entirely decided by the System Administrator according to the company’s security policy. Most corporate LANs would not allow incoming packets from reaching an internal server until the System Administrator has given the permission to do so. Some company’s network may not even allow devices such as the PLC to open a connection to the Internet to access external data. If your application requires the PLC to access the Internet or to be accessible from the Internet, then you would need to consult your system administrator on the required procedure.

2.9 Installing a Control Web Page Into the Fx PLC

The Fx PLC web server space can host up to 1.7M bytes of data files which can be HTML, JPG, JS (JavaScript) files etc. Thanks to its support of the "web services" commands, a programmer can create its own sophisticated control webpages using only standard HTML and JavaScript .

TRi has created an example control webpage which can be downloaded from:

http://www.tri-plc.com/download/webapp/webapp02.zip

When you have successfully unzipped all the files from the Webapp02.zip into your hard disk, you will need to transfer the files to the PLC's web server using a FTP client program. The following sections describe how to use the free FileZilla FTP program to transfer these files and section 2.10 will describe how to customize the control web pages.

NOTE: The "M.JS" JavaScript file used with the new preloaded 0.HTM file is no longer provided in the new web app download because it is loaded remotely. In the past, M.JS had been stored inside the PLC and was available directly for editing. However, now the user modifications in the HTML file are more expanded and in order to save space for additional files as well as protect against accidental modification of the M.JS file, it is now stored outside PLC memory.

If you are an experienced web programmer, or you plan to outsource web programming services to customize your PLCs web interface, then you may obtain a copy of this file for more advanced editing by writing to support@triplc.com and providing your purchase references (purchaser's name, company name, invoice # etc).

Of course, you may also like to build your own HTML and JavaScript/Jquery control structure, which is entirely feasible for any experience web programmer.

2.9.1 Installing the FileZilla program

First please download the FileZilla client from: http://filezilla-project.org

Once you have gone to the web page from the above link, you will need to select the client version to download. Then you need to select the software version for your computers platform, typically the Windows version, and download it to your computer. This is the installation setup file, which you will need to execute after it has downloaded. Please perform a default installation by following the installation steps.

2.9.2 Configure FileZilla to Communicate with the Fx2424

After installing the FileZilla client, please open it from the start menu or desktop icon and then go to the "File" menu and select "Site Manager". The following menu will then pop up:

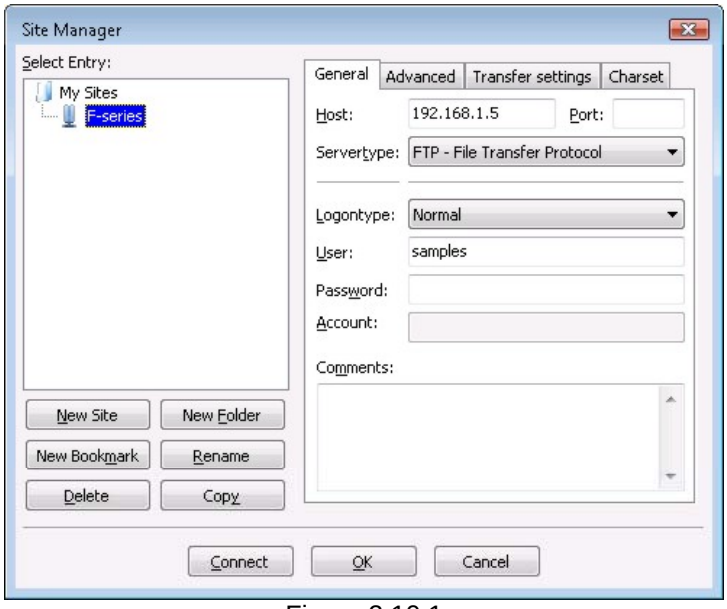


Figure 2.10.1

Under the default “General” tab, you will need to enter the Host IP address and configure the “Logontype”.

The Host should be the IP address of the Fx2424 (default IP is shown in the example). The Port can be left blank and the Servertype can be left as the default as shown.

The Logon type should be set to Normal and the User and Password should be set to the same as for the login to the F-server.

Nothing else needs to be configured in this area. Next go to the “Transfer settings” tab.

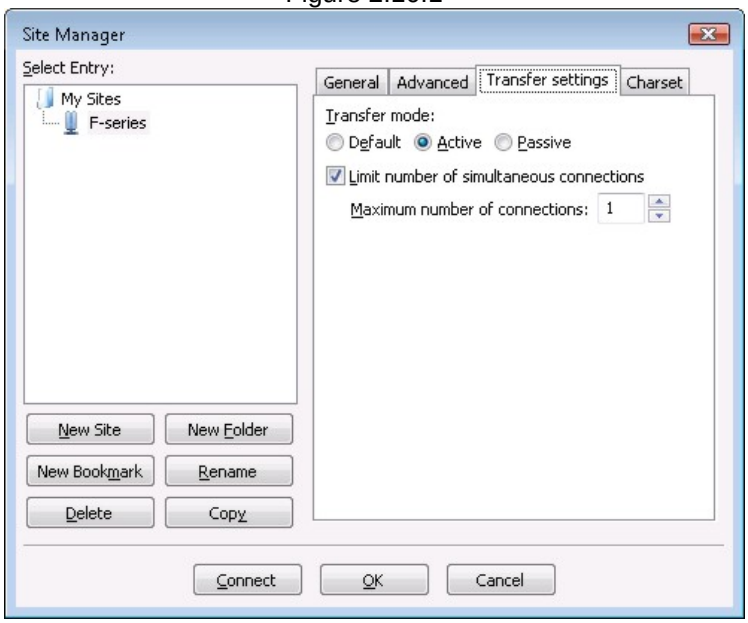


Figure 2.10.2

Under the “Transfer settings” tab you need to set the Transfer mode to “Active” and check the box to Limit number of simultaneous connections. The maximum number of connections should be 1.

This is everything that needs to be configured, so you can now click on connect.

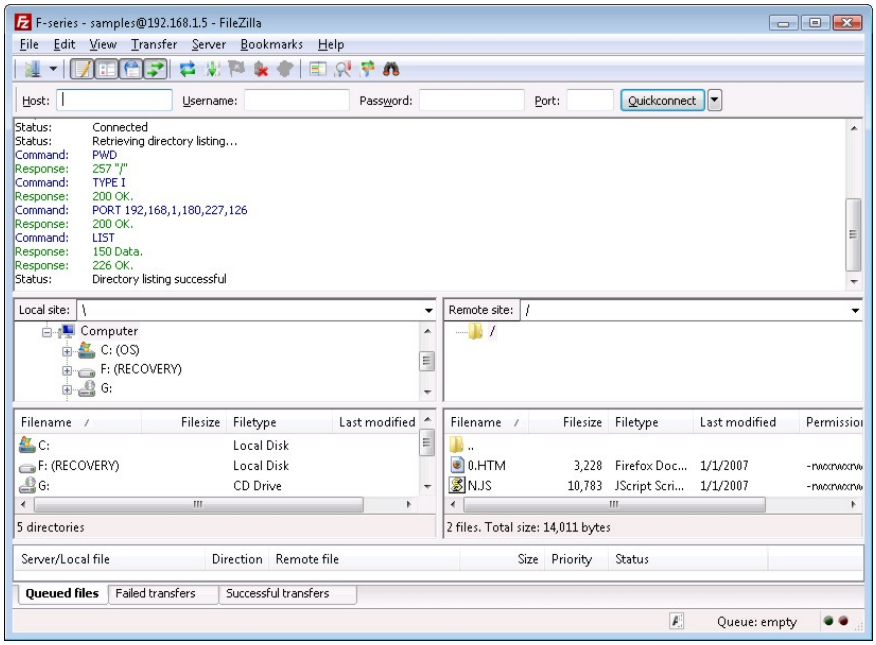


Figure 2.10.3

If FileZilla has made a successful connection, the status should be: "Connected", and "Directory listing successful".

Also, you should be able to see the preloaded files in the remote site directory, which is the bottom right window.

If you are not able to connect to the PLC's web server or if you can't get a successful directory listing, then you should first double check the settings you have configured for FileZilla compared to the settings configured in the F-Server. The next thing to check is your Windows firewall, which must allow the PLC to make a connection back to the PC (due to the use of active transfer settings). You can disable the firewall temporarily to check if it is affecting your FileZilla connection. If you have confirmed that disabling the firewall allows you to successfully connect to the PLC server and view the directory listing, then you can add the PLC's IP address as well as FileZilla program to the firewall exception list so that you can leave your firewall enabled if you wish.

To add FileZilla to the Windows firewall exception list, you will need to open the Windows firewall from your network settings configuration area. Then you will need to click on "Change settings", which will open up a new window. You will need to go to the "Exceptions" tab in the new window and look for FileZilla in the list of programs. If you see FileZilla in the list, then you will need to check the box beside it to add it to the exception list. Otherwise, you will need to manually add it to the list by clicking "Add program" and searching for FileZilla in its installation directory. Here is a picture of the Windows Firewall status window and the Change settings window with the Exceptions tab:

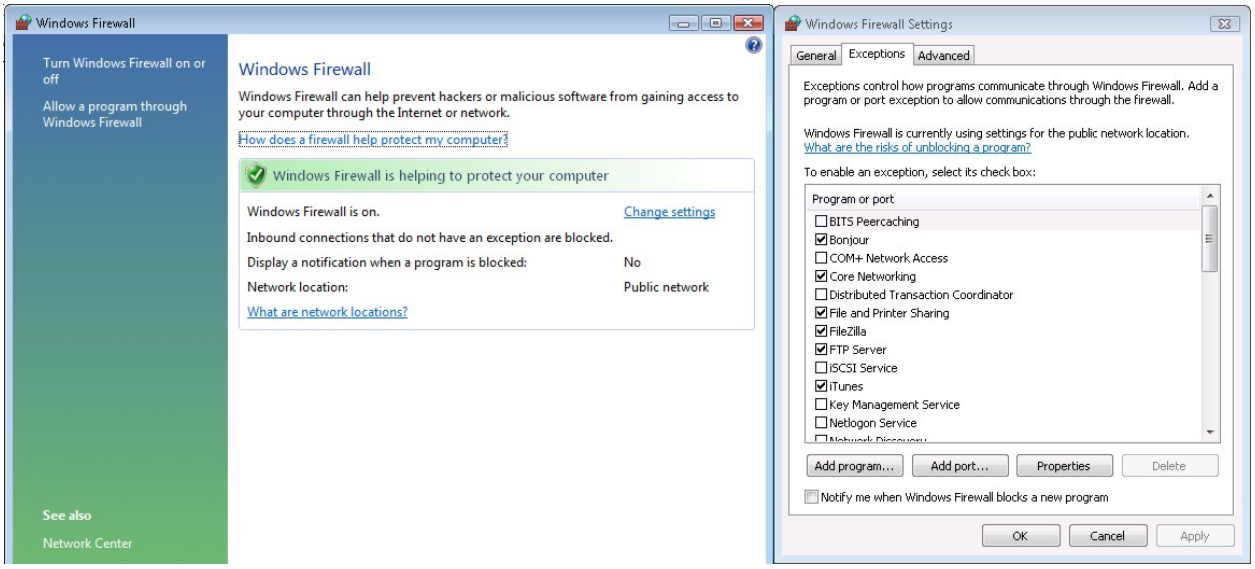


Figure 2.10.4

2.9.3 Transferring and Retrieving Files from the PLC Web Server

Now that you have a connection to the PLC's web server and can view the directory listing, you should be able to see any files that have been preloaded into the PLC.

Downloading Files

The first thing to do is to download the preloaded files from the web server and store them somewhere on your computer. To do this you will need to open up the folder on your computer that you want to store the files in. Then you can drag the files from the web server via FileZilla into your destination folder. It is best to make a copy of these files as a backup on your computer so that you have the original copies available in case you need them.

Uploading Files

When you are ready to transfer your own web pages or applet to the PLC, you just need to drag it from the folder it is saved to on your computer into the bottom right window of FileZilla where the current files are shown. If you didn't change the filename, you can tell the new file to overwrite the old file. Otherwise, you will need to manually delete the old file from the PLC's web server in FileZilla by right clicking on it and selecting Delete.

2.9.4 Troubleshooting FileZilla File Transfer Problems

1) FileZilla appears to have connected to the PLC's FTP server, but it cannot list the directory or transfer any file to the PLC.

The reason almost always have something to do with the PC's software firewall. You need to configure the Windows Firewall (and any software anti-virus firewall on your PC) to allow incoming connections to the FileZilla program. You can try to disable the firewall temporarily to test the connection and if does work, you can then be sure that it is the firewall configuration issue that need to be resolved.

The following paragraphs explain how the firewall can affect the FTP communications for those who are interested:

The File Transfer Protocol is unique in that it requires **two** socket connections between two devices that are communicating via FTP. When FileZilla is connected to the PLC's FTP server on port 21, it establishes a "command" channel and it is through this command channel that FTP commands are being sent.

However, once the command channel is established (FileZilla is "connected" to the FTP Server), a second socket connection (known as the "data channel") needs to be made between FileZilla and the PLC. All data, such as the directory information and content of any files to be transferred between FileZilla and the PLC will need to go through the data channel. They are two possible ways of establishing this data channel, one is called the "Active Transfer" and the other is known as "Passive Transfer".

FileZilla is able to operate in either transfer mode, but the PLC FTP Server can only operate in "Active Transfer" mode. "Active Transfer" mode requires that the client (FileZilla) provides a listening socket for the server (PLC) and the server (PLC) will then try to connect to this socket to establish the data connection. So if the FileZilla program sits behind the software firewall and no exception has been configured, then the PLC FTP Server will not be able to make a connection to the FileZilla data socket because it is blocked by the firewall and the connection will therefore fail. This explains why FileZilla seemingly able to connect to the PLC but yet is unable to send any data or list the directory – because there is no data channel connection for it to do so.

However, sometimes the problem could persist after disabling all firewalls and in that case the only option is to revert to an older version of FileZilla, and then transfer the files by doing a right-click download or right-click upload (if drag and drop doesn't work). You can do this as follows :

a. Download the FileZilla client version 3.1.5.1 from http://www.oldapps.com/filezilla.php

b. Remove the current version and then install version 3.1.5.1

c. Connect to the PLC

d. There should be two directories (same as always) : local site (your PC) and remote site (the PLC). You should see your files in the PLC directory.

e. Open the folder in the local site that you would like to save the files to.

f. Right click on a file in the PLC directory such as 0.HTM and select download. You should see the file in the folder on your local site.

g. To move files from the PC to the PLC, open the folder in the local site where the file is located and right click it to select upload. You should see it in the PLC or you will be asked to overwrite it if the same file name is there already.

2) FileZilla Can Connect The First Time But Could Not Re-connect After Time-Out

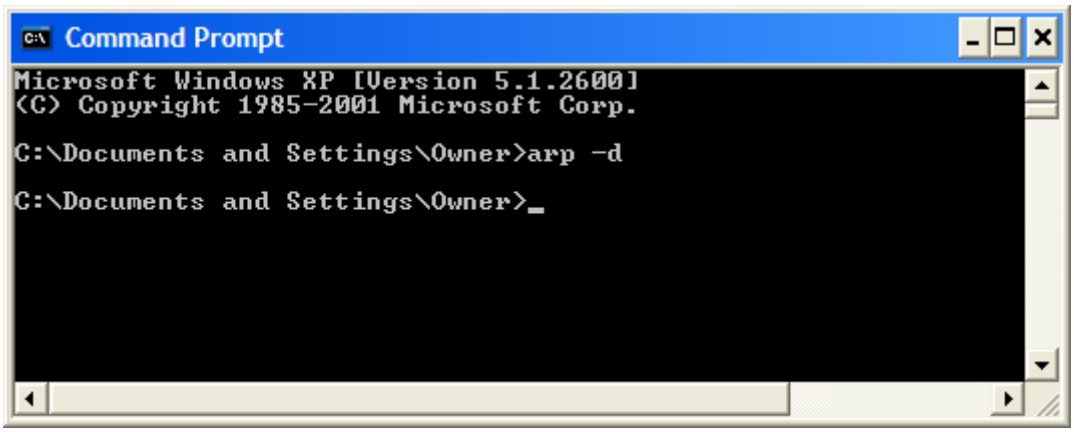
FileZilla is normally setup to time out after 1 minutes of no activity. However, Since the PLC's FTP server can only handle a single FTP connection at a time, it is good to avoid letting FileZilla time out due to no activity. This is because when FileZilla times out it cutoff the connection to the PLC. But if for whatever reason the PLC were to miss the disconnection information it will be left in a "half-open" state and will not be able to accept a new connection. When this happens and if you try to reconnect to the PLC again after the connection has been dropped by FileZilla, you will most likely be unable to connect to the PLC again until after the PLC time out its FTP connection due to no activity. The only quick fix is to power-on reset the PLC so that it starts up fresh and can accept new connection again.

You can set the FileZilla timeout settings to a larger number (e.g. 300 seconds) using the FileZilla's "Edit->Settings" menu item so that it will not time-out automatically too quickly. You should always select "disconnect" after a file transfer to properly close the connection and then the PLC will be properly disconnected and is ready to accept new connection.

3) Problem Using FileZilla To Transfer Files to Multiple PLCs Configured To The Same Default IP Address Even Though Only One PLC Is Connected To The Router At A Time.

When you connect FileZilla to a PLC with a particular IP address (e.g. 192.168.1.5) for the first time, Windows will memorize the MAC-ID address of that PLC and associate it with this IP address in its "ARP cache" (ARP = Address Resolution Protocol). So when you power off one PLC and immediately plug another PLC with the same IP address to the network, then Windows will pass to FileZilla the MACID of the **PREVIOUS** PLC instead of the new PLC using data in its ARP cache memory. When FileZilla tries to connect to the right IP address but a wrong MACID, it of course would not be able to connect properly.

In such a scenario, you must clear the Windows ARP cache first using a command prompt as follow:



The "**arp –d**" command tells windows to delete its ARP cache data. So when you use FileZilla to connect to a new PLC Windows will do the necessary to properly connect to the new PLC that has a different MACID from the previous PLC.

Note that the same issue applies to using I-TRiLOGI to transfer program to multiple PLCs that are all set to the same default IP address, even though only one PLC will be connected to the network at a time. You will need to clear the ARP cache after every transfer to avoid problem. Or else you have to wait for windows to expire its ARP cache data before connecting the next PLC.

2.10 Accessing and Customizing the HTML Web Interface for Control and Monitoring

The second generation of Web control application is now available free for any TRi PLC user to download and install into their PLC. In the previous section (2.9 Installing a Control Web Page Into the F-series PLC) you were shown how to transfer html files to the FServer.

Click on the following link to download this new web application.

http://www.tri-plc.com/download/webapp/webapp02.zip

The "Webapp02.zip" file you downloaded above (or in Section 2.9) for Fx PLCs contains the following:

- Four sample control web pages (0-C001-02.HTM, 0-C002-02.HTM, 0-C002-02-NoLCD.HTM, and 0-C003-02.HTM)
- A readme file (Readme.txt)
- A download link (UserGuide.htm) for the quickstart modification guide
- PLC program files for TRiLOGI 6 and 7 (webAppTest.PC6 and webAppTest.PC7) that can be loaded in the PLC

The user guide describing how you can modify the control webpage can be downloaded directly from:

http://www.triplc.com/documents/WebApp_UserGuide.pdf

2.10.1 Next Generation of Super PLC Web Control

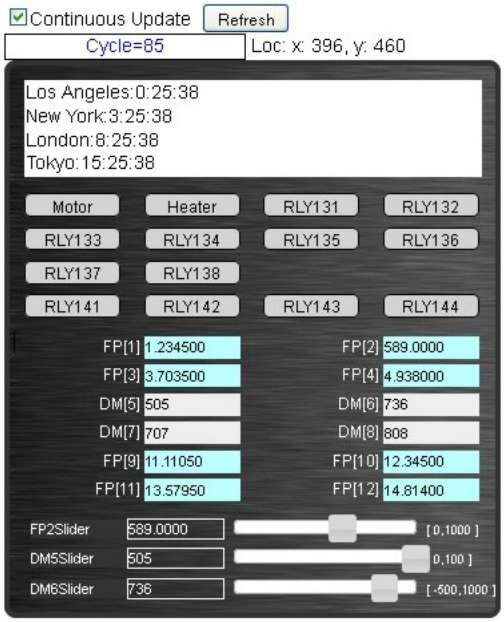
These web files can be accessed from any standard web browser that supports AJAX technology (Internet Explorer, Firefox, Chrome, Safari) locally as long as your Fx2424 PLC is connected to the LAN (local area network) or from anywhere in the world as long as your Fx2424 PLC is connected to the Internet. Even the iPhone, Android phones, and other smartphones that support AJAX can be used to access these web pages, which means your application can be controlled and monitored from anywhere while on the go.

This new version makes use of the newer Jquery UI and no longer utilizes the M.JS JavaScript file. Instead it calls the Jquery file remotely (no longer stored inside the PLC).

Either way, the web files are only accessible from standard browsers that support AJAX technology. However, the HTML files are designed to be easily modified by anyone, even if you don't have any programming experience. The purpose is to be able to easily customize the web page layout for your specific application by defining some label names and a background image without having to worry about the programming required to interface to the PLC, which is already taken care of.

The basic HTML file "0.HTM" is the file where simple modifications can be made. All of the provided HTML files are actually 0.HTM because only the first character of the name is used, so 0-C001-02.HTM becomes 0.HTM.

The first release of this new Web control application accepts up to: 16 I/O buttons linked to Relay #128 to #144; 16 floating point FP[] variables; 16 DM[] variables, and 8 sliders that can be attached to display and/or control any of the FP[1] to FP[16] or DM[1] to DM[16] variables. Any of these control elements can be selectively displayed and their position, size and color are all user customizable using a simple text editor, with no programming required at all. See below for a screenshot of 0-C001-02.HTM.



Refer to the user guide linked above for more information on the available user modifications.

2.10.2 Advanced Customization

The “M.JS” JavaScript file used with the new preloaded 0.HTM file is no longer provided in the new web app download because it is loaded remotely. In the past, M.JS had been stored inside the PLC and was available directly for editing. However, now the user modifications in the HTML file are more expanded and in order to save space for additional files as well as protect against accidental modification of the M.JS file, it is now stored outside PLC memory.

If additional modifications need to be made beyond what can be done in the HTML file, then the following two options are available:

1) Enlist Design Services

Contact our Solution Partner, Sparrow Design LLC, for browser/app control design services.

2) Request the JavaScript Files.

If you are an experienced web programmer, or you plan to outsource web programming services to customize your PLCs web interface, then you may obtain a copy of this file for more advanced editing by writing to support@triplc.com and providing your purchase references (purchaser’s name, company name, invoice # etc).

Of course, you may also like to build your own HTML and JavaScript/Jquery control structure, which is entirely feasible for any experience web programmer.

Users only need to refer to chapter 16 in this manual for the list of Hostlink commands to be sent in web query form (refer to section 2.7) in order to implement web based data exchange with TRi PLCs.

Chapter 3 I/O and Internal Relays Programming

3 PROGRAMMING I/O AND INTERNAL RELAYS

3.1 Introduction

The Fx2424 will have a certain no. of physical digital inputs and outputs depending on the particular model, but all will have 512 internal relays available in both ladder logic and BASIC.

3.2 Programming DIO with Ladder Logic

The physical I/O and internal relays can be programmed in ladder logic in a few simple steps.

3.2.1 For Physical I/O

1. Edit the label names
2. Place the input contact(s) into the ladder logic circuit
3. Place the output coil at the end of the ladder logic circuit

3.2.2 For Internal Relays (Non-Latching)

1. Edit the label names
2. Place the relay contact(s) into the ladder logic circuit
3. Place the relay coil at the end of the ladder logic circuit

3.2.3 For Internal Relays (Latching)

1. Edit the label names
2. Place the activating input/relay contact into the ladder logic circuit
3. Place the latching relay in parallel with the activating contact
4. Place the relay coil at the end of the ladder logic circuit

3.2.4 Programming Examples:

3.2.4.1 Example 1 – Editing Label Names

The Digital I/O can be named by selecting “I/O Table” from the “Edit” menu and choosing the particular digital I/O that you want to name. In Figure 3.1, physical input #1 is being named “Input1”.

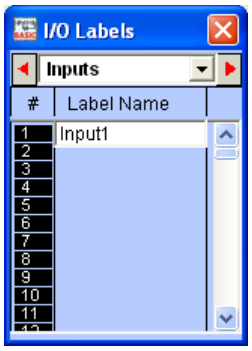


Figure 3.1: I/O Table

3.2.4.2 Example 2 – Creating a Simple Ladder Logic Circuit

You can place components in the circuit by clicking in the green area to the right of the red arrow, as shown in Figure 3.2 below. This will bring up the component tool bar in the gray area above the green circuit area.

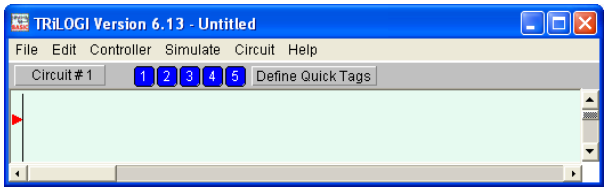


Figure 3.2: Creating Ladder Circuit

Once the component toolbar is shown, you can place your input/relay contact by selecting the #1 component from the toolbar and then selecting the digital input from the I/O Table. The contact will then be automatically placed in the ladder logic circuit. The same can be done for the output coil by selecting the #7 component from the toolbar and then selecting an output that has been entered into the I/O Table. After selecting one input and one output, the ladder logic circuit should like something like Figure 3.3, below:

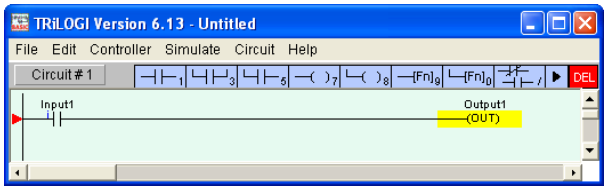


Figure 3.3: Completed Ladder Circuit

3.2.4.3 Example 3 – Creating a Latching Relay Circuit

The first part of the circuit follows the same procedure as the previous example, except that the #7 coil should be a Relay coil. So it should look similar to the circuit in Figure 3.3. The next part requires a parallel contact to be added to the Input1 contact. This is done by selecting the Input1 contact (or whichever contact was used) and then adding the #3 contact , as shown in Figure 3.4 below.

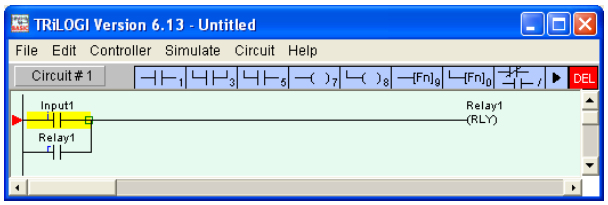


Figure 3.4: Completed Latching Circuit

3.3 Programming DIO in a Custom Function

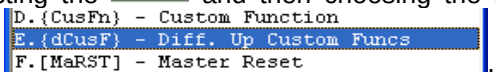
In order to program digital I/O or anything in a custom function, a custom function must be created in the I/O Table and added in a ladder logic circuit. Custom functions act the same way as coils in ladder logic, in that that they need a contact to activate them. Once they are activated, the code inside them will execute.

To create a custom function circuit, follow these 3 steps:

1. Edit the name of the custom function in the I/O Table
2. Place the activating contact in the ladder logic circuit
3. Place the custom function at the end of the circuit

3.3.1 Editing Label Names:

This is the same as for the digital I/O, except that the I/O table window needs to be scrolled to the custom function area for editing custom function names.

Placing the custom function in the circuit is done the same way as other ladder logic contacts and coils, by selecting the and then choosing the Differential custom function {dCusF} from the pop-up window  . The circuit should look something like below:

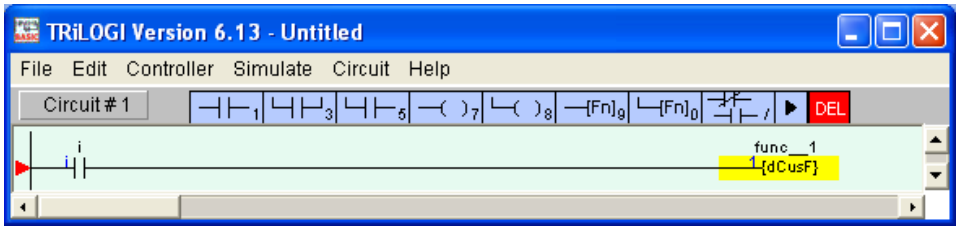
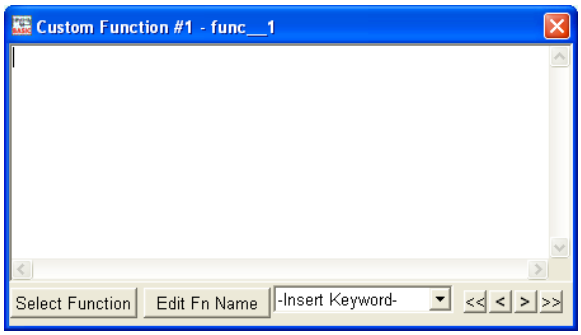


Figure 3.5: Circuit with custom function {dCusF}

3.3.2 Controlling I/O from Custom Functions:

An empty custom function looks like this:



TBASIC code is entered into the custom function, which allows the possibility of total control of all of the PLCs functions and hardware.

There are 7 TBASIC functions available to control all of the digital I/O, which are:

1. SETIO *labelname*
2. CLRIO *labelname*
3. TOGGLEIO *labelname*
4. TESTIO (*labelname*)
5. SETBIT *v,n*
6. CLRBIT *v, n*
7. TESTBIT (*v, n*)

Each function has its own advantage depending on what needs to be done to a digital I/O. Each of these functions is explained in the programmer's reference manual, which should be referred to for further information. Here are some examples of how to control digital I/O using these functions.

3.3.3 Example 1 – Turn on/off an Output

This can be done using both the SETBIT *v,n* / CLRBIT *v,n* command and the SETIO *labelname* / CLRIO *labelname* command.

1. Using SETBIT *v,n* / CLRBIT *v,n*

```
SETBIT OUTPUT[1], 0      'This will turn on the first output using the output[] register
CLRBIT OUTPUT[1], 7      'This will turn off the 8th output using the output[] register
```

2. Using SETIO *labelname* / CLRIO *labelname*

```
SETIO out1       'This will turn on the output out1
CLRIO out5       'This will turn off the output out5
```

In this case, out1 and out5 would need to be entered in the I/O Table as an output. Otherwise, there will be a compilation error.

3.3.4 Example 2 – Toggle an Output

```
TOGGLEIO light      'This will change the output, light, from off to on or on to off
```

The output, light, would need to be entered into the I/O Table as an output as well and could represent any desired output.

3.3.5 Example 3 – Test the Status of an Output

This can be done using both the TESTBIT (*v, n*) and TESTIO (*labelname*) command.

Using TESTBIT (*v, n*)

```
X = TESTBIT (INPUT[2], 1)      'status of input #10 (on = 1, off = 0) is stored in variable X
```

Using TESTIO (*labelname*)

```
X = TESIO (button)  'status of input button (defined in the I/O table) is stored in variable X
```

Chapter 4 Timers, Counters & Sequencers

4 TIMERS, COUNTERS AND SEQUENCERS

4.1 Introduction

4.1.1 Timer Coils

A timer is a special kind of relay that, when its coil is energized, must wait for a fixed length of time before closing its contact. The waiting time is dependent on the "Set Value" (SV) of the timer. Once the delay time is up, the timer's N.O. contacts will be closed for as long as its coil remains energized. When the coil is de-energized (i.e. turned OFF), all the timer's N.O. contacts will be opened immediately. However, if the coil is de-energized before the delay time is up, the timer will be reset and its contact will never be closed. When the last aborted timer is re-energized, the delay timing will restart and use the SV of the timer rather than continue from the last aborted timing operation.

4.1.2 Counter Coils

A counter is also a special kind of relay that has a programmable Set Value (SV). When a counter coil is energized for the first time after a reset, it will load the value of SV-1 into its count register. From there on, every time the counter coil is energized from OFF to ON, the counter decrements its count register value by 1. Note that the coil must go through an OFF to ON cycle in order to decrement the counter. If the coil remains energized all the time, the counter will not decrement. Hence, a counter is suitable for counting the number of cycles an operation has gone through. When the count register hits zero, all of the counter's N.O. contacts will be turned ON. These counter contacts will remain ON regardless of whether the counter's coil is energized or not. To turn OFF these contacts, you have to reset the counter using a special counter reset function [RSctr].

4.1.3 Sequencers

A sequencer is a highly convenient feature for programming machines or processes that operate in fixed sequences. These machines operate in a fixed, clearly distinguishable step-by-step order, starting from an initial step, progressing to the final step, and then restarting from the initial step again. At any moment, there must be a "step counter" to keep track of the current step number. Every step of the sequence must be accessible and can be used to trigger some action, such as turning on a motor or solenoid valve, etc. As an example, a simple Pick-and-Place machine that can pick up a component from point 'A' to point 'B' may operate as follow:

Step #	Action
0	Wait for "Start" signal
1	Forward arm at point A
2	Close gripper
3	Retract arm at point A
4	Move arm to point B
5	Forward arm at point B
6	Open gripper
7	Retract arm at point B
8	Move arm to point A

4.2 Programming timers and counters on Ladder Logic

The timers and counters can be programmed in ladder logic in a few simple steps.

4.2.1 For Timers

1. Edit the label names
2. Place the input contact(s) into the ladder logic circuit
3. Place the timer coil at the end of the ladder logic circuit
4. Place the timer contact in one or more ladder logic circuits

4.2.2 For Counters

1. Edit the label names
2. Place the input contact(s) into the ladder logic circuit
3. Place the counter coil at the end of the ladder logic circuit
4. Place the counter contact in one or more ladder logic circuits (optional)

4.2.3 Example 1 – Creating a Simple Timer Circuit in Ladder Logic

You can place components in the circuit by clicking in the green area to the right of the red arrow, as shown in Figure 4.1 below. This will bring up the component tool bar in the gray area above the green circuit area.

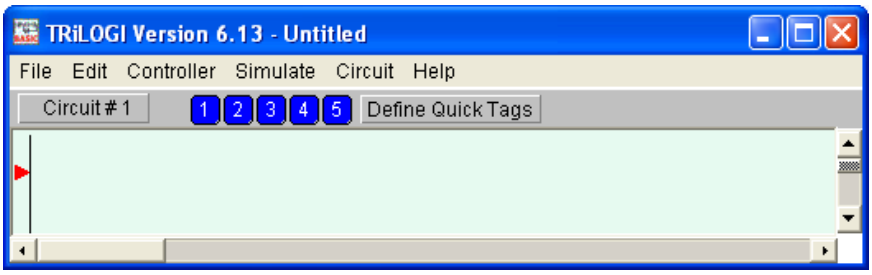


Figure 4.1: Creating Timer Circuit

Once the component toolbar is shown, you can place your activating contact by selecting the #1 component from the toolbar and then selecting the activating contact from the I/O Table. The contact will then be automatically placed in the ladder logic circuit. The same can be done for the timer coil by selecting the #7 component from the toolbar and then selecting a timer that has been entered into the I/O Table. Then a timer contact needs to be added as an input to a ladder logic circuit. This contact will activate once the timer counts down. This could be used to turn on an output a certain amount of time after the timer coil is activated. Placing a timer contact in a circuit is the same as placing any contact in a ladder circuit, except that the corresponding timer should be selected from the “Timers” section of the I/O table.

After creating a ladder circuit that contains one input and one timer output and another ladder circuit that contains one timer contact and one output, the ladder logic circuit should like something like Figure 4.2, below:

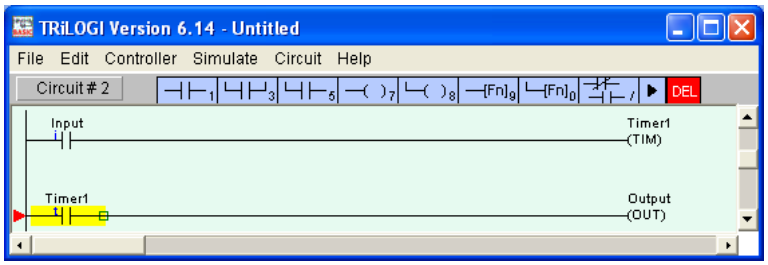


Figure 4.2: Completed Timer Circuit

4.2.4 Example 2 – Creating a Simple Counter Circuit in Ladder Logic

You can place components in the circuit by clicking in the green area to the right of the red arrow, as shown in Figure 4.3 below. This will bring up the component tool bar in the gray area above the green circuit area.

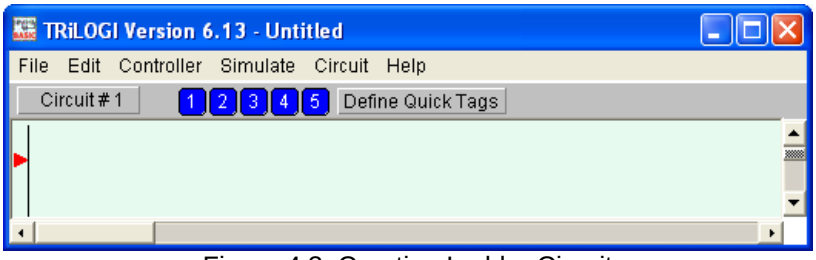


Figure 4.3: Creating Ladder Circuit

Once the component toolbar is shown, you can place your activating contact by selecting the #1 component from the toolbar and then selecting the activating input from the I/O Table. The contact will then be automatically placed in the ladder logic circuit. The same can be done for the counter coil by selecting the #7 component from the toolbar and then selecting a counter that has been entered into the I/O Table. Then a counter contact needs to be added as an input to a ladder logic circuit. This contact will activate once the counter counts down. This could be used to turn on an output after a certain count is reached. Placing a counter contact in a circuit is the same as placing any contact in a ladder circuit, except that the corresponding counter should be selected from the “Counters” section of the I/O table.

After creating a ladder circuit that contains one input and one counter output and another ladder circuit that contains one counter contact and one output, the ladder logic circuit should like something like Figure 4.4, below:

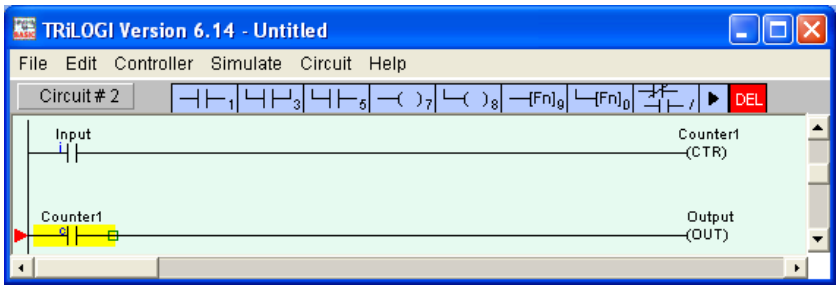


Figure 4.4: Completed Counter Circuit

4.3 Programming timers and counters in Custom Function

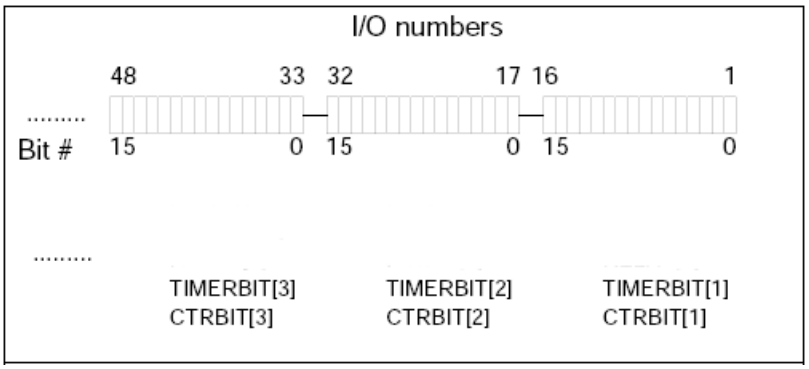
4.3.1 Timers and Counters Present Values

The present values (PV) of the 64 timers and 64 counters in the PLC can be accessed directly as system variables:

```
timerPV[1] to timerPV[64], for timers' present value
ctrPV[1] to ctrPV[64], for counters' present value
```

4.3.2 Inputs, Outputs, Relays, Timers and Counters Contacts

The bit addressable I/Os elements are organized into 16-bit integer variables TIMERBIT[n] and CTRBIT[n] so that they may be easily accessed from within a CusFn. These I/Os are arranged as shown in the following diagram:



4.3.3 Changing The Timer and Counter Set Values in a Custom Function

You can use the SetTimerSV and SetCtrSV functions to change the Set Value (SV) for a timer and counter respectively. An example of this is shown below:

```
SetTimerSV 1,500  'Define Timer #1 to have a Set Value of 500
SetCtrSV 10,1000  'Define Counter #10 to have a Set Value of 1000
```

4.3.4 Controlling a Timer or Counter in a Custom Function

You can activate a timer or a counter directly from within a custom function simply by assigning their present value counter to a desired value.

E.g. To start a 50 seconds timer:

```
TIMERPV[2] = 500  ' Timer #2 will time out 50.0 seconds later.
```

E.g. To decrement a counter or a sequencer:

```
CTRPV[10] = CTRPV[10] - 1 ' Counter #10 is decremented by 1.
```

4.4 Programming Sequencers on Ladder Logic

4.4.1 Introduction

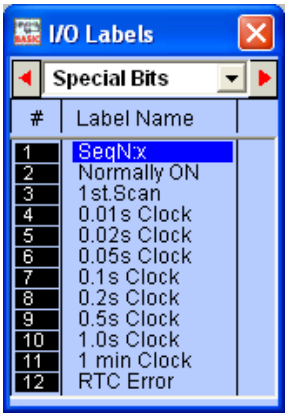
TRiLOGI Version 6+ supports eight sequencers of 32 steps each. Each sequencer uses one of the first eight counters (Counter #1 to Counter #8) as its step counter. Any one or all of the first eight counters can be used as sequencers "Seq1" to "Seq8".

To use a sequencer, first define the sequencer name in the Counter table by pressing the <F2> key and scroll to the Counter Table. Any counter to be used as sequencer can only assume label names "Seq1" to "Seq8" corresponding to the counter numbers. For e.g. if Sequencer #5 is to be used, Counter #5 must be defined as "Seq5". Next, enter the last step number for the program sequence in the "Value" column of the table.

A circuit that uses the special function "Advance Sequencer" [AVSeq] will need to be constructed. The first time the execution condition for the [AVseq] function goes from OFF to ON, the designated sequencer will go from inactive to step 1. Subsequent changes of the sequencer's execution condition from OFF to ON will advance (increment) the sequencer by one step. This operation is actually identical to the [UPctr] instruction.

The upper limit of the step counter is determined by the "Set Value" (SV) defined in the Counter table. When the SV is reached, the next advancement of sequencer will cause it to overflow to step 0. At this time, the sequencer's contact will turn ON until the next increment of the sequencer. This contact can be used to indicate that a program has completed one cycle and is ready for a new cycle.

Accessing individual steps of the sequencer is extremely simple when programming with TRiLOGI. Simply create a "contact" (NC or NO) in ladder edit mode. When the I/O window pops up for you to pick a label, scroll to the "Special Bits" table as follow:



The "Special Bits" table is located after the "Counters" table and before the "Inputs" table. Click on the "SeqN:x" item to insert a sequencer bit. You will be prompted to select a sequencer from a pop-up menu. Choose the desired sequencer (1 to 8) and another dialog box will open up for you to enter the specific step number for this sequencer.

Each step of the sequencer can be programmed as a contact on the ladder diagram as "SeqN:X" where N = Sequencers # 1 to 8 and X = Steps # 0 - 31.

e.g. Seq2:4 = Step #4 of Sequencer 2.
Seq5:25 = Step #25 of Sequencer 5.

Although a sequencer may go beyond Step 31, if you define a larger SV for it, only the first 32 steps can be used as contacts to the ladder logic. Hence it is necessary to limit the maximum step number to not more than 31.
Quite a few of the ladder logic special functions are related to the use of the sequencer. These are described below:

4.4.2 Advance Sequencer - [AVseq]

Increment the sequencer's step counter by one until it overflows. This function is identical to (and hence interchangeable with) the [UpCtr] function.

4.4.3 Resetting Sequencer - [RSseq]

The sequencer can also be reset to become inactive by the [RSseq] function at any time. Note that a sequencer that is inactive is not the same as sequencer at Step 0, as the former does not activate the SeqN:0 contact. To set the sequencer to step 0, use the [StepN] function described next.

4.4.4 Setting Sequencer to Step N - [StepN]

In certain applications it may be more convenient to be able to set the sequencer to a known step asynchronously. This function will set the selected sequencer to step #N, regardless of its current step number or logic state. The ability to jump steps is a very powerful feature of the sequencers.

4.4.5 Reversing a Sequencer

Although not available as a unique special function, a sequencer may be stepped backward (by decrementing its step-counter) using the [DNctr] command on the counter that has been defined as a sequencer. This is useful for creating a reversible sequencer or for replacing a reversible "drum" controller.

4.4.6 Program Example

Assume that we wish to create a running light pattern which turns on the LED of Outputs 1 to 4 one at a time every second in the following order: LED1, LED2, LED3, LED4, LED4, LED3, LED2, LED1, all LED OFF and then restart the cycle again. This can be easily accomplished with the program shown in Figure 4.5.

The 1.0s clock pulse bit will advance (increment) Sequencer #2 by one step every second. Sequencer 2 should be defined with Set Value = 8. Each step of the sequencer is used as a normally open contact to turn on the desired LED for the step. A "Stop" input resets the sequencer asynchronously. When the sequencer counts to eight, it will become Step 0. Since none of the LEDs are turned ON by Step 0, all LEDs will be OFF.

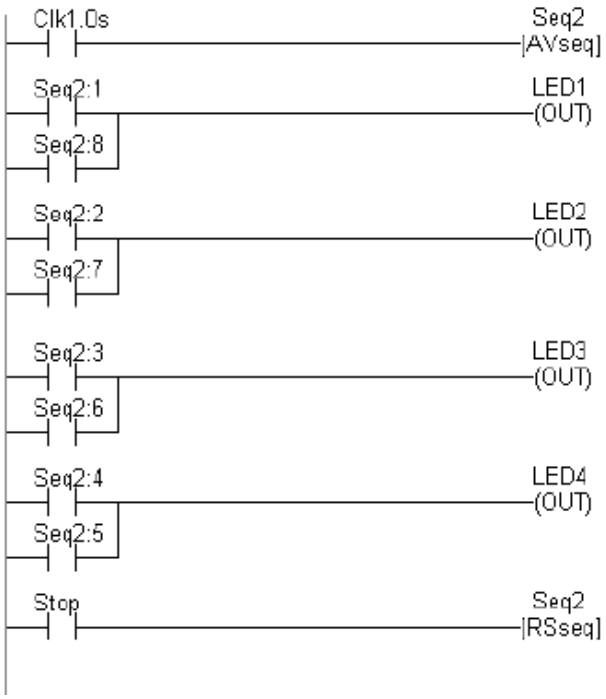


Figure 4.5

4.5 Programming Sequencers in Custom Function

You can change the current step of Sequencer easily from within a Custom Function by changing the present value of their equivalent counter. E.g. Sequencer #3 is the same as Counter #3, thus if you wish to assign Sequencer #3 to Step 10, you can achieve it as follows:

```
CTRPV[3] = 10
```

Chapter 5 Analog Inputs and Outputs

5 ANALOG INPUTS AND OUTPUTS

5.1 Analog Power Supply

The analog power of the PLC is derived from the same 12 to 24VDC power supply as the CPU. It will generate a stable 5V, (±1% accuracy) regulated DC voltage source that will be used internally as voltage reference and is available externally for use by other analog input devices. The reference voltage output is available on the analog I/O connector pin #15, and may be used as the source voltage for connecting to potentiometers. Its current is limited to 10mA only. Thus if you need more current for your analog device, you will need to supply your own quiet +5V DC source.

The analog reference voltage is generated by a stable voltage reference IC (LM4040-5.0) on the carrier board and is not user-adjustable.

5.2 Analog Inputs

Each Fx2424 PLC is equipped with 2 independent, 12-bit A/D (Analog to Digital) converters. Each A/D converter multiplexes with up to 4 Analog input pins, giving a total of 8 analog input channels. The 2 A/D converters run simultaneously and thus save conversion time, even when all 8 analog channels are used. All analog input channels operate in the range of 0-5V full-scale. The pin assignment for these analog outputs is described in Section 1.2.1.

Electrical Characteristics

No. of A/D channel	8
Resolution	12-bit
Input Impedance Ch #1 to 4 Ch #5 to 8	 >> 10 M ohms 20.00K Ohms
Moving Average	1 to 9 points (user definable)
Conversion Time	< 2 μs for one channel. < 6 μs for all 8 channels (2 A/D converters run simultaneously)

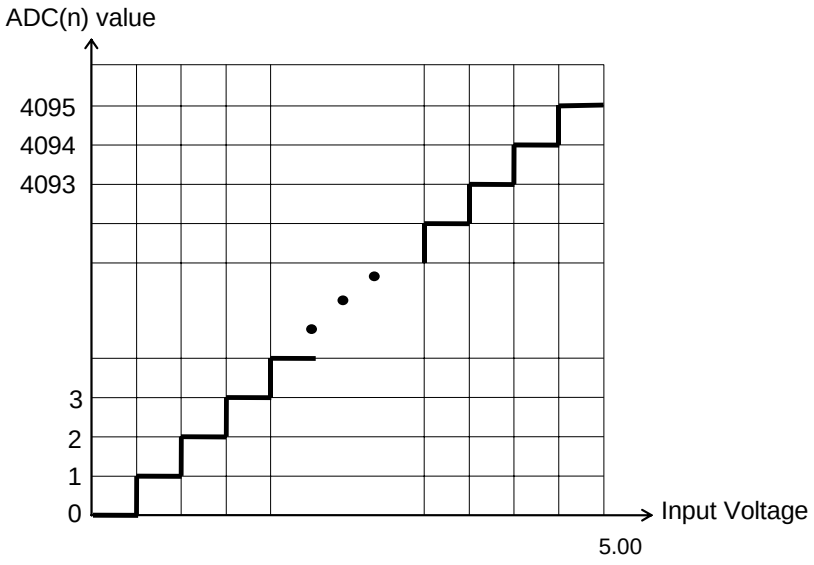
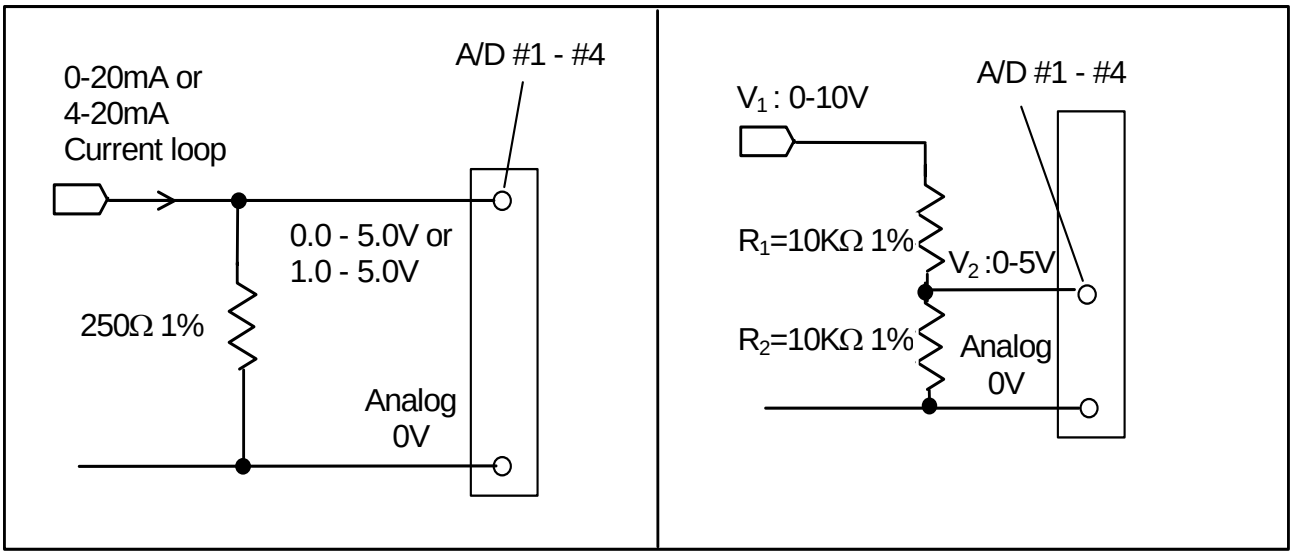


Figure 5.1 Transfer Function for 12-bit ADC.

Electrically, the A/D #1 to 4 are designed differently from A/D #5 to 8. This would affect how these 8 analog inputs interface to different analog signal sources, as described in the following subsections.

5.2.1 A/D #1 to 4

The first four ADCs are high impedance inputs (>>10 Mega Ohms), which of course can be connected directly to any 0-5V analog source. It is also very easy to interface to other analog voltage or current sources by using external resistors as voltage dividers or current-to-voltage converters, as shown in the following diagram:



Converting 20mA current-loop to 0-5V *Converting 0-10V signal to 0-5V*

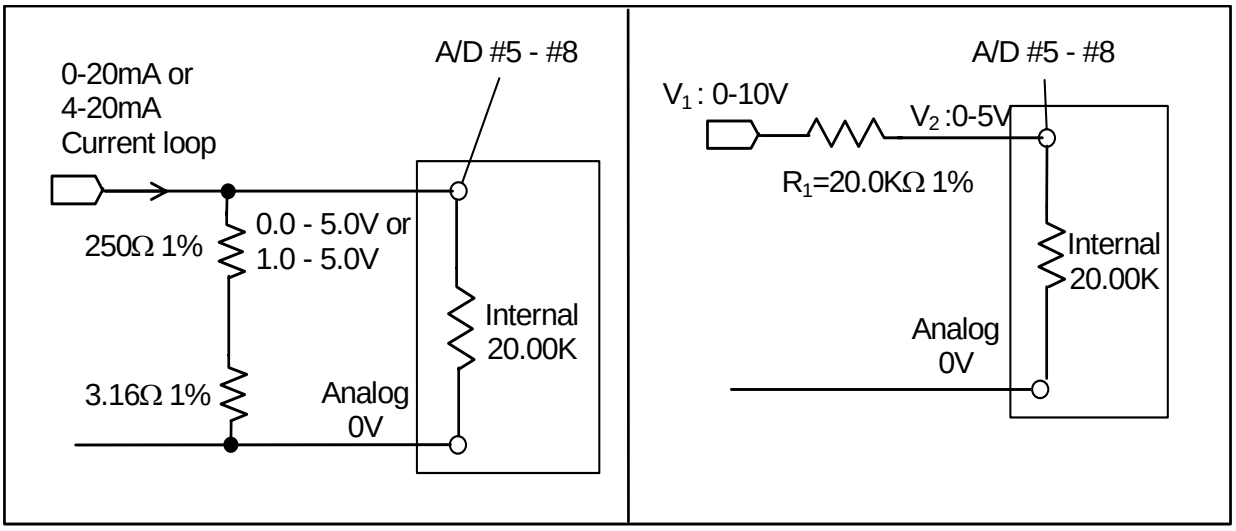
Figure 5.2

Note that if the sensor has its own power supply, then you must connect the sensor "ground" to analog "AV_{SS}" terminal to provide a common ground. All the analog inputs to the Fx2424 PLC are internally current-limited to guard against transient over-voltage damage. However, care should be taken to prevent applying excessive over-voltage to the analog inputs for a prolonged period of time, which can lead to permanent damage of the ADC input.

5.2.2 A/D #5 to 8

The DC input impedance of Analog inputs #5 to 8 are all 20.00K ohms (0.1%). This is not a problem when connected to a 0-5V low impedance analog source.

However, if you need to connect to 0-10V inputs or 4-20mA analog source to A/D #5 to #8, you have to take into consideration the low A/D input impedance in your design. The following figures show how to connect 4-20mA current source signals and 0-10V signals to the A/D inputs #5 to 8.
:



Converting 20mA current-loop to 0-5V *Converting 0-10V signal to 0-5V*

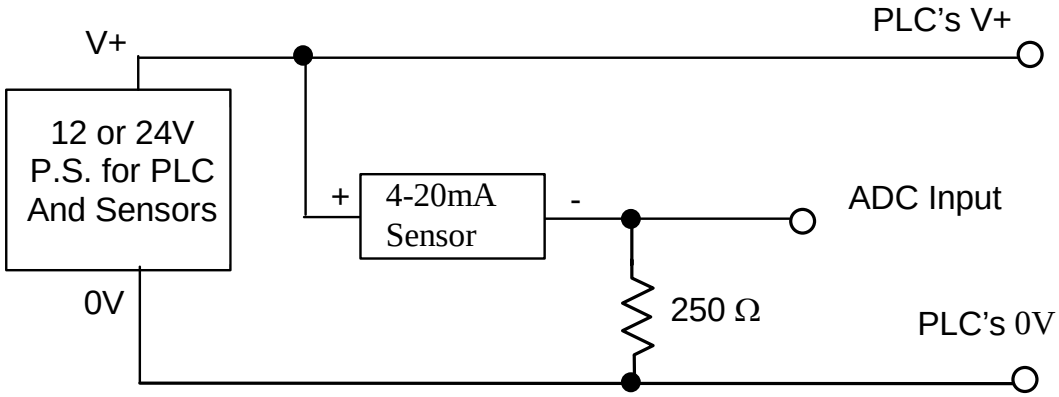
Figure 5.3

You can see that interfacing to 0-10V analog signal is extremely simple since all you need is to add a 20.0K ohm series resistor and it will be divided into 0-5V when it enters the AD #5 to #8.

However, to convert 0-20mA or 4-20mA current source into 0-5V or 1-5V voltage signal, you should use a 253.16 ohm resistor which, when paralleled with the 20K ohm of internal impedance, will yield a 250.0 ohm total resistance. You can obtain 253.16 ohm by combining a 250 ohm and a 3.16 ohm metal-film resistor.

5.2.3 Interfacing to two-wire 4-20mA sensors

Many 4-20mA analog sensors only have two wire connections and are designed to be powered by the 4-20mA output current that flows through it. These types of sensors can be interfaced easily to the 0-5V analog inputs of the PLC as shown in the following diagram:



* For ADC #5 to #8, use a 253.16 ohm resistor instead of 250 ohms as explained in 5.2.2.

The sensor output will be converted to a 1-5V analog voltage and can be read by the PLC using the ADC(n) statement, which will return readings of between 819 and 4095.

5.2.4 Using Potentiometer to Set Parameters

A potentiometer can provide a very low cost means for users to input parameters to the PLC such as temperature settings, timer or counter preset values, etc. The diagram above shows how easy it is to implement such a device using the 5V reference output and an analog input. Very accurate parameters can be set if the LCD display is used as visual feedback of the settings.

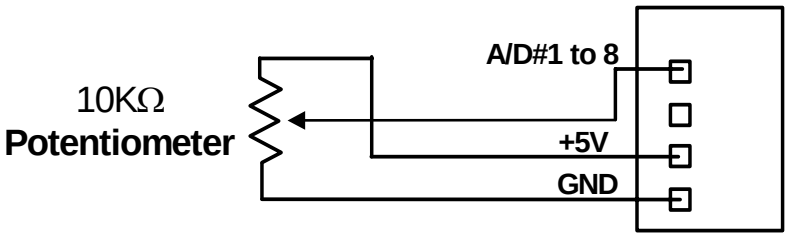


Figure 5.4

5.2.5 Reading Analog Input Data

The 8 analog input signals are read by the TBASIC command ADC(1) to ADC(8). The ADC(n) function will return a number between 0 and 4095(12-bit resolution), which corresponds to the measured voltage at any of the analog inputs n. The resolution of a 12-bit ADC is 1/4096, this means that for the 0-5V ADC range, the resolution is 5/4096V = 1.22mV.

That means that if you apply a 2.500V to the PLC's analog input #3, ADC(3) should return a value of 2.500/5.000 x 4096 = 2048.

Note that the CPU only accesses the analog input #n when the TBASIC function ADC(n) is called. Hence, in order to monitor the analog input, you have to execute the ADC function periodically. The frequency that the ADC function is called is known as the "sampling rate" and it depends on how fast the analog data changes. If the analog data changes slowly (such as room temperature) then there may be no need to sample the analog at high frequency.

A very simple example of sampling the analog input #1 to #4 every second and converting the data into voltage readings of 0 to 5000 (which represents 0 to 5.000V) is shown as follow:

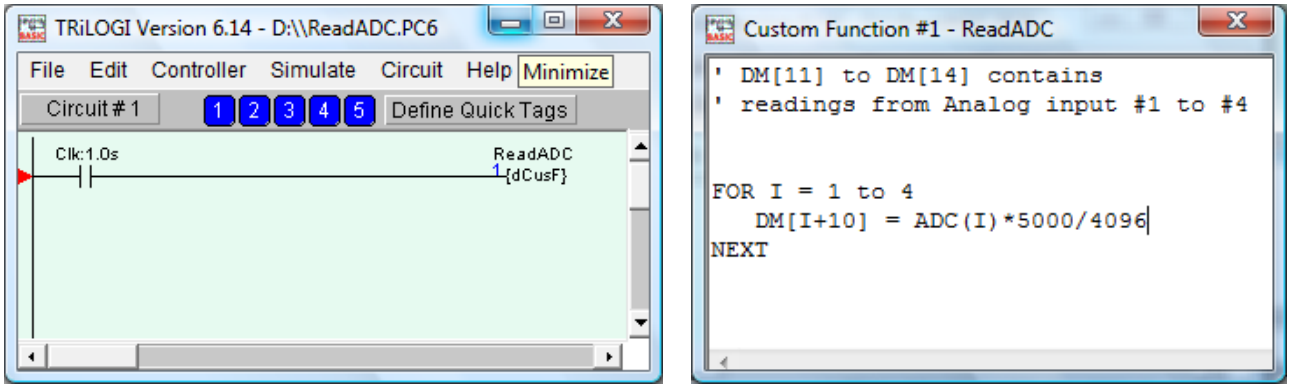


Figure 5.5

You can examine the readings of DM[11] to DM[14] from the "Online Monitoring - View Variables-DM[n]" screen. These readings represent the voltages measured at the analog input pins. You can also read the

raw ADC readings (which will change in the range between 0 to 4095) from the “View Variables - integer” screen.

5.2.6 Moving Average

The Fx2424 PLC offers a built-in “Moving Average” computation routine for each ADC channel. When moving average is enabled the PLC firmware would store the past analog readings for each channel in its own historical memory array, and each new instantaneous reading would overwrite the oldest reading. When you run the ADC(n) function, the PLC firmware would return the average of these past readings instead of the instantaneous new reading.

You can define a moving average of 1 to 9 points using the procedure described in Section 5.4.5

Defining a larger moving average can better help to even out fluctuations in ADC readings that can be caused by interference from digital noise. However, the larger the moving average, the slower the ADC(n) function can detect a sudden change in the amplitude of the analog signal due to the averaging effect.

For a system that needs to quickly detect a signal change, consider using either a smaller number of moving average points or call the ADC(n) function more frequently so that a sudden change can be detected earlier.

5.2.7 Scaling of Analog Data

The 12-bit analog inputs on the Fx2424 return data in the range of 0 to 4095, which corresponds to the full range of the voltage input presented at the analog pin. However, very often a user needs a formula to translate this numeric data into units meaningful to the process (e.g. degree C or F, psi etc). To do so, you need to know at least two reference points of how the native unit maps to the PLC's ADC reading.

Reference Point	ADC Reading
x1	a1
x2	a2

Hence, for any reading A = ADC(1), the corresponding X is derived from:

$$\frac{X - x1}{x2 - x1} = \frac{A - a1}{a2 - a1} \quad ==> \quad X = (x2 - x1)*(A - a1)/(a2 - a2) + x1$$

Note that since x1, x2, a1, and a2 are all constants the actual formula is much simpler then it appears above.

E.g. Temperature measurement

Temp	ADC(n)
30	200
100	3000

So for any ADC readings A, the temperature is:

X = 70*(A - 200)/2800 + 30

Note: To get better resolution , you can represent 30 degree as 300 and 100 degree as 1000 so if X = 123 it means 12.3 degree.

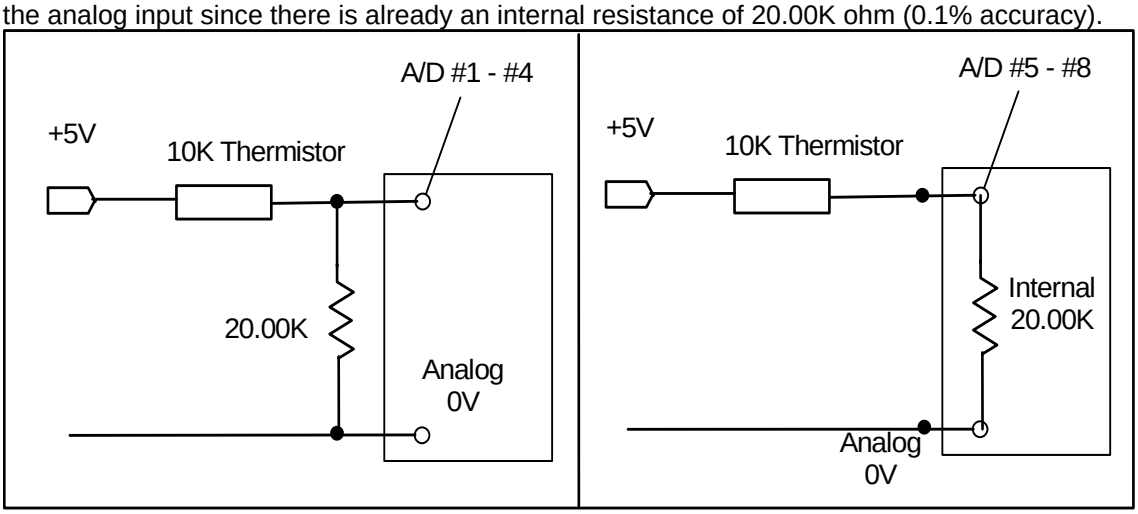
5.3 Temperature Measurement Using Analog Inputs

5.3.1 Thermistor Temperature Sensors

A thermistor is a kind of resistor whose resistance decreases when its surrounding temperature increases. It is a very low cost and stable device that can be used to measure a wide range of ambient temperature from freezers to hot water boilers, which are commonly used in HVAC applications.

In order to convert the resistance changes into voltage readings to be read by the PLC's analog input, you can use it to form an arm of a voltage divider circuit which would present a variable voltage to the analog input when the temperature changes. A type of thermistor that measures 10.0K ohm at 25 degree C (simply called 10K thermistor) is especially suitable for use with the Fx2424 PLC, as illustrated below:

For ADC #1 to #4, you can use an external 20 K Ohm resistor to form the voltage divider with the 10K thermistor. For ADC #5 to #8, it is even simpler because you can connect the 10K thermistor directly to the analog input since there is already an internal resistance of 20.00K ohm (0.1% accuracy).



Connecting 10K Thermistor to ADC 1-4 *Connecting 10K Thermistor to ADC 5-8*

Figure 5.6

Note that since the thermistor resistance value vs. temperature change is a non-linear function, you cannot simply use a formula to calculate the temperature from the voltage value. For better accuracy you need to use a look up table plus a linear interpolation technique to determine the temperature based on the ADC readings. The look up table and interpolation method can be implemented using TBASIC quite easily. For your convenience, we have provided a sample TBASIC program that you can download from the following web page:

http://www.tri-plc.com/appnotes/F-series/ThermistorSensorFPLC.zip

This example uses the R-T (Resistance-Temperature) graph of the Precon Type III thermistor to implement the temperature look up. We have provided an Excel file that computes the ADC reading vs ambient temperature for this thermistor type. The TBASIC program uses these ADC readings to determine the temperature.

The sample program is structured such that the lookup table values are stored in the FRAM and you can readily adapt it to other types of thermistors with a different R-T graph. The program only implements lookup for a temperature range of –10°F to 100°F, but you can also easily change the temperature range of interest.

5.3.2 Using LM34 Semiconductor Sensor

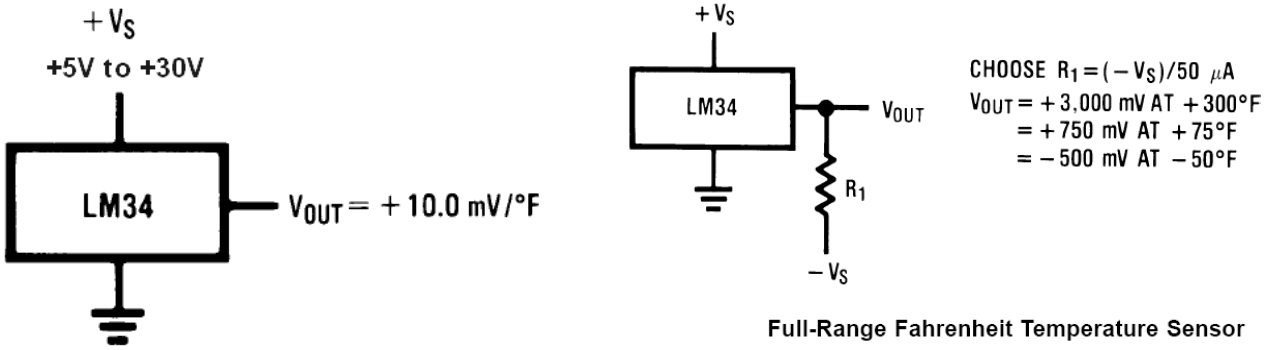


Figure 5.7

The LM34 is a wonderful, low cost semiconductor temperature sensor with a range of –50 ^{o}F to 300 ^{o}F. It is extremely easy to use for measuring temperature above 0 ^{o}F. You simply connect one pin to the positive voltage (+5 to +30V) and the other pin to 0V, and the signal pin will output a voltage that is directly proportional to the ambient temperature in ^{o}F. The output voltage is 10mV per degree F.

So at room temperature of 72 degrees F, the device will output a voltage of 72 x 0.01 = 0.72V. If you connect this to the PLC's analog input, you can obtain the temperature in degree F or degree C using the formula:

```
        F = ADC(1)*500/4096      ' in degree F
OR      C =  (F – 32)*5/9                ' in degree C
```

Although another part number LM35 can output temperature in 10mV per degree C, the output falls into even lower ADC range for ambient temperature measurement since the same 72 degree F is only 22 degree C, which means LM35 only outputs 22 x 0.01 = 0.22V. Hence for better accuracy and resolution we recommend using LM34 instead of LM35 and if need be then convert it to degree C using TBASIC.

Another advantage for using LM34 instead of LM35 is that you can measure down to 0 ^{o}F (-17 degree C) without using a negative voltage source as shown in the circuit on the right in Figure 5.7.

5.3.3 Using Thermocouple

Thermocouples are very rugged devices that are widely used in the industry because of their stability, accuracy, and wide functional temperature range. They are commonly used in measuring temperature in ovens that may go up to several hundred degrees C.

However, thermocouple output signals are in the range of tens of microvolts to mille-volts, which is too small to be measured by the Fx2424's analog input directly. You will need a "signal conditioner" that can amplify the thermocouple output to 0-5V, which can then be connected to the PLC's analog input.

5.3.4 Using PT100 Temperature Sensor

PT100 is a positive temperature coefficient thermistor that is made from platinum. It has the advantage of being very stable and highly accurate. It is usually connected to a signal conditioner in a balanced bridge configuration and the signal conditioner will convert temperature changes to 0-5V output for the PLC.

5.4 Analog Outputs

The Fx2424 PLC features 4 channels of 0-5V DC, 12-bit analog outputs (Digital-to-Analog Converters or D/A). Unlike in the M-series PLC, these 4 D/A outputs each have their own dedicated pins and, therefore, do not take up any Analog input pins. The pin assignment of these analog outputs is described in Section 1.2.1.

Note that the D/A #1 to #4 are all high impedance outputs that would not be able to source more than 1mA of current. You would need to connect them to external buffer amplifiers if you want to use them to deliver load current to drive external loads.

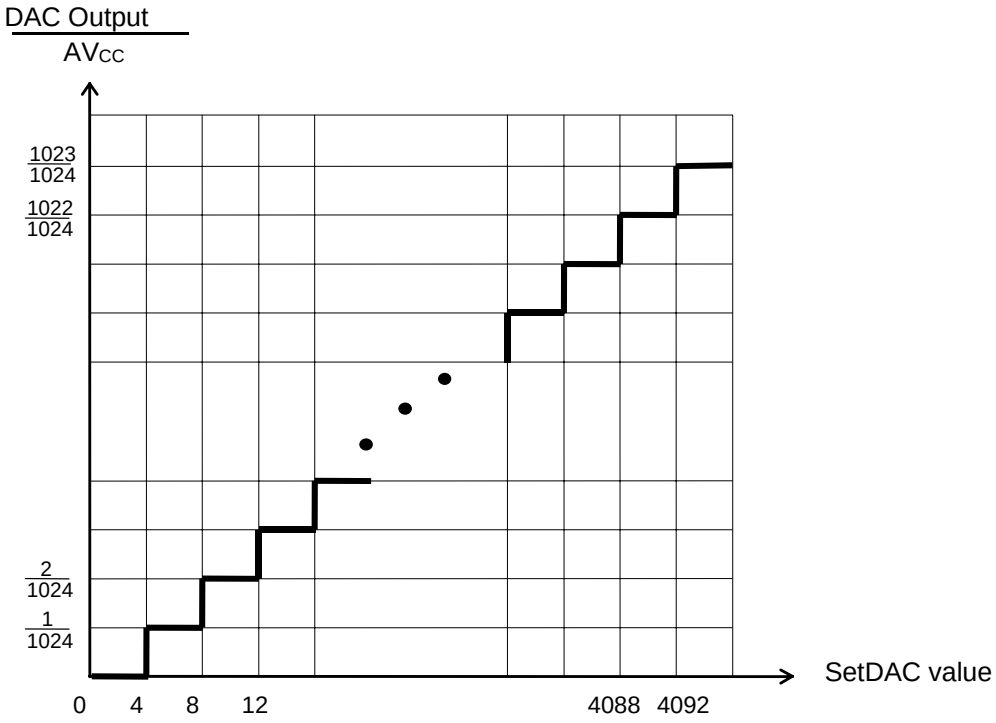


Figure 5.8

5.4.1 Programming The Analog Output

You can use the TBASIC command SETDAC n,x to set D/A #n to a 12-bit value x. (x = 0 to 4095).

You only need to execute the SETDAC statement once and the D/A output will be set to a voltage proportional to the given set value.

E.g. SETDAC 1, 1000

The D/A #1 will output 1000/4096 * 5V = 1.221V

5.5 Calibration of ADC and DAC & Moving Average Definition

The ADC and DAC are factory-calibrated such that a voltage of 2.500V should return a value of 2048 when read by the ADC(n) function. Likewise, an output voltage of 2.500V should be present at the DAC pin when you SETDAC to 2048. However, if there is a need to re-calibrate the ADC and DAC you can follow the procedure outlined below.

To perform calibration of an ADC channel, you need to supply a precise DC reference voltage to the ADC channel, and then check the analog readings obtained via the ADC(n) function and compare it to the expected value. If there is an error, you can apply a correction factor to it.

To perform the ADC or DAC calibration, you would need to use the "F-Series Ethernet Configuration" software mentioned in Chapter 2.1. Click on the "Advanced" button on the Basic Configuration screen and open the "F-series Ethernet Advanced Configuration" screen. The bottom half of the screen contains ADC and DAC calibration constants that you can enter and transfer to the PLC, as shown below:

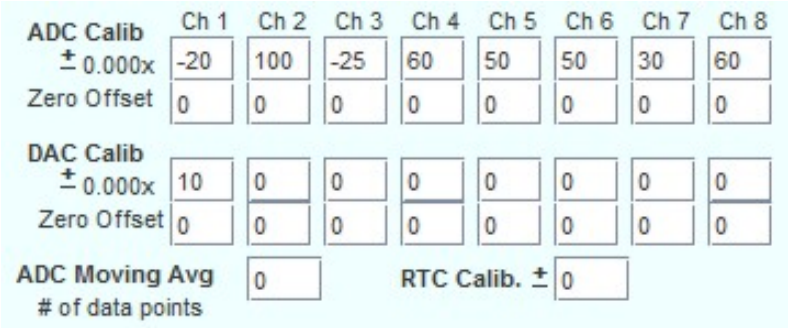


Figure 5.9

5.5.1 ADC Calib.

These fields are used to apply a multiplication factor to the value returned by ADC function. The multiplication factor = (1+ x/10000).

Example 1: If you apply 2.500V to ADC #1, you would expect the value returned by ADC(1) to be 2048. But the actual average reading centers around 2060.

Proportional Error = 2060/2048 = 1.005859
Multiplication factor required to correct this error = 1/1.005859 = 0.9942 = (1 – 58/10000)
=> x = -58

You should therefore enter a value of 58 into the "ADC Calib" field for Ch #1 and save it to the PLC.

After the PLC has rebooted, the CPU would apply the multiplication factor of 0.9942 to the readings it received, which would correct the reading to: 2060 x 0.9942 = 2048.

Example 2: If you apply 4.000V to ADC #8, you would expect the ADC(8) function to return a value of 4.000/5.000 x 4096 = 3277. However, your program returned a value of 3230 from ADC(8).

Proportional Error = (3230)/3277 = 0.985658
Multiplication factor required to correct this error = 1/0.985658 = 1.0146 = (1+ 146/10000)
=> x = +146

To compensate for this error, enter a value of 146 in the "ADC Calib." for Ch 8 and save it to the PLC.

After the PLC has rebooted, the CPU would apply the multiplication factor of 1.0146 to the readings it received, which would correct the reading to: 3230 x 1.0146 = 3277.

Notes:

1) We have created an MS-Excel spreadsheet file "AnalogCalibration.xls" to facilitate the computations of the correction factor, X, used in the ADC and DAC Calibration. This file can be downloaded from the following web page:

   http://www.tri-plc.com/appnotes/F-series/AnalogCalibration.xls

2) Changes to the ADC calibration data only take effect after the PLC has been cold-booted. You can either power cycle the PLC or simply check the "Reboot PLC After Save" checkbox and the PLC will re-boot after you have transferred the parameters to it.

5.5.2 ADC Zero Offset

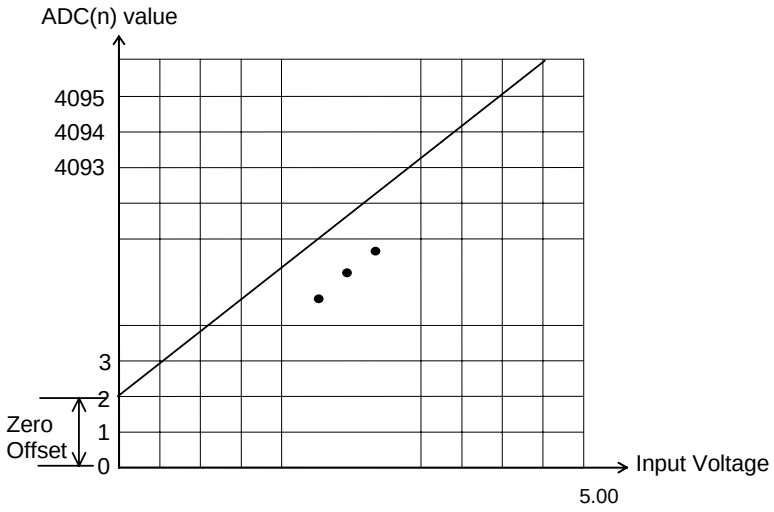


Figure 5.10

The zero offset error can be corrected by entering a value into the "ADC Zero Offset" field. Any number between –100 and 100 can be entered here. The ADC(n) function would add the zero-offset value that you entered here to the measured value and return the total sum to the calling routine.

5.5.3 DAC Calib.

A value x entered in each of these fields represents the multiplication factor (1 + x/10000) that the PLC will apply to the "*value*" parameter, which is executed by the command "SETDAC n, *value*" before actually writing to the DAC output. This allows the user to apply a correction factor to the DAC output if there is a problem with the DAC voltage.

Example 1: If you execute "SETDAC 1,2048", you would expect the DAC #1 to output 2.500V, but instead you only get 2.480V.

Proportional Error = 2.480/2.500 = 0.992
Multiplication factor required to correct the DAC output = 1/0.992 = 1.0081 = (1+81/10000)

You can therefore enter the value “81” into the “DAC Calib.” field corresponding to DAC #1 and save the parameters to the PLC. After the PLC has rebooted, when you execute the statement “SETDAC 1,2048”, the CPU would apply the multiplication factor of 1.0081 to the actual digital value it sends to the DAC output. The actual DAC output = 2.480V x 1.0081 = 2.500V.

Example 2: Same as Example 1 but you measure 2.515V when you expect 2.500V

Proportional Error = 2.515/2.500 = 1.006
Multiplication factor required to correct the DAC output = 1/1.006 = 0.9940 = (1 – 0.0060)

You should therefore enter the value –60 into the “DAC Calib” field corresponding to DAC #1 and save it to the PLC. After the PLC has rebooted, when you execute the statement “SETDAC 1,2048”, the PLC will apply the multiplication factor of 0.9940 to 2048 before it writes to the DAC hardware.

The actual DAC output = 2.515V x 0.994 = 2.500V

5.5.4 DAC Zero Offset

If you plot the line graph for the output voltage versus the DAC set value, the line should normally pass through the origin. But if there were any zero offset error, then the line would be above or below the origin.

The DAC output on the Fx2424 should not have any zero offset error and you normally should just leave these fields set to “0”.

However, if for any reason there is a need to apply a zero offset error correction, you can enter a value between –100 to +100 into the “DAC Zero Offset” field. The CPU would add the zero-offset value you enter into this field to the “X” in the “SETDAC n, x” statement and only send the sum to the actual DAC hardware.

5.5.5 A/D Moving Avg

This field lets you define the number of points of moving average that the Fx2424 CPU firmware uses to compute the value returned by the ADC(n) function (Please see explanation of moving average in Section 5.2.5).

A larger number of moving Average points has the positive effect of filtering out large noise spikes seen at the analog input, but the disadvantage is that the PLC would be slower in noticing a sudden step change at the analog input. If you specify a moving average of 1 point, that means no moving average will be used and the ADC(n) function will return the most recently sampled data at the analog input #n.

Chapter 6 Special Digital I/Os

6 SPECIAL DIGITAL I/Os

12 of the first 16 ON/OFF inputs of the Fx2424 can be configured as "special inputs" such as High Speed Counters, Interrupts and Pulse Measurement. 12 of the 16 outputs can also be configured as PWM, or stepper motor controller/driver pulse-outputs. If these special I/Os are not used, then they can be used as ordinary ON/OFF type I/O in the ladder diagram. The High Speed Counters and Pulse measurement inputs share physical inputs, but they can be used simultaneously as HSC and PMON (unlike M-Series). Note that if any other two special functions share the same I/O then only one of them can be active at any one time. The location of these special I/Os are tabulated as follows:

Special Inputs

Input #	High Speed Counter	Interrupt	Pulse Measurement
1	Ch #1: Phase A	Ch #1	Ch #1
2	Ch #1: Phase B	Ch #2	Ch #2
3	Ch #2: Phase A	Ch #3	Ch #3
4	Ch #2: Phase B	Ch #4	Ch #4
5	Ch #3: Phase A	Ch #5	Ch #5
6	Ch #3: Phase B	Ch #6	Ch #6
7	-	-	-
8	-	-	-
9	-	Ch #9	-
10	-	Ch #10	-
11	-	Ch #11	-
12	-	Ch #12	-

Note: While inputs 1-6 can be used simultaneously as High Speed Counters **and** Pulse Measurement pins, any pins defined as interrupts can only be interrupts.

Special Outputs

Output #	Stepper Pulse/Dir outputs	Stepper Driver Outputs	PWM output
1	-	-	-
2	-	-	-
3	-	-	-
4	-	-	-
5	Ch #1 Direction	Ch #1 **A** phase	Ch #1
6	Ch #1 Pulse	Ch #1 $\overline{\mathbf{A}}$ phase	Ch #2
7	-	Ch #1 **B** phase	Ch #3
8	-	Ch #1 $\overline{\mathbf{B}}$ phase	Ch #4
9	Ch #2 Direction	Ch #2 **A** phase	-
10	Ch #2 Pulse	Ch #2 $\overline{\mathbf{A}}$ phase	-
11	-	Ch #2 **B** phase	-
12	-	Ch #2 $\overline{\mathbf{B}}$ phase	-
13	Ch #3 Direction	Ch #3 **A** phase	-
14	Ch #3 Pulse	Ch #3 $\overline{\mathbf{A}}$ phase	-
15	-	Ch #3 **B** phase	-
16	-	Ch #3 $\overline{\mathbf{B}}$ phase	-

These special I/Os therefore share the same electrical specifications as the ON/OFF type I/Os, which have already been described in the Chapter 1 - Installation Guide. We will describe each of these special I/Os in greater details in the following chapters.

Chapter 7 High Speed Counters

7 HIGH SPEED COUNTERS

Technical Specifications:

No. of Channels	3
Maximum acceptable pulse rate	10KHz per channel
Quadrature signal decoding	Automatic
Relevant TBASIC Commands	HSCDEF, HSCOFF, HSCPV[ ]

7.1 Introduction

Digital inputs #1 + 2, #3 + 4, and #5 + 6 form three channels of high speed counter inputs which can interfaced directly to a rotary encoder that produces "quadrature" outputs. A quadrature encoder produces two pulse trains at a 90^o phase shift from each other as follows:

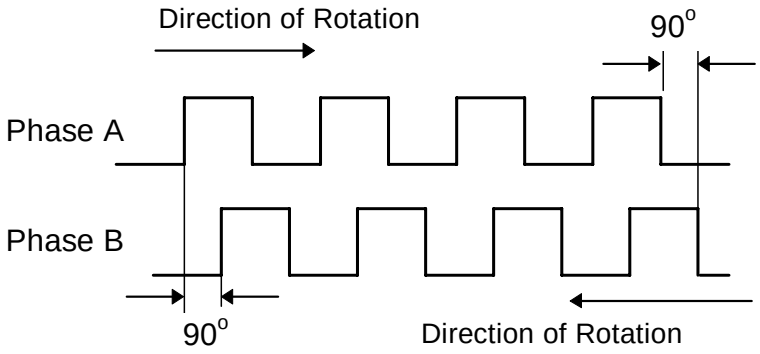


Figure 7.1

When the encoder shaft rotates in one direction, phase A leads phase B by 90 degrees. When the shaft rotates in the opposite direction, phase B will lead phase A by 90 degrees. The quadrature signals therefore provide an indication of the direction of rotation.

The Fx2424 handles the quadrature signals as follows: if the pulse train arriving at input #1 leads the pulse train at input #2, the High Speed Counter (HSC) #1 increments on every pulse. If the pulse train arriving at input #1 lags the pulse trains at input #2, then the HSC #1 decrements. Note that if input #2 is OFF, then pulse trains arriving at input #1 are considered to lead the input #2 and HSC #1 will be incremented. Likewise if input #1 is OFF, then pulse trains arriving at input #2 will decrement HSC #1.

Inputs #3 and #4 form the inputs for High Speed Counter channel #2 and Inputs #5 and #6 form the inputs for High Speed Counter channel #3. HSC#2 and HSC#3 operate in the same way as HSC#1 described above.

The fact that the Fx2424 automatically takes care of the direction of rotation of the quadrature encoder greatly simplifies the programmer's task of handling high-speed encoder feedback. The HSCDEF statement can be used to define a CusFn to be executed when the HSC reaches a certain pre-defined value. Within this CusFn you can define the actions to be taken and define the next CusFn to be executed when the HSC reaches another value. Please note that the HSDDEF statement will also activate the Pulse Measurement hardware as described in the Pulse Measurement section.

A programming example of the HSC can be found in your iTRiLOGI program folder:

```
C:\TRiLOGI\TL6\usr\samples\HighSpeedCtr.PC6
```

7.2 Enhanced Quadrature Decoding

The default method in which the PLC handles quadrature signals as described above is somewhat simplistic. It does not take into consideration the "jiggling" effect that occurs when the encoder is positioned at the transition edge of a phase. Mechanical vibration could cause multiple counts if the rotor shaft "jiggles" at the transition edge of the phase, resulting in multiple triggering of the counter. This simplistic implementation, however, does have the advantage that the HSC can also be used for single-phase high-speed counting.

For the Fx2424, an enhanced quadrature decoding routine is provided which will lock out multiple counting by examining the co-relationship between the two phases. You can configure the Fx2424 to use the enhanced quadrature counting by using the SETSYSTEM command, as follows:

SETSYTEM 4, n

The value of n at bit 0, 1, and 2 respectively defines if the HSC channel 1, 2, and 3 is to run in "Simple" (when the bit is 0) or "Enhanced" (when the bit is 1) mode. As such:

N (bit 2,1,0)	HSC #3	HSC #2	HSC #1
0 (000)	Simple	Simple	Simple
1 (001)	Simple	Simple	Enhanced
2 (010)	Simple	Enhanced	Simple
3 (011)	Simple	Enhanced	Enhanced
4 (100)	Enhanced	Simple	Simple
5 (101)	Enhanced	Simple	Enhanced
6 (110)	Enhanced	Enhanced	Simple
7 (111)	Enhanced	Enhanced	Enhanced

7.3 Interfacing to 5V type Quadrature Encoder

If you have a choice, you should select an encoder that can produce 12V or 24V output pulses so that they can drive the inputs #1,2,3,4,5 or 6 directly. If you have a 5V type of encoder only, then you need to add a transistor driver to interface to the PLC's inputs. The simplest way is to use an IC driver ULN2003 connected as shown in Figure 7.2.

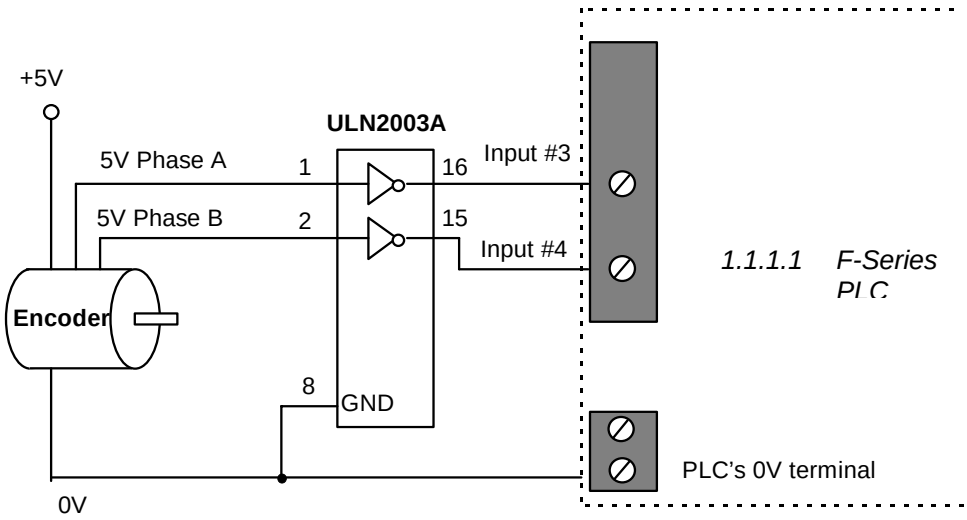


Figure 7.2 Interfacing 5V type Rotary Encoder

Chapter 8 Frequency / Speed Measurement

8 FREQUENCY / SPEED MEASUREMENT

The Fx2424 provides a very straightforward means to measure the pulse width (of the ON cycle), the frequency, or the period of a rectangular-wave pulse-train arriving at its Pulse Measurement (PM) inputs #1,2,3,4,5, or #6. (Which are mapped to digital inputs #1 to #6 – see Chapter 6).

8.1 Programming of PM Input

1) To use the PM input to measure pulse width or frequency, execute the PMON statement ONCE to configure the relevant input to become a pulse measurement input. You usually put the PMON statement in the init custom function and execute it with a “1st.Scan” pulse.

2) Thereafter the pulse width (in μs) or the pulse frequency (in Hz) can be easily obtained from the PULSEWIDTH(n) or PULSEFREQUENCY(n) functions. You can also obtain the pulse period (inverse of frequency) using the PULSEPERIOD function.

   E.g. A = PULSEWIDTH(1)
   B = PULSEPERIOD(1)
   C = PULSEFREQUNCY(1)

3) All PM inputs by default return the measured pulse width and pulse period in unit of microsecond. However, for those who desire better resolution, you can define PM #1 to #4 to return the measured pulse width and pulse period in 0.1 microsecond resolution by executing the following command once only during initialization:

   SETSYSTEM 20, 1

   Once the above statement is executed, if PUSLEWIDTH(1) - PULSEWIDTH(4) returns the value 1234 it means the measured pulse width is 123.4 μs. However, PULSEWIDTH(5) and PULSEWIDTH(6) would still only return the measured width in microsecond.

A sample program can also be found on your i-TRiLOGI installation folder at:

C:\TRiLOGI\TL6\usr\samples\PulseMeasurement.PC6

8.2 Applications

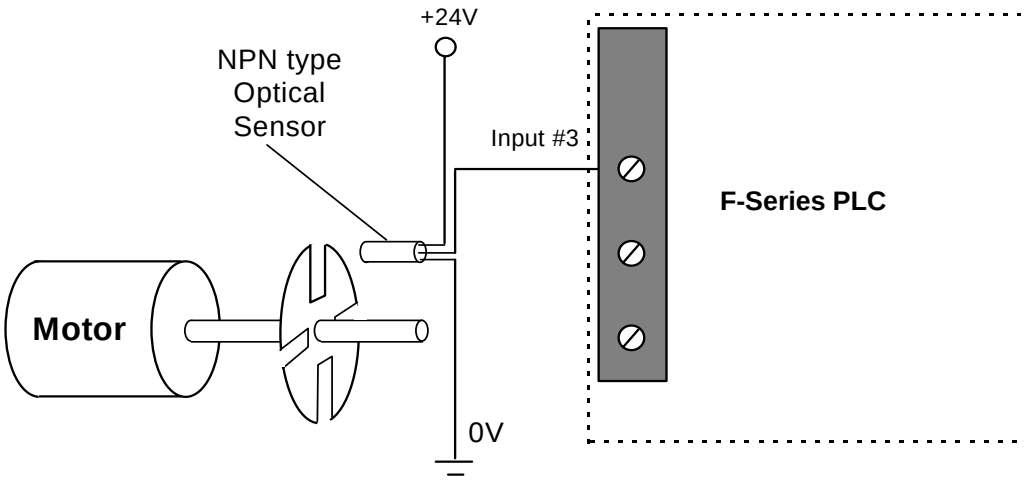


Figure 8.1 Setting Up a Simple Tachometer or Encoder

8.2.1 Measuring RPM Of A Motor

One useful application of the PM capability is to measure the speed of rotation (RPM) of a motor. A simple optical sensor, coupled with a rotating disk with slots fitted to the shaft of a motor (See Figure 8.1) can be fabricated economically. When the motor turns, the sensor will generate a series of pulses. The frequency of this pulse train directly measures the rotational speed of the motor (RPM = Frequency x 60) and can be used to provide precise speed control.

Note that the above setup can also double as a low cost position-feedback encoder when used with the high speed counter, since the number of pulses counted can be used to determine the displacement. With the Fx2424, the pulses can be both counted and measured **simultaneously** on the same input.

8.2.2 Measuring Transducer with VCO Outputs

Some transducers incorporate a Voltage-Controlled-Oscillator (VCO) type of output that represents the measured quantities in terms of varying frequency of the output waveform. Such transducers may be used conveniently by the Fx2424 using the pulse measurement capability. However, the frequency of such signals should be below the maximum input pulse rate.

8.2.3 Measuring Transducer with PWM Outputs

Some transducers may output the readings of their measurands (the quantity that is being measured) in the form of “pulse-width modulated” outputs. This means that the transducer would send rectangular pulses with varying duty cycles that are proportional to the measured quantities. You can then easily use the PULSEWIDTH and PULSEPERIOD functions to compute the duty cycle of the incoming PWM pulses and readily convert it to the actual units of the measurands. The Fx2424 PLC should be able to measure with reasonable accuracy the pulse width of incoming pulses not exceeding 10KHz. Although the PLC should be able to measure incoming pulse frequency of up to 20KHz, the 100pF noise filtering capacitors used at the digital inputs 1-16 would distort the signal at high frequency such that the measured pulse width would be of somewhat poor accuracy.

8.3 Frequency Measurement on High Speed Counter Inputs

For applications that require frequency measurement and pulse counting of the same signal, you only need to feed the pulse input into any pair of the inputs #1 & 2, or inputs #3 & 4, or inputs #5 & 6 and define it as a High Speed Counter (see Chapter 7). This is because an input pin that has been defined as an HSC will automatically be enabled for pulse measurement.

In other words, if you need to use the HSC and the Pulse Measurement on the same channel, then you don't need to execute both the HSCDEF and PMON, you only need to execute the HSCDEF. The HSCDEF function will automatically start the Pulse Measurement hardware so it is not necessary to use the PMON function. However, if you use only the PMON function, it would not enable the HSC function. However, if you execute the HSCDEF function and then execute the PMON function, the HSC will be disabled, even though it was previously enabled.

Chapter 9 Interrupts

9 INTERRUPTS

9.1 Input Interrupts

During normal PLC ladder program execution, the CPU scans the entire ladder program starting from the first element, progressively solving the logic equation at each circuit until it reaches the last element. After which it will update the physical Inputs and Outputs (I/O) at the end of the scan. Hence, the location of a logic element within the ladder diagram is important because of this sequential nature of the program execution.

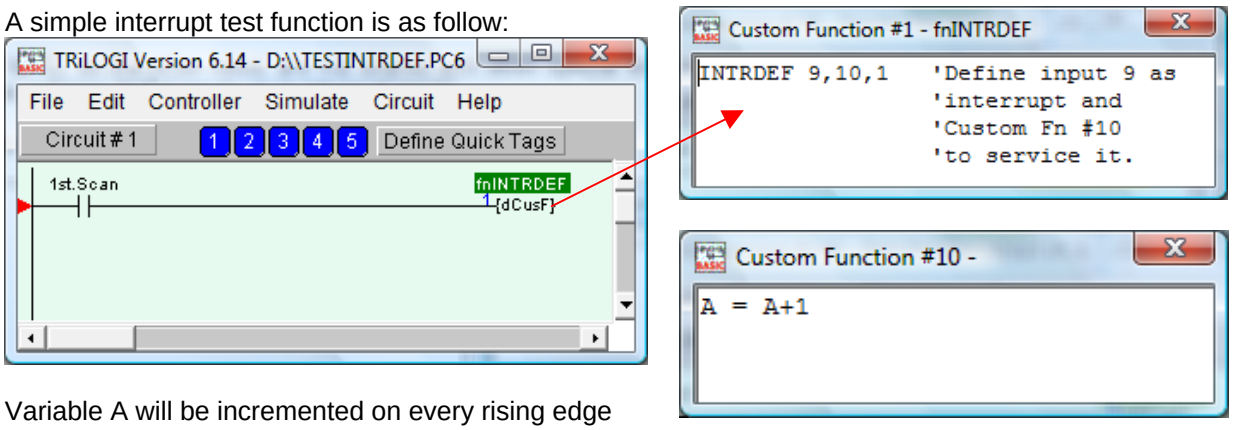
When scanning the ladder program, the CPU uses some internal memory variables to represent the logic states of the inputs obtained during the last I/O refresh cycle. Likewise, any changes to the logic state of the outputs are temporarily stored in the output memory variable (not the actual output pin) and will only be updated to the physical output during the next I/O refresh.

You can see that the CPU will only notice any change to the input logic state when it has completed the current scan and starts to refresh its input variables. The input logic state must also persist for at least one scan time to be recognized by the CPU. In some situations this may not be desirable because any response to the event will take at least one scan time or more.

An interrupt input, on the other hand, may occur randomly and the CPU will have to suspend whatever it is doing as soon as it can and start “servicing” the interrupt. Hence, the CPU responds much faster to an interrupt input. In addition, interrupts are “edge-triggered”, meaning that the interrupt condition occurs when the input either changes from ON to OFF or from OFF to ON. Consequently, the input logic state need not persist for longer than the logic scan time for it to be recognized by the CPU.

The Fx2424 PLC supports up to 10 interrupt inputs: Any one or all of digital inputs #1 to #6 and #9 to #12 can be defined as interrupt inputs using the INTRDEF statement. The Interrupt inputs may also be defined as either rising-edge triggered (input goes from OFF to ON) or falling-edge triggered (input goes from ON to OFF). When the defined edges occur, the defined CusFn will be immediately executed irrespective of the current state of execution of the ladder program.

A simple interrupt test function is as follow:



Variable A will be incremented on every rising edge sensed on Input #9.

Note: Since inputs 1 to 6 can also be used as other special inputs such as High Speed Counter (HSC) inputs and/or Pulse Measurement (PM) as described in Chapter 6, 7 and 8, if these inputs are defined as interrupts using the INTRDEF statement, then they will lose their other special function. I.e., they can only be defined either as a HSC/PM or as an interrupt input and not both. Inputs 9 to 12, on the other hand,

are not shared with other special functions. Hence, these can be used either as ordinary digital inputs or as interrupt inputs.

9.2 Periodic Timer Interrupt (PTI)

The Periodic Timer Interrupt (PTI - not available on T100M+ PLC) lets you define a custom function that will be executed by the CPU precisely every *x* number of milliseconds (ms). The syntax for setting up a PTI is as follow:

```
INTRDEF 18, cfnum, x  ' Interrupt 18 is reserved for PTI
```

cfnum - custom function number to execute when PTI event takes place.
x - The period in number of milliseconds between two PTI events.

E.g. `INTRDEF 18, 101, 15 ' call function #101 every 15 ms.`

The Periodic Timer Interrupt runs independently of the ladder logic and its execution is therefore not affected by the total PLC program scan time.

When the PTI timer times up, the CPU will suspend the execution of the ladder logic or a (non-interrupt) TBASIC function and immediately calls up the custom function defined by the INTRDEF 18 statement. However, if the CPU is currently executing a user-interrupt service routine (e.g. an input interrupt or HSC interrupt), then the CPU will have to complete the current interrupt service routine before it will run the PTI interrupt function.

Notes:

1) Limit the use of PTI only for critical code that requires precise timing between two events. Program bugs that occur due to problems in the PTI interrupt routine may be quite hard to debug.

2) For normal periodic routines, such as checking for temperature or checking serial port for incoming bar code data every few seconds, it is better to use the system clock pulses e.g. “Clk:1.0s” to trigger a {dCusF}.

3) Always try to keep your interrupt service routine short and ensure that it will not end up in an endless loop. The TBASIC custom function execution time should be much shorter than the period of the PTI events. Otherwise, you may find that the CPU will be spending most of its time servicing the PTI interrupt routine, leaving very little time for scanning the ladder program, and that will have an adverse impact on the CPU performance.

9.3 Power Failure Interrupt (PFI)

The Fx2424 CPU has a power failure sensing circuit that will call a custom function when it detects an impending power failure. This allows you to save critical data to the PLC’s non-volatile memory (see Section 1.7.2) just before power failure. The syntax for the PFI is as follow:

```
INTRDEF 17, cfnum, 1  ' Interrupt 17 is reserved for PFI
```

cfnum - custom function number to execute when PTI event takes place.

E.g. `INTRDEF 17, 256, 1 ' call function #256 when power failure occur.`

Chapter 10 Stepper Motor Control

10 STEPPER MOTOR CONTROL

10.1 Technical Specifications:

No. of Channels (control signal)	3
No. of Channels (full driver)	3
Max. Pulse Rate (pps)	10000 (single channel running)
Continuous Current per phase	4A @24V DC
Peak Current per phase	8A @24V DC
Driver Breakdown Voltage	+55V
Velocity Profile (Defined by STEPSPEED)	Trapezoidal -accelerate from 1/8 max pps to max pps. -decelerate from max pps to 1/8 max pps)
Maximum number of steps	2 ~ 2^{31} (= 2.1 x 10^9)
TBASIC commands	STEPSPEED, STEPMOVEABS, STEPCOUNTABS( ), STEPMOVE, STEPSTOP, STEPCOUNT( )

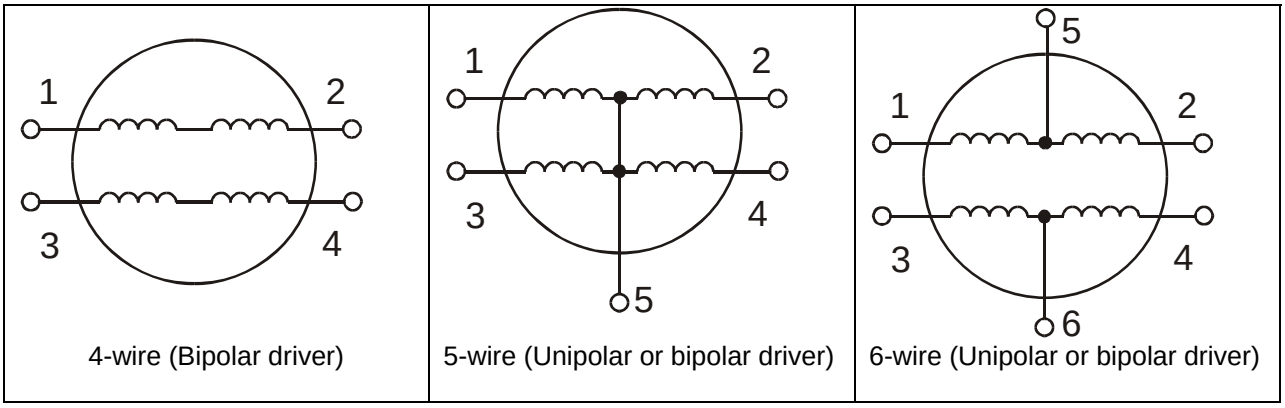
It is essential to understand the difference between a stepper motor “Controller” and a stepper motor “Driver”. A stepper motor “Driver” comprises the power electronics circuitry that provides the voltage, current, and phase rotation to the stepper motor coils.

Unlike the FMD PLCs, the Fx2424 have both the stepper motor driver and controller hardware. Now a stepper motor can be driven **directly** from the PLC or the PLC can simply supply the control signals to an external stepper motor driver.

10.2 Stepper Motor Driver

The Fx2424 can drive up to 3 stepper motors directly, using digital outputs #5 to #16. Each stepper motor requires 4 digital outputs so a total of 12 digital outputs are required to drive 3 stepper motors. All these outputs are capable of driving stepper motors of up to 4A per phase. Note that any outputs that have been defined for use with a particular stepper channel would no longer be available as regular digital outputs in the ladder or TBASIC program. Please refer to Chapter 6 for a definition of the list of special digital I/Os.

10.2.1 Stepper Motor Wiring Types



4-wire (Bipolar driver) | 5-wire (Unipolar or bipolar driver) | 6-wire (Unipolar or bipolar driver)

Figure 10.1

The most commonly available stepper motors today are either 4-wire, 5-wire or 6 wires types and their windings are as shown in Figure 10.1

- 4-wire stepper motors require drivers that can reverse current flow from coil terminal 1 to 2 or from 2 to1, as well as from terminal 3 to 4 or from 4 to 3. These kinds of drivers are known as “bipolar drivers”.

- 5-wire stepper motors typically connect the common terminal (5) to the power (e.g. +24V DC) and 4 current sink drivers are connected to terminals 1 to 4 to sink the current to 0V. The sequence in which the four coils are activated controls the rotation of the stepper motor. These kinds of drivers are known as “unipolar” drivers.

- 6-wire stepper motors are the most flexible and may be connected either to bipolar drivers (leaving terminals 5 and 6 unconnected) or to unipolar drivers (connecting terminals 5 & 6 together to “+” terminal.

Since all the Fx2424 digital outputs are NPN (current sink) type, only 5- or 6-wire type stepper motors can be used.

If you have a stepper motor with 6 wires (most common) but you do not have the wiring diagram, you may be able to figure out which wire is which terminal by using a multi-meter to measure the resistance between coils. E.g. the resistance between terminals 1 and 2 will be twice the resistance between terminals 1 and 5 or between 2 and 5, likewise for terminals 3,4 and 6. There will be no connectivity between terminal 1 and 3 and between terminal 2 and 4. It will however be more tricky to figure out the actual wiring of a 5-wire stepper motor since it is difficult to figure out which is the coil 1-2 and which is the coil 3-4, so some trial and error may be required to get the motor working.

10.2.2 Damping Circuits (Inductive Bypass)

Since the stepper motor windings are all inductors, it is important to connect damping diodes across each winding to absorb the inductive kicks produced when the windings are turned OFF. There are several ways to connect the damping circuit; the simplest way is to connect a diode from the PLC’s output to the Stepper motor power supply V+, as shown below in Figure 10.2.

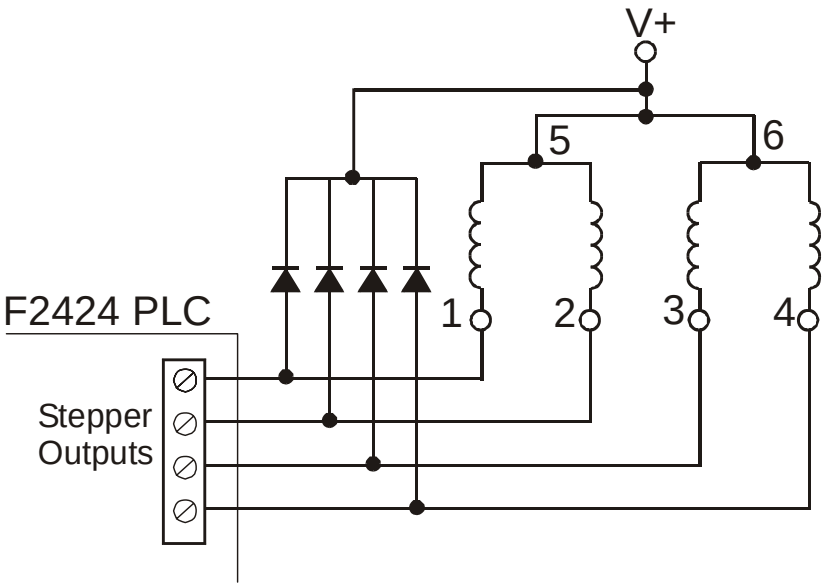


Figure 10.2

The above circuit works best when the stepper motor power is the same voltage as or is a slightly higher voltage than the PLC's power supply. (Note: The maximum stepper motor's power supply voltage must not exceed the PLC's stepper output driver breakdown voltage of 55V)

If the stepper motor's power supply were lower than the PLC's power supply, then connecting the output shown in Figure 10.2 would result in the PLC's output LED to light up, even when the output driver is off. This is because the PLC's output LED is tied to the PLC's power supply and a lower voltage at the load's power supply would cause current to flow out of the PLC's output terminal into the load's power supply and therefore turning on the output LEDs. Although this should not affect the driver's function it can be misleading to the user as to the PLC's output state. Hence, we recommend connecting a diode in series with each of the PLC's outputs to block such reverse current flow:

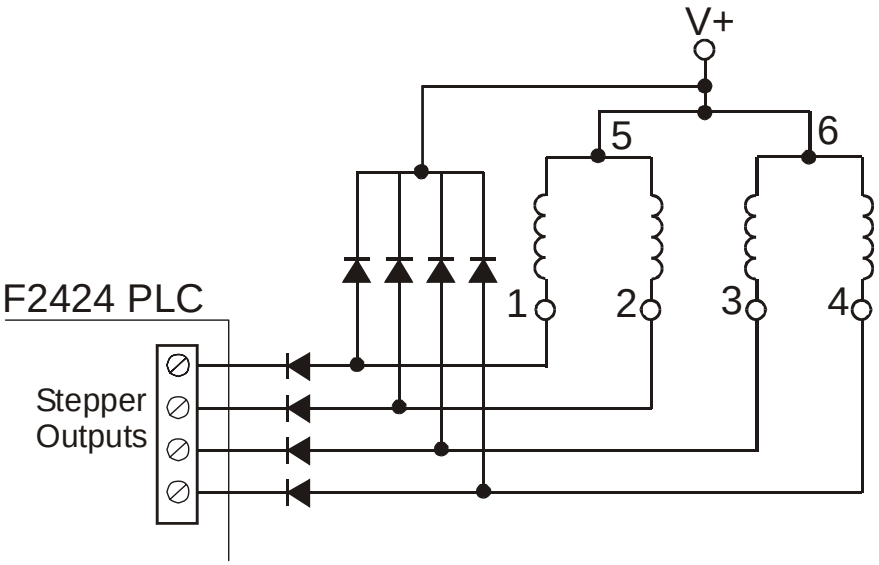


Figure 10.3 – If V+ is lower voltage than PLC's power supply voltage.

10.3 Stepper Motor Controller

If the built-in stepper driver is not suitable for your application (e.g. if you need micro-stepping or you need to drive a very high power stepper motor), then you can use an external stepper motor driver and use the PLC to send direction and output pulses to the stepper motor driver.

When configured as a Stepper-Motor Controller, the PLC would generate the required number of "pulses" and sets the direction signal according to the defined acceleration and maximum pulsing rate specified by "STEPSPEED" and "STEPMOVE" commands. The "pulse" and "direction" outputs are not meant to be connected directly to the stepper motor. Instead, you will need a stepper motor "driver", which you can buy from the motor vendor. Depending on the power output, the number of phases of the stepper motor, and whether you need micro-stepping, the driver can vary in size and cost. Most stepper motor drivers have opto-isolated inputs which accept a direction signal and stepping-pulse signal from the "Stepper Motor Controller". In this case the Fx2424 is the "Stepper Motor Controller" which will supply the required pulse and direction-select signals to the driver.

Note that the digital outputs #5, #9, and #13 automatically become the direction-select signals for the Stepper controller channels #1, #2, and #3, respectively, when the stepper controllers are being used. The direction pin is turned ON when the motor moves in the negative direction and turned OFF when the stepper motor moves in the positive direction. The STEPMOVEABS command makes it extremely simple

to position the motor at an absolute location, while the STEPMOVE command lets you implement incremental moves in either direction for each channel.

10.3.1 Interfacing to 5V Stepper Motor Driver Inputs

Some stepper motor drivers accept only 5V signals from the stepper motor controller. In such a case, you need to determine whether the driver's inputs are opto-isolated. If they are, then you can simply connect a 2.2K current limiting resistor in series with the path from the PLC's output to the driver's inputs, as shown in Figure 10.4.

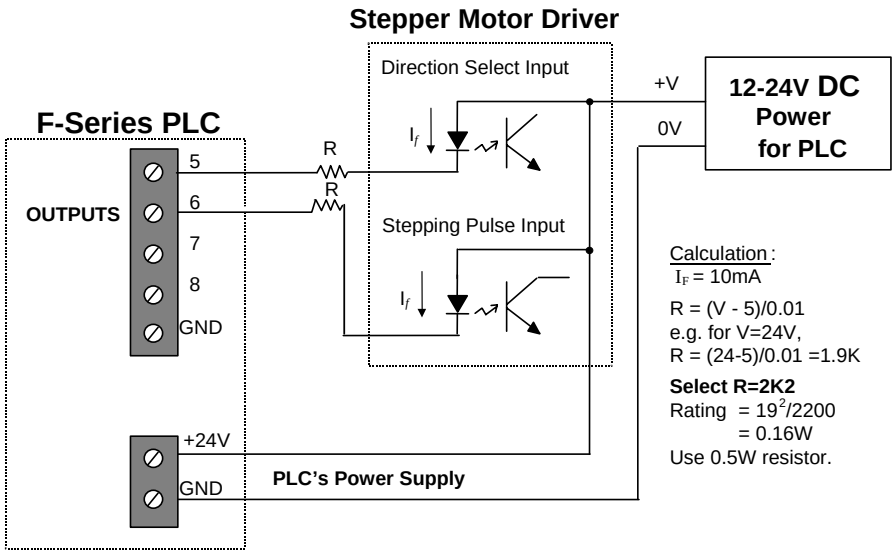


Figure 10.4

However, if the stepper motor driver input is only 5V CMOS level and non opto-isolated, then you need to convert the 12-24V outputs to 5V. This can be achieved using a low cost transistor such as a 2N4403. A better way is to use an opto-isolator with a logic level output, as shown in Figure 10.4. This provides a galvanic isolation between the PLC and the stepper motor driver.

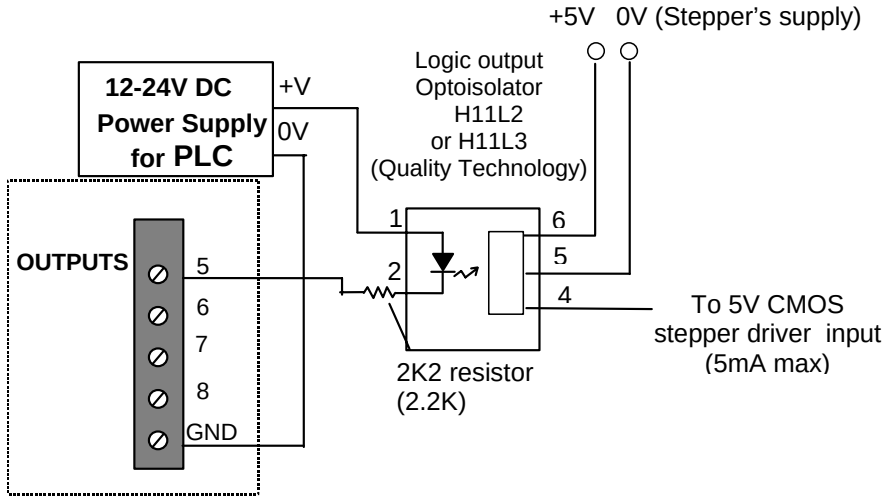


Figure 10.5 Conversion of F-Series outputs to 5V logic level

10.4 Programming Stepper Control Channel

10.4.1 Introduction

Any of the 3 stepper-motor channels can be configured as either a Controller (direction/pulse) or a Driver; and as a Driver, can drive in FULL STEP mode or HALF STEP mode. The stepper channels are controlled by the PLC program using the "STEPMOVE" and "STEPMOVEABS" commands. These commands have the same parameters as they did when they were used on M-Series PLCs. For example,

STEPMOVE *ch, count, r*
STEPMOVEABS *ch, position, r*

The *ch* parameter is the channel. This is how the PLC knows which stepper motor channel to turn on. Now this parameter is also used to set the stepper motor channel to controller mode, full-step driver mode, or half-step driver mode.

To set the stepper motor channel to controller mode, the *ch* parameter should be a "1", "2", or "3". This is the same as for the M-Series PLCs, except that channel 3 is now available.

To set the stepper motor channel to driver mode, the *ch* parameter needs to have a "1" prefixed to it for full-step mode and a "2" prefixed for half-step mode.

10.4.2 Full-Step Driver Mode:

```
STEPMOVE 11, 100, 1    'Stepper channel 1 will be driven 100 full steps
STEPMOVE 12, 100, 1    'Stepper channel 2 will be driven 100 full steps
STEPMOVE 13, 100, 1    'Stepper channel 3 will be driven 100 full steps
```

10.4.3 Half-Step Driver Mode:

```
STEPMOVE 21, 100, 1    'Stepper channel 1 will be driven 100 half steps
STEPMOVE 22, 100, 1    'Stepper channel 2 will be driven 100 half steps
STEPMOVE 23, 100, 1    'Stepper channel 3 will be driven 100 half steps
```

The same format should be used for STEPMOVEABS, except that the count parameter is changed to the position parameter (more details provided in the Programmers Reference manual and also in the program examples section of this chapter).

10.4.4 Setting the Acceleration Properties

Whether the stepper motor channels are being used to control a stepper motor driver or to drive the motor directly, the STEPSPEED command must be executed first in order to define the acceleration settings. Once this command has been executed, the acceleration settings will not change until another STEPSPEED command is executed with different settings or if the PLC is powered down. If the PLC is powered down, the STEPSPEED command will need to be executed again before the stepper motor channels can be used. However, if a software reset executed (from online monitoring or within the program), the STEPSPEED command does not need to be re-executed.

STEPSPEED *ch, pps, acc*

The *ch* parameter should be a value between 1 and 3. Speed, *pps*, is based on the no. of pulses per second (*pps*) output by the pulse generator. The *pps* parameter should be set to a value between 1 and 10000 (max rated *pps* for the Fx2424). The acceleration, *acc*, determines the total number of steps taken to reach full speed from a standstill and the number of steps from full speed to a complete stop. The stepper motor controller calculates and performs the speed trajectory according to these parameters when the STEPMOVE or STEPMOVEABS commands are executed.

To set stepper motor channel #3 to a speed of 100 pps in 50 steps, the command would be as follows:

```
STEPSPEED 3, 100, 50
```

This would be equivalent to an acceleration of 100 pulses/s2, which can be calculated using the following expression:

$$A = V^2/2S = 100^2/2*50 = 100pps^2$$

10.4.5 Using the STEPMOVE Command

Once the STEPSPEED command is executed, the STEPMOVE command can be used to move the stepper motor forwards or backwards. The STEPMOVE command is also used to specify whether the PLC is a motor controller, a full-step motor driver, or a half-step motor driver.

STEPMOVE *ch, count, r*

The *ch* parameter specifies which stepper motor channel is being used and whether it is being used as a controller, full-step driver, or half-step driver.

The *count* parameter specifies how many pulses the motor will move. If the motor were in half-step driver mode, then count would be in half steps.

The *r* parameter specifies which internal relay will be activated once the motor has moved *count* number of steps. This relay would be cleared when the STEPMOVE command is executed in case it was already activated.

10.4.5.1 Example: PLC as a Motor Controller

In this example, the PLC would be a controller and would send the control pulse and direction pulse to a driver board.

Moving Forwards

```
STEPMOVE 1, 500, 101   ' channel 1 would send 500 pulses to a driver and then turn on relay 101
STEPMOVE 2, 750, 102   ' channel 2 would send 750 pulses to a driver and then turn on relay 102
STEPMOVE 3, 200, 103   ' channel 3 would send 200 pulses to a driver and then turn on relay 103
```

Moving Backwards

```
STEPMOVE 1, -500, 101 ' channel 1 would send 500 pulses to a driver and then turn on relay 101
STEPMOVE 2, -750, 102 ' channel 2 would send 750 pulses to a driver and then turn on relay 102
STEPMOVE 3, -200, 103 ' channel 3 would send 200 pulses to a driver and then turn on relay 103
```

10.4.5.2 Example: PLC as a Full-Step Motor Driver

Moving Forwards

```
STEPMOVE 11,8000,101     ‘ channel 1 would send 8000 pulses to a motor and then turn on relay 101
STEPMOVE 12,50,102         ‘ channel 2 would send 50 pulses to a motor and then turn on relay 102
STEPMOVE 13,900,103       ‘ channel 3 would send 900 pulses to a driver and then turn on relay 103
```

Moving Backwards

```
STEPMOVE 11,-300,101      ‘ channel 1 would send 300 pulses to a motor and then turn on relay 101
STEPMOVE 12,-1000,102   ‘ channel 2 would send 1000 pulses to a motor and then turn on relay 102
STEPMOVE 13,-7500,103   ‘ channel 3 would send 7500 pulses to a motor and then turn on relay 103
```

10.4.5.3 Example: PLC as a Half-Step Motor Driver

Moving Forwards

```
STEPMOVE 21,8000,101   ‘ channel 1 to send 8000 forward pulses to motor and turn on relay 101
STEPMOVE 22,50,102       ‘ channel 2 to send 50 forward pulses to a motor and turn on relay 102
STEPMOVE 23,900,103     ‘ channel 3 to send 900 forward pulses to motor and turn on relay 103
```

Moving Backwards

```
STEPMOVE 21,-300,101      ‘ channel 1 to send 300 backward pulses to motor and turn on relay 101
STEPMOVE 22,-1000,102   ‘ channel 2 to send 1000 backward pulses to motor and turn on relay 102
STEPMOVE 23,-7500,103   ‘ channel 3 to send 7500 backward pulses to motor and turn on relay 103
```

10.4.6 Using the STEPMOVEABS Command

This command allows you to move the stepper motor # *ch* to an absolute position indicated by the position parameter. At the end of the move the relay # *r* will be turned ON. Position can be between -2^{31} to $+2^{31}$ (i.e. about ± 2 x 10^9). The absolute position is calculated with respect to the last move from the "HOME" position. (The HOME position is set when the STEPHOME command is executed). The speed and acceleration profile are determined by the STEPSPEED command as in the original command set.

This command automatically computes the number of pulses and the direction required to move the stepper motor to the new position with respect to the current location. The current location can be determined at any time by the STEPCOUNTABS( ) function.

Once the STEPMOVEABS command is executed, re-execution of this command or the STEPMOVE command will have no effect until the entire motion is completed or aborted by the STEPSTOP command. The STEPMOVEABS command is also used to specify whether the PLC is a motor controller, a full-step motor driver, or a half-step motor driver.

STEPMOVEABS *ch, position, r*

The *ch* parameter would specify which stepper motor channel is being used and whether it is being used as a controller, a full-step driver, or a half-step driver.

The *position* parameter specifies how many pulses the motor will move relative to its home position. If the motor were in half-step driver mode, then *position* would be in half steps.

The *r* parameter specifies which internal relay will be activated once the motor has moved to its new position. This relay would be cleared when the STEPMOVEABS command is executed in case it was already activated.

10.4.6.1 Example: PLC as a Motor Controller

In this example, the PLC would be a controller and would send the control pulse and direction pulse to a driver board.

```
STEPMOVEABS 1,500,101     ‘ Stepper #1 to move fwd 500 steps from home and turn on relay 101
STEPMOVEABS 2,-75,102     ‘ Stepper #2 to move back 75 steps from home and turn on relay 102
STEPMOVEABS 3,200,103     ‘ Stepper #3 to move fwd 200 steps from home and turn on relay 103
```

10.4.6.2 Example: PLC as a Full-Step Motor Driver

```
STEPMOVEABS 11,300,101 ‘ Stepper #1 to move fwd 300 pulses from home and turn on relay 101
STEPMOVEABS 12,100,102 ‘ Stepper #2 to move fwd 100 pulses from home and turn on relay 102
STEPMOVEABS 13,-90,103    ‘ Stepper #3 to move back 90 pulses from home and turn on relay 103
```

10.4.6.3 Example: PLC as a Half-Step Motor Driver

```
STEPMOVEABS 21,300,101    ‘ Stepper #1 to move fwd 300 pulses from home and turn on relay 101
STEPMOVEABS 22,-10,102     ‘ Stepper #2 to move back 10 pulses from home and turn on relay 102
STEPMOVEABS 23,50,103      ‘ Stepper #3 to move fwd 50 pulses from home and turn on relay 103
```

10.4.7 Demo Program for Stepper Motor Control

A demo program for programming the Stepper Motor Controller and Driver:

“Stepper Motor - F2424.PC6”

can be found in the FPLCsamples.zip file which can be downloaded from:

http://www.tri-plc.com/trilogi/FPLCsamples.zip

Chapter 11 Pulse Width Modulated Outputs

11 PULSE WIDTH MODULATED OUTPUTS

11.1 Introduction

Pulse-Width Modulation (PWM) is a highly efficient and convenient way of controlling output voltage to devices with large time constants, such as controlling the speed of a DC motor, the power to a heating element, or the position of a proportional valve.

The PWM works by first turning on the output to full voltage for a short while and then shutting it off for another short while and then turning it on again, and so on, in consistent and accurate time intervals. This can be illustrated with the following diagram:

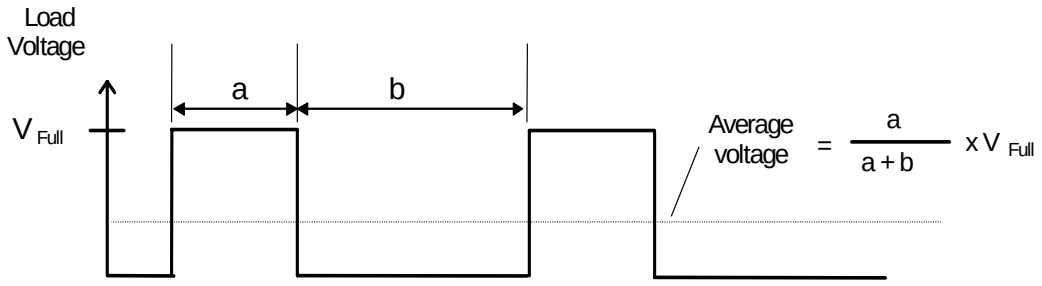


Figure 11.1

The average voltage seen by the load is determined by the "duty cycle" of the PWM waveform. The duty cycle is defined as follow:

$$\text{Duty Cycle} = \frac{a}{a + b} \times 100\%$$

Period = (a + b)
Frequency = 1/period Hz

Average voltage = % duty cycle multiplied by the full load voltage $V_{Full.}$ Since the voltage applied to the load is either "Fully ON" or "Fully OFF", it is highly efficient because the switching transistors are working in their saturated and cut-off region and dissipate very little power when it is fully turned ON.

11.2 Fx2424 PLC PWM Outputs

Technical Specifications:

No. of Channels	4
Duty Cycle range	0.00 to 100.00
Worst case resolution	0.1%
Available Frequencies (Hz) % Frequency Errors :	50Hz to 50 KHz, < ± 0.01% @ 100Hz < ± 0.5% @ 10KHz < ± 2% @ 50KHz
Relevant TBASIC commands	**SETPWM**

Unlike in the T100M+ PLCs, which only support 8 fixed frequencies settings, the PWM channels in the Fx2424 PLC can generate pulses with frequency ranging from 1Hz all the way to 50KHz. At the lower frequency range, the output frequency can be extremely accurate (less than 0.01% error). Even at 10KHz the output frequency error is less than 0.5%. This makes it possible to use the PWM channels to generate square wave pulses of a certain frequency.

For example, in Section 17.4.3 we show you how to use the PWM output to generate the 38KHz carrier wave pulses for sending encoded IR remote commands. This enables the PLC to control your home TV, DVD player, and air conditioner!

Usually it is better to select as high a frequency as possible because the resulting effect is smoother for higher frequencies. However, some systems may not respond properly if the PWM frequency is too high, in such cases a lower frequency should be selected.

The TBASIC **SETPWM** statement controls the frequency and duty-cycle settings of the PWM channel. The Fx2424 PLC features four channels of PWM on its digital outputs #5 (PWM channel #1) to output #8 (PWM channel #4 – please see Chapter 6 for the pin assignment).

Since all four PWM outputs are high voltage, high current outputs (24V, 8A peak, 4A continuous) they can be used to directly control the speed of a small DC motor. They can also directly drive proportional (variable position) valves whose opening is dependent on the applied voltage. **Note:** When using the PWM output to drive a motor or solenoid valve, please take note of the need to add a bypass diode to absorb the inductive kick that will occur when the output current to the load is turned OFF, as mentioned in Chapter 1.5.3.

11.3 Increasing Output Drive Current (Opto-Isolated)

The advantage of using the PWM is that you can easily amplify the drive current to a larger load such as a larger permanent magnet DC motor by using a power transistor or power MOSFET to boost the current switching capability. If the load is of a different voltage and the load current is high, you should use an opto-isolator to isolate the PLC from the load, as in Figure 11.2

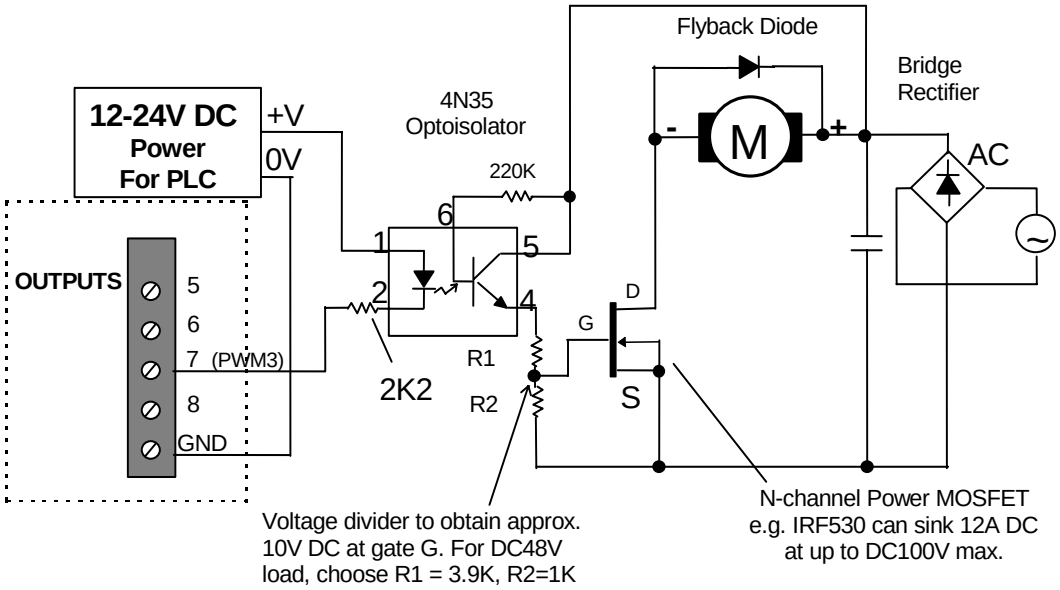


Figure 11.2 PWM Speed Control of a large DC Motor.

Note:

1. The opto-isolator must be able to operate at a frequency matching that of the PWM frequency, otherwise the resulting output waveform will be distorted and effective speed control cannot be attained.
2. The simple PWM speed control scheme described above is the open-loop type and does not regulate the speed with respect to changing load torque. Closed-loop speed control is attainable if a tachometer (either digital or analog) is used which feeds back to the CPU the actual speed. Based on the error between the set point speed and the actual speed, the software can then adjust the PWM duty cycle accordingly to offset speed variation caused by the varying load torque. A PID function may also be invoked to provide sophisticated PID type of speed control.
3. The Fx2424's PWM can be used to control the speed of small motors only (up to the maximum current limit that the PLCs output can safely drive). For larger motors, industrial-grade variable-speed drives should be used instead.

11.4 Position Control Of RC Servo Motor

RC Servo is a class of DC servo motor commonly used in remote control (hence the term RC) for positioning a device at a desired location. It is often termed "proportional control" because the position where the motor will turn to is directly proportional to the pulse width of the control signal. When chosen appropriately, RC Servo can provide an extremely inexpensive and versatile solution for positioning a device. For example, for controlling the percentage opening of a HVAC damper, or to rotate the angle of window blinds or to position a solar panel to track the sun light. There are many sizes of RC Servo available in the market, from those that weigh just a few grams to those for controlling an industrial scale unmanned vehicle (e.g. UAV or unmanned submarine). A small, self-contained RC servo typically cost **less than $20** retail price and is incredibly easy to control using the PWM output on the Fx2424.

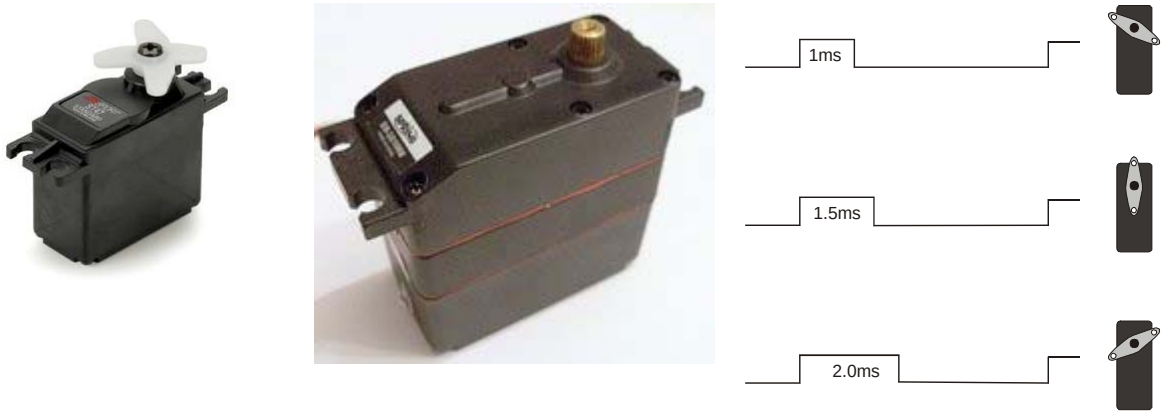


Figure 11.3 RC Servo and Control Signal

RC Servo typically only have 3 wires: Power " + " (typically 4.8 to 6V), " – " and a "Control" input.

To position the RC Servo to a position within its range of travel (Some are 0 to 90 degree and there are those that can go from 0 to 180 degree), you send a positive pulse with pulse width between 1.00 ms to 2.00 ms to the "Control" input once every 20ms. The servo will position the actuator to one end when it receives a pulse of 1.00ms and the other end when it receive a pulse of 2.00ms. If you send it a pulse of 1.50ms it will position the actuator to the center.

In PWM term, 1.00ms pulse width every 20ms means a duty cycle of 1.00/20.00 = 5.00%. 2.00ms pulse width every 20ms means a duty cycle of 2.00/20.00 = 10.00% duty cycle. So to control the position of the actuator one only needs to send it a positive PWM signal with frequency = 1/0.02 = 50Hz and duty cycle between 5% and 10%.

The gear and servo feedback mechanism within the RC Servo produces a huge amount of torque relative to its weight for positioning the actuator to the position determined by the pulse width at the "control" input. Yet once it is in the correct position the servo draws only minimum current required to maintain its position. Hence being a closed-loop controller an RC Servo is actually a much more efficient and effective positional control device (for a limited range) than a stepper motor which relies on a constant current in its winding to provide the "holding torque" for positioning. The open loop nature of stepper motor means that it does not know if the device is actually being knocked out of its desired location whereas in the Servo any deviation from its desired location is instantly being corrected by the servo mechanism.

11.4.1 Using Fx2424 PWM Output To Control RC Servo (Non-Isolated)

As you probably have realized by now that you can use a single PWM output to very easily position the RC Servo to whatever position within its range of travel.

However, since the Fx2424 power supply is 12 to 24V (vs 4.8 to 6V on the typical RC Servo) and the PWM output is NPN (current sink) type, the signal to the servo is actually inverted if you connect the PLC's output to the Servo's "control" input directly.

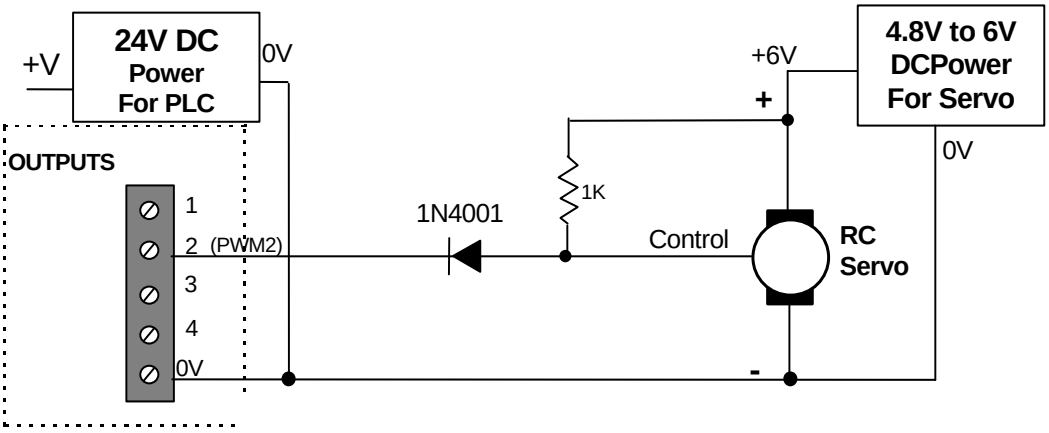


Figure 11.4 Non-isolated Interace to RC Servo

As shown in the above figure, we use a 1K ohm resistor to pull up the "Control" input to the RC Servo's power supply so that when the PLC's output is OFF, the Control input is +6V and when the PLC output is ON, the "Control" input is pulled to low.

The 1N4001 diode is to prevent the 24V weak pullup signal at the PLC output from entering the servo's "Control" input. As such, when the PLC output is ON, the "Control" input is pulled down to about 1 diode drop (about 0.7V).

In other words, the RC Servo connected above are controlled by the PLC's PWM output but the duty cycle is inverted. I.e. To send a 5% positive PWM control pulse to the Servo, you can run the following statement:

```
SETPWM 2, 9500, 50     ' Set PWM 2 output to 95% duty cycle at 50 Hz
```

Likewise, to send a 10% positive PWM control pulse to the Servo, you will need to run the following statement:

```
SETPWM 2, 9000, 50     ' Set PWM 2 output to 90% duty cycle at 50 Hz
```

11.4.2 Using Fx2424 PWM Output To Control RC Servo (Opto-Isolated)

You can also use the PLC's PWM output to drive an optocoupler (such as 4N35) and the output from the optocoupler is use to provide control pulse to the RC Servo, as shown in Figure 11.5

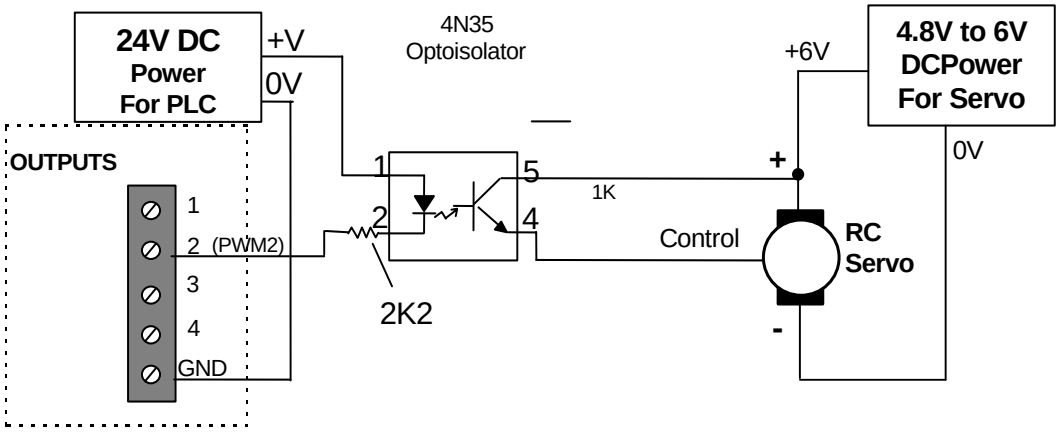


Figure 11.5 Opto-Isolated Interace to RC Servo

In the opto-isolated interface shown above, the power supply to the PLC and that to the RC Servo are completely isolated (no common ground required). Also since the interface inverts the output signal from the PLC, no software inversion is necessary. i.e When the PLC NPN output is OFF, the "control" input to the RC Servo will be OFF, and when the PLC NPN output is ON, the "control" input to the RC Servo = +6V.

Hence, to send a 5% positive PWM control pulse to the Servo, you can run the following statement:

```
SETPWM 2, 500, 50     ' Set PWM 2 output to 5% duty cycle at 50 Hz
```

Likewise, to send a 10% positive PWM control pulse to the Servo, you can run the following statement:

```
SETPWM 2, 1000, 50     ' Set PWM 2 output to 10% duty cycle at 50 Hz
```

Of course you can replace the duty cycle with a value (e.g. DM[1]) and run statement such as:

```
SETPWM 2, DM[1], 50   ' Set PWM 2 output to DM[1]/100% duty cycle at 50 Hz
```

11.4.3 RC Servo Positioning Resolution

Since the resolution on the Fx2424 PLC's PWM output is precise to 0.01% at 50Hz, this means that you can get a maximum of **500** discrete positions within the travel range of the RC Servo. This should be sufficiently accurate for many applications such as HVAC damper control or control of proportional valves. The actual positioning accuracy and precision, however, will depend on the quality of the RC Servo.

Chapter 12 Real Time Clock

12 REAL TIME CLOCK

12.1 Introduction

A Real Time Clock (RTC) is a device that keeps accurate date and time information down to the second. The RTC in the Fx2424 is battery backed so that time will continue to run if PLC power is lost with or without warning.

On the FMD Series and Nano-10 PLCs, it is not possible to keep the RTC running after power loss unless either the FRAM-RTC-0 or FRAM-RTC-256 module is installed. However, the FRAM-RTC modules are not compatible and are not required with the Fx2424 because a battery is used to power the on-board RTC. Due to safety restrictions by air-transportation, the battery may or may not be included with your Fx2424. If included, it may not be installed on the PLC in the factory and, in that case, you will need to install it yourself. Please refer to Section 1.9 for information on both purchasing and installing the RTC battery.

NOTE:

If the RTC battery is not installed or the battery is removed for more than 15 seconds when the PLC is powered down, then the PLC will lose its real time clock data when it is powered up again. When this happens the "RTC Error" status LED on the CPU board will light up and the RTC.Err bit in the ladder logic special bit will be ON so that user program can use it to alert the operator.

12.2 TBASIC variables Used for Real Time Clock

Date		Time	
YEAR	DATE[1]	HOUR	TIME[1]
MONTH	DATE[2]	MINUTES	TIME[2]
DAY	DATE[3]	SECOND	TIME[3]
Day of Week	DATE[4]		

There are 7 registers available in TBASIC that are used to access and configure the date and time. These registers, which are shown above, can be read from and written to just like any other integer variable. The data for these registers are in integer format.

`DATE[1]` : may contain four digits (e.g. 1998, 2003 etc).
`DATE[4]` : 1 for Monday, 2 for Tuesday, ... 7 for Sunday.

12.3 RTC Error Status On Ladder Logic

There is a special bit available in TRiLOGI that allows you to notify the PLC program if the RTC Error event occurs and the RTC Error status light is turned on. The RTC Error event occurs if the RTC is corrupted or damaged (see section 12.8 for more detail) or if the battery is not installed. The special bit is called RTC.Err and can be obtained from the “Special Bits” I/O Table. The RTC.Err contact can be used

to activate an alarm of some kind. The following ladder logic circuit is an example of this using the RTC.Err bit as an input that controls an output called RTC_Alarm:

In the circuit above, RTC.Err is a special bit that cannot be renamed in your program. The output RTC_Alarm is a user-defined output that could be named to anything. If the RTC Error event occurs for any reason, the RTC.Err bit would activate the RTC_Alarm output.

12.4 Setting the RTC Using TRiLOGI Software

The RTC date and time can be easily set within TRiLOGI by selecting “Set PLC’s Real Time Clock” from the “Controller” menu. A window will pop up with default values entered as shown below in Figure 12.1. All of these values can be edited and then written to the PLC by clicking on “Set PLC’s Clock”

Figure 12.1: Set Real Time Clock

12.5 Setting the RTC Using TBASIC

The PLCs RTC can be set from TBASIC using the DATE[] and TIME[] registers shown in section 12.2. Figure 12.2: Set Date & Time in TBASIC, shown below, is an example of a custom function where the date is set to October 1st 2008, the day of the week is Wednesday, and the time is 14:30:01 (2:30:01 pm).

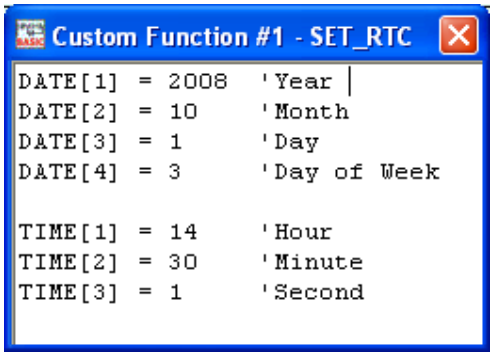


Figure 12.2: Set Date & Time in TBASIC

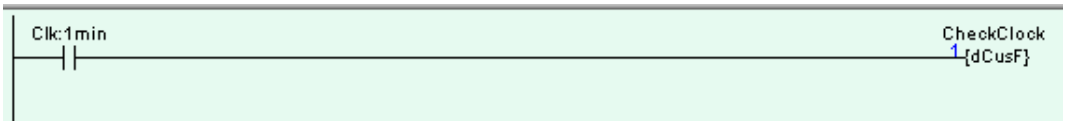
12.6 Setting the RTC from Internet Time Server

Please refer to Section 2.6 for more detailed information on how your PLC may be able to automatically set its own real time clock using the timeserver data available on the Internet. A sample program is also included in that section.

12.7 Setting up an Alarm Event in TBASIC

Since, to the TBASIC program, the RTC data are simply integer arrays DATE[1] to DATE[4] and TIME[1] to TIME[3], they are fully accessible at any time by your PLC program. Therefore, if you want your program to execute a certain routine on a specific date and or/time, you would need to periodically check these variables against the desired settings and activate the action when the RTC variable(s) reach the set value.

Example: Set up a 1 minute clock pulse to monitor the RTC as follows:



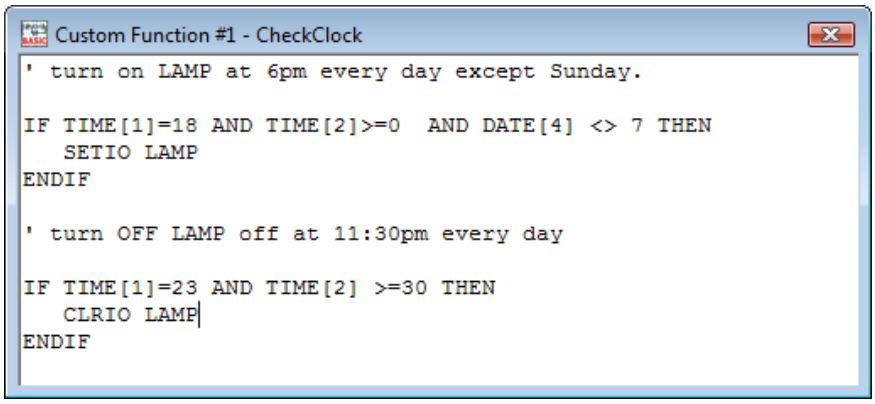
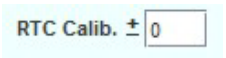


Figure 12.3: Using Real Time Clock Example

12.8 RTC Calibration

The Fx2424 PLC uses a battery-back real time clock that derived its clock from a 32.768KHz crystal, which should provide reasonably good accuracy for normal use. However, if you like the RTC to be of greater accuracy, you can calibrate it using the “F-series Ethernet Advanced Configuration” screen in the PLC configuration software mentioned in Chapter 2. The data field to be used is the “RTC Calib” textbox as shown below:



In the above field you will need to enter the number of seconds that you want the PLC to add or subtract over a period of 72 hours. Therefore, first you must check the PLC’s RTC reading against a super

accurate clock source (e.g. a clock that regularly updates itself with atomic clock data in the airwaves) and find out how many seconds the clock would have gained or lost over 72 hours.

For example, if the RTC is too slow and it loses 5 seconds over 72 hours, you would want the RTC to add 5 seconds over 72 hours and you therefore should enter the value of +5 in the “RTC Calib ± “ field. If the RTC is too fast and gains 8 seconds over the 72 hours, then you should enter a value of “-8” in this field to compensate for the inaccuracy.

Note that the RTC does not compensate for temperature variation; hence, its accuracy is temperature-dependent. If you require the PLC’s RTC to be very accurate you should keep the PLC under a constant operating temperature.

If your Fx2424 is connected to a LAN, then you also have the option of having the RTC periodically set itself using the very accurate RTC data that can be obtained from a time-server on the Internet (See Section 12.6).

12.9 Troubleshooting the RTC

If the RTC Error light comes on and the battery is properly installed, then the RTC could be corrupted or damaged. This could happen if there was any damage to the components on the board, such as I/O drivers, communications drivers, or any of the IC’s or PCB circuitry. Also, if a voltage spike came into the PLC through its I/O, power supply, or communications ports, that could cause some corruption even if there was no component damage.

If the RTC is only corrupted, then you should only have to reset the RTC to resolve the problem. This can be done most easily by using the TRiLOGI method of setting the RTC that was described in section 12.4. After setting the RTC, the PLC should be powered off and then powered on again. The RTC Error light should be off at this point, but if it isn’t, then there is a possibility of damage to the PLC. At this point you should contact Triangle Research tech support or if you have purchased the unit from a local distributor, then you should contact the distributor for assistance.

Chapter 13 LCD Display Programming

13 LCD DISPLAY PROGRAMMING

13.1 SETLCD Command

The SETLCD *y, x, string* TBASIC command allows you to easily display any string of up to 20 characters on the y^{th} line starting from the x^{th} column. E.g., to display the message "T100MD-888+" on the 3rd line starting from the 5th character position from the left end of the screen, you use the command:

```
SETLCD 3, 5, "T100MD-888+"
```

Normally, y = 1,2,3, 4; x = 1, 2, …. 20. Integers must be converted to strings using the STR$() or HEX$() function before they can be displayed using SETLCD. You can use the concatenation operator "+" to combine a few components together in the command. E.g.

```
SETLCD 1,1,"Rm Temp = "+ STR$(ADC(1)/100,3)+CHR$(223)+"C"
```

The function STR$(ADC(1)/100,3) reads the content of ADC channel #1, divides it by 100 and converts the result into a 3-digit string. The CHR$(223) appends a special character which corresponds to the ' ^{o} ' symbol. E.g. if ADC(1) returns the value 1234, the final string being displayed will be :

Rm Temp = 012 ^{o}C.

13.2 Special Commands For LCD Display

If you use the SETLCD command with line #0, then the strings will be treated as special "instructions" to be sent to the LCD module to program it for various modes of operation. This includes: blinking cursor, underline cursor or no cursor, as well as display shift mode. You have to refer to the LCD manufacturer's data sheet for the detailed commands. Some of the most useful commands are listed below:

Action	Command
1. Clear screen	SETLCD 0,1, CHR$(1)
2. No cursor	SETLCD 0,1, CHR$(12)
3. Underline Cursor	SETLCD 0,1, CHR$(14)
4. Blinking Cursor	SETLCD 0,1, CHR$(13)
5. Underline + Blinking Cursor	SETLCD 0,1, CHR$(15)

13.3 Displaying Numeric Variable With Multiple Digits

The "SETLCD *y, x, string"* command only overwrites the exact number of characters in the *string* parameter to the LCD display, and thereafter the cursor is placed back to the location specified by the *x* and *y* parameters. Thus if there exists some old characters right after the last character it can cause confusion, especially if you are displaying a number. E.g. If you first display the following string at row 1 and column 1:

```
Pressure = 12345
```

And if you subsequently display the string "Pressure = 983" at the same location without first clearing the line, then you will see the following string being displayed:

```
Pressure = 98345
```

What happens is that the string "`Pressure = 983`" is correctly displayed but the two old characters "45" left over from previous display would appear to be part of the new data. This can cause confusion.

There are several ways you can eliminate such a display problem:

1) Clear the line first before overwriting with a new string. You can create a custom function just to clear a particular line. E.g. if you pass the parameter DM[100] to the custom function and inside the custom function you do the following:

```
SETLCD DM[100],1, "                    "
```

   Hence calling this custom function with DM[100] = 1,2,3, or 4 would clear the corresponding line.

2) A " quick and lazy" way to do it is to add a few more characters to the back of the string to be displayed which will wipe out old characters that could be present adjacent to the new string.

```
E.g.  SETLCD 1,1, "Money = $"+ STR$(D) + "      "
```

3) If there is data to the right of the currently displayed string that you cannot overwrite with spaces, then you can restrict the number of digits that a numeric variable may be converted to using the two parameters from the STR$ or HEX$ command.

```
E.g SETLCD 1,1, "Temp = "+ STR$(T,4)
```

   The STR$(T,4) function will always return 4 digits of the data stored in T. If T is less than 4 digits, then one or more preceding "0"'s will be added. E.g. if T = 12 then this function will return the string "0012. Note that for negative numbers the negative sign is counted as part of the digit count so you need to provide enough spaces to take care of the sign if you need to handle both positive and negative numbers.

13.4 Displaying Floating-point

The STR$(x#) and STR$(x#,n) can convert a floating point number x# into a text string which can therefore be displayed immediately using the SETLCD command. Please refer to the STR$() command in the TL7 Programmer's Reference guide for details.

If you wish to control the number of digit being displayed then you could use the MID$ command to extract the number of significant digits and the exponents from the string returned by STR$(x#, n) function.

Chapter 14 Serial Communications

14 SERIAL COMMUNICATIONS

14.1 Introduction:

There are 3 serial ports on the Fx2424, including 2 RS485 ports and 1 RS232 port. All three of these serial ports have full Modbus ASCII/RTU and Host Link Protocol drivers so that all three serial ports can be used independently. All three serial ports on this PLC can also be programmed to accept or send ASCII or binary data using the TBASIC built-in commands such as INPUT$(n), INCOMM(n), PRINT #n, OUTCOMM n, d.

The first serial port (COMM1) is an RS232C port, which is compatible with most PC's RS232C ports. The second serial port (COMM2) is a two-wire RS485 port that allows multiple PLCs to be connected to a single host computer or a master PLC for networking or to implement a distributed control system. The third serial port (COMM3) is identical to the second one (COMM2).

Note: Each of the 3 serial ports has its own pair of comm. LED indicators that will light up whenever the corresponding serial channel sends or receives data. This makes it extremely convenient for troubleshooting serial communication problems.

14.2 COMM1: RS232C Port with Female DB9 Connector

This port is configured as a DCE (Data Communication Equipment) and is designed to connect directly to the PC's serial port without the need for a null modem cable. COMM1 communicates with the host computer at a default baud rate of *38,400* bit-per-second with 8 data bits, 1 stop bit and no parity. The communication baud rate can be changed in software from 1200 bps to 115.2K or 230.4K bps, which is compatible with most PCs and industrial equipment.

The COMM1 port is the main communication port used by the PLC configuration software (See Chapter 2.1) as well as for program transfer and on-line monitoring of the PLC program when it is not connected to an Ethernet router or hub. The pin connections with the host PC are shown below:

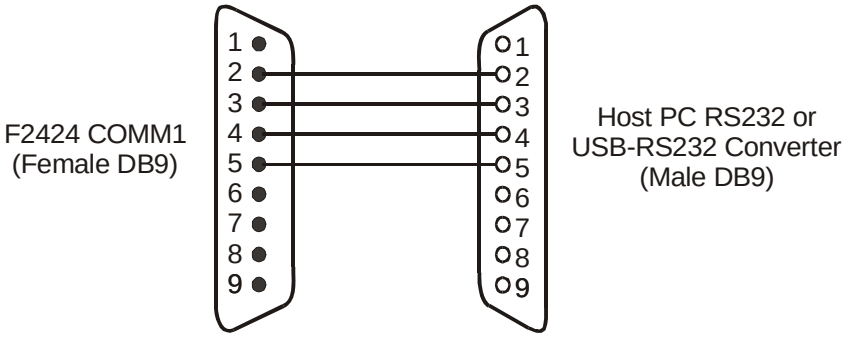


Figure 14.1 Connecting COMM1 with PC

However, to connect COMM1 to another DCE device (e.g., a modem), you need to make a special cable which swaps the transmit and receive signals, as follows:

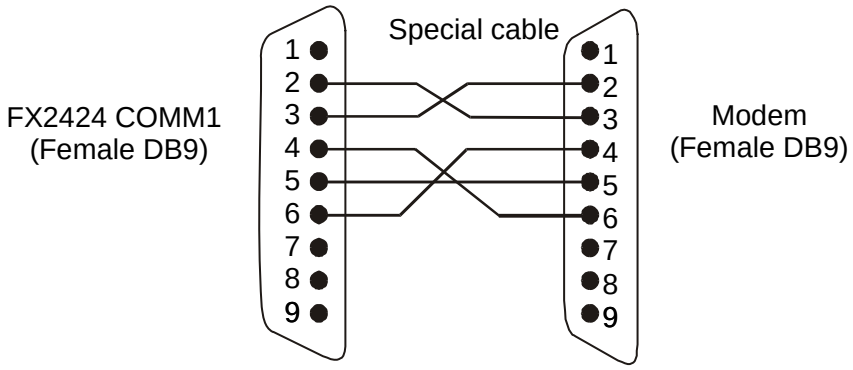


Figure 14.2 Connecting COMM1 to a MODEM

Pin 4 and 6 are handshaking signals whose presence may be required by some modems to work properly, so these pins are connected as shown in the diagram.

14.3 COMM2 / COMM3: Two-wire RS485 Port

These two half-duplex RS485 ports are meant for serial bus type networking or for connecting to optional peripherals such as a serial LCD message-display unit (e.g. MDS100-BW), external analog modules (e.g. I-7018), touch panel HMI, or for inter-communication between PLCs.

Up to 255 RS232/RS485 peripheral devices may be linked together in an RS485 network on both COMM2 and COMM3.

The RS485 ports are available on 2-way screw terminals between the RS232 (COMM1) port and the analog I/O DB15 connector (please refer to Chapter 1- Installation Guide). For successful communication using either of the RS485 ports, you need to correctly connect the '+' and '-' terminals to the RS485 equipment using a twisted pair cable. If you are using the PC as the network host, you will need an RS232C-to-RS485 converter such as the "Auto485".

The following describes some possible uses of the RS485 ports.

14.3.1 PROGRAMMING AND MONITORING

A Fx2424 can be programmed via its RS485 port on a one-to-one or multi-drop manner. Since most PCs only have RS232 port(s) (or USB ports for which USB-to-RS232 converters are readily available), you need to purchase an RS232-to-RS485 converter in order to program the PLC via its COMM2 and COMM3 port. The most commonly available types of RS485 converters today use the RTS signal to control the RS485 transmitter direction, which is supported by the TLServer software. However, we strongly recommend an auto-turnaround type of converter such as the Auto485 adapter (configured in 'Auto' mode) for use with Windows programs. Under Windows, the application software does not have very deterministic control of the RTS pin at precise timing and that can contribute to the occasional loss of communication when the RS485 transceiver is not switched at the proper moment.

Programming via COMM2 or COMM3 is particularly useful if COMM1 is already assigned to other tasks such as interfacing to modem, bar code readers, a SCADA system or MMI *. The programmer can continue to program and monitor the PLC using its RS485 port while its COMM1 is actively communicating with other devices. This makes it much easier to troubleshoot communication problems

at COMM1 since you can continuously monitor the data exchange between the PLC and the external devices connected to its COMM1.

* Of course, since the Fx2424 PLC has a built-in Ethernet port, you can always program it via the local area network. But the fact that all 3 serial ports can be programming ports makes them truly flexible for many industrial applications.

14.3.2 Accessing 3rd Party RS485-based Devices

There are more and more industrial devices such as electric power meters, analog I/O modules (e.g. the I-70xx modules by ICPDAS), variable frequency drives, etc that allows data communication via their RS485 port. The Fx2424 PLC can use either COMM2 or COMM3 port to access these devices. The PLC has many built-in commands for reading/writing to these serial ports, including built-in commands for communicating with devices that use the MODBUS ASCII or RTU protocols.

14.3.3 Interfacing Other Devices to Modbus Host or to the Internet

Since the Fx2424 supports MODBUS protocols, it can operate as a master PLC and serve as a gateway to interface non MODBUS-enabled PLCs (such as the E10+ PLCs, or the I-7000 analog modules) to third party SCADA software or MMI hardware that speaks MODBUS. The Fx2424 also makes it easy for these devices to be controlled or monitored over the Internet. The master Fx2424 will use its RS485 port to pull data from these devices into its data-memory. The data memory in the Fx2424 is in turn accessible by a SCADA program using the Modbus serial or Modbus/TCP protocol. Through the Fx2424, these other connected devices are also accessible from the Internet using the i-TRiLOGI client/server software.

14.3.4 Distributed Control

Another important use of the RS485 port will be to connect a Fx2424 to other F-series, M-series, H-series or E10+ PLCs. One Fx2424 will act as the master and all other PLCs will act as slaves. Each PLC must be given a unique ID. The master will send commands to all the slaves using the “NETCMD” or READMODBUS, WRITEMODBUS, READMB2, WRITEMB2 statements and coordinate information flow between the PLCs. In this way, a big system can be built by employing multiple units of F, M, E or H-series PLCs connected in a network. This results in more elegant implementation of complex control systems and simplifies maintenance jobs.

14.4 Changing Baud Rate and Communication Formats: Use of the SETBAUD Statement

The Fx2424 PLC’s COMM ports are highly configurable. All 3 COMM ports can be set to a wide range of baud rates. You can also program them to communicate in either 7 or 8 data bits, 1 or 2 stop bits, odd, even or no parity. The baud rate and communication formats of the serial ports are set by the following command:

SETBAUD *ch, baud_no*

ch represents the COMM port number (1, 2, or 3 only). The *baud_no* parameter takes a value from 0 - 255 (&H0 to &HFF), which allows for additional configuration of the communication format. The upper 4 bits of *baud_no* specify the communication format (number of data bits, number of stop bits, and parity) and the lower 4 bits represent the baud rate. Hence the *baud_no* for 8 data bit, 1 stop bit, and no parity is the same as the old models, providing compatibility across the PLC families.

Once the new baud rate has been set, it will not be changed until execution of another SETBAUD statement. The baud rate is not affected by a software RESET but the settings will be lost when the power is turned OFF. The available baud rates and their corresponding baud rate numbers for COMM1, 2, and 3 are shown below:

Format	*baud_no*
8, 1, n	0000 xxxx
8, 1, e	0100 xxxx
8, 1, o	0110 xxxx
7, 1, n	1000 xxxx
7, 1, e	1100 xxxx
7, 1, o	1110 xxxx

Format	*baud_no*
8, 2, n	0001 xxxx
8, 2, e	0101 xxxx
8, 2, o	0111 xxxx
7, 2, n	1001 xxxx
7, 2, e	1101 xxxx
7, 2, o	1111 xxxx

Where xxxx represents the baud rate of the comm port, as follow:

x x x x	0000	0001	0010	0011	0100	0101	0110	0111
Baud Rate	2400	2400	4800	9600	19200	31250	38400	57600

x x x x	1000	1001	1010	1011	1100	1101	1110	1111
Baud Rate	100K	115.2K	230.4K	110	150	300	600	1200

A table of all the available baud rates and COMM formats is shown in the following page. The communication format written as “7,2,e”, which means 7 data bits, 2 stop bits and even parity. Likewise, “8,1,n” means 8 data bits, 1 stop bit and no parity. You can use the table to select the baud number for a certain baud rate and COMM format.

Baud No Table (All numbers in Hexadecimal: &H00 to &HFF)

Format / Baud	8,1,n	8,1,e	8,1,o	7,1,n	7,1,e	7,1,o	8,2,n	8,2,e	8,2,o	7,2,n	7,2,e	7,2,o
110	0B	4B	6B	8B	CB	EB	1B	5B	7B	9B	DB	FB
150	0C	4C	6C	8C	CC	EC	1C	5C	7C	9C	DC	FC
300	0D	4D	6D	8D	CD	ED	1D	5D	7D	9D	DD	FD
600	0E	4E	6E	8E	CE	EE	1E	5E	7E	9E	DE	FE
1200	0F	4F	6F	8F	CF	EF	1F	5F	7F	9F	DF	FF
2400	01	41	61	81	C1	E1	11	51	71	91	D1	F1
4800	02	42	62	82	C2	E2	12	52	72	92	D2	F2
9600	03	43	63	83	C3	E3	13	53	73	93	D3	F3
19200	04	44	64	84	C4	E4	14	54	74	94	D4	F4
31250	05	45	65	85	C5	E5	15	55	75	95	D5	F5
38400	06	46	66	86	C6	E6	16	56	76	96	D6	F6
57600	07	47	67	87	C7	E7	17	57	77	97	D7	F7
100K	08	48	68	88	C8	E8	18	58	78	98	D8	F8
115K2	09	49	69	89	C9	E9	19	59	79	99	D9	F9
230K4	0A	4A	6A	8A	CA	EA	1A	5A	7A	9A	DA	FA

E.g. To set the baud rate of COMM3 to 19200, 7 data bit, 1 stop bit and even parity, execute the statement: SETBAUD 3, &HC4

Important: Since the three COMM ports are independent, they can be set to different formats and baud rates from each other. Please note that if you change the baud rate or communication format to something that is different from that set in the **TLServer**, then both the TLServer and TRiLOGI will no longer be able to communicate with the PLC via this COMM port. You will have to either configure the TLServer’s serial port setting using its “Serial Communication Setup” routine to match the PLC, or you can cycle the power to the PLC to reset the COMM port to the default format (38,400, 8,n,1).

If you had used “1st.Scan” contact to activate the SETBAUD command than you will need to cycle the power to the PLC with DIP switch #4 set to ON to halt the execution of the SETBAUD command. When the PLC is reset this way, its COMM1 will power up at the default format of 38,400, 8,n,1 only so you will need to configure TLServer’s serial port to 38,400bps to communicate with it.

14.5 Support of Multiple Communication Protocols

The Fx2424 is a real communication wizard! It has been designed to understand and speak many different types of communication protocols, some of which are extremely widely used *de facto* industry standards, as follows:

a) NATIVE HOST LINK COMMAND
b) MODBUS ASCII (Trademark of Modbus.org)
c) MODBUS RTU* (Trademark of Modbus.org)
d) OMRON C20H protocols. (Trademark of Omron Corp of Japan)

The command and response formats of the “NATIVE” protocols are described in detail in Chapter 15 and 16. The other protocols and their address mapping to the Fx2424 are described in Section 14.7. The three independent COMM ports 1, 2, & 3 support all of the above protocols. Each COMM port can communicate using the same or different protocols, independent of the other. The most wonderful feature of Fx2424 is that the support of all the above-mentioned protocols can be fully automatic and totally transparent to the users. There is no DIPswitch to set and no special configuration software to run to configure the port for a specific communication protocol. The following describes how the automatic protocol recognition scheme works:

1. When the PLC is powered ON, all three COMM ports are set to the “AUTO” mode, which means that they are open-minded and listen to all serial data coming through the COMM ports. The CPU tries to determine if the serial data conforms to a certain protocol and if so, the COMM mode is determined automatically.

2. Once the protocol is recognized, the CPU sets that COMM port to a specific COMM mode, which enables it to process and respond only to commands that conform to that protocol. Error detection data such as the “FCS”, “LRC” or CRC are computed accordingly which method is used to verify the integrity of the received commands. If errors are detected in the command, the CPU responds in accordance with the action specified in the respective protocols.

3. When the COMM port enters a specific COMM mode, it will regard commands of other protocol as errors and will not accept them. Hence, for example, if COMM #1 has received a valid MODBUS RTU command (which puts it in an “RTU” mode), it will no longer respond to TRiLOGI’s attempts to communicate with it using the “NATIVE” mode. You will receive a communication error if you try to use TRiLOGI to access a PLC COMM port that has just been communicating in other protocol modes.

4. To improve the flexibility of switching from one COMM mode to another, The Fx2424 incorporates a COMM mode self-reset timer such that a specific COMM mode will time out automatically and enters into “AUTO” mode after 10 seconds if no more commands are received from that COMM port. When a user wants to switch from one COMM mode to another, he/she often will be changing the serial connector from one device to another. During this time there is no data received by the COMM port, which presents an opportunity for it to reset its COMM mode. However, the surest way to reset the specific COMM mode is to cycle the power to the PLC so that its COMM port will be reset to “AUTO” mode and ready to communicate with any supported protocols.

5. If you wish to use the COMM port for serial data input only, you can use the **SETPROTOCOL** command to set the COMM port to **NO PROTOCOL**. This can prevent the PLC from erroneously treating some serial data as the header of an incoming communication protocol and respond to it automatically.

The **SETPROTOCOL** command can also be used to set the PLC to a specific protocol. This may be desirable if the COMM port has a specific role and you do not want it to enter other modes by mistake. Please refer to the TBASIC Programmer's Reference manual for a detailed description of the SETPROTOCOL command.

Note: If you fix a COMM port to a non-native, non-auto mode, TRiLOGI will not be able to communicate with the PLC anymore. You may have to power-cycle the PLC to reset the COMM mode. If you use the "1st.Scan" contact to activate the SETPROTOCOL command, then you will need to cycle the power to the PLC with DIP switch #4 set to ON to halt the execution of the SETPROTOCOL command. (Also remember that when the PLC is reset this way, its COMM1 will power up at default format of 38,400, 8,n,1 only so you will need to configure TLServer's serial port to 38,400bps to communicate with it.)

14.6 Accessing the COMM Ports from within TBASIC

Besides responding automatically to specific communication protocols described in section 14.5, all three serial ports COMM #1, #2, and #3 are fully accessible by the user program using the TBASIC commands: INPUT$, INCOMM, PRINT # and OUTCOMM. It is necessary to understand how these commands interact with the operating system, as follow:

When a COMM port receives serial data, the operating system of the Fx2424 automatically stores them into a 256 bytes circular buffer so that user programs can retrieve them later. The serial data are buffered even if they are incoming commands of one of the supported protocols described in section 14.5. In addition, processing of a recognized protocol command does not remove the characters from the serial buffer queue so these data are still visible to the user's program.

Each COMM port has its own separate 256-byte serial-in buffer. As long as the user-program retrieves the data before the 256-byte buffer is filled up, no data will be lost. If more than 256 bytes have been stored, the buffer wraps around and the oldest data is overwritten first and so on. The following describes how INCOMM and INPUT$, PRINT # and OUTCOMM functions interact with the serial buffer:

a) INCOMM (n)

Every execution of the INCOMM(n) function removes one character from the circular buffer. When no more data is available in the buffer this function returns a -1. The data removed by INCOMM will no longer be available for the INPUT$ command.

b) INPUT$(n)

When the INPUT$(n) function is executed, the CPU checks the COMM #n buffer to see if there is a byte with the value 13 (the ASCII CR character) which acts as a terminator for the string. If a string is present, all the characters that make up the string will be removed from the COMM buffer. If a completed string is not present, then the COMM buffer will not be affected, and INPUT$(n) returns a null string. This ensures that before a complete string is received, the serial characters will not be lost because of the unsuccessful execution of the INPUT$(n) function.

c) INPUT$(n) in Blocking Mode

INPUT$(n) is designed to be non-blocking and to return immediately – i.e. either it returns a complete string or it returns an empty string. This means that INPUT$(n) will not suspend the CPU and wait for a valid string from the COMM port. However, in real world communication very often you will need to send a command to a device and it takes a while for the device to be able to send back a response string.

The most efficient way of handling such serial communication exchange with other devices is to send a command string using the PRINT #n and then start a timer and let the PLC program continue to scan the ladder program. When timer times out, the timer contact is used to activate a custom function and use INPUT$(n) to read the return message from the device. This is most efficient use of the CPU time since the program will not have to waste time to wait for response string from the device and that's the reason for INPUT$(n) to default to non-blocking mode.

Another way to deal with this is to use the NETCMD$ function (terminated with "~"). NETCMD$ sends a command string and will return immediately when it receives a response string or if more than 0.15s has passed. If the device is slow to response the CPU basically sits in a loop to wait for the response for up to a maximum of 0.15s. NETCMD$ function also re-tries the communication several times if it does not receive a response in the first try.

A third way of handling serial exchange with other devices is to use the INPUT$(n) command in **blocking mode**. Starting from CPU firmware r72 and above, if you run the command

SETSYSTEM 19, t

This will configure the INPUT$(n) command to block the CPU for up to maximum of (t x 10) milliseconds to wait for a valid string from 3rd party device. Although this command also wastes CPU cycles to wait for a response and hence it is not as efficient compared to using the timer-activated method mentioned above, it nonetheless is very simple to implement and can be used for non time-critical applications.

You only need to execute SETSYSTEM 19, t once and INPUT$(n) will block for the same t x 10 millisecond every time it is executed until SETSYSTEM 19,t is run again. To return the INPUT$(n) to non-blocking mode, run SETSYSTEM 19, 0.

d) PRINT #n

The PRINT statement transfers its entire argument to a 256 byte serial-out buffer, which is separate from the serial-in buffer. The PRINT statement, therefore, does not affect the content of the serial buffer for incoming characters. The operating system handles the actual transfer of each byte of data out of the serial-out buffer in a timely manner. Again each COMM port has its own independent 256-byte serial-out buffer and, hence, the two serial ports can operate totally independently of each other.

Note that the PLC automatically enables the RS485 transmit driver when it sends serial characters out of its COMM 2 and 3 ports. When the stop bit of the last character in the serial-out buffer has been sent out, the operating system immediately disables the RS485 driver and enables the receiver. This greatly eases the use of the RS485 port since there is no need for a user to bother with the often-critical timing of controlling the RS485 driver/receiver direction.

e) OUTCOMM

This command sends only a single byte out of the serial COMM port without going through the serial out buffer. For RS485 port, it enables the RS485 transmitter before sending the character and disables it immediately after the stop bit has been sent out.

14.7 Using The PLC As a Modbus / Omron Slave – SCADA, HMI Applications

The Fx2424 support a subset of the OMRON™ C20H and MODBUS™ (Both ASCII and RTU modes are supported) compatible communication protocols so that it can be easily linked to third-party control software/hardware products such as SCADA software, LCD touch panels etc. The PLC automatically recognizes the type of command format and will generate a proper response. These are accomplished without any user intervention and without any need to configure the PLC at all!

Both MODBUS and Omron protocols use the same **device ID address** (00 to FF) as used by the native "Hostlink Command" protocol described in Chapter 15. Since the addresses of I/O and internal variables in the Fx2424 are organized very differently from the OMRON or Modicon PLCs, we need to **map** these addresses to the corresponding memory areas in the Modbus address space so that they can be easily accessed by their corresponding protocols.

All I/Os, timers, counters, internal relays, data memory DM[1] to DM[4000], and floating point data FP[1] to FP[1000] are mapped to the Modbus Holding Registers space. The Inputs, Outputs, Relays, Timers and Counters bits are mapped to the MODBUS Bit address space as shown in Table 14.1. Note that input and output bits are always mapped according to Table 14.1 whether it is MODBUS function 01, 02 or 05.

However, 32 bit variables and string variables are not mapped since they are fundamentally quite different in their implementation among different PLCs. Internal variables that are not mapped can be still be accessed by copying the contents of these variables to unused data memory DM[n] (this can be easily accomplished within a CusFn) so that they can be accessed by these third party protocols.

14.7.1 MODBUS ASCII Protocol Support

The Fx2424 support MODBUS ASCII protocols with the following command and response format:

START	Address	Function	Data	LRC Check	CRLF
:	2 chars	2 chars	# chars	2 chars	2 chars

The following Function Codes are supported:

01/02	Read I/O bit (Use Bit Address Mapping in Table 14.1)
03/04	Read I/O Word registers
05	Force I/O Bit (Use Bit Address Mapping in Table 14.1).
06	Preset Single Word Register
16	Preset Multiple Word Registers

The exact command/response format of the MODBUS protocol can be found at http://www.modbus.org. However, if your only purpose is to interface the PLC to other MODBUS hosts such as an LCD touch panel or SCADA software then there is no need to know the underlying protocol command format. All you need to know is which PLC's system Variable is mapped to which MODBUS register, as shown in Table 12.1.

Table 14.1: PLC I/O & Variable to Modbus Address Mapping

Fx2424	I/O #	MODBUS Holding Register (Modicon Convention)	MODBUS Bit Addr.
Input	**n**		**N**
	1 to 16	40001.1 to 40001.16	1 to16
	17 to 32	40002.1 to 40002.16	17 to 32
	33 to 48	40003.1 to 40003.16	33 to 48
	49 to 64	40004.1 to 40004.16	49 to 64
	65 to 80	40005.1 to 40005.16	65 to 80
	81 to 96	40006.1 to 40006.16	81 to 96
Output	**n**		**256 + n**
	1 to 16	40017.1 to 40017.16	257 to 272
	17 to 32	40018.1 to 40018.16	273 to 288
	33 to 48	40019.1 to 40019.16	289 to 304
	49 to 64	40020.1 to 40020.16	305 to 320
	65 to 80	40021.1 to 40021.16	321 to 336
	81 to 96	40022.1 to 40022.16	337 to 352
Timer	**N**		**512+n**
	1 to 16	40033.1 to 40033.16	513 to 528
	17 to 32	40034.1 to 40034.16	529 to 544
	33 to 48	40035.1 to 40035.16	545 to 560
	49 to 64	40036.1 to 40036.16	561 to 576
Counter	**n**		**768 + n**
	1 to 16	40049.1 to 40049.16	769 to 784
	17 to 32	40050.1 to 40050.16	785 to 800
	33 to 48	40051.1 to 40051.16	801 to 816
	49 to 64	40052.1 to 40052.16	817 to 832
Relay	**n**		**1024 + n**
	1 to 16	40065.1 to 40065.16	1025 to 1040
	17 to 32	40066.1 to 40066.16	1041 to 1056
	33 to 48	40067.1 to 40067.16	1057 to 1072
	49 to 64	40068.1 to 40068.16	1073 to 1088
	65 to 80	40069.1 to 40069.16	1089 to 1104
	81 to 96	40070.1 to 40070.16	1105 to 1120
	97 to 112	40071.1 to 40071.16	1121 to 1136
	113 to 128	40072.1 to 40072.16	1137 to 1152
	129 to 144	40073.1 to 40073.16	1153 to 1168
	145 to 160	40074.1 to 40074.16	1169 to 1184
	161 to 176	40075.1 to 40075.16	1185 to 1200
	177 to 192	40076.1 to 40076.16	1201 to 1216
	193 to 208	40077.1 to 40077.16	1217 to 1232
	209 to 224	40078.1 to 40078.16	1233 to 1248
	..	..	..
	497 to 512	40097.1 to 40097.16	1521 to 1536

* MODBUS is a registered trademark of Groupe Schneider.
OMRON is a registered trademark of OMRON Corporation.

Fx2424	Variables	Modbus Holding Register (Modicon convention)	Modbus Holding Register (Binary address)
Timer Present Values	1 to 64	40129 to 40192	128 to 191
Counter Present Values	1 to 64	40257 to 40320	256 to 319
Clock	TIME[1] TIME[2] TIME[3]	40513 40514 40515	512 513 514
Date	DATE[1] DATE[2] DATE[3] DATE[4]	40517 40518 40519 40520	516 517 518 519
Data Memory	DM[1] DM[2] …. DM[4000]	41001 41002 …. 45000	1000 1001 … 3999
Floating-point Array	FP[1] FP[2] … FP[501] ….. FP[1000]	45001 & 45002 45003 & 45004 ….. 46001 & 46002 …. 46999 & 47000	5000 & 5001 5002 & 5003 ….. 6000 & 6001 …. 6998 & 6999

14.7.1.1 BIT ADDRESS MAPPING

All the Fx2424 I/O bits are mapped identically to both the MODBUS “0x” and 1x space. The bit register offset is shown in the last column of Table 14.1. Although MODBUS names the 0x address space as “Coil (which means output bits) and the “1x” address space as “Input Status” (which means input bits only), the Fx2424 treat both spaces the same. Some MODBUS drivers only allow you to “read” from 0x space and “write” to 1x space but you still use the same offset shown on Table 14.1.

Example:

To map a lamp symbol to PLC Input 5, you select the MODBUS register address 0-0005. You can also map a lamp symbol to the PLC’s output #2. In that case, you should map it to MODBUS register address 0-0258.

To map a toggle switch symbol to the PLC input #5, if you are restricted to select only MODBUS 1x address space, then you will have to map the switch to 1-0005, and likewise you can map the switch to output #2 using MOBDUS address 1-0258. But if the driver allows the switch to be mapped to 0x space, then you can use MODBUS register space 0-0005 and 0-0258 for the mapping, with identical results.

14.7.1.2 WORD ADDRESS MAPPING

As shown in Table 14.1, to access DM[1] from the PLC, you use MODBUS address space 4-1001 (Modicon convention, 1000 in binary address) and so on. To access the Real Time Clock Hour data (TIME[1]), use 4-0513. The I/O channels can also be read or written as 16-bit words by using the addresses from 4-0001 to 4-0320.

Some MODBUS drivers (such as National Instruments "Lookout" software) even allow you to manipulate individual bit within a 16-bit word. So it is also possible to map individual I/O bits to the "4x" address space. E.g. Input bit #1 can be mapped to 4-0001.1 and output bit #2 is mapped to 4-0257.2, etc. This is how it is shown in Table 14.1. However, if you do not need to manipulate the individual bit then you simply use the address 4-0001 to access the system variable INPUT[1] and address 4-0257 to access the system variable OUTPUT[1]. Note that INPUT[1] and OUTPUT[1] are TBASIC system variables and they each contain 16 bits that reflect the on/off status of the actual physical input and output bits #1 to #16.

14.7.2 MODBUS RTU Protocol Support

The Fx2424 also support the MODBUS RTU protocol. The difference between the ASCII and RTU protocols is that the latter transmits binary data directly instead of converting one byte of binary data into two ASCII characters. A message frame is determined by the silent interval of 3.5 character times between characters received at the COMM port. Other than that, the function codes and memory mappings are identical to the MODBUS ASCII protocol. Table 14.1 therefore applies to the MODBUS RTU protocol as well.

MOBBUS RTU has the following command and response format:

Start	Address	Function	Data	CRC 16	END
Silence of 3.5 char times	1 byte	1 byte	# byte	2 bytes	Silence of 3.5 char times

The following Function Codes are supported:

01/02	Read I/O bit (Use Bit Address Mapping in Table 14.1)
03/04	Read I/O Word registers
05	Force I/O Bit (Use Bit Address Mapping in Table 14.1).
06	Preset Single Word Register
16	Preset Multiple Word Registers

14.7.3 OMRON Host Link Command Support

Command Type	Header	Level of Support
a) TEST	TS	Full support
b) STATUS READ	MS	Full support
c) ERROR Read	MF	Dummy (always good)
d) IR Area READ	RR	Full support (0000 to 1000)
e) HR, AR, LR Area & TC Status READ	RH	Dummy (always returns "0000")
f) DM AREA READ	RD	Full support
g) PV READ	RC	Dummy (always returns "0000")
h) Status Write	SC	Dummy (always OK)

l) IR Area WRITE	WR	Full Support
j) HR, AR, LR Area & TC Status WRITE	WH, WJ, WL, WG	Dummy (always OK)
k) DM Area WRITE	WD	Full Support (from DM0001-DM4000)
l) FORCED SET	KSCIO KRCIO	Full Support for IR Area only Dummy for other areas.
m) Registered I/O Read for Channel or Bit	QQMR/ QQIR	Full Support for IR and DM only Dummy for other areas (always 0000)

Some OMRON host link commands are described in Section 16.40. For other commands please refer to the Omron C20H/C40H PLC Operation manual published by OMRON Corporation. If your purpose is only to use the PLC's OMRON mode with SCADA or HMI, then there is no need to learn the actual command/response format.

14.7.4 Application Example: Interfacing to SCADA Software

SCADA software or MMI systems (also known as LCD Touch Panels) normally use an object-oriented programming method. Graphical objects such as switches indicator lights or meters, etc., are picked from the library and then assigned to a certain I/O or internal data address of the PLC. When designing a SCADA system, first you need to define the PLC type. You can choose the MODBUS ASCII, MODBUS RTU or OMRON C20H. Once a graphical object has been created, you will need to edit its connection and at this point you will be presented with a selection table that corresponds to the memory map of that PLC type.

Example 1: To connect an indicator lamp to Input #9 of the PLC.

You will need to program the switch to connect to IR00.8 for the OMRON protocol. However, If you have defined the PLC as MODBUS type, then this indicator lamp should be connected to bit address 1-0265. In either case there is no need to learn about the actual command format of the protocol itself, as the SCADA software will automatically generate the required commands to access the input address that has been chosen for the object.

Example 2: To display readings from ADC #3 as a bar graph on SCADA.

Since the data from ADC #3 is not directly mapped to MODBUS or OMRON in Table 14.1, you need to add a statement in the custom function that reads ADC #3 and copies it into a data memory, e.g.,

```
DM[100] = ADC(3)
```

Now you can program the bar graph on the SCADA screen to be connected to DM[100] if you use the OMRON protocol. For the MODBUS protocol, the object should be connected to the address: 4-1100

14.8 Using The PLC As a Modbus Master

– Getting Data From Power or Flow Meters

The Fx2424 support for the MODBUS protocol goes beyond being a MODBUS slave only. You can use the TBASIC READMOBUS and WRITEMODBUS commands, as well as READMB2 and WRITEMB2 to send out MODBUS ASCII or RTU commands to access any other F-series, FMD or Nano-10 PLCs or any third party MODBUS slave devices. The READMODBUS or READMB2 commands use MODBUS Function 03 (this can be changed to function 04 using SETSYSTEM 6,4 command) to read from the slave, and WRITEMODBUS or WRITEMB2 use MODBUS Function 16 to write to the slave.

Note that when using the READMODBUS or WRITEMODBUS commands, the 40001 address stated in Table 14.1 should be interpreted as address 0000, and 40002 as address 0001, and 41001 as address 1000, etc. This is in accordance with the specifications stated in MODBUS protocol. MODICON defined zero offset addresses for the MODBUS protocol, yet in their holding register definition these are supposed to start from address 40001 - hence the unusual correspondence. But to maintain compatibility with the MODBUS specifications, we have to adhere to their definitions.

14.8.1 Fx2424 PLC As MODBUS RTU Master

The Fx2424 PLC can also act as a MODBUS RTU master! The same READMODBUS and WRITEMOBUS commands can be used to send and receive MODBUS RTU commands. What you need to do is add 10 (decimal) to the COMM port number to signal to the processor that you wish to use MODBUS RTU instead of MODBUS ASCII to talk to the slaves.

In other words, you should specify port #11 to use RTU commands on COMM1, and specify port #13 to use RTU commands on COMM3.

E.g. the statement DM[10] = READMODBUS (13, 8, 16)

will access, via COMM3, the slave with ID = 08 and read the content of register #16. This register corresponds to MODICON address 40017 and is the OUTPUT[1] of the slave PLC.

The ability to speak MODBUS RTU greatly extends the type of peripherals that can be used with a Fx2424 PLC. You can now make use of many off-the-shelf, third party RTU devices to extend the PLC capability.

14.9 Using Modem to Remotely Program/Monitor The PLC

TLServer 3.x supports remote dial up to M-series and F-series PLCs via standard, off-the-shelf modems. It takes two modems to communicate between two devices. The host end of the modem setup and configuration is handled by the TLServer software itself, whereas on the PLC side, the PLC has to configure the modem so that it can successfully communicate with the host computer running TRiLOGI.

14.9.1 Wiring

The modem is often connected to the PLC's COMM1. Since the serial port on most modems are DCE type, you will need a make a special cable (also known as null-modem) to connect them as shown in Figure 14.2. If the modem only has a DB25 connector, you can connect the wires as shown in the following diagram:

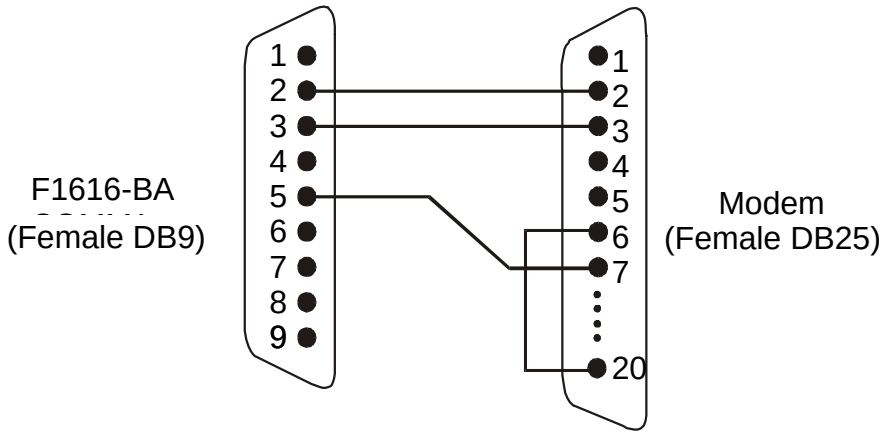


Figure 14.3 Connecting Fx2424's COMM1 to a modem's DB25 port

Note that pin 6 (DSR) and pin 20 (DTR) at the modem end are tied together. This is often required to inform the modem that the device is ready for operation so that the modem can work properly. A modem may also be connected to COMM2 or COMM3 for multi-drop remote programming and monitoring using TRiLOGI 6 or 7 software. However, you will need to purchase an auto-turnaround type RS232-to-RS485 converter, such as: the "Auto485" (http://www.tri-plc.com/auto485.htm).

14.9.2 Programming

Please refer to the Internet i-TRiLOGI version Programmer's Reference guide, Chapter 3 for programming details for the PLC to communicate with the PC via modem.

Chapter 15 Host Link Communication Protocol

15 HOST LINK PROTOCOL INTRODUCTION

While a Fx2424 is running, it may receive ASCII string commands that read or write to its inputs, outputs, relays, timers, counters, and all the internal variables from a host computer or another F-series, FMD or Nano-10 PLC. These ASCII commands are known as the "Host-link commands" and are to be serially transmitted (via RS232C or RS485 port) to and from the controller. The default serial port settings of the Fx2424 for host-link communication are: *38400 baud, 8 data bit, 1 stop bit, no parity*. The baud rate and the communication format may be changed using the “SetBAUD” TBASIC command described in the i-TRiLOGI Programmer’s Reference.

15.1 Multiple Communication Protocols

The Fx2424 supports many different communication protocols to allow maximum application flexibility. In addition to its own native set of communication protocols, the Fx2424 also understands and speaks the following protocols:

a) ***MODBUS**™ ASCII mode compatible communication protocol.

b) ***MODBUS**™ RTU mode compatible communication protocol.

c) ***OMRON**™ Host Link Commands for the C20H PLC family.

*Note: all trademarks belong to their respective owners.

The native host link command protocol will be described in detail in this and the next chapter. The MODBUS and OMRON compatible protocols have already been discussed in Section 14.7.

15.2 Native Mode Communication Protocols

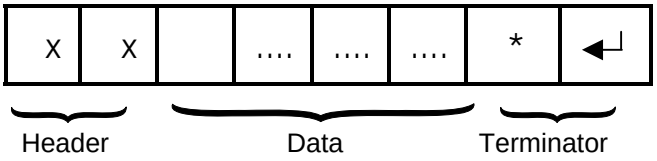
When the Fx2424 receives a native host-link command via COMM1, COMM2 or COMM3, it will automatically send a response string corresponding to the command. This operation is totally transparent to the user and does not need to be handled by the user’s program.

All Fx2424 support both the point-to-point (one-to-one) and multi-point (one-to-many) communication protocols. Each protocol has a different command structure as described below.

15.3 Point-To-Point Communication Format

In a point-to-point communication system, the host computer's RS232C serial port is connected to the PLC’s COMM1. At any one time, only one controller may be connected to the host computer. The host-link commands do not need to specify any controller ID code and are therefore of a simpler format, as shown below:

15.3.1 Command/Response Frame Format (Point to Point)



Each command frame starts with a two-byte ASCII character header, followed by a number of ASCII data and ends with a terminator which is comprised of a '*' character and a carriage return (ASCII value = 13_{10}). The header denotes the purpose of the command. For example, RI for Read Input, WO for Write Output, etc. The data is usually the hexadecimal representation of numeric data. Each byte of binary data is represented by two ASCII characters (00 to FF).

To begin a communication session, the host computer must first send one byte of ASCII character: Ctrl-E (=05Hex) via its serial port to the controller. This informs the controller that the host computer wishes to send a (point-to-point) host-link command to it. Thereafter, the host computer must wait to receive an echo of the Ctrl-E character from the controller. Reception of the echoed Ctrl-E character indicates that the controller is ready to respond to the command from the host computer. At this moment, the host computer must immediately send the *command frame* to the controller and then wait to receive the *response frame* from the controller. The entire communication session is depicted in Figure 15.1.

After the controller has received the command, it will send a response frame back to the host computer and this completes the communication session. If the controller accepts the command, the response frame will start with the same header as the command, followed by the information that has been requested by the command and the terminator.

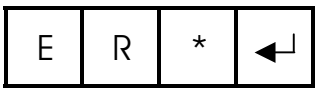
As you can probably see, proper handshaking using the Ctrl-E character between the host and the PLC is important to communicate successfully using the Point-to-point protocol.

Although the “Multi-point” format discussed in the next section seems more complex, the communication exchange using multi-point protocol is actually simpler than point-to-point since it involves only a single exchange of command/response string. We therefore recommend using the multi-point format if you are writing your own communication program.

Note: TBASIC has a built-in command “NETCMD$” that lets a Fx2424 access another slave PLC using the multipoint Host-link protocol format very easily.

If an unknown command is received or if the command is illegal (such as access to an unavailable output or relay channel), the following **error response** will be received:

15.3.2 Error Response Format



The host computer program should always check the returned response for possibilities of errors in the command and take necessary actions.

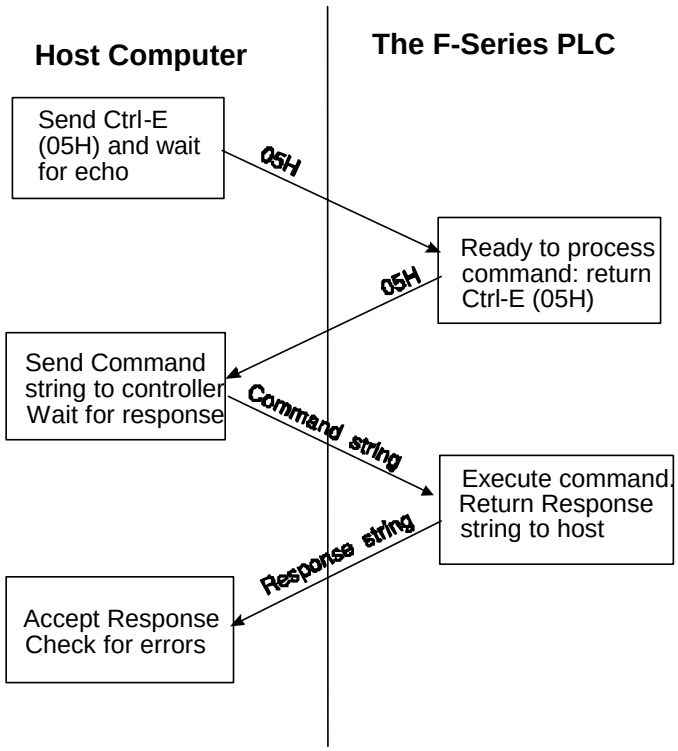
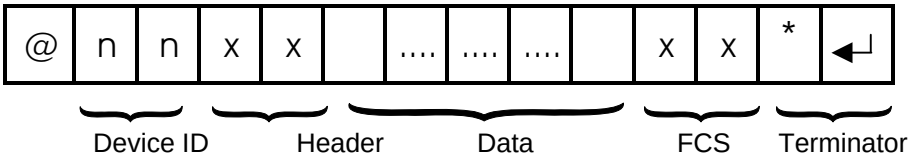


Figure 15.1

15.4 MULTI-POINT COMMUNICATION SYSTEM

In this system, one host computer may be connected to either a single PLC (via either RS232 or RS485) or multiple PLCs on an RS485 network.

15.4.1 Command/Response Frame Format (Multi-point)

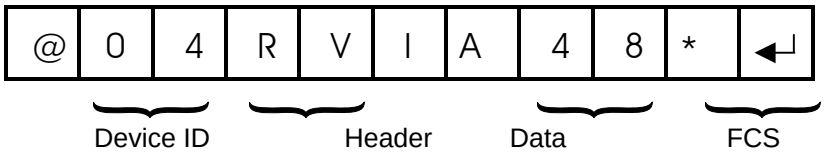


Each command frame starts with the character "@" and two-byte hexadecimal representation of the controller's ID (00 to FF), and ends with a two-byte "Frame Check Sequence" (FCS) and the terminator. FCS is provided for detecting communication errors in the serial bit-stream. If desired, the command frame may omit calculating the FCS simply by putting the characters "00" in place of the FCS.

Note: we call “00” the “wildcard” FCS, which is available when the PLC is in “auto protocol” mode. This is to facilitate easy testing of the multi-point protocol. However, the wildcard FCS can be disabled if the PLC has executed the SETPROTOCOL *n*, 5 to put its COMM port *n* into pure native mode. In that case you will have to supply the actual FCS to your command string.

15.4.2 Calculation of FCS

The FCS is 8-bit data represented by two ASCII characters (00 to FF). It is a result of performing an Exclusive OR on each character in the frame sequentially, starting from @ in the device number to the last character in the data. An example is as follows :



@	0100 0000
	XOR
0	0011 0000
	XOR
4	0011 0100
	XOR
R	0101 0010
	XOR
V	0101 0110
	XOR
I	0100 1001
	XOR
A	0100 0001
	0100 1000 = 48_{16}

Value 48_{16} is then converted to ASCII characters '4' (0011 0100) and '8' (0011 1000) and placed in the FCS field.

FCS calculation program example

The following C function will compute and return the FCS for the "string" passed to it.

```
unsigned char compute_FCS(unsigned char *string){
   unsigned char result;
   result = *string++;    /*first byte of string*/
   while (*string)
      result ^= *string++; /* XOR operation */
   return (result);
}
```

A **Visual Basic** routine for FCS computation is included in the source code of a sample communication program you can download from:

http://www.tri-plc.com/applications/VBsample.htm#VB6sample

15.4.3 Communication Procedure

Unlike the point-to-point communication protocol, the host computer must **NOT** send the CTRL-E character before sending the command frame. After the host computer has sent out the multi-point host-link command frame, only the controller with the correct device ID will respond. Hence it is essential to ensure that every controller on the RS485 network assumes a different ID (If a master PLC is used, then

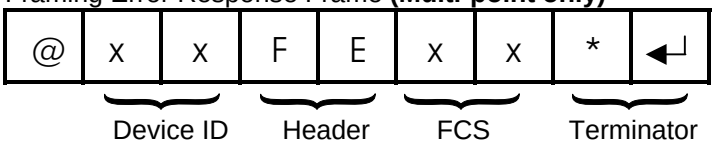
the master PLC should also have a different ID from all the slaves). Otherwise, contention may occur (i.e., two controllers simultaneously sending data on the receiver bus, resulting in garbage data being received by the host). On the other hand, if none of the controller IDs match that specified in the command frame, then the host computer will receive no response at all.

The PLC automatically recognizes the type of command protocols (point-to-point or multi-point) sent by the host computer and it will respond accordingly. If a multi-point command is accepted by the controller, the response frame will start with a character '@', followed by its device ID and the same header as the command. This will be followed by the data requested by the command, a response frame FCS and the terminator.

15.4.4 Framing Errors

When the controller receives a multi-point host-link command frame, it computes the FCS of the command and compares it with the FCS field received in the command frame. If the two do not match, then a "framing error" has occurred. The controller will send the following Framing Error Response to the host:

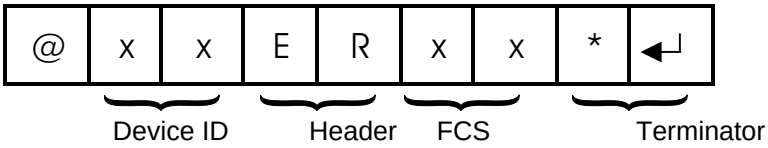
Framing Error Response Frame **(Multi-point only)**



15.4.5 Command Errors

If an unknown command is received or if the command is illegal (such as an attempt to access an unavailable channel), the following **error response** will be received:

Error Response Format



The host computer program should always check the returned response for possibilities of errors in the command and take necessary action.

15.4.6 SHOULD YOU USE POINT-TO-POINT OR MULTI-POINT PROTOCOL?

Although at first the point-to-point protocol appears simpler in format (having no ID and no FCS computation), the communication procedure is actually more complex since it involves the need to synchronize the two communicating devices by exchanging the Control-E character. The lack of error checking also makes the protocol less reliable especially in noisy environment.

In fact, the TLServer software will only accept multi-point communication protocol from the client software with the exception of the “IR*” command, which is needed to obtain the ID of a PLC with unknown ID.
Hence, if you were to write your own communication program to talk to the PLCs, we would strongly recommend using only the multi-point protocol exclusively due to its simplicity and built-in error checking capability.

15.5 RS485 Primer

15.5.1 RS485 Network Interface Hardware

The built-in RS-485 interface allows the Fx PLCs to be networked together using very low cost twisted-pair cables. Since the Fx PLCs are fitted with a 1/8-power RS485 driver such as the 75HVD3082, up to 256 devices can be connected together. The twisted-pair cable goes from node to node in a daisy chain fashion and should be terminated by a 120-ohm resistor as shown below.

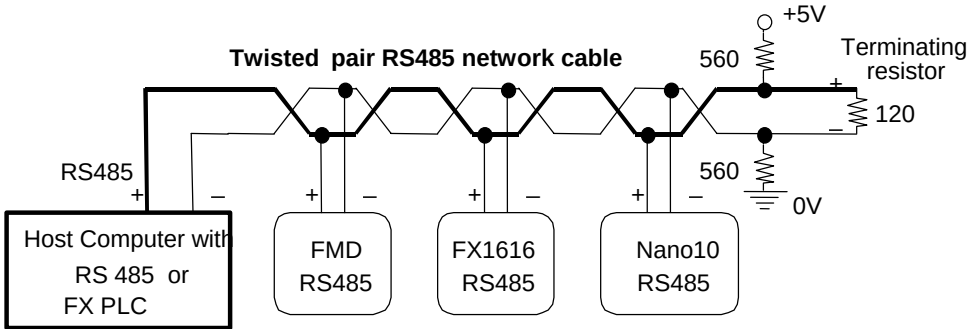


Figure 15.2

Note that the two wires are not interchangeable so they must be wired the same way to each controller. The maximum wire length should not be more than 1200 meters (4000 feet). RS-485 uses balanced or differential drivers and receivers, which means that the logic state of the transmitted signal depends on the differential voltage between the two wires and not on the voltage with respect to a common ground.

As there will be times when no transmitters are active (which leaves the wires in "floating" state), it is good practice to ensure that the RS-485 receivers will indicate to the CPUs that there is no data to receive. In order to do this, we should hold the twisted pair in the logic '1' state by applying a differential bias to the lines using a pair of 560Ω to 1KΩ biasing resistors connected to a +9V (at least +5V) and 0V supply as shown in Figure 15-2. Otherwise, random noise on the pair could be falsely interpreted as data.

The two biasing resistors are necessary to ensure robust data communication in actual applications. Some RS485 converters may already have biasing built-in so the biasing resistors may not be needed. However, if the master is a Fx2424 then you should use the biasing resistor to fix the logic states to a known state. Although in a lab environment the PLCs may be able to communicate without the biasing resistors, their use is strongly recommended for industrial applications.

15.5.2 Protection of RS485 Interface

The simple, direct multi-drop wiring shown in Figure 15-2 will work well if all the networked PLCs are in close proximity and they all share a common power supply. They will even work for long distance as long as there are no wiring errors. However, in an industrial environment, the PLCs are most likely far apart and may each have their own power supply. Since processes are often modified regularly, should somebody on one occasion by mistake short one of the PLC's RS485 to high voltage, all the PLCs connected to the same RS485 wiring will be fried simultaneously. This can result in very costly down time for the whole process because all of the PLCs connected to the network will need to be repaired.

Hence, for networking over long distances and involving more than a few PLCs, it is important to either strengthen or protect the RS485 interface, as described below:

1. You can replace the standard RS485 driver (75HVD3082) on the PLC with a fault-tolerant RS485 driver IC; part number LT1785AIN8. This 8 pin IC is made by Linear Technology and can

withstand wrong voltages of up to ±60V! The LT1785AIN8 is a 1/4 power RS485 driver, which means up to 128 PLCs can be connected together.

Unfortunately this IC is much more expensive than 75HVD3082 and hence it is not provided as standard component on the PLC. You can purchase the IC from any major electronic catalog company such as www.digikey.com.

2. When using non fault-tolerant RS485 drivers such as SN75176 or SN75HVD3082, we strongly recommend the following protection circuit to be added between every PLC's RS485 and the twisted pair multi-drop network cable:

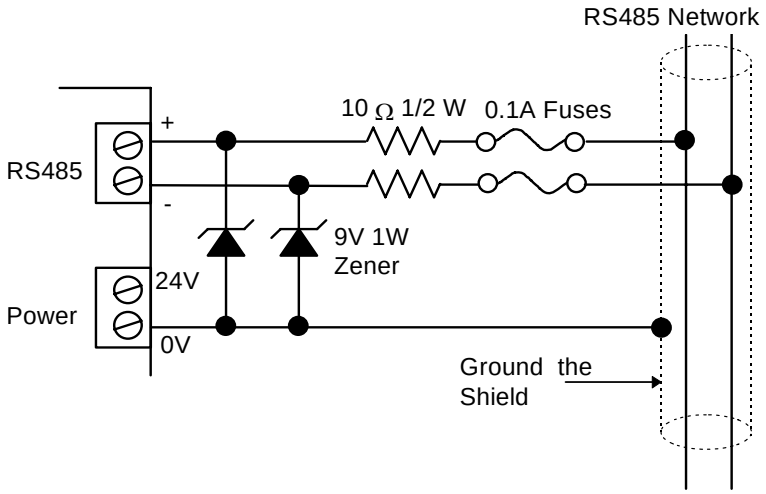


Figure 15.3

Note:

As can be seen from the circuit, the two 9V Zener diodes clamp the signal voltage on the PLC's RS485 interface to between +9V and - 0.7V. If the high voltage persists, the 0.1A fuse will blow, effectively disconnecting the PLC from the offending voltage on the network.

Even if you choose to replace the RS485 driver by an LT1785AIN8 IC instead of using the zener/fuse pair wiring, you should still use shielded twisted pair cables as the multi-drop network "backbone" and connect the shield to the 0V (DC ground) power terminal of every PLC. The grounded shield then provides a common ground reference for all the different PLCs' power supplies. Even though the RS485 network may still work without a common ground reference because the signal wire pair will somehow "pull" all the RS485 devices to some reference point. **Failure to provide a common ground is a potential source of serious trouble** as signal wires with a floating ground easily induce large voltage differences between nodes when subjected to electromagnetic interference. Hence, for reliable operation it is important to provide the common ground. A grounded shield also has the additional advantage of shielding the electrical signals from EMI.

15.5.3 Single Master RS485 Networking Fundamentals

RS485 is a half-duplex network, i.e., the same two wires are used for both transmission of the command and reception of the response. Of course, at any one time, only one transmitter may be active. The Fx2424 implements a master/slave network protocol. The network requires a master controller, which is typically a PC equipped with an RS485 interface. In the case of a PC, you can purchase an RS-485 adapter card or an RS232C-to-RS485 converter and connect it to the RS232C serial port. A Fx2424 can also be programmed to act as the master, it can communicate with other PLCs by executing the "NETCMD$" function or the "READMODBUS" or the "WRITEMODBUS" commands (the latter two are for communicating using MODBUS protocols only and are covered in Section 14.8).

Only the master can issue commands to the slave PLCs. To transmit a command, the master controller must first enable its RS-485 transmitter and then send a multi-point command to the network of controllers. After the last stop bit has been sent, the master controller must relinquish the RS485 bus by disabling its RS485 transmitter and enabling its receiver. At this point the master will wait for a response from the slave controller that is being addressed. Since the command contains the ID of the target controller, only the controller with the correct ID would respond to the command by sending back a response string. For the network to function properly, it is obvious that no two nodes can have the same ID. You can use the “Setup Serial Port” command in TLServer to set the ID for each Fx2424. You can also use the "IW” Host Link command to set the device ID. Also, all nodes must be configured to the same baud rate and communication format.

Care should be taken to ensure that the power supplies for all the controllers are properly isolated from the main so that no large ground potential differences exist between any controllers on the network.

15.5.4 Multi-Master RS485 Networking Fundamentals

Since the Fx2424 is capable of sending out network commands, the obvious question is whether multiple masters are allowed on the RS485 network? It is possible to have multiple masters on a single RS485 network provided the issues of collision and arbitration are taken care of. There are several means to achieve these objectives.

15.5.4.1 Multiple Access with Collision Detection

There is nothing to stop any PLC from sending out host-link commands to other PLCs. However, if more than one PLC simultaneously enables their transmitters and send out host-link commands, then the signals will conflict and the messages will be garbled up. If the network traffic is low, then the solution may be a matter of having the master check for the correct response after sending out a command string. If there is error in the response string, the master should back off the network for a short while (use different timing for different PLCs) and then re-send the command until a correct response string is obtained.

Fortunately, the “NETCMD$” function of Fx series PLC automatically senses the RS485 lines until they are free before sending out the command string to reduce the chance of a collision. It also checks the integrity of the response string for correct FCS (Frame Check Sequence) characters before returning the string (Please refer to the Programmer’s Reference for detail description of the NETCMD$( ) function).

However, the program must still check the following items in the response string to verify that the string returned from NETCMD$( ) function indeed comes from the PLC that it had talked to and not from another PLC (which tries to send a command to someone else):

i) The ID is correct
ii) The header is identical to the command string
iii) The length of response string is correct.

Pros and Cons: This method does not incur any hardware cost, but it requires careful programming and strict checking of the response string and hence requires more effort to program. It is also the least desirable if the network traffic is moderately high as many collisions will occur and there is danger of some undetected error being allowed to pass through.

15.5.4.2 Token Awarding Scheme

A “token” is a software means of telling a PLC that it has been given the right to temporarily act as the master. A Fx PLC or a host PC can serve as the token master. An internal relay bit or a variable of the PLC can be defined as the token. The token master will begin by giving the token (i.e., by setting the token relay bit to ‘1’ or the token variable to some fixed value) to the first PLC on the list. The PLC that has the token can then send host-link commands to other PLCs. When it has finished the job it can then send a command to the token master to relinquish its token. If it is based on a fixed timing scheme the master can assume that the PLC will complete its job after a fixed time (say 0.1 seconds) and turn off its corresponding token relay bit.

The token master then passes the token to the next PLC on the list and so on until the last PLC has relinquished its token, and the token is passed back to the first PLC on the list again. This way at any one time there will only be one active network master (the one with the token) and hence there is no danger of conflicting signals or garbled messages to handle.

Pros and Cons: This method also does not incur any hardware cost, but it requires the programmer to draw up a plan on what internal relay or variable to use as the token and how the PLC can relinquish its token to the token master. (It could be by fixed timing or by returning a message to relinquish the token) It is a challenging job for programmers unfamiliar with networking schemes, but with some experimentation it can be achieved readily.

15.5.4.3 Rotating Master Signal

In this scheme we make use of the digital inputs of the Fx PLCs to grant the PLC the right to act as the network master. Lets call this input the “Be the Master” input. We can use a low cost H-series PLC running a sequencer to activate the “Be the Master” input line of each PLC one at a time. Each PLC is given a fixed amount of time to be the master (e.g. 0.1s each). Only when the “Be the Master” input is ON can the Fx PLC start sending out host-link commands to other PLCs. So at any one time there will only be one master on the network and no conflict will occur as a result.

Pros and Cons: This method is the easiest to program since there is no need to handle the token with the token master or perform extensive error check on the response string. However, this method uses one input of each PLC and as many outputs on the master-signal generator PLC as there are PLC masters. It also requires wiring the PLCs to the master-signal generator PLC.

15.5.5 TROUBLE-SHOOTING AN RS485 NETWORK

1) Single faulty device

If a single device on the RS485 network becomes inaccessible, problems can be isolated to this particular device. Check for loose or broken wiring or wrong DIP switch settings. Also double check the device ID using the host-link command `"IR*"` sent via the RS232C port of the PLC. If all attempts fail, either replace the entire PLC or the SN75176 chip that handles the RS485 interfacing and try again.

2) Multiple faulty devices

If all the PLCs are inaccessible by the host computer, it may possibly be due to a faulty RS232C-to-RS485 converter at the PC. If this is the case, disconnect the RS485 converter from the network and check it using a single PLC. Replace the converter if it is confirmed to be faulty. Next check the wire from the converter to the beginning of the network. A broken wire here can lead to the failure of the entire network.

Since an RS485 network links many PLCs together electrically and in a daisy chain fashion, problems occurring along the RS485 network sometimes affect the operation of the entire network. For example, a

broken wire at the terminal of one node may mean that all the PLCs connected after this node become inaccessible by the master. If the RS485 interface of one of the PLCs has short-circuited because of component failure, then the entire network goes down with it too. This is because no other node is able to assert proper signals on the two wires that are also common to the shorted device.

Hence, when trouble-shooting a faulty RS485 network, it may be necessary to isolate all the PLCs from the network. Thereafter, reconnect one PLC at a time to the network, starting from the node nearest to the host computer. Use the TRiLOGI program to check communication with each PLC until the faulty unit has been identified.

Chapter 16 Host Link Protocol Format

16 HOST LINK PROTOCOL FORMAT

This chapter describes the detailed formats of the command and response frames for all Fx PLC host link commands. Only the formats for the point-to-point communication protocol are presented, but all these commands are available to the multi-point protocol as well.

To use a command for multi-point system, simply add the device ID (@nn) before the command header and the FCS at the end of the data (See Chapter 3 for detailed descriptions of multi-point communication command format).

16.1 Device ID Read

Command Format

I	R	*	↵

Response Format

I	R	16^1	16^0	*	↵

Device ID (00 to FF)

The device ID is to be used for the multi-point communication protocol so that the host computer can selectively communicate with any controller connected to a common RS485 bus (see Chapter 3 for details). The ID has no effect for point-to-point communication. The device ID is stored in the PLC's non-volatile memory and, therefore, will remain with the controller until it is next changed.

16.2 Device ID Write

Command Format

I	W	16^1	16^0	*	↵

New Device ID (00 to FF)

Response Format

I	W	*	↵

E.g. To set the PLC's ID to 0A, send command string "`IW0A*`" to PLC.

16.3 Read Digital Input Channels

Command Format

R	I	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	I	16^1	16^0	*	↵

8-bit Data (Hex)

16.3.1 Definition of Input Channels

The following table shows the input numbers as defined in TRiLOGI's Input entry table corresponding to the input channel number.

	Bit7			Input/Output Numbers				Bit0
CH00:	8	7	6	5	4	3	2	1
CH01:	16	15	14	13	12	11	10	9
CH02:	24	23	22	21	20	19	18	17
CH03:	32	31	30	29	28	27	26	25
CH04:	40	39	38	37	36	35	34	33
CH05:	48	57	56	45	44	43	42	41
CH06:	56	55	54	53	52	51	50	49
CH07:	64	63	62	61	60	59	58	57
CH08:	72	71	70	69	68	67	66	65
CH09:	80	79	78	77	76	75	74	73
$CH0A_{16}$:	88	87	86	85	84	83	82	81
$CH0B_{16}$:	96	95	94	93	92	91	90	89
$CH0C_{16}$:	104	103	102	101	100	99	98	97
….	..	..	..	..	..	..	..	..
$CH1E_{16}$:	248	247	246	245	244	243	242	241
$CH1F_{16}$:	256	255	254	253	252	251	250	249

The 8-bit inputs of each channel are represented by a two byte ASCII text expression of its hexadecimal value. For example: if inputs 1 to 3 are logic '0's, inputs 4 to 10 are logic '1's and all other inputs are logic '0's, then if you send command "`RI00*`", you will get the response "RIF8*" ($F8_{16}$ =1111 1000_2).

16.4 Read Digital Output Channels

Command Format

R	O	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	O	16^1	16^0	*	↵

8-bit data (Hex)

Please refer to the Input/Output vs Channel Number table described in the section “16.3. Read Digital Input Channels” for details.

16.5 Read Internal Relay Channels

Command Format

R	R	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	R	16^1	16^0	*	↵

8-bit data (Hex)

16.5.1 Definition of Internal Relay Channel Numbers

All Fx PLCs support 512 internal relays, the channel definition of the first 256 internal relays is the same as the inputs and the outputs. The remaining relays and their assigned channels are shown in the following table:

	bit7			Relay numbers				bit0
$CH20_{16}$:	264	263	262	261	260	259	258	257
$CH21_{16}$:	272	271	270	269	268	267	266	265
$CH22_{16}$:	280	279	278	277	276	275	274	273
$CH23_{16}$:	288	287	286	285	284	283	282	281
$CH24_{16}$:	296	295	294	293	292	291	290	289
$CH25_{16}$:	304	303	302	301	300	299	298	297
$CH26_{16}$:	312	311	310	309	308	307	306	305
$CH27_{16}$:	320	319	318	317	316	315	314	313
$CH28_{16}$:	328	327	326	325	324	323	322	321
$CH29_{16}$:	336	335	334	333	332	331	330	329
$CH2A_{16}$:	344	343	342	341	340	339	338	337
$CH2B_{16}$:	352	351	350	349	348	347	346	345
..	..	..	..	..	..	..	..	..
$CH3A_{16}$:	488	487	486	485	484	483	482	481
$CH3D_{16}$:	496	495	494	493	492	491	490	489
$CH3E_{16}$:	504	503	502	501	500	499	498	497
$CH3F_{16}$:	512	511	510	509	508	507	506	505

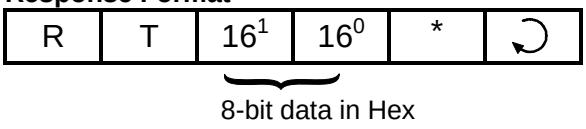
16.6 Read Timer Contacts

Command Format

R	T	n	n	*	↵

8-bit Channel # (Hex)

Response Format



16.6.1 Definition of Timer-Contact Channel Numbers

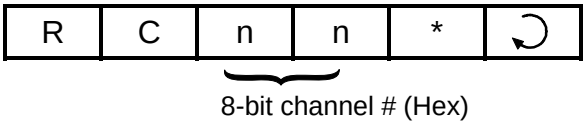
A timer contact is a single bit of memory and 8 timer contacts are grouped into one 8-bit channel similar to that of the inputs, outputs etc.

The following table shows the timer numbers defined in TRiLOGI's Timer entry table and their corresponding channel numbers.

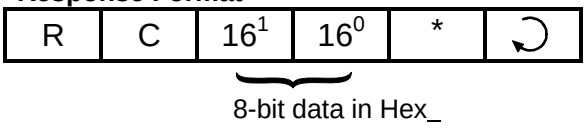
	bit7			Timer Contact numbers				bit0
CH0:	8	7	6	5	4	3	2	1
CH1:	16	15	14	13	12	11	10	9
CH2:	24	23	22	21	20	19	18	17
CH3:	32	31	30	29	28	27	26	25
CH4:	40	39	38	37	36	35	34	33
CH5:	48	57	56	45	44	43	42	41
CH6:	56	55	54	53	52	51	50	49
CH7:	64	63	62	61	60	59	58	57

16.7 Read Counter Contacts

Command Format



Response Format

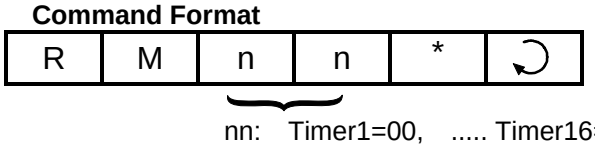


16.7.1 Definition of Counter-Contact Channel Numbers:

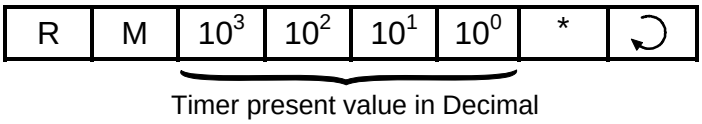
The 64 counter contacts are assigned channel #'s in exactly the same way as the 64 timers. Please refer to the last section: "16.6. Read Timer Contacts" for details.

16.8 Read Timer Present Value (P.V.)

Command Format



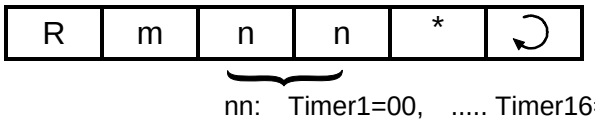
Response Format



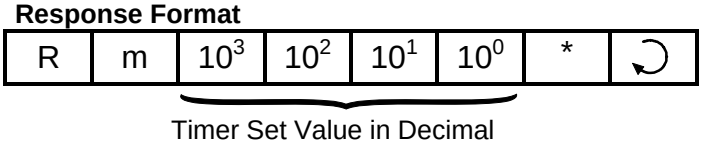
The present value (PV) of the specified timer is returned in decimal form as four byte ASCII text characters from 0000 to 9999.

16.9 Read Timer Set Value (S.V.)

Command Format



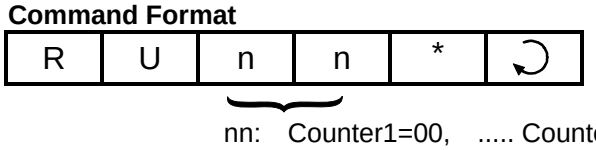
Response Format



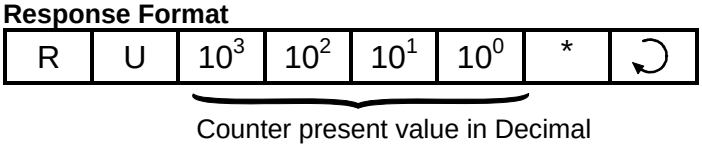
The Set Value (S.V.) of the specified timer is returned in decimal form as four byte ASCII text characters from 0000 to 9999. Note that this command header contains **small letter "m"** instead of "M" in the "RM" command.

16.10 Read Counter Present Value (P.V.)

Command Format



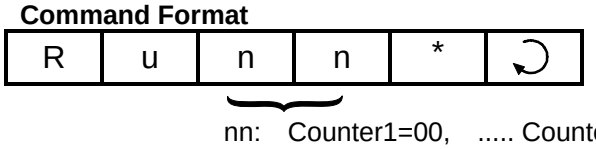
Response Format



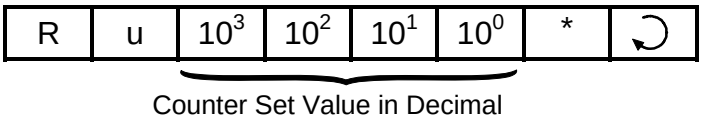
The Present Value of the specified counter is returned in decimal form as four byte ASCII text characters from 0000 to 9999.

16.11 Read Counter Set Value (S.V.)

Command Format



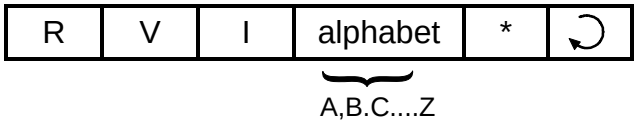
Response Format



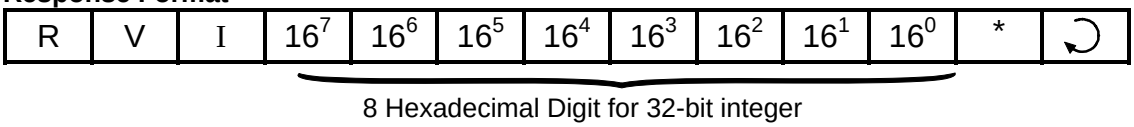
The Set Value of the specified counter is returned in decimal form as four byte ASCII text characters from 0000 to 9999. Note that this header contains **small letter "u"** instead of "U" in the "RU" command.

16.12 Read Variable - Integers (A to Z)

Command Format



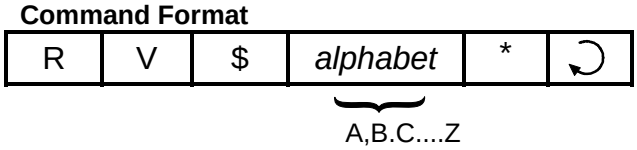
Response Format



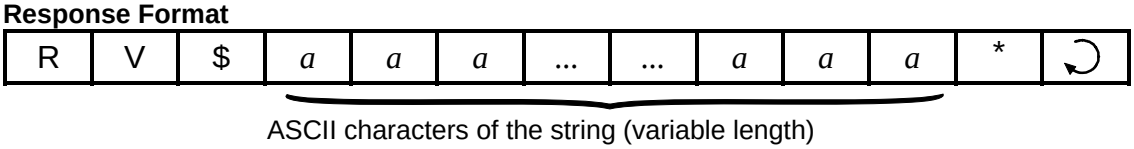
E.g. To read the value of the variable "K", send host-link command "`RVIK*`". If variable K contains the value 123456_{10} (=$1E240_{16}$), the PLC will send the response string as "`RVI0001E240*`".

16.13 Read Variable - Strings (A$ to Z$)

Command Format



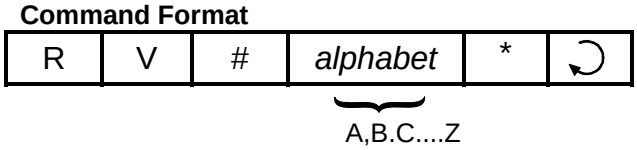
Response Format



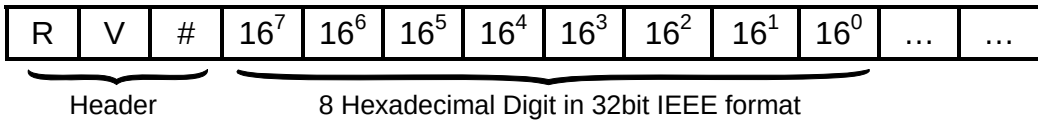
E.g. To read the value of the string variable "M$", send host-link command "`RV$M*`". If variable M$ contains the string "Hello World", the PLC will send the response string as "`RV$Hello World*`".

16.14 Read Variable – Floating Point (A# to Z#)

Command Format



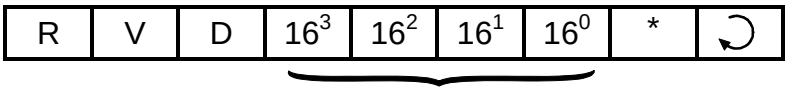
Response Format



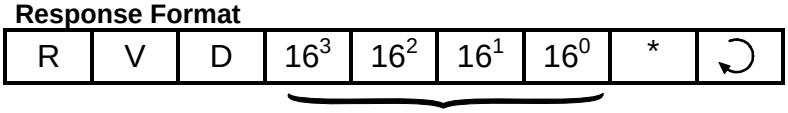
E.g. To read the value of the string variable "M$", send host-link command "RV$M*". If variable M$ contains the string "Hello World", the PLC will send the response string as "RV$Hello World*".

16.15 Read Variable – Single Data Memory (DM[1] to DM[4000])

Command Format



Response Format

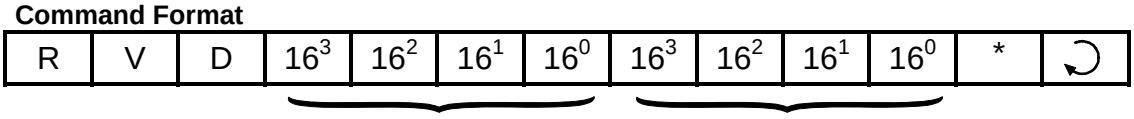


E.g. To read the value of DM[3600], send host-link command "RVD0E10*". If variable DM[3600] contains the value 12345_{10} (=3039_{16}), PLC will send the response string as "RVD3039*".

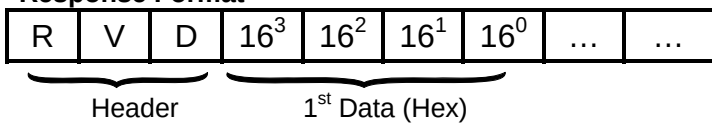
16.16 Read Variable – Multiple Data Memory (DM[1] to DM[4000])

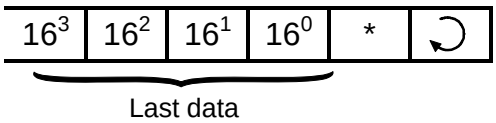
(Note: This new command is available only to Fx PLC)

Command Format



Response Format



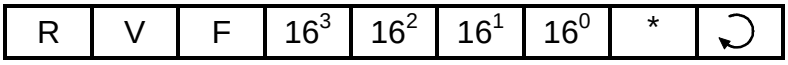


E.g. To read the value of DM[3600] to DM[3609], send host-link command "RVD0E10000A*".

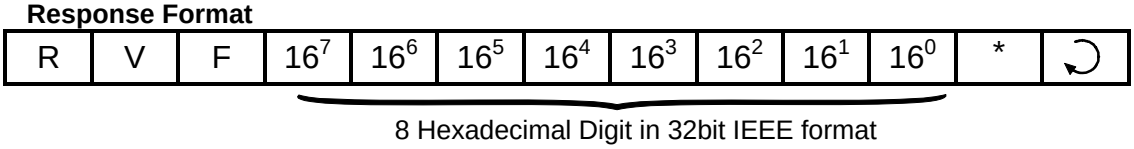
16.17 Read Variable – Single Floating Point Array FP[1] to FP[1000]

(Note: This new command is available only to Fx PLC)

Command Format



Response Format

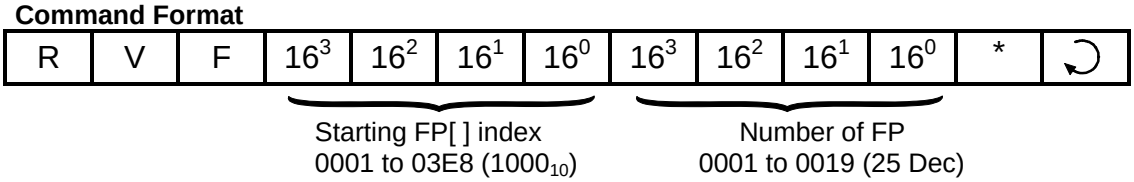


E.g. To read the value of FP[20], send host-link command "RVF0014*". If variable FP[20] contains the value 1.2345E-2 (0.012345), PLC will send the response string as "RVF3C4A42AF*".

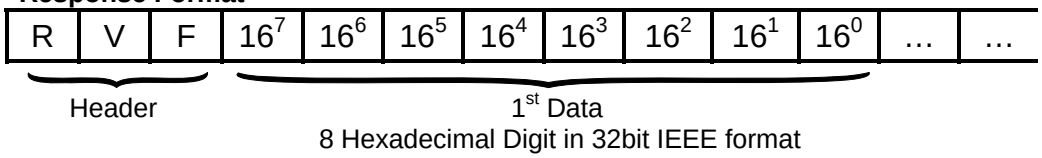
16.18 Read Variable – Multiple Floating Point Data FP[1] to FP[1000]

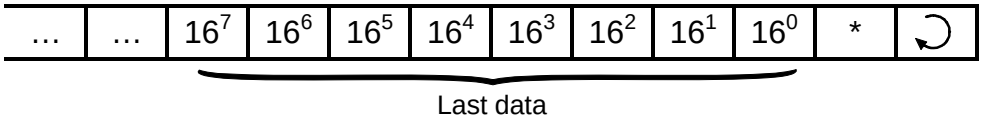
(Note: This new command is available only to Fx PLC)

Command Format



Response Format





E.g. To read the value of FP[101] to FP[110], send host-link command "RVF0065000A*".

16.19 Read Variable - System Variables

This command allows you to read all the Fx PLC's 16-bit system variables such as the inputs[ ], outputs[ ], relays[ ], counters[ ], timers[ ], timers' P.V., counters' P.V., CLK[ ] and DATE[ ]. Although inputs, outputs etc. are also accessible via the "RI", "RO", "RR"... commands, the RVS command can access them as 16-bit words instead of as 8-bit bytes in those commands. For the 32-bit system variable HSCPV[ ], use the "RVH" command described in the next section to access it. It may be more conventional for some SCADA software driver to use a single header command "RVS" to access all the I/O, varying only the "type" number to access different I/O types.

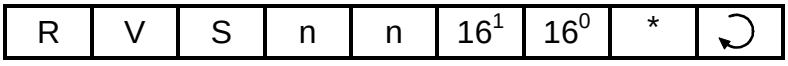
The RVS command also can be used to access the internal variables used to store ADC, DAC and PWM values obtained during the latest execution of the ADC(), setDAC or setPWM statement. These are however not system variables in the TBASIC sense. E.g. it is illegal to use ADC[2] to access the ADC channel #2 in TBASIC (you have to use the ADC(2) function instead). An 8-bit hexadecimal number is used to denote the "type" of system variable, as shown in the following table:

System Variable	type
input[ ]	01
output[ ]	02
relay[ ]	03
timer[ ]	04
ctr[ ]	05
timerPV[ ]	06
ctrPV[ ]	07

System Variable	type
clk[ ]	08
date[ ]	09
-	0A
ADC*	0B
DAC*	0C
PWM*	0D

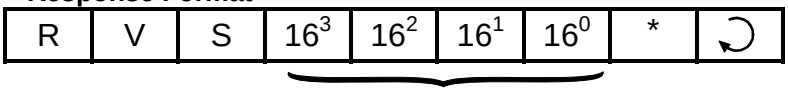
*Not a system variable in TBASIC, but readable.

Command Format



type (01 to 0D) - denotes the type of system variable to access,
index (01 to 1F) - index into the array, starting from 01.

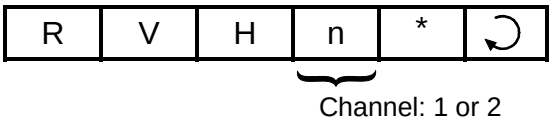
Response Format



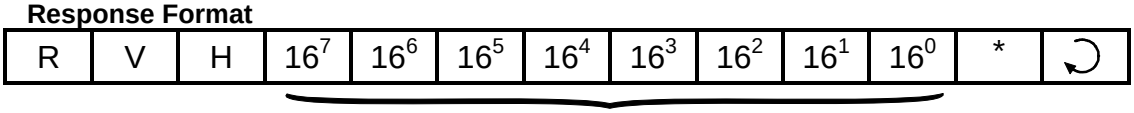
Example: To read the value of DATE[2] (which represents the month of the RTC), send command "RVS0902*" and if the PLC responds with "RVS0005", it means the month is May.

16.20 Read Variable - High Speed Counter HSCPV[ ]

Command Format



Response Format



E.g. To read the value of HSCPV[2], send hostlink command "RVH2*". If variable HSCPV[2] contains the value 123456_{10} ($=1E240_{16}$), the PLC will send the response string as "RVH0001E240*".

16.21 Write Inputs

Command Format

W	I	n	n	16^1	16^0	*	↵

Channel # (00 to 0F) — Data

Response Format

W	I	*	↵

16.22 Write Outputs

Command Format

W	O	n	n	16^1	16^0	*	↵

Channel # (00 to 0F) — Data

Response Format

W	O	*	↵

16.23 Write Relays

Command Format

W	R	n	n	16^1	16^0	*	↵

Channel # — Data

Response Format

W	R	*	↵

16.24 Write Timer-contacts

Command Format

W	T	n	n	16^1	16^0	*	↵

Channel # (00 to 07) — Data

Response Format

W	T	*	↵

16.25 Write Counter-contacts

Command Format

W	C	n	n	16^1	16^0	*	↵

Channel # (00 to 07) — Data

Response Format

W	C	*	↵

16.26 Write Timer Present Value (P.V.)

Command Format

W	M	n	n	10^3	10^2	10^1	10^0	*	↵

Timer1=00, ... Timer64=3F (Hex) — New timer PV

Response Format

W	M	*	↵

Please note that the timer number starts from 00, which represent timer #1, then 01 represents timer #2... and so on.

16.27 Write Timer Set Value (S.V.)

Command Format

W	m	n	n	10^3	10^2	10^1	10^0	*	↵

Timer1=00, ... Timer64=3F (Hex) — New timer SV

Response Format

W	m	*	↵

Note: the 2nd character is a lower case “m” instead of the upper case “M” of “WM” command.

16.28 Write Counter Present Value (P.V.)

Command Format

W	U	n	n	10^3	10^2	10^1	10^0	*	↵

Counter1=00, New PV
....
Counter64=3F (Hex)

Response Format

W	U	*	↵

16.29 Write Counter Set Value (S.V.)

Command Format

W	u	n	n	10^3	10^2	10^1	10^0	*	↵

Counter1=00, New Counter SV
....
Counter64=3F (Hex)

Response Format

W	u	*	↵

Note: the 2nd character is a lower case “u” instead of the upper case “U” of the “WU” command.

16.30 Write Variable - Integers (A to Z)

Command Format

W	V	I	*alphabet*	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

A,B.C....Z 8 Hexadecimal Digit for 32-bit integer

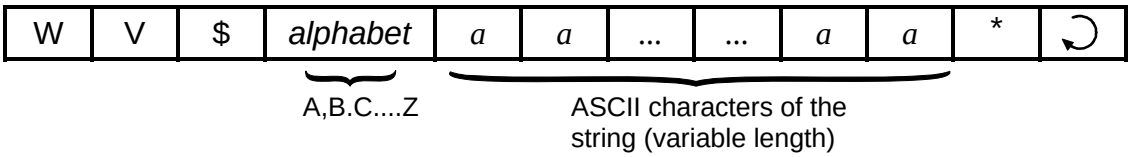
Response Format

W	V	I	*	↵

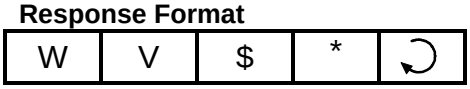
E.g. To assign variable “K” to number 56789_{10}(=$0DD5_{16}$), send hostlink command “`WVIK00000DD5*`”.

16.31 Write Variable - Strings (A$ to Z$)

Command Format



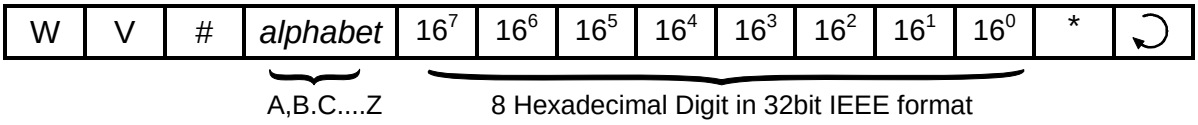
Response Format



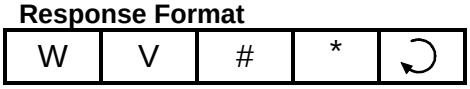
E.g. To assign the string " Super PLC" to the string variable P$, send hostlink command "WV$P Super PLC*".

16.32 Write Variable – Floating-point (A# to Z#)

Command Format



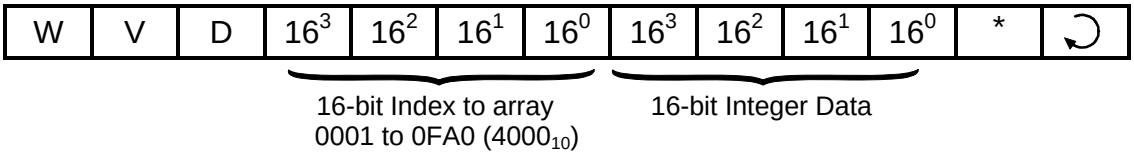
Response Format



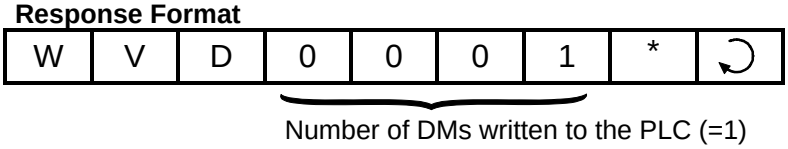
E.g. To write to A# the value 6.265599E9, send command "WV#A4FBABABABA*" where "4FBABABABA" is the IEEE 32-bit representation of the floating-point number 6.265599E9. See TL7 Programmer's Reference on how float number are represented using 32-bit integer in IEEE single precision format in I-TRiLOGI.

16.33 Write Variable – Singe Data Memory (DM[1] to DM[4000])

Command Format



Response Format



E.g. To write the value 1234_{10} (=$4D2_{16}$)to DM[1000], send hostlink command "WVD03E804D2*". (1000_{10} = $3E8_{16}$)

Note: The response to WVD command in the Nano-10, FMD and F-series was simply "WVD*" without the variable count.

16.34 Write Variable – Multiple Data Memory (DM[1] to DM[4000])

(Note: This new command is available only to Fx PLC)

Command Format

W	V	D	16^3	16^2	16^1	16^0	16^3	16^2	16^1	16^0	16^3	16^2	…	…

16-bit Index to array 0001 to 0FA0 (4000_{10}) — 1st 16-bit Integer — 2nd 16-bit Integer

…	…	16^3	16^2	16^1	16^0	*	↵

last 16-bit integer

Response Format

W	V	D	16^3	16^2	16^1	16^0	*	↵

Number of DMs written to the PLC

16.35 Write Variable – Singe Float Array (FP[1] to FP[1000])

(Note: This new command is available only to Fx PLC)

Command Format

W	V	F	16^3	16^2	16^1	16^0	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

16-bit Index to array 0001 to 03E8 (1000_{10}) — 8 hexadecimal digit in 32-bit IEEE format

Response Format

W	V	F	0	0	0	1	*	↵

Number of FP[ ] written to the PLC (=1)

E.g. To write the value 6.265599E9 (=4FBABABABA in IEEE) to FP[1000], send hostlink command "WVF03E84FBABABABA*".

16.36 Write Variable – Multiple Float Array (FP[1] to FP[1000])

(Note: This new command is available only to Fx PLC)

Command Format

W	V	F	16^3	16^2	16^1	16^0	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	…	…

16-bit Index to array — 1st IEEE float data — 2nd float

…	…	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

last IEEE float data

Response Format

W	V	F	16^3	16^2	16^1	16^0	*	↵

Number of FP[ ] written to the PLC

16.37 Write Variable - System Variables

System Variable	type
input[ ]	01
output[ ]	02
relay[ ]	03
timer[ ]	04
ctr[ ]	05
timerPV[ ]	06
ctrPV[ ]	07

System Variable	type
clk[ ]	08
date[ ]	09
-	0A
ADC*	0B
DAC*	0C
PWM*	0D

*Not a system variable in TBASIC

Command Format

W	V	S	n	n	16^1	16^0	16^3	16^2	16^1	16^0	*	↵

type Index 16-bit Integer Data

type (01 to 0D) - denotes the type of system variable to access,
index (01 to 1F) - index into the array, starting from 01.

Response Format

W	V	S	*	↵

E.g. To set clk[1] (which represents the hour of the RTC) to 14, send the command "WVS0801000E*" to the PLC.

16.38 Write Variable - High Speed Counter HSCPV[ ]

Command Format

W	V	H	n	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

1 or 2 8 Hexadecimal Digit for 32-bit integer

Response Format

W	V	H	*	↵

E.g. To clear the value of HSCPV[2], send hostlink command "WVH200000000*".

16.39 Halting the PLC

Command Format

C	2	*	↵

Response Format

C	2	*	↵

When the PLC receives this command, it temporarily halts the execution of the PLC's ladder program after the current scan. However, the PLC continues to scan the I/Os and processes host link commands sent to it and will report the current I/O data and internal variables to the host computer.

16.40 Resume PLC Operation

Command Format

C	1	*	↵

Response Format

C	1	*	↵

When the PLC receives this command, it will resume execution of the ladder program if it had been halted previously by the "C2" command. Otherwise, this command has no effect.

16.41 Read Analog Input

This command forces the PLC to refresh the value of its ADC data at the analog channel before returning its value in the response string (i.e. no need for the PLC to execute ADC(n) function to refresh the analog input)

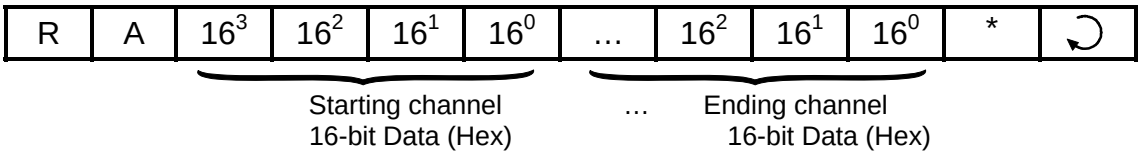
Command Format

R	A	n	n	c	c	*	↵

Starting Analog Channel # (01-08h)

Channel count (01 to 08h)

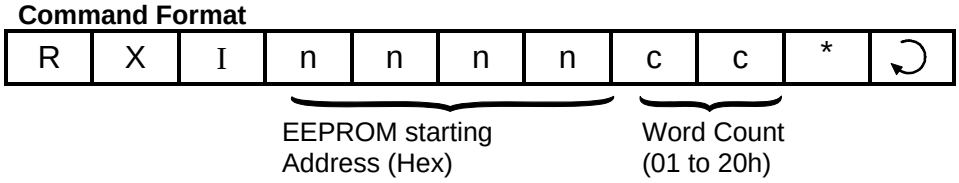
Response Format



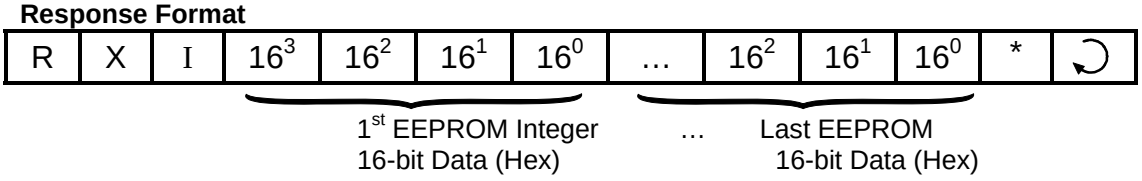
E.g. To read 4 channels of Analog inputs starting from Ch #2, Send "`RA0204*`". The response string will contain 4 sets of data for channel 2, 3, 4 and 5.

16.42 Read EEPROM Integer Data

Command Format



Response Format

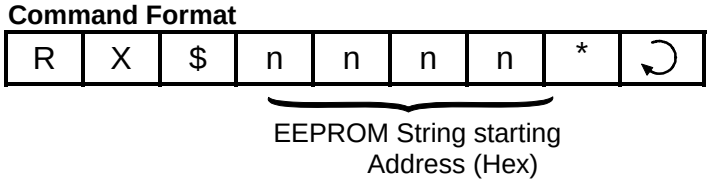


Maximum allowable word count per command is 32 (01 to 20 Hex). If "count" is > 32, only the first 32 words will be returned.

E.g. To read the 10 words of EEPROM data starting from address 100, send host-link command "`RXI00640A*`". The response string will contain 10 sets of 16-bit data (4 ASCII hex digit per set).

16.43 Read EEPROM String Data

Command Format



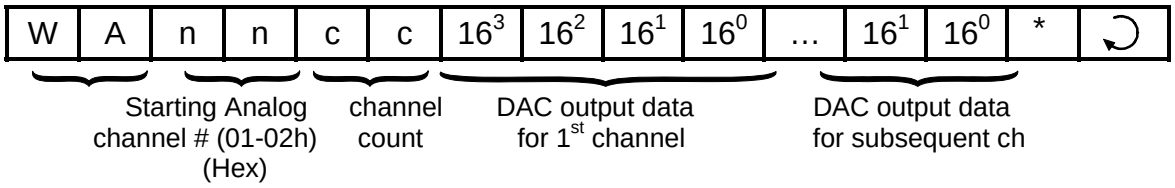
Response Format

E.g. To read the string data stored at EEPROM address 10, send host-link command "`RX$000A*`". The response string will contain string data stored in the EEPROM (maximum 40 characters).

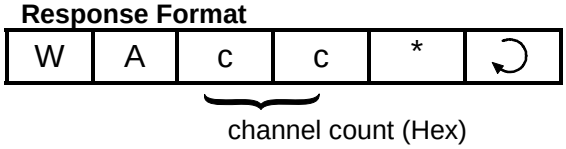
16.44 Write Analog Output

Upon receiving this command, the PLC updates the value of its DAC data at the analog output channel (i.e. no need for PLC to execute SETDAC to update the analog output).

Command Format

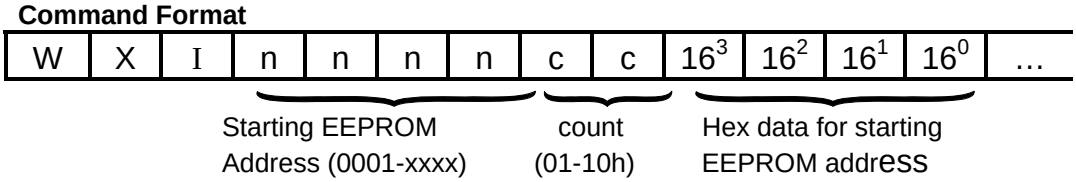


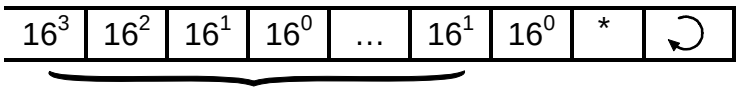
Response Format



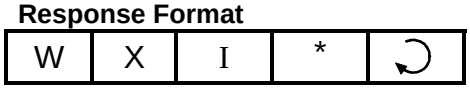
16.45 Write EEPROM Integer Data

Command Format





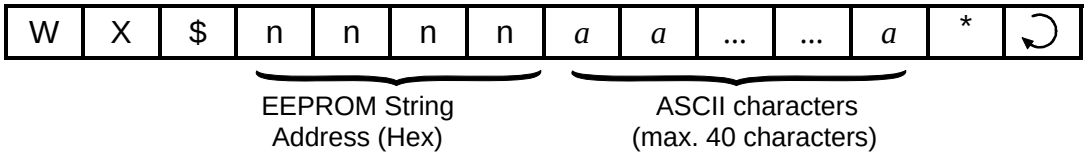
Response Format



Maximum allowable word count per command is 16 (01 to 10 Hex).

16.46 WRITE EEPROM String Data

Command Format



Response Format

W	X	$	*	↵

E.g. To write the string data "Hello TRi" at EEPROM String address 12, send host-link command "`RX$000CHello TRi*`".

16.47 Force Set/Clear Single I/O Bit

This new "Wbnnnnxx" command allows you to change a single I/O bit on the PLC. You can force set or clear any single input, output, relay, timer or counter bit. This has an advantage over other write commands such as WI, WO, etc that affects the entire group of 8 or 16-bits organized into "channels".

Command Format

W	b	n	n	n	n	x	x	*	↵

I/O Bit address (Hex)

00 – Clear I/O bit (OFF)
FF – SET I/O bit (ON)

Response Format

W	b	*	↵

I/O Type	**Bit address nnnn (Hex)**
Input #1 to #256	0000 to 00FF
Output #1 to #256	0100 to 01FF
Timer #1 to #256	0200 to 02FF
Counter #1 to #256	0300 to 03FF
Relay #1 to #256	0400 to 04FF
Relay #257 to #512	0500 to 05FF

E.g. To force output 1 to ON, send "`Wb0100FF*`". To turn it OFF, send "`Wb010000*`"

16.48 Using OMRON Host Link Commands

Since the PLCs also support OMRON C20H Host Link commands, which are very similar in construct to our multi-point command/response format, you can also make use of OMRON commands to supplement the native host link commands.

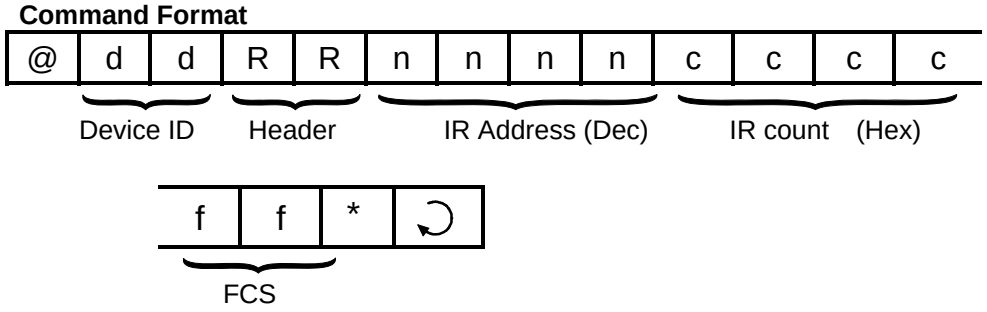
We will only discuss four of the OMRON host link commands "RR", "WR", "RD" and "WD" in this section because these commands can be used by users to read/write to **multiple** I/O registers and data memory in a single command (Note: maximum length of command string should be <=80 characters).

Note: Fx PLC native protocol command set now includes reading and writing of multiple DM and FP that are not available on F2424/F1616-BA, FMD and Nano-10 PLCs. To maintain backward compatibility with FMD and Nano-10 PLCs that only support read/write of single DM (there is no FP[ ] in these PLCs), these OMFRON host link commands may be used to read/write multiple DM locations in a single command.

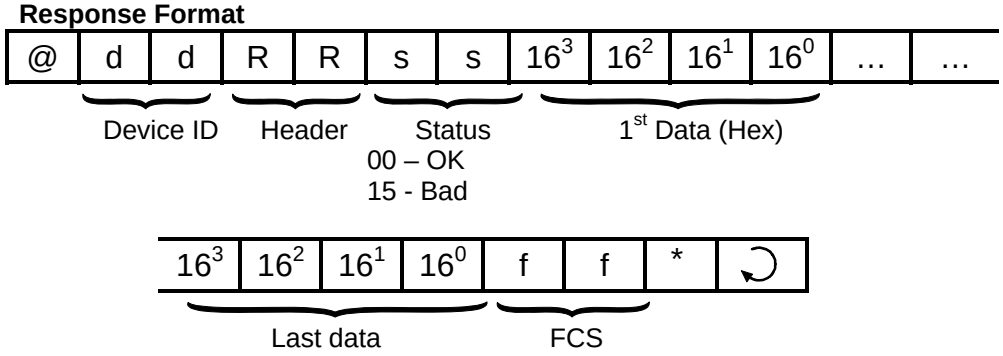
16.48.1 Read IR Registers

This command refers to Table 14.1 in Chapter 14 to map the PLC's I/Os to OMRON IR register space from IR0 to IR519

Command Format



Response Format



E.g. To read Timer PV #1 to #7 using this command, send:

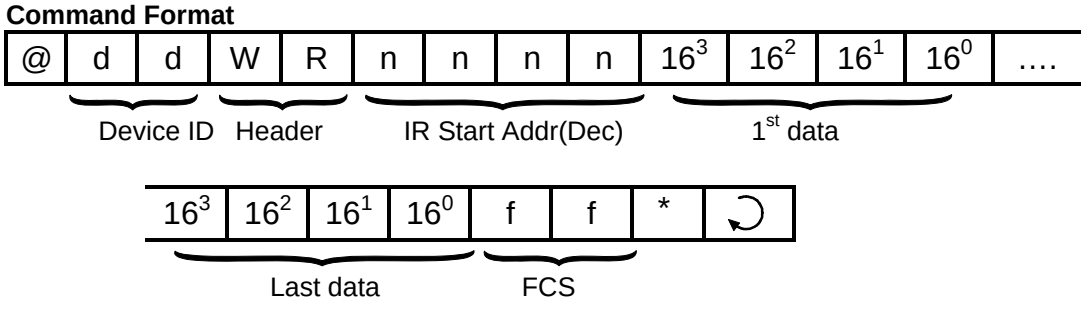
"@01RR012800074D*"

The PLC will send return a response "@01RR00xxxxyyyyzzzz….*"

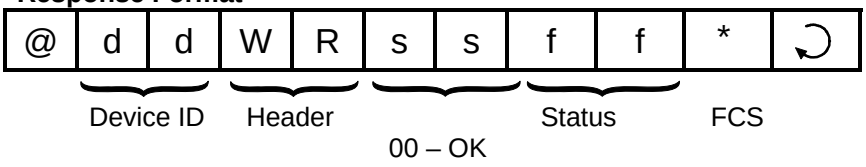
16.48.2 WRITE IR Registers

This command refers to Table 14.1 in Chapter 14 to map the PLC's I/Os to OMRON IR register space from IR000 to IR519

Command Format



Response Format

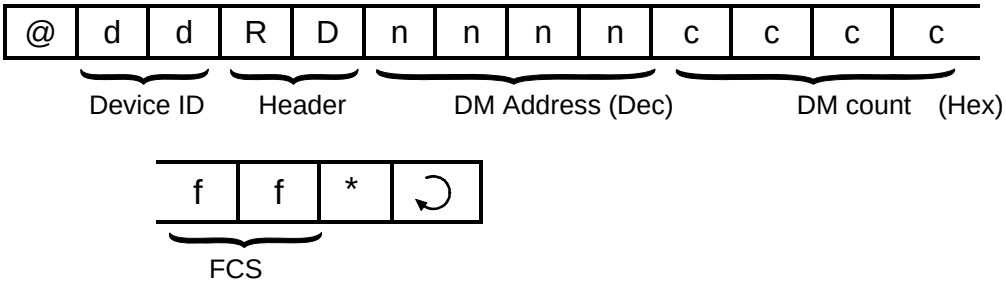


E.g. To Write to CtrPV #1 to #2 using this command, send: "@01WR0256xxxxyyyyff*"

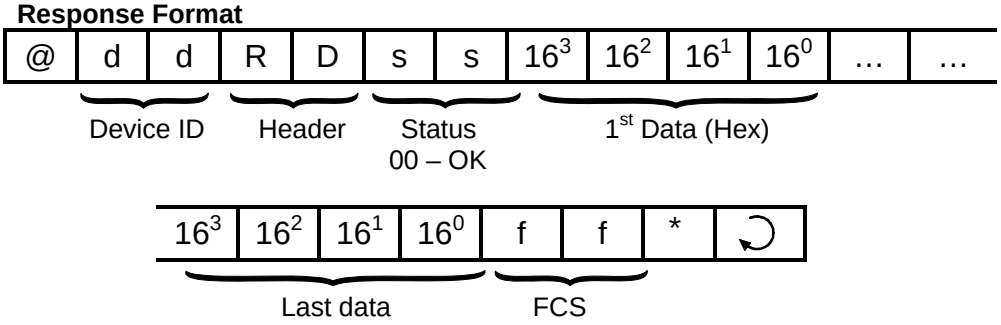
where xxxx and yyyy are the hex values to be written to CtrPV 1 & 2.

16.48.3 Read Data Memory DM[1] to DM[4000]

Command Format



Response Format

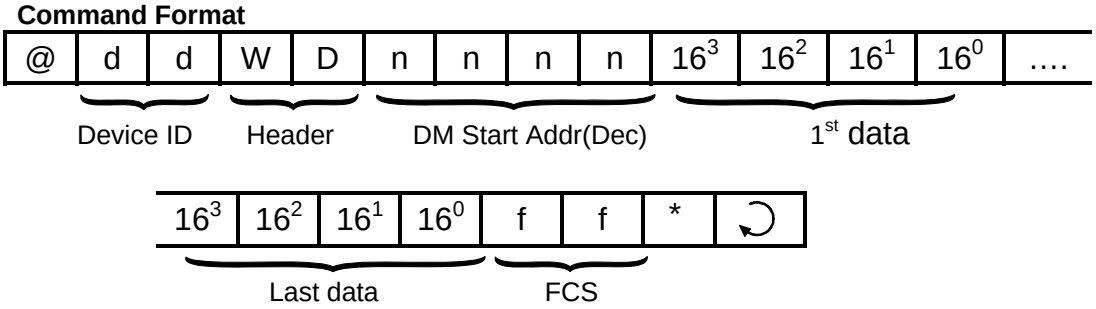


E.g. To read DM#112 to #130 (19 words), send: "@01RD0112001357*"

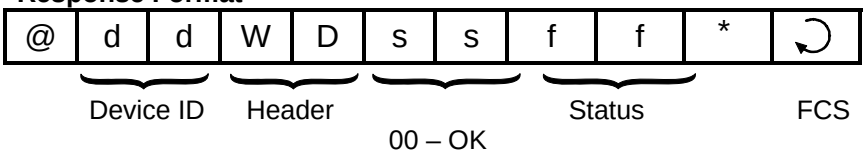
The PLC will send return a response "@01RD00xxxxyyyyzzzz…*"

16.48.4 WRITE Data Memory DM[1] to DM[4000]

Command Format



Response Format



E.g. To Write to DM#1200 to #1201 using this command, send: "`@01WD1200xxxxyyyyff*`"

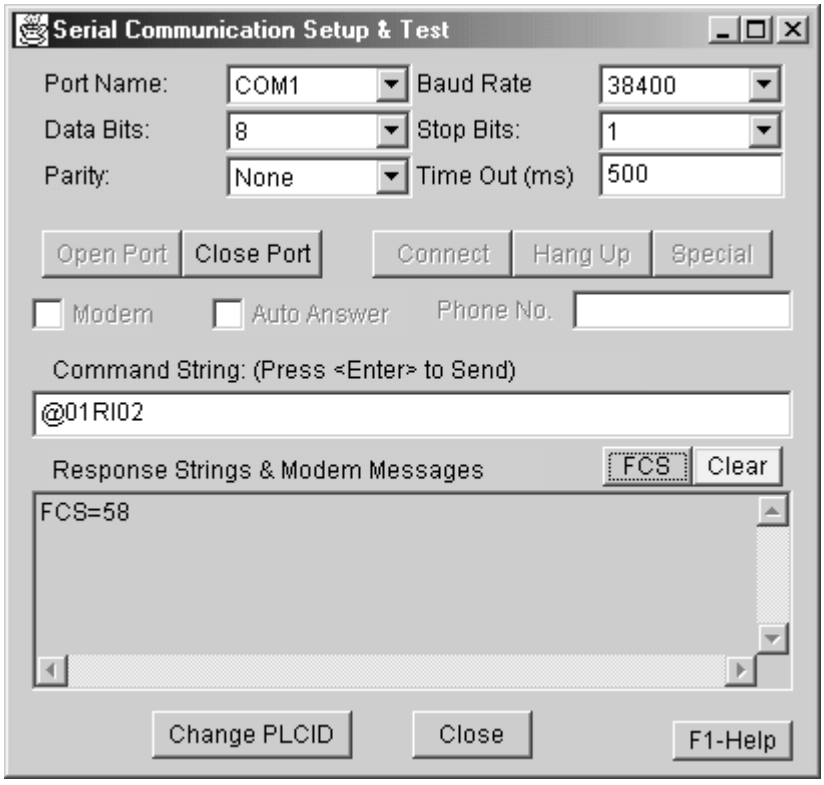
where xxxx and yyyy are the values to be written to DM[1200] & DM[1201].

16.49 Testing of Host Link Commands

You can try out all the Host-Link commands using TLServer's "Serial Communication Setup". However, TLServer is designed to accept only multi-point protocol commands except the "IR*" command (which is necessary to obtain the device ID from the PLC). You, therefore, have to enter all your host link commands in multi-point format.

Since the multi-point protocol requires a FCS (frame check sequence) character to be appended to the end of the command string, you may be able to get around it by using the "wildcard" FCS "00" in place of the actual FCS. E.g. To read input channel 02 from a PLC with ID = 01, you can enter the command string as "`@01RI0200*`".

For TLServer version 2.1 and above, there is an "FCS" button that lets you compute the actual FCS for the string in the command string text field. You can then use the actual FCS with the command string to completely test your command. E.g. If you type in the string "`@01RI02`" in the command string (but do not press Enter) and then click on the "FCS" button, the FCS for this string will be computed and shown as "FCS = 58", as shown in the following figure:



You now can enter the complete command string as "@01RI0258*" and it will be accepted by TLServer. (Note: If the PLC has executed a SETPROTOCOL *n*,5 to configure its serial port into pure native mode, then wildcard FCS will not be accepted and you must use the actual FCS with your command. The FCS button makes it much easier than computation by hand).

If you have changed some data using the write command, then activate On-Line Monitoring and examine the changes made using the "View Variables" window.

16.50 Visual Basic Sample Program

To help users get started writing their own Visual Basic program to communicate with the PLC, we have created a sample Visual Basic program with full source code listing. Please visit the following web page to download the visual basic sample program.

http://www.tri-plc.com/applications/VBsample.htm

16.51 Inter-PLC Networking Using NETCMD$ Command

All Fx PLCs are able to send out host link commands to other Fx PLCs, M-series or H-series PLCs using the built-in TBASIC function NETCMD$(). This function accepts host link commands in multi-point format and automatically computes the Frame Check Sequence (FCS) characters, appends them to the command string and sends out the whole command string together with the terminators. The function then waits for a response string and checks the integrity of the received response string for errors. This function returns a string only if a proper response string has been received. Please refer to the TBASIC Reference for a detailed explanation of this command.

The NETCMD$() function therefore greatly simplifies the programming tasks for handling networking between PLCs. The programmer only needs to construct the correct command string according to the formats described in this chapter, pass the formatted string to the NETCMD$() function, and then check for the response string. A Fx PLC may use the NETCMD$ to map the I/O of another PLC into its internal relays and use the other PLC as its remote I/O.

There are some programming examples in your "TRILOGI\TL6\usr\samples" folder that illustrate the use of NETCMD$() to map I/Os of a slave PLC to the master. Please study the two examples: "RemoteIO-Hseries.PC6" and "RemoteIO-Mseries.PC6" carefully to understand the mechanism of mapping I/Os between the PLCs. The TRiLOGI program "REMOTE-Hseries.PC6" will work on the H-series, M-series or F-series PLCs as slaves, whereas the program "REMOTE-Mseries.PC6" will only work with F-series, FMD or Nano-10. This is because the Fx PLC host link command set is a superset of the H-series host link command set, and this example uses the more efficient M-series host link commands to read/write 16-bit data for networking between M-series PLCs.

An application note and example programs demonstrating how to use our other PLC models as slave remote I/O for the F- or M-series PLC can be found at the following web page:

http://www.tri-plc.com/appnotes/AppnoteMain.htm

16.52 Inter PLC Networking Using MODBUS Protocols

The PLCs may also pass data to each other using special MODBUS commands, which are even simpler to use than NETCMD$ but are restricted to accessing variables that are mapped into MODBUS address structure. Please refer to the Section 14.7 and 14.8 as well as the TBASIC Reference manual for details on using the READMODUS, WRITEMODBUS, READMB2 and WRITEMB2 commands.

Chapter 17 IR Remote Control

17 IR REMOTE CONTROL

17.1 Introduction

The Fx2424 PLC features a built-in, Infra Red remote control sensor with 38KHz modulation band pass filter (part # TSOP32138), located as shown in the following picture:

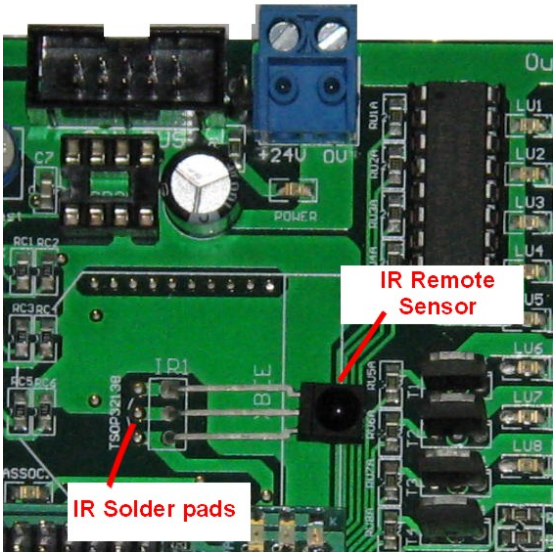


Figure 17.1

The IR remote sensor enables the PLC to do the following :

1. Receive and decode IR remote control signals sent by hand-held IR remote controls that are encoded with Sony Corporation's™ SIRC protocol (which really is just any hand held remote control that comes with your SONY TV or VCR). The decoded signal can be used to trigger any actions under your program control and it is unbelievably easy to program!

2. "Learn" (i.e. records) up to 128 IR remote codes (can be a mixture of different brands of devices) it receives into non-volatile flash memory and playback* the recorded IR remote codes under your program control to turn ON/OFF or to adjust AV devices, air-conditioning settings etc.

   * To playback the recorded IR remote codes you need to purchase a low cost IR LED and a resistor and connect it to one of the PLC's PWM output channels (see Section 17.4 for details). The IR LED emitter is not included with the PLC.

17.2 Applications of IR Remote Control

1. Use your IR remote control to Start/Stop your machine or activate some command remotely.

2. Use the remote control as a keypad to enter parameters into the PLC (preferably use one with an LCD display so that you can see what you have entered).

3. Use the remote control to enter a multi-digit security code to unlock certain routines or to run some program code (e.g. setup or calibration routine) that is not normally accessible by the equipment operator.

4. Teach the PLC different remote control codes so that the PLC can be programmed to control a combination of different devices by sending out IR signal to turn them ON/OFF or to adjust their settings. This is extremely useful for home automation, HVAC or AV control tasks

17.3 Decoding Sony IR Remote Control Signal

You really don't have to learn anything about how the IR signal is encoded and decoded in the Sony SIRC protocol in order to use it with the Fx2424 PLC. This is because the PLC's operating system takes care of decoding the signal and it simply presents the data to you as a single 16-bit binary word. However, for those who are interested in how the IR signals are encoded by a Sony IR remote control you can refer to this web site: http://www.sbprojects.com/knowledge/ir/sirc.htm

To enable the IR Remote decoding in the PLC, you need to run the following command once.

```
SETSYSTEM 17,1  ' enable IR remote
```

This command enables the PLC's IR remote interrupt routine and it will sample the incoming IR signal. This interrupt routine will be disabled when the PLC is turned OFF, or a SETSYSTEM 17,0 command is run, or when an IR playback function is invoked (see section 17.4).

The IR signal received can be decoded and read by the PLC program using the TBASIC command "STATUS(17)".

```
E.g.        X = STATUS(17)     ' read IR code into variable X
```

THAT'S IT!!!

When you press a different key on the Sony IR remote control, the STATUS(17) function will return a different number. E.g. If you press the '1' key on the TV remote control, STATUS(17) returns the hexadecimal number "&H80", the "2" key returns "&H81"…. and if you press the '0' key on the TV remote, STATUS(17) returns "&H89". Do note that a Sony VCR remote would return a different set of data because a VCR has a different "device" code from the TV. You can then simply use the IF.. THEN statement to perform different tasks based on the key being pressed.

Note:

1. If no key is pressed, STATUS(17) returns –1.

2. If a key is pressed down and not released, then the first read of the STATUS(17) returns the code correctly, but a repeat read of the STATUS(17) function would return –1 until the key has been released and a new key or the same key is being pressed again.

A sample program "`IRremoteControl.PC6`" that turns on output 1 to 10 in response to a key-press of 0 to 9 received from a Sony TV remote can be downloaded from: http://www.tri-plc.com/trilogi/FPLCsamples.zip . This program also demonstrates how to turn the key presses on the IR remote into a numerical keypad. The entered number is displayed on the LCD display along with the IR code. You can use the "volume-up" button to accept the entry into variable B and use the "volume-down" button to delete the wrongly entered number. The sample program is ready to use and you can transfer it to the PLC and use a Sony TV remote control to test it immediately.

17.3.1 Relocating The IR Remote Sensor

If the PLC is installed inside an opaque enclosure and you want to be able to aim your remote at it without opening the enclosure, then you need to relocate the IR sensor to the external surface of the enclosure. We enable you to do this easily by installing the IR sensor with long leads so that you can cut the lead and relocate it to a desirable location. You will need to connect 3 wires to the leads of the sensor and the other ends of the wires are to be soldered to the Fx2424 PCB. We have provided 3 blank solder pads next to the original sensor solder pad just to make life a little easier for you, as shown in Figure 17.1

17.4 Learning and Playing back IR Remote Control Signals

Unfortunately there isn't any international standard for the IR remote control protocol and each brand of TV, VCR, amplifier and air conditioning, etc manufacturer actually implements their own proprietary protocols as they wish. Fortunately, most of the IR remote controls encode the digital '1' and '0' using pulses of IR beams modulated at a frequency between 36 to 42 KHz, with a 38KHz carrier frequency being the most common. The main difference among most manufacturers' implementation is the duration of the ON and OFF time of these modulated IR beams that are used to represent logic '0' and logic '1'.

The IR remote control sensor used on the Fx2424 is optimized for a 38KHz carrier frequency but it is also able to receive signals encoded by a wide range of carrier frequencies between 36KHz and 42 KHz.

Although the PLC firmware is designed to only decode IR signal based on Sony SIRC protocol, the IR remote sensor has another important use, which is to "learn" (record) IR remote control signals sent by most other brands of IR remote control. The PLC does not attempt to decode these signals. Instead, it simply measures the pulse width of the ON and OFF times of the carrier pulses and store this timing information into the PLC's internal flash memory. This pre-recorded IR signal data can then be called up under the user's program control to be played back through an IR LED emitter (user-supplied) connected to one of the PLC's digital outputs, #5 to #8.

The PLC uses the PWM capability of any one of these 4 outputs to generate the 38KHz modulating signal at a 33% duty cycle and uses the recorded timing information to re-construct the IR signals.

Up to 128 IR remote control signal sets, each with up to 128 bits of data can be recorded by the Fx2424. Any of the pre-recorded IR data can be called up in any order by a user's TBASIC program. This makes the Fx2424 extremely useful for home automation tasks that involve controlling devices that can only be controlled by an IR remote control signal.

17.4.1 Recording IR Remote Control Data Set

After the `SETSYSTEM 17,1` statement has been executed as mentioned in Section 17.3, the PLC will automatically record the IR remote data set into the "Current IR Buffer". This buffer can only hold one IR data set of not more than 128 bits of information at any one time. A new data set received will always overwrite the last data set.

The Fx2424 CPU has dedicated 64Kbytes of flash memory to store up to 128 data sets. The data set in the "Current IR Buffer" can be saved to any of the 128 locations, n, using the following command:

```
SETSYSTEM 17, n + 2000
```

That is to say, if you run the `SETSYSTEM 17,xxx` with the second parameter that is between 2001 and 2128, the CPU will record the data set in the "Current IR Buffer" into the reserved flash memory.

17.4.2 Loading IR Remote Control Data Set

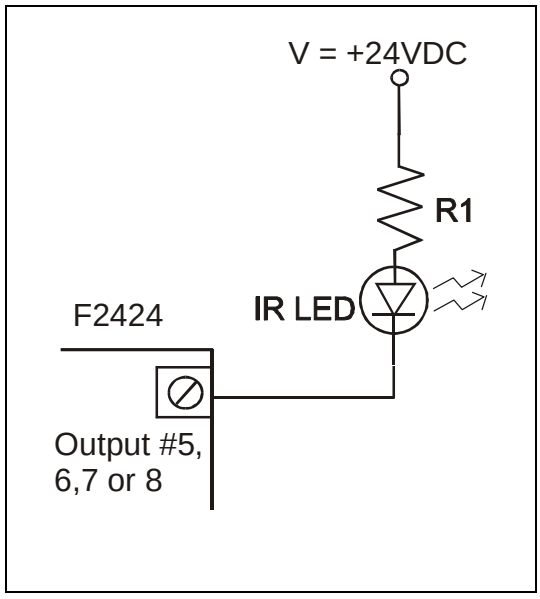
Any of the 128 data sets that are previously stored in the flash memory can be loaded back into the "Current IR Buffer" using the following command:

```
SETSYSTEM 17, n + 1000
```

That is to say, if you run the `SETSYSTEM 17,xxx` command with the second parameter that is between 1001 and 1128, the CPU will load the previously recorded data back into the "Current IR Buffer". You can use the techniques discussed in the next subsection to "playback" the IR remote signal that is currently in the "Current IR Buffer".

17.4.3 Playback of IR Remote Control Signal

If your application requires the PLC to send out IR signals to your devices, then you have to construct an IR emitter as shown in the following diagram:



Power Supply: Can be any voltage from +5V to +24V DC, but most convenient to use the same +V of the PLC's power supply.

IR LED: (Not included) - Most Infra Red LEDs that you purchase from Digikey can be used. You may also salvage one from an old remote control.

R1 : It depends on the current, `I`, that the LED can handle and the output power you want. Typically 20 to 50mA should be adequate.

```
R1 =  (V  - 2) / I
```

E.g. for I = 50mA, V = 24V, R1 = 440 ohms.
R1 power rating = I^2R = 1.1 W => Use a 3W resistor.

The LED emitter circuit is to be controlled by one of the PLC's PWM outputs.

To playback the IR remote data set in the "Current IR Buffer" through PWM channel #n, you simply execute the following command:

```
SETSYSTEM 17, n + 10
```

i.e. If you run `SETSYSTEM 17,11` , the PLC will re-construct the IR signal using the data in the "Current IR Buffer" and send it out via IR emitter connected to its PWM channel #1 (D/O #5).

Likewise, if the IR emitter is connected to PWM #4 (D/O #8), then you need to run the command "`SETSYSTEM 17, 14`" to turn ON the IR emitter.
The PLC actually uses the PWM output(s) to turn the IR LED ON and OFF to produce the 38KHz carrier frequency at 33% duty cycle.

Important Notes:

1) The IR remote control sensing interrupt will be disabled automatically when you playback the IR control signal. This is to prevent the reflected IR signals from corrupting the data in the "Current IR Buffer". Hence if you need the PLC to re-learn a new IR remote signal after you have done a playback, you will need to execute "SETSYSTEM 17, 1" again to enable the interrupt service routine.

2) It is possible to connect multiple IR emitters to the same PWM outputs especially if these emitters are going to different rooms. The PMW outputs on the Fx2424 are capable of sinking up to 8A peak and 4A continuous current and hence it is able to support many IR emitters simultaneously.

3) Since any of the 4 PWM outputs can be used, it is also possible to use more than one PWM output to control multiple devices selectively even though they may all share the same IR codes. For example, you have 4 bedrooms and all of them are installed with same Panasonic brand of air conditioner. If you connect 4 IR emitters to the same PWM output, then all 4 air conditioners will be activated at the same time when the PWM channels output the IR control signals. But if these 4 emitters are each connected to a different PWM output, then you can easily select which air conditioner to turn ON or OFF.

17.5 SETSYSTEM 17, xxx Command Summary

The SETSYSTEM 17,xxx command is used by the Fx2424 PLC exclusively (not available on M-series PLCs) for controlling its IR remote control capability. The following summarizes the various parameters xxx for this purpose.

Xxx	Functions
0	Disable IR Remote Sensor
1	Enable IR Remote Sensor
11 – 14	Playback current IR data through PWM channel #1 to #4
1001 – 1128	Load the stored IR data set #1 to #128 into "Current IR Buffer"
2001 – 2128	Store the data in the "Current IR Buffer" to flash memory data set location #1 to #128.
*101 to 150	PWM carrier frequency in KHz (Default is 138 which means 38KHz)
*233 to 250	PWM duty cycles. (Default is 233 which means 33% duty cycle)

* You normally should not change these parameters from their default values.

Chapter 18 Light Dimmer Control

18 LIGHT DIMMER CONTROL

18.1 Introduction

You may have used one of those commercial light dimmers to control the brightness of incandescent light bulbs. Modern light dimmers are mostly based on controlling the “firing angle” of a type of semiconductor switch known as “TRIAC” (which stands for “AC transistor”).

A TRIAC is a semiconductor switch that conducts current when a trigger voltage is applied to its “Gate” terminal. However, what makes the TRIAC different from a typical transistor is that once the TRIAC starts conducting current, it cannot be switched off even if the voltage to the gate is removed. I.e. The TRIAC switch is “latched” once it is turned ON. That also means that you only need to apply a very short trigger pulse to the “Gate to turn it ON. In control term this is often known as “firing” of the TRIAC.

Since the TRIAC is latched ON and cannot be turned OFF regardless of what voltage or no voltage is applied to the “Gate”, it will be quite useless for switching DC current as the switch cannot be turned OFF! However, when a TRIAC is used to switch AC current, since at every half cycle the AC voltage must change polarity, that means the TRIAC will be switch OFF automatically during zero-crossing. Once the TRIAC is switched off, it can only be switched ON again by applying a triggering voltage pulse (also known as “firing pulse”) to the “Gate” again and this process must be repeated every half cycle. This is illustrated in the following two figures.

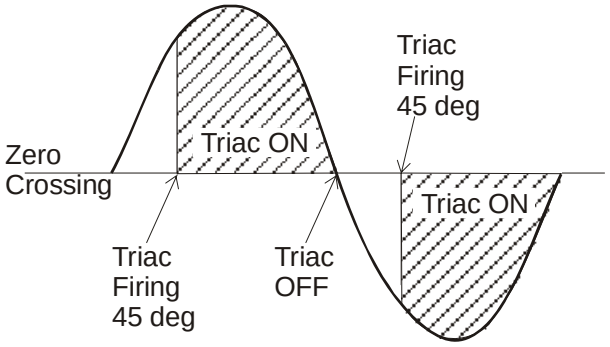


Figure 18.1(a)

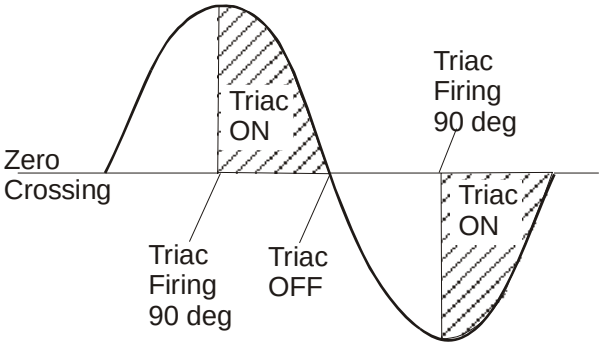


Figure 18.1(b)

In Figure 18.1(a) the TRIAC is triggered at the phase angle = 45 degree, whereas in Figure 18.1(b) the TRIAC is fired at phase angle = 90 degree. The TRIAC conducts the AC current immediately upon receiving the firing pulse and will only turn itself OFF at the next zero crossing. In the next half cycle the TRIAC needs to be fired again in order to conduct and this process repeats itself indefinitely.

You can easily see that the TRIAC in Figure 18.1(a) is turned ON earlier compared to the TRIAC in Figure 18.1(b). This means that when a load such as an incandescent lamp is connected to the TRIAC, it will be turned ON a greater proportion of time using the firing angle shown in Figure 18.1(a) and therefore will appear brighter compared to the that in Figure 18.1(b). By controlling the “firing angle” applied to the TRIAC you can achieve the objective of controlling the brightness of the lamp. The lamp will be brightest when the firing angle is near 0 degree and will be dimmest when the firing angle is near 180 degree.

Controlling lamp brightness this way is highly efficient. Unfortunately the transfer function is non-linear (i.e. the brightness is neither proportional non inversely proportional to the firing angle) and therefore it has limited applications in precision digital control system. But it can provide low cost control for devices such as lamps, since the operator who receives instant visual feedback can vary the firing angle manually until a satisfactory brightness is obtained.

In a commercial light dimmer, a potentiometer is typically used with a capacitor to vary the timing where the TRIAC is allowed to start conducting, as shown in the example circuit in Figure 18.2:

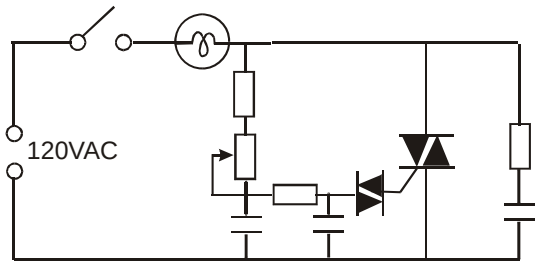


Figure 18.2 – Commercial Light Dimmer Circuit

On the Fx2424 we will make use of the computational power of the CPU to control the firing angle and therefore achieving the same effect as light dimmer control.

18.2 Control of 120V/240V AC TRIAC

Since 120V/240V AC voltages power most incandescent light bulbs, it is essential to provide galvanic isolation between the AC lamp power and the PLC's low voltage electronics. What you should use is an opto-isolated TRIAC that can be controlled by a low DC voltage between 3 and 30V. These kinds of TRIACs are commonly known as "Solid State Relay" (SSR), as illustrated in the following diagram:

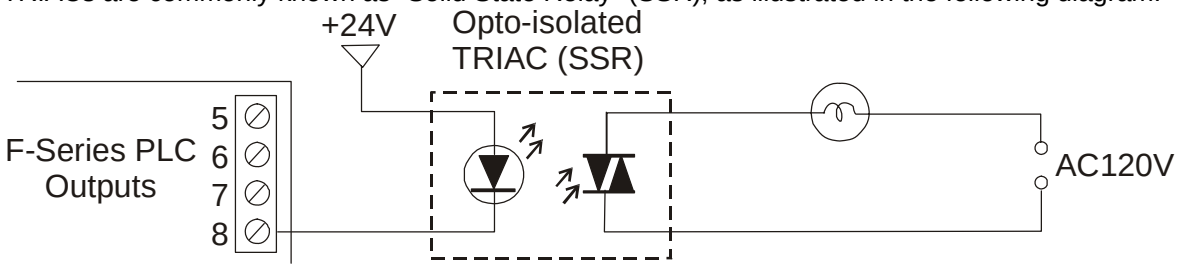


Figure 18.3

In the above circuit, if the PLC turns on its output #8, it will provide continuous trigger voltage to the TRIAC gates and therefore the lamp can be switched on at 100% brightness. When the PLC turns off its output #8 the lamp will be turned OFF completely.

To achieve light dimming control, the PLC must determine the precise moment that the AC voltage crosses the zero voltage point and the PLC will then schedule a time delay and then send a short firing pulse via its digital output to the TRIAC to turn it on for the rest of the half cycle. Since the TRIAC will turn itself OFF in the next zero crossing, the PLC must do this repeatedly at every zero crossing in order to maintain the fractional power being applied to the lamp.

The Fx2424 is capable of controlling up to 12 SSR for light dimmer control using its digital outputs #5 to #16. Section 18.4 explains how to use a new i-TRiLOGI function to readily program these light dimmers ,

Important Note: There are two types of SSR on the market – one type is called "Zero Crossing SSR" which only switches ON when the AC voltage crosses the zero volt line. Zero crossing type SSR **CANNOT** be used by the PLC for light dimmer control. Instead you must purchase the "Non-Zero Crossing" type of SSR for this purpose.

18.3 Zero-crossing Detection

As explained in 18.2, the Fx2424 CPU must know the precise moment the zero crossing occurs so that it can schedule the time interval when it needs to send out the firing pulse to turn on the SSR. This can be accomplished by feeding the AC power through a bi-directional opto-isolator and the output would drive input #6 of the Fx2424 PLC. The falling edge of the signal detected at input #6 would send an interrupt to the CPU to start the scheduling of the various time slots that each light dimmer output is to fire a triggering pulse. This means that if you want to use the light dimmer control, you must configure input #6 as an interrupt input for capturing the zero crossing signal and therefore input #6 will no longer be available for other functions (such as for HSC or Pulse Measurement)

Note that since all Fx2424 inputs are 24V NPN only, you will need to construct the following interface circuits to feed the AC signal into the PLC's input #6:

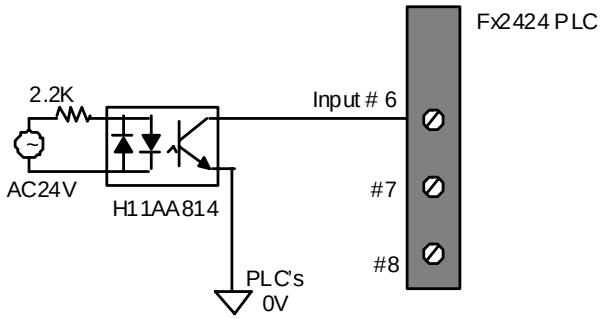


Figure 18.4

The above shows the wiring diagram of using an external opto-isolator to provide the zero-crossing signal to the PLC's NPN input #6.

18.4 Programming Light Dimmer Control

18.4.1 Enable Zero-Crossing Interrupt

Before programming any of the light dimmer channels, you first need to enable the zero-crossing interrupts for the zero-crossing signal fed into input #6. This is accomplished using the following command:

```
SETSYSTEM 22,1   ' Enable zero crossing interrupt
```

18.4.2 Programming Individual Dimmer Channel

You can program up to 12 of the Fx2424 PLC's outputs #5 to #16 to control SSR for dimmer control purposes using the SETDIMMER command (available only on i-TRiLOGI version 6.23 onwards). The syntax is as follows:

SETDIMMER *ch*, *value*

ch = Light dimmer channel (1 to 12)
This is mapped to the Fx2424 PLCs outputs as follows:

ch	F-series Output #
1	5
2	6
3	7
4	8
5	9
6	10
7	11
8	12
9	13
10	14
11	15
12	16

value = Integer multiplier of an internal timer resolution (approximately 53 microsecond) to schedule the next firing time from the last zero crossing.

For 50Hz AC signal, range = 1 to 188
For 60Hz AC signal, range = 1 to 156

The lamp will be fully ON (100% brightness) when **value* = 1 and fully OFF (0% brightness) when ***value*** is near the maximum stated above. **DO NOT** set any of the dimmer channel with ***value*** greater than the maximum allowed for the corresponding AC frequency as it will lead to poor interrupt timing of the CPU and can affect all the dimmer channel (if you see the light flickering repeatedly then some of the dimmer channel could have been set with an out of range value.)

* Note: Since the CPU needs a minimum amount of time delay to fire the TRIAC after the last zero crossing, the minimum ***value*** cannot equal 0.

Also, please be reminded again that the voltage applied to the load is non-linear with respect to the firing angle controlled by the ***value*** parameter.

18.4.3 Programming Example

Please find the demo program “SetDimmer.PC6” in the zip file that you can download from:

http://www.tri-plc.com/trilogi/FPLCsamples.zip

The demo program lets you experiment with turning a lamp slowly ON (Fade ON effect) and slowly OFF (Fade OFF effect) as well as letting you experiment with manual control of the lamp’s brightness using two push buttons.

Chapter 19 ZigBee Networking

19 ZIGBEE NETWORKING

19.1 Introduction

ZigBee™ (trademark of ZigBee Alliance) is the commercial name of a specification for a suite of high-level communication protocols using small, low-power digital radios based on the IEEE 802.15.4 standard. Please refer to Wikipedia for a good fundamental description of the ZigBee networking protocols.

ZigBee is gaining popularity with building automation and HVAC equipment suppliers as a low cost, secure and open standard intercommunication medium among devices used for the building industry. More equipment will soon appear with built-in support for communication using the ZigBee protocol.

19.2 Installation of the XBEE Module

The Fx2424 is designed to easily interface to the XBEE™ family of ZigBee modules produced by Digi International, Inc. There are two rows of 10-pin solder pads on the Fx2424's PCB and to each you can solder a 2mm pitch, single-in-line socket that you can purchase from http://www.digikey.com (Part number : S5751-10-ND), as shown in the following diagrams:

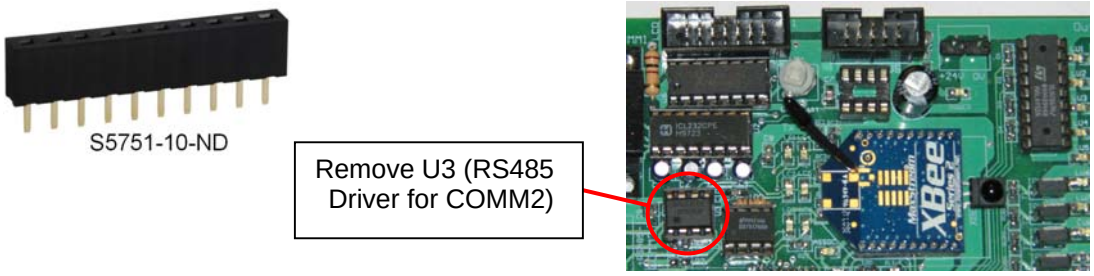


Figure 19.1

Once you have soldered the two rows of sockets, you can purchase the XBEE module directly from Digi International (http://www.digi.com) and plug the XBEE module into these two sockets. Through these connector pins the F2424 PLC provides the 3.3V power to the XBEE module as well as connects its COMM2 Receive and Transmit data lines to the XBEE module. However, the PLC does not support the XBEE serial port hardware handshaking.

The Fx2424 PLC uses only its COMM2 serial port to communicate with the XBEE module. It can use the PRINT #2, OUTCOMM 2, INCOMM(2) and INPUT$(2) commands to configure the XBEE module for Transparent mode or API mode to integrate it with a ZigBee network.

However, since the Fx2424's COMM2 port is already pre-installed with an RS485 driver IC at the U3 socket (IC part number is 75HVD3082 or equivalent), you **MUST REMOVE** the COMM2 RS485 driver IC if you wish to install the XBEE module to prevent signal conflicts. You can easily remove the 8-pin RS485 driver IC from its DIP socket and store it in your parts bin as spare parts. Since the Fx2424's COMM3 port also uses the same RS485 driver chip, the RS485 driver you just removed from COMM2 port would come in handy if the COMM3 driver happen to be damaged due to bad wiring.

Note:

1. **DO NOT** be tempted to solder the XBEE board directly to the F2424's PCB solder pad. You will need to be able to remove the XBEE from F2424 and put into the Digi' interface board whenever

you need to upgrade the firmware or change the role of the XBEE module (e.g. configure it to be a coordinator, a router or an end device, to operate in AT mode or API mode etc).

2. Although the XBEE socket can also accept the XBEE-PRO module, the current requirement for the XBEE-PRO module at maximum transmission power exceeds the peak current capability of the PLC's voltage regulator and therefore we do not recommend installing the XBEE-PRO OEM module into the F2424 PLC's XBEE socket. If you must use an XBEE-PRO, you can install the XBEE-PRO module onto the RS232 carrier board provided by Digi and interface to the PLC via the PLC's RS232 or RS485 port (use Auto485 to convert the signal).

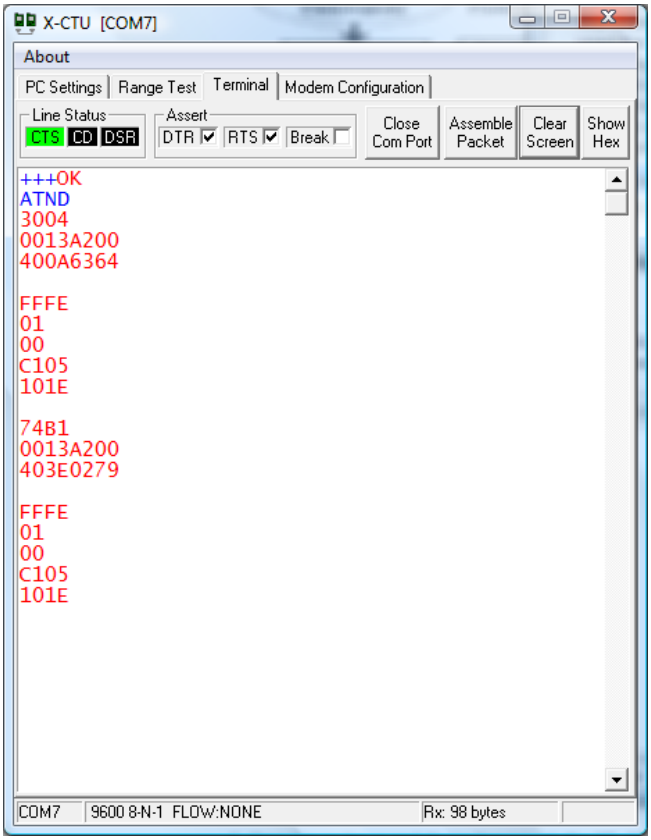
19.3 Using The XBEE's Transparent Mode

The simplest way that the Fx2424 can use the XBEE modules is to put the XBEE into "Transparent Mode". The Fx2424 can configure the XBEE module to make a transparent connection to any other XBEE module on the same ZigBee Personal Area Network (PAN). After a transparent connection has been established, A PC will be able to connect to any Fx PLC via ZigBee, or any two Fx PLCs will be able to communicate with each other **wirelessly** using the same NETCMD$, READMODBUS and WRITEMODBUS commands, as if they have been connected by physical wires.

19.3.1 Connecting to Fx2424 Using XBEE 64-bit Addressing

1. Use the X-CTU software and the USB/RS232 interface board provided in the XBEE Development Kits to configure one XBEE module as ZIGBEE Coordinator AT (e.g. firmware version 2041) and another one or more XBEE module as ZIGBEE Router AT (e.g. firmware version 2241). Follow the procedure outlined in the XBEE user's manual to setup a ZigBee PAN (Personal Area Network). When setup properly, the Coordinator will blink its red LED once every second while the Routers will be blinking their red LED twice a second. The red LED on the interface board is the "Associate" LED that is used to indicate that a router or end device has successfully joined a PAN.

2. With the X-CTU "Terminal" connected to the Coordinator, enter command mode by entering "+++" and the Coordinator should return "OK", then quickly enter "ATND" followed by a carriage return. After a while the XBEE coordinator will return the 64-bit serial numbers of the devices are that joined to it. In our example shown on the right, the two devices are: 0013A200-400A6364 and 0013A200-403E0279. Note down these two numbers, as they are needed for our tests.



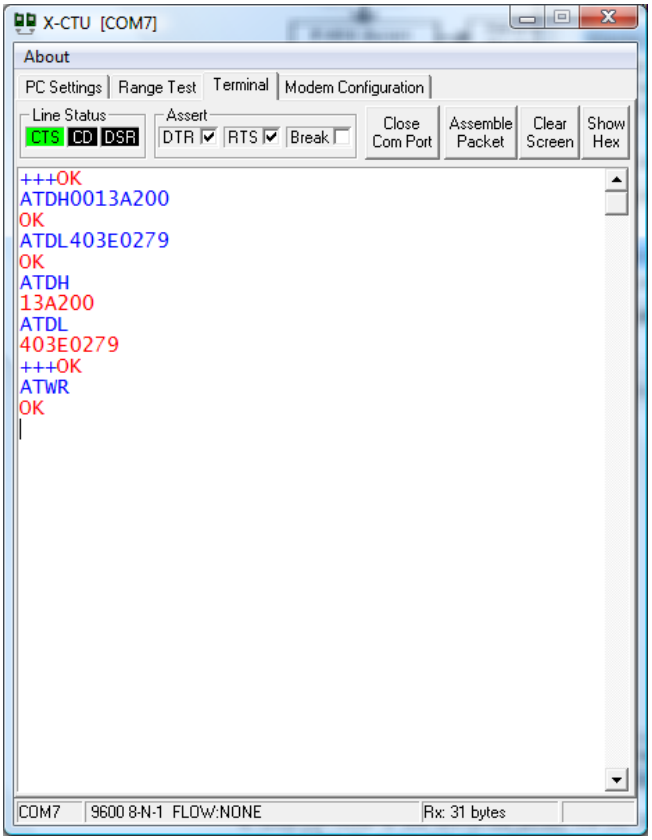
3. The 64-bit serial number is unique for each XBEE device produced by the manufacturer, so your XBEE module will definitely have a different serial number from ours (the upper 32-bit typically would be the same for devices from the same manufacturer). For Digi ZigBee products it is "0013A200" but the lower 32-bit will be different for each device.

   In transparent mode we can force the coordinator to make a transparent connection to a specific router or end device by specifying the serial number in two commands: ATDH and ATDL (destination address High and destination address Low). Each command will specify the 32 bit half of the 64-bit serial number.

4. Next, use the X-CTU terminal to enter the following commands:

   ATDH0013A200
   ATDLXXXXXXXX

   where XXXXXXXX is the lower 32-bit address of the router device that you want to connect to. In our example we entered ATDL403E0279, which is the serial number of the router that we want the coordinator to connect to. You can confirm that the destination address has been correctly setup by entering "ATDH" and "ATDL" without any parameters and the XBEE module would return the previously defined destination address. You can enter "ATWR" if you wish to store the destination address into non-volatile memory.

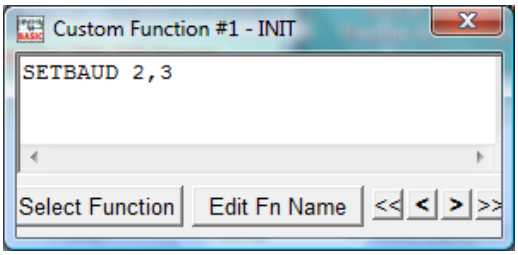


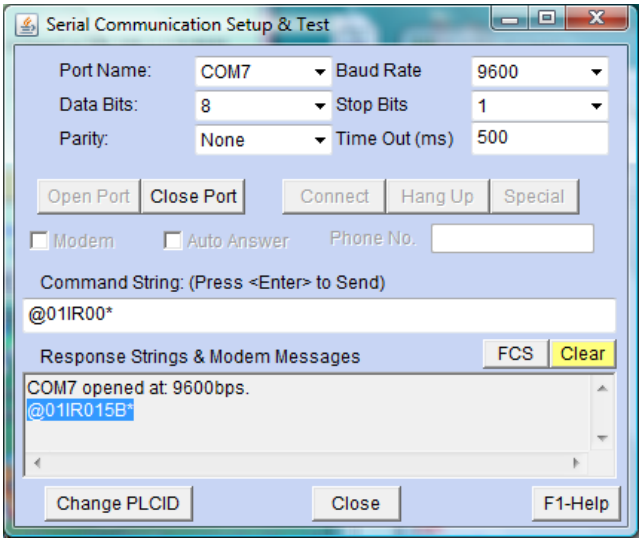
Note: XBEE automatically exits command mode within 10 seconds of no activity so if you enter a command and did not see any response, then you need to re-enter command mode by typing "+++" again and then re-enter the command.

5. If you have another PC connected to the Router's interface board via RS232, you can run the HyperTerminal program (default to 9600 bps, 8 data bit, 1 stop bit and no parity) to test the connection. Once the coordinator is connected to the router designated by the ATDH & ATDL command, anything you type on the terminal at the coordinator terminal side will appear at the router terminal screen and vice versa. This means that you now have a proper transparent connection between the coordinator and the router.

6. Next you can turn off power to the Router interface board, remove the XBEE module and plug it into the Fx2424's XBEE socket (make sure power to PLC is turned OFF before doing this). Turn on power to the Fx2424 and you should see the green "Assoc." LED located next to the lower left corner of the XBEE socket starting to blink twice a second. This means that the XBEE module on the Fx2424 is connected to the PAN setup by the Coordinator.

7. We will next attempt to use the TRiLOGI software to connect to the PLC via the XBEE connections. Remember that the XBEE module is physically connected to COMM2 of the Fx2424 PLC with the default serial baud rate of 38,400, 8, 1, n. But the XBEE module's serial port is default to 9600, 8,1,n so either one of them must change so that the XBEE module can talk to the PLC's COMM2 using the same serial settings. You can change the PLC's baud rate by adding the statement "SETBAUD 2,3" to initialize the PLC's COMM2 port to 9600,8,1. You need to transfer this program to the PLC using the regular program transfer method (e.g. via Ethernet port or comm. Port) before our test.





8. You should now click the "Close Com Port" button to close the COM Port at the X-CTU and start up TLServer. Click the "Serial Port Setup" button on TLServer and select the same COM port that X-CTU uses to connect to the coordinator. Set the serial port to 9600,8,n,1. Then enter the command "@01IR00*" and press the Enter key (assuming the PLC's ID is 01) and the PLC should send a response string "@01IR015B*". This shows that TLServer is now connected to the PLC via Zigbee radio.

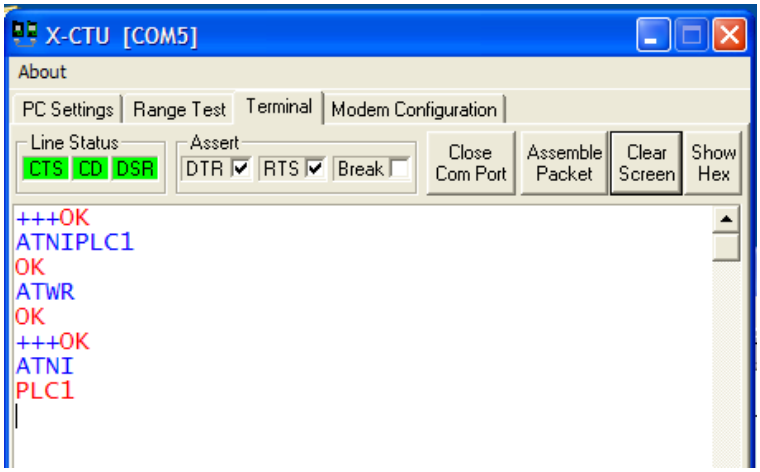
9. You can now use TRiLOGI to perform online monitoring or programming of the PLC via TLServer in exactly the same way as if the PLC is connected to the PC serial port directly. However, in this case the connection is done wireless via the two XBEE modules. The XBEE modules thus perform the same task as a pair of radio modems, except that the actual connection is via a ZigBee mesh network instead of a point-to-point radio link.

10. If you have more than one PLCs installed with the XBEE module, you can change the destination address by repeating the ATDH and ATDL command as mentioned in step #4 and you would then be able to connect to a different PLC wirelessly. While this is certainly not a practical way of using the XBEE module, it is a good way to gain some understanding of the XBEE addressing method. In the next example we are going to simplify the connection process using "Node Identifier".

19.3.2 Defining Node Identifier for XBEE Module

In the last example we showed you how to use the ATDH and ATDL commands to set the 64-bit destination address of a slave XBEE module. It is certainly not convenient having to enter so many digits just to change a destination node to connect to. To simplify the connection procedure, XBEE offers a very useful "Node Identifier" command that allows you to assign alphanumeric names (up to 20 characters with no spacing) to each XBEE module using the ATNI command.

1. First, turn off power to the PLC and then move the XBEE module to its RS232 interface board. Next, use the X-CTU terminal screen to enter the command as shown below:



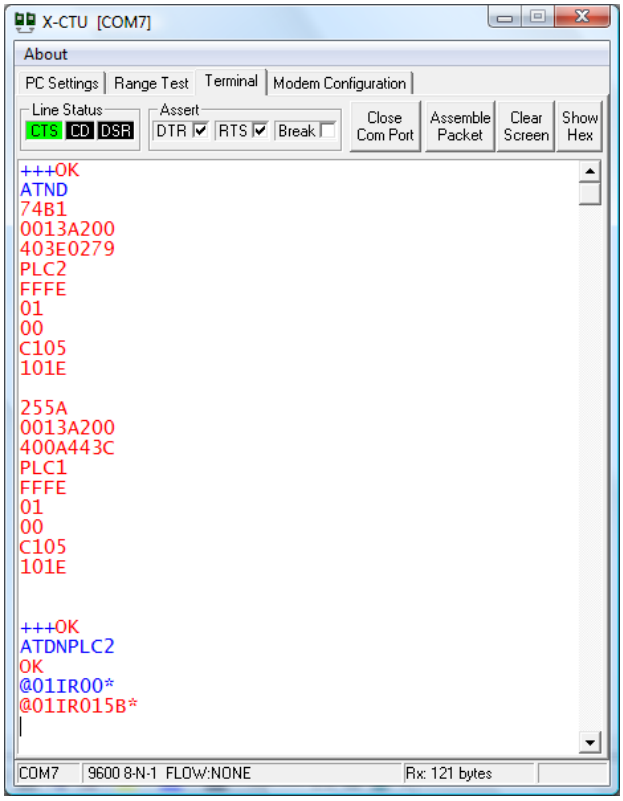
2. By using the command “ATNIPLC1”, we have defined this XBEE module with the identifier “PLC1”. We next issued the ATWR command to save the identifier to the non-volatile memory. To confirm that the identifier has been saved, you can power off the XBEE module and then on again and enter the command “ATNI” and the XBEE module will report the pre-defined identifier, as shown in our example.

3. If you have more than one Fx2424 PLC with an XBEE module, you can define them similarly with a different name suitable for your application. In our case we have simply defined another XBEE module as “PLC2”.

4. Next, turn off power to the interface board and move the XBEE modules to the Fx2424 PLCs and then turn on power to the PLCs. The PLCs’ XBEE modules should still be joined to the coordinator, which is indicated by the blinking “Assoc.” LED on the PLC.

5. Now run the X-CTU terminal program and connect to the Coordinator, enter the “node discovery” command “ATND” and you will see the details of the XBEE nodes that are joined to this PAN. This time you can see that the PLC’s node identifiers, “PLC1” and “PLC2”, have been listed together with their 64-bit and 16-bit addresses.

6. Now, instead of using ATDH & ATDL to enter the 64-bit node address, we can simply use the ATDN command followed by the node identifier to set the destination address. In our example we show a host link command “@01IR00*” (blue) being sent from the X-CTU terminal to PLC2 and the response string received from PLC2 (red).



7. You can see that using the Node Identifiers greatly simplify the connection procedure. Note that for nodes PLC1 and PLC2, their destination address by default all points to address 0, which is

the default network address of the coordinator. Therefore, although we didn't specify the destination address for the Router nodes PLC1 and PLC2, they would automatically send all their responses back to the coordinator.

8. If two Routers XBEE modules need to communicate with each other instead of with the Coordinator, then it is necessary to set the destination node ID of **each** XBEE module to that of the **other** module so that the two nodes can talk to each other directly to exchange data. However, two nodes connected this way still would not exclude other nodes from sending data to either of them. I.e. Any node, including the coordinator, can still send data to either PLC1 or PLC2 as long as it sets its destination address to either node. But any response string returned from these two PLCs will not go back to the third sender since their destination address is not pointing to the sender. You should remember this in order not to be surprised if the slave AT appears not to respond to a host link command that it received via the XBEE channel.

19.3.3 Wring TBASIC Program To Configure XBEE

So far we have been using the X-CTU Terminal software provided by Digi to manually enter command mode to configure the XBEE module, such as to set the destination address. This is useful as it gives us the visibility of what's going on when you send commands to the XBEE module.

For real world applications you may want to program the Fx2424 PLC to configure the XBEE module instead of doing it manually by X-CTU software. To put the XBEE module into transparent mode, the PLC needs to send three "+" character to COMM2, while observing the guard time before and after the "+++" characters. (We also find that adding a small delay between each "+" character works better). If successful, the XBEE module would return an "OK" string that is terminated with a carriage return. You can use the INPUT$(2) command to read the "OK" string as a confirmation that the command mode has been entered. Next, you can use the PRINT #2 statement to send the commands to the XBEE module and the INPUT$(2) statement to read the response.

We have created a sample program "XBEE-COMM2.PC6" that will put the XBEE module into command mode and send a command string stored in A$ to the XBEE module and return the response string from XBEE in A$ back to the calling program. You can download the sample program from http://www.tri-plc.com/trilogi/XBEE-demo.zip - the program has been tested and fine-tuned to match the XBEE behavior and is easy for you to incorporate into your own software.

19.3.4 Practical Applications of XBEE Modules In Transparent Mode

1) One possible application is to setup a Fx2424 PLC installed with an XBEE Coordinator as a master PLC. A number of F-series PLCs installed with XBEE router or end device modules can be spread across the control environment to perform local control independently, but at the same time they also act as ZigBee slave controllers. All ZigBee slave controllers would set their destination node address to the Coordinator attached to the master PLC. If a ZigBee PAN is properly setup and operational, then the master PLC can access any slave controller to read/write data simply by setting the Destination node to the slave controller's node identifier and then send host link commands to the slave controller. However, due to the time taken to get in and out of the command mode to set the destination node address, such a system is only suitable if the master PLC does not need to access the slave PLC frequently, or if there are only a small number of slave controllers it needs to access. It would not be suitable for implementing remote I/Os where the master controller needs constant status updates of remote I/Os.

2) A modification of a system described above is to program each of the slave PLCs to monitor its local I/O. Then it can then send an ASCII string to the master PLC if there are changes that require it to inform the master PLC to take action. You can design the ASCII string format in whatever way

suitable to your application as long as it contains enough information to inform the master who is calling for attention and what the master needs to do.

In such a system, the master PLC must regularly read incoming ASCII strings using the INPUT$(2) function and check to see if it is of the correct user-designed format (you can use a certainly starting characters e.g. “AT#xxxx” where “AT#” means “Attention to node #” and “xxxx” is the node identifier of the slave that requires attention) and if so, interprets the content of the string to decide what it needs to do with the slave controller. Based on the information in the received string, the master can selectively set its Node Destination to the slave node identifier so that it can issue host link commands to access the designated slave PLC’s data.

Such a system would be quite efficient for collecting data from slave I/O since the slave I/O can simply embed the information inside the ASCII string and the master PLC will receive the data almost immediately. For example, a slave controller could be monitoring outdoor temperature and it will only send the new temperature information to the master PLC if the temperature varies more than +/- one degree C. If the master PLC does not receive updated information, it can then assume that the temperature did not change.

Of course, in order to ensure that all the slave PLCs are still operational and are connected in the network, you can also design the system in such a way that the master PLC expects the slave PLC to always send a message within a certain time (say every 5 minute). The master will query the slave if it did not see any update after a certain amount of time and it could sound an alarm if it discovers that the slave is no longer accessible on the ZigBee network.

We have created a simple demonstration project based on the abovementioned arrangement. You can download the demo programs from the following URL.

http://www.tri-plc.com/trilogi/XBEE-TransparentModeMasterSlaveDemo.zip

In this project two F-series PLCs with XBEE Router modules are used as slave controllers and they will only send ASCII strings to the master PLC when its digital input status has changed. The master PLC will interpret the ASCII string and extract the input status of the two PLCs and then echo them on the master’s PLC digital output 1-16 for the first PLC and 17-32 for the second PLC. It is a simple project but does illustrate the concept well and you should be able to easily modify the programs to suit your application needs.

19.4 Using The XBEE’s API Mode

(To be Appended)

Chapter 20 Extended File System

20 EXTENDED FILE SYSTEM

Before you begin, please download a sample I-TRiLOGI program that will be referred to throughout this manual from our website:

http://www.tri-plc.com/trilogi/ExtendedFileSystem.zip

20.1 Introduction

A F-series PLC with r77 or later firmware now supports up to 1,740K bytes of file space that can be used for storing up to 435 web page control files!

More importantly, a F-series PLC with a new r77 firmware can open local data files in this file space and write/append data to it. The PLC can therefore log a large amount of data into one or more data files, which can be retrieved for analysis anytime.

There are two ways to retrieve the stored data files from the PLC:

1. Download the file from the PLC's built-in web server: The file created by the PLC can be downloaded from the PLC's built-in web server using any web browser. This allows the user to access the data file at any time of the day.

2. Automatic FTP upload from the PLC to an external web-server: You can program the F-series PLC to make a FTP client connection to any web server on the local network or on the Internet/Cloud and upload the data file it has created to the web-server using any filename.

Imagine what the Fx2424 can do with this new uploading capability! The ability to log data locally and automatically upload the data to a web server transforms the PLC into a potent data-logger! The PLC can be programmed to capture daily, weekly or monthly data and then periodically upload the data file to an Internet web server with a unique, time-stamped filename (E.g. "temperaturelog2012-01-01.xls"). This allows the PLC to log data completely unattended.

The data uploaded by the PLC to the external web server can therefore be viewed or downloaded into a PC using any web browser, anywhere in the world. This allows you to carry out analysis of past logged data file for performance or diagnostic analysis at any time without having to physically access the PLC to retrieve the logged data.

20.1.1 Advantage of Using Data Uploading

1) Although it is possible to directly access the PLC's internal web server to download the data file it has created, this does require active action by the user and to ensure that the data are retrieved before the file is full and deleted by the PLC to create space to log new data.

2) By programming the PLC to upload the data periodically the PLC can delete the file after it has successfully uploaded the data file to free up space to accept new data. In other words the PLC will never run out of data space to log data since it can store the logged files on any server including the Cloud!

3) To directly access the file stored on the PLC from outside of the LAN, you will need to setup the router or firewall to "forward" the PLC's server port (e.g. 9080) to the PLC. If you have multiple PLCs logging data, then each PLC will need to have a different port number in order to properly forward the data. This not only complicates the setup, but also is often frown upon by System Administrator and may not even be permitted by the corporate network security policy.

4) The PLC is designed to upload data to any web server via FTP passive mode by providing the login username and password. Using FTP passive mode allows the PLC to open a network connection to an external web-server to upload a file and then close the connection immediately. It does not require opening a port on your router to permit external access to the PLC from the Internet. Hence there is no complicated router setup involved as there is no port forwarding required. It also eliminates the security risk from someone trying to take control of the PLC from outside of your LAN and is generally much more acceptable to the System Administrator.

If you have multiple PLCs in use, you can program each PLC to upload data to a different directory or append a different file name prefix, or to a different server, and once programmed all PLCs will happily log data unattended indefinitely!

20.2 File Structure and File Naming of The Extended File System

Fx2424 uses a very simple file system, where the 1740K bytes of file space are divided into 435 slots of **4K** bytes each. The file system does not use any kind of file allocation table; hence it must follow a strict naming scheme because the file name directly relates to where it is stored in the file space. The followings describe how the files are named and where they are stored.

1. The first 35 files are named "0.yyy", "1.yyy"…."X.yyy", "Y.yyy".

2. The 'yyy' part of the file name is the "extension", and only the following MIME extension are accepted by the PLC:

   `HTM, JPG, GIF, CSS, JS, BIN, TXT, JAR, ZIP, XLS`

   Any other extension names will be replaced by "???". The MIME extension are respected by the web browser when you download a file from the built-in web server so you should always only store files with one of the above extension.

3. The file "0.yyy" occupies the first 4K byte slot, "1.yyy" occupies the second slot… the "Y.yyy" occupies the 35th slot.

4. The next **400 files** can only use the file name "Zxxx.yyy". where the xxx part of the filename is a 3 digit decimal number which can be any number from 000 to 399. The yyy part is an extension and the same as what's described in (2) above.

5. "Z000.yyy" occupies the 36th slot, "Z001.yyy" occupies the 37th slot .., and the file "Z399.yyy" occupies the last 4K slot.

6. Any file may occupy more than 1 slot so you can specify "0.yyy" to occupy the entire 1740K bytes of the file space. Or you can configure "Z000.yyy" to occupy 20K (=5 slots) from the 36th slot, and the next valid file should only start from Z005.yyy. i.e. If you want to allocate 20K bytes of file space to file Z000.yyy you cannot use any file name Z001, Z002, Z003 and Z004, otherwise these file will corrupt the file space of Z000.yyy.

7. This means that the programmer will have to design the file space carefully and determine how to best use the file structure to provide the right balance between the number of data files and the amount of data space allocated to each data file.

Note: The file name restriction only applies to files stored on the PLC's internal file space. When you use the FTP upload function described later, you can specify **any destination filename** as long as they are acceptable to the external FTP server.

20.3 Transferring Files To The PLC's Web Server

If you are only using the extended file space for the purpose of storing additional web pages on the built-in web server, then you can use the FTP client software such as FileZilla client to transfer the program to the PLC as described in Section 2.10.

20.4 Accessing The Extended Data Files Using TBASIC

TBASIC program on a F-series PLC with r77 firmware can directly access any data file with names from **"Z000.yyy to Z399.yyy"**. **File with names from "0.yyy" to "Y.yyy" are reserved for storing web pages only** and hence are not accessible by TBASIC.

The PLC can open a new file for writing new data (essentially deleting the old file content), or open an existing file and append data to the end of the file. It can also open a file and read data from the file as ASCII strings. It can achieve this by using the **`PRINT #8`** and **`INPUT$(8)`** functions, which will be described in details in the following sections. We have created a sample program: "ExtendedFileSystem.PC6" (click here: http://www.tri-plc.com/trilogi/ExtendedFileSystem.zip to download) that demonstrates all these new capabilities.

20.4.1 Open A File For Writing New Data

Syntax: **`PRINT #8 "<WRITE Zxxx.yyy>"`**

where "Zxxx.yyy" is the file name. If successfully executed, the "<WRITE>" command will open the file and set the file pointer to the beginning of the file. Thereafter the PLC can start writing ASCII data to the file using the PRINT #8 <string data> command. [Note: the PRINT #8 command automatically appends a carriage return to the end of the string data unless the string data is terminated with a semi-colon (';')].

When the PLC has completed writing data, it must close the file by executing the command: PRINT #8 “</>”.

E.g.
```
PRINT #8  “<WRITE Z005.TXT>”
PRINT #8  “The current Greenwich Mean Time is”
PRINT #8  STR$(TIME[1]);”:”;STR$(TIME[2]);”:”;”00”
PRINT #8  “</>”
```

The CPU should use the STATUS(2) command to check whether the <WRITE> has been successfully executed before begin writing data to it. STATUS(2) command returns a 1 if “<WRITE>” operation is successful and returns a ‘0’ if the operation failed. The CPU can only write or access to a single file at a time so any opened file must be closed by the PRINT #8 “</>” command before another file can be opened for writing.

20.4.2 Open A File For Appending Data To The End Of The File

Syntax: **PRINT #8 “<APPEND Zxxx.yyy>”**

where “Zxxx.yyy” is the file name. If successfully executed, the “<APPEND>” command will open the file and set the file pointer to the end of the file. Thereafter any string data following a PRINT #8 command will be appended to the end of the file. When the PLC has completed appending data, it must close the file by executing the command: **PRINT #8 "</>"**.

As per the “<WRITE>” command, the CPU should also use the STATUS(2) command to check whether the <APPEND> command has been successfully executed before begin writing data to it.

Example

```
PRINT #8  "<APPEND Z"+STR$(F,3)+".txt>"
S = STATUS(2)      ‘ Status(2) returns 1 if successful.
IF S <> 1 RETURN: ENDIF

FOR I = 1 to 100
  PRINT #8  STR$(I,4)+":This is the Appended first line"
  PRINT #8  STR$(I,4)+":This is the Appended second line"
  SETLCD 1,1, "Append  #"+STR$(I,4)
NEXT
PRINT #8  "</>"     ‘ close the file
```

20.4.3 Delete A File

Syntax: **PRINT #8 “<DELETE Zxxx.yyy>”**

where “Zxxx.yyy” is the file name of the file to be deleted. There is no need to close a deleted file.

20.4.4 Open A File For Reading

Syntax: **PRINT #8 “<READ Zxxx.yyy>”**

where “Zxxx.yyy” is the file name of the file to be opened for reading. If the file has been successfully opened for reading after execution of the PRINT #8 “<READ>” command, the PLC

can start to retrieve ASCII data from the file line-by-line using the INPUT$(8) command. A line is either a string that is terminated with a Carriage Return character (ASCII 13), or is a 70-character long string (which is the maximum length of any string variables A$ to Z$) without carriage return. In either case the return string does not contain the CR character itself.
The PLC can check if a file has been successfully opened for reading using the STATUS(2) function **AFTER** executing the PRINT #8 “<READ>” command. STATUS(2) will only return a 1 if a file has been successfully opened.

The PLC can determine if the End-of-File (EOF) has been reached using the STATUS(2) function after every INPUT$(8) command has been executed. STATUS(2) returns a 255 if the EOF has been reached. The PLC should then close the file by executing the “PRINT #8 “</>” command.

```
A$ = "<READ Z"+STR$(F,3)+".txt>"
S = STATUS(2)
IF S <> 0
   SETLCD 1,1, "Failed to Open File"
   GOTO @100
ENDIF
C = 0
PRINT #8 A$
SETLCD 1,1, A$
WHILE 1
  A$ = INPUT$(8)
  S = STATUS(2)
  IF S = 255 EXIT : ENDIF   ' S = 255 means EOF
  SETLCD 2,1,A$
  DELAY 50      ' So that reader can read from the screen.
  C = C+1
ENDWHILE

SETLCD 1,1, "Read ” +STR$(C) + " lines "
@100
PRINT #8 "</>"   ' close the opened file
```

20.5 Setting Up The FileZilla FTP Server

One important capability of the new PLC with r77 firmware is the ability to upload file created by the PLC to an external server on a local area network or on the Internet via the FTP protocol. If you have access to a FTP username and password on your company's server (or if the SysAdmin is authorized to set up an account for you) you can certainly use your own account for testing. If not, you can download the free Filezilla FTP Server and set it up for testing. The "ExtendedFileSystem.PC6" has a FTP upload demo and it was configured to work with a FileZilla FTP server.

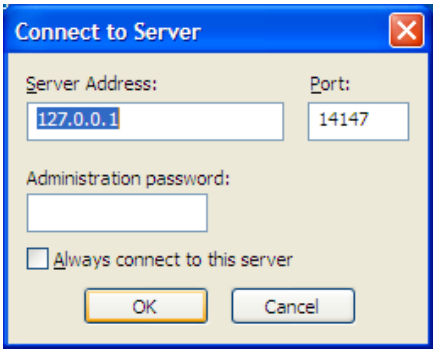
Using Filezilla has the advantage that you can see the login sequence performed by the PLC when it attempts to connect to the FTP Server so that it is easier to troubleshoot connection problem. (For professional grade troubleshooting, one handy program to have is the "Wireshark" program which is a TCP/IP packet sniffer that allows you to look at the actual TCP/IP packets sent between your PC and the PLC). However, it is important to setup the Filezilla server program properly to minimize connection trouble.

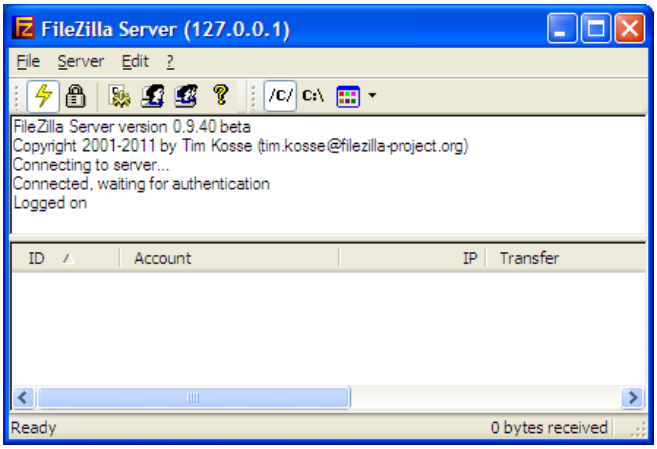
20.5.1 Download and Setup FTP Server

1. First download the FileZilla server installer from the following website:

   http://filezilla-project.org/download.php?type=server

2. Run the "FileZilla Server Interface" program which is meant for managing the FTP Server settings.

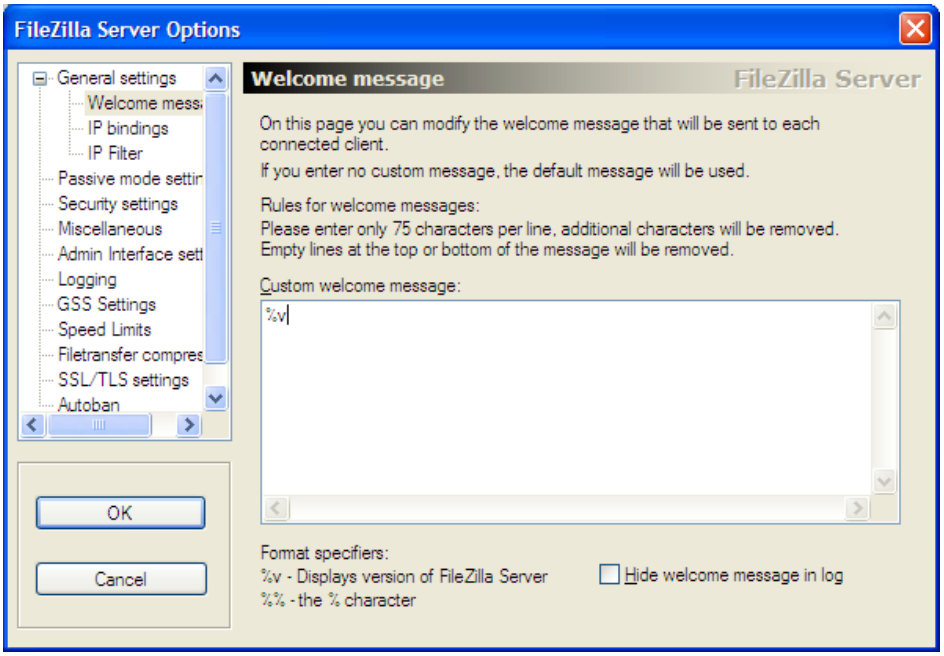




3. If the FileZilla Server is running on the same PC that you are running the FileZilla Server Interface program then you can use the localhost IP address which is 127.0.0.1 – the Port can be anything since this is a client port that the Interface program is using to interact with the FTP Server (don't be confused with the FTP **Server** listening port which is by default = 21).

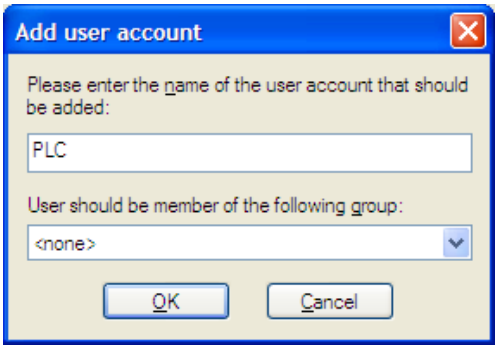
4. If this is the first time you run the program after setting up Filezilla FTP Server there will be no Administration password so you can leave it blank. Click OK to connect.

5. Click "Edit->Settings" and then "General Settings -> Welcome message" – leave only one line of welcome message so that the PLC has less work to do. Then click OK to accept the change.

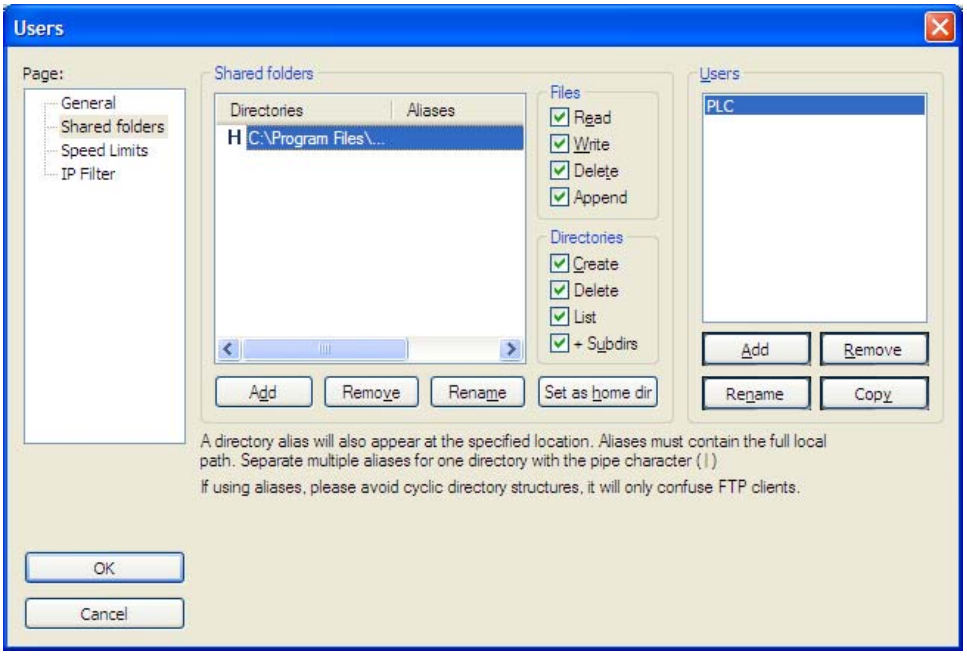


6. Click "Edit –> Users" in order to setup a user name and password for your test. The "ExtendedFileSystem.PC6" program uses a username = "PLC" and password = "1234" so for a quick test you may like to setup the same username and password:

7. At the "General" page of the setup screen, click "Add" button at the "Users" pane to add a username "PLC". Since no group has been defined simply leave the default as "<none>" in the second text box as shown in the following diagram.



8. Next click "Shared folders" page and you must setup a folder that is to be used to receive uploaded file. Click "Add" at the "Directories" pane to add the folder. You can choose any folder on your PC to be used for the FTP upload and you just have to remember the location so that you can look for the uploaded file later in the test. However, make sure that you check all the check boxes for "Read", "Write", "Append" etc as shown in the diagram.



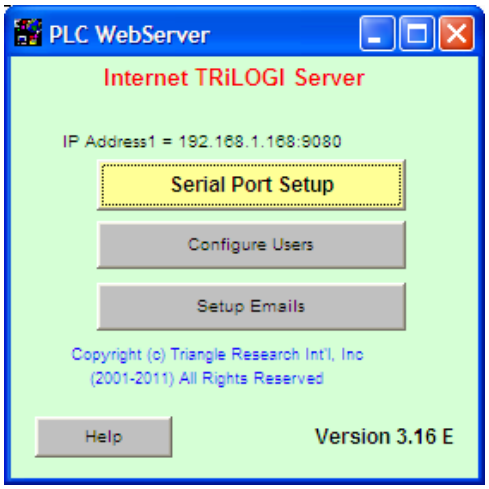
9. Click “OK” to complete the setup. The FTP server should now be waiting for connection.

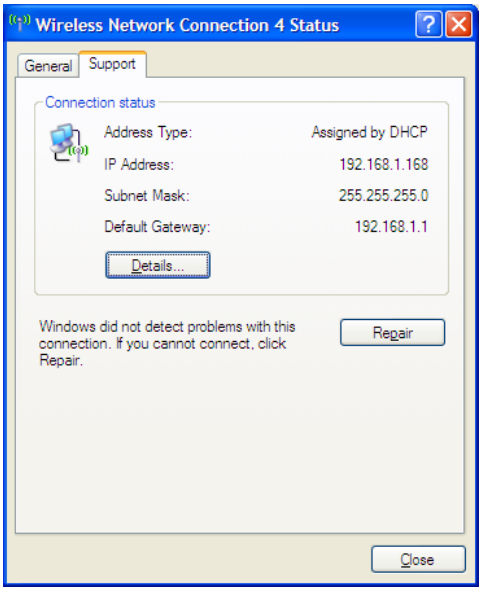
20.5.2 Testing Connection To The FTP Server Using Telnet

You can now test the FTP Server using the FileZilla client as mentioned in Chapter 2.9 of the PLC User’s Manual.

But a better way to test is to use the “Telnet” program on your PC (if you are running Windows Vista or Windows 7 you may need to enable the Telnet program since it is disabled by default – do a quick Google search on how to enable the Telnet client software on your PC).

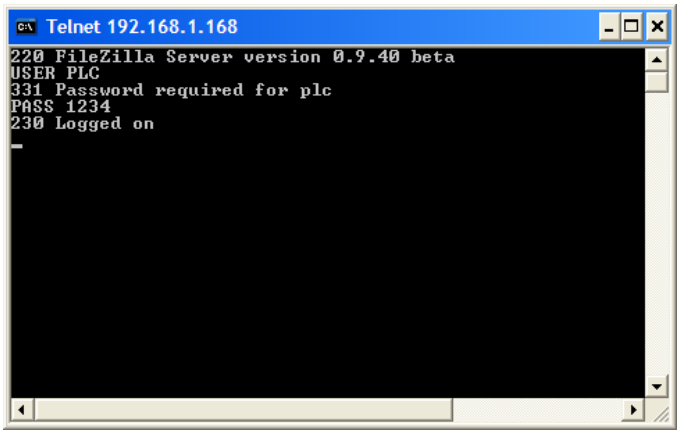
Also you may want to find out the IP address of your PC that is running the FileZilla Server. If you have TLServer running on your PC your IP address is reported on the TLServer’s front panel. You can also get the IP address from the Windows “Network Connection Status” as shown below. Our test PC has an IP address = 192.168.1.168 which will be used in the following tests as well as used in the PLC program to connect to the FTP Server.

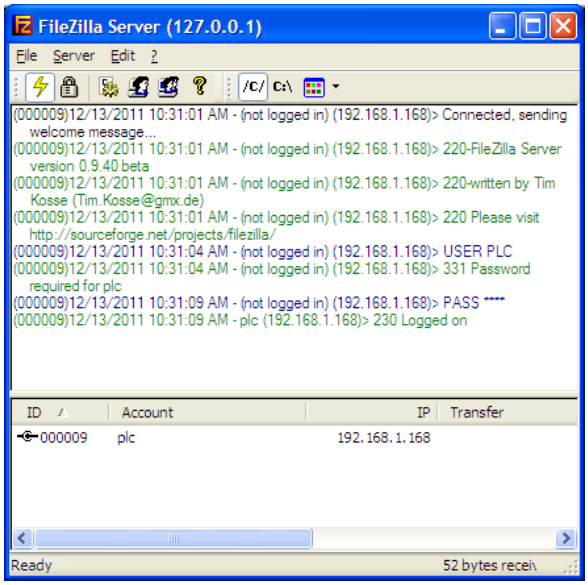




1. First open a command prompt window and then type "**`telnet 192.168.1.168 21`**" - this will open a telnet connection to the FTP server on our test PC with IP address 192.168.1.168 and listening at port 21. Please replace the IP address with the actual IP address of **your** PC. The following screen shot captures the test sequence. Note that the same command/response sequence with the server is also shown on the FileZilla Server Interface program front panel:

USER PLC
PASS 1234





2. Once you get the "230 Logged On" message you know that the FTP setup is done correctly. Note that the welcome message from the FTP Server shows only one line "**`220 FileZilla Server Version x.xx`**" which is what we have set it up to be. You can now disconnect from the FTP server by typing "Quit" at the command prompt.

3. There is one more things you need to do before you proceed to test the FTP upload features of the PLC to avoid connection problem – that is to temporarily **TURN OFF the Windows Firewall and any**

software firewall setup by anti-virus software during your test. You can always re-enable your software firewall(s) after the test if you wish. PC operating system are designed to run client program normally instead of acting as a server so Windows Firewall by default is to block all incoming connections to the FTP Server. Thus it can give you a lot of headache when you are trying to connect to the FTP server operating behind the Windows Firewall.

Notes:

a) The main purpose of Windows Firewall is to protect your PC when you are connected to say a public wi-fi network. But if your PC is connected to the Internet via a router at work or at home, the router hardware itself would act as a firewall to isolate your PCs and a software Firewall is actually redundant. (If a hacker tried to connect to a port to your public Internet IP address what he reached would be the port on the router and he would not be able to reach your PC, unless you have specifically set up to forward all TCP/IP messages sent to that port to a specific PC).

b) If you really want to use the Filezilla FTP Server as a permanent server on a PC to receive files, and still want to have the Windows Firewall enabled, you can refer to the last section in this chapter which describes how to do it.

20.6 Uploading File From PLC to FileZilla FTP Server Directory

20.6.1 Overview of The FTP Protocol

The FTP protocol requires **two** socket-connections between the devices performing the file transfer. One connection is the "command" channel where FTP commands such as STOR or DELE and the responses are sent as plain ASCII text strings between the FTP client and the FTP server. The second connection is the "data" channel where only the file content or the file directory data are being transferred.

There are two transfer modes: "Active" mode and "Passive" mode. Active mode requires that the **server** establish a data connection back to the **client.** Passive mode on the other hand, requires that the client also be the one to establish the data connection. i.e. For passive mode, both the command and the data connections are performed by the client (the PLC in this case).

The PLC has been designed to use only **passive** mode to transfer file to the FTP server. Passive mode is preferred because that is the only way to transfer file if the FTP Server is located on the Internet. The alternative active mode transfer requires the server to make a data connection back to the PLC that is sitting behind a router firewall, and that can be problematic unless the router is specifically configured to forward the data port to the PLC.

20.6.2 PLC FTP Upload Procedure

In order to upload file to the FTP Server, the PLC would use the PRINT #4 "<TCPConnect xxx.xxx.xxx.xxx:21>" command tag to connect to the FTP server port 21 to establish the "command" connection to the FTP Server. The PLC uses its PRINT #4 to send and INPUT$(4) to receive ASCII text strings from the FTP server via the command channel. The PLC would then send the command "PASV" to inform the FTP server that it wants to transfer a file in passive mode.

At this point you can use the PRINT #4 command to send any valid FTP command to the server, including changing directory (CWD command – make sure the directory exist), deleting a file (DELE command – beware of what you are deleting!) etc.

When the PLC is ready to start a file transfer to the FTP server, the server will in turn provide the PLC with the port number that it has opened for the PLC to connect to establish a data connection. Upon receiving this port number the PLC will make a second TCP connection to the given port and then the actual file transfer will begin.

A new, network service command tag named "FTPUPLD" handles the negotiation between the FTP Server and the PLC as well as handling of the data transfer from the PLC to the FTP server. The following is the syntax:

```
PRINT #4 "<FTPUPLD Zxxx.yyy  [destination file name]>"
```

Where: Zxxx.yyy is the file name of the extended file that the PLC has access to. The [destination file name] can be any legal name acceptable to the server so you can attach a date or time stamp to the file name for easy identifications.

When the above **<FTPUPLD>** command is run, the PLC will send the actual "STOR" command in the background to the FTP server and then obtain the port number from the server and it will then make a data connection to it, and file transfer can then begin.

20.6.3 Monitoring The FTP Upload Progress

Once the file transfer begins the PLC firmware will handle the rest of the file transfer until either the file has been completely transferred or the transfer is aborted due to a network or server trouble. You can monitor the progress of the file transfer using either the **STATUS(4)** or **STATUS(20)** functions.

STATUS(4)= 0 : FTP client was idle or last FTP failed
1 : FTP data transfer just started
2 : 1st FTP segment transferred, now transferring the rest
3 : FTP data transfer completed.

STATUS(20) > 0**:** Number of bytes uploaded to FTP Server. Transfer is in progress.
< 0**:** Total number of bytes uploaded. Transfer completed.

For example, If 2,345 bytes has been uploaded to the server and the transfer has ended, **STATUS(20)** will return the number = –2345.

Since file transfer can take substantial amount of time to complete, it is not wise to run a loop to wait for the file transfer to complete since this will block the PLC from processing any other part of the program. The demo "ExtendedFileSystem.PC6" shows you how to setup a monitoring function to periodically monitor the progress of the file transfer and report the transfer status on the LCD display.

Please refer to the comments in the custom function "fnConnFTP" and "fnMonFTP" of the "ExtendedFileSystem.PC6" program for more detailed descriptions of each command involved in the FTP file transfer.

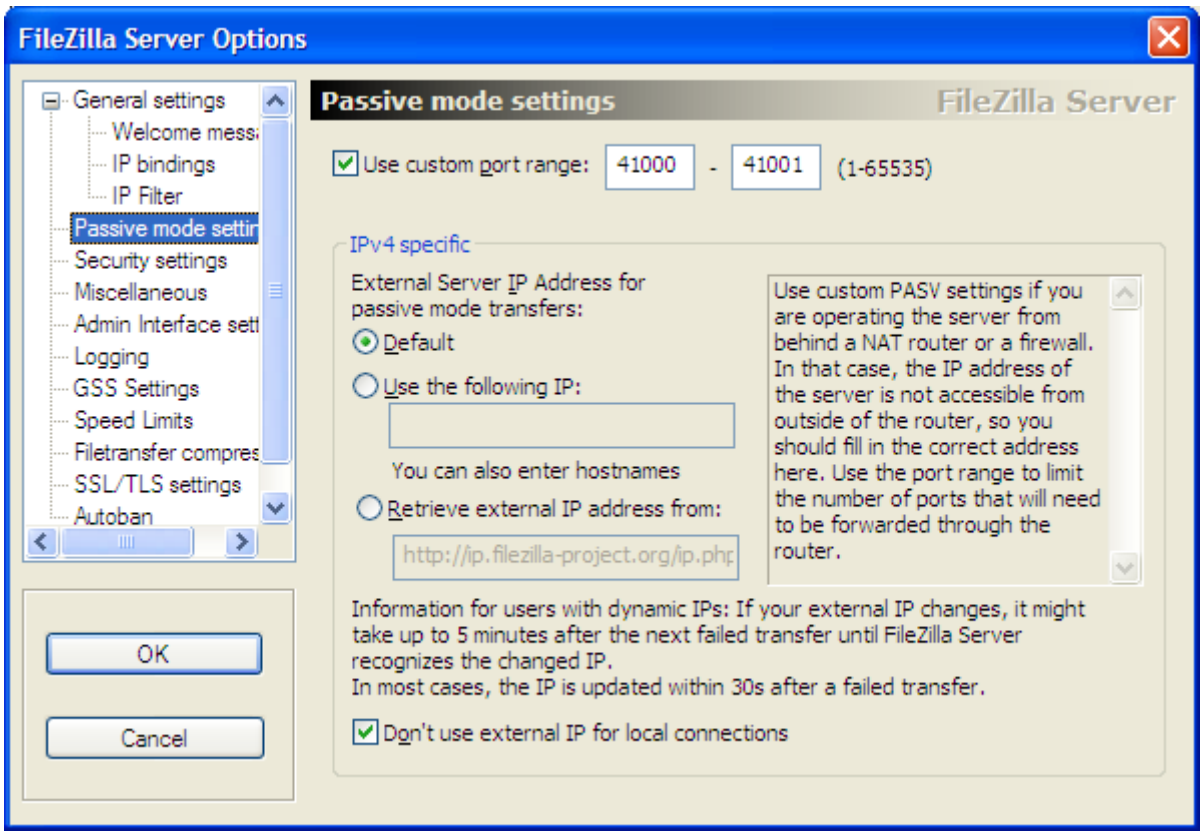
20.7 Setting Up A FTP Server Behind a Windows Firewall.

Please refer to the following Microsoft document describing issues and solutions related to FTP server behind the Windows Firewall.

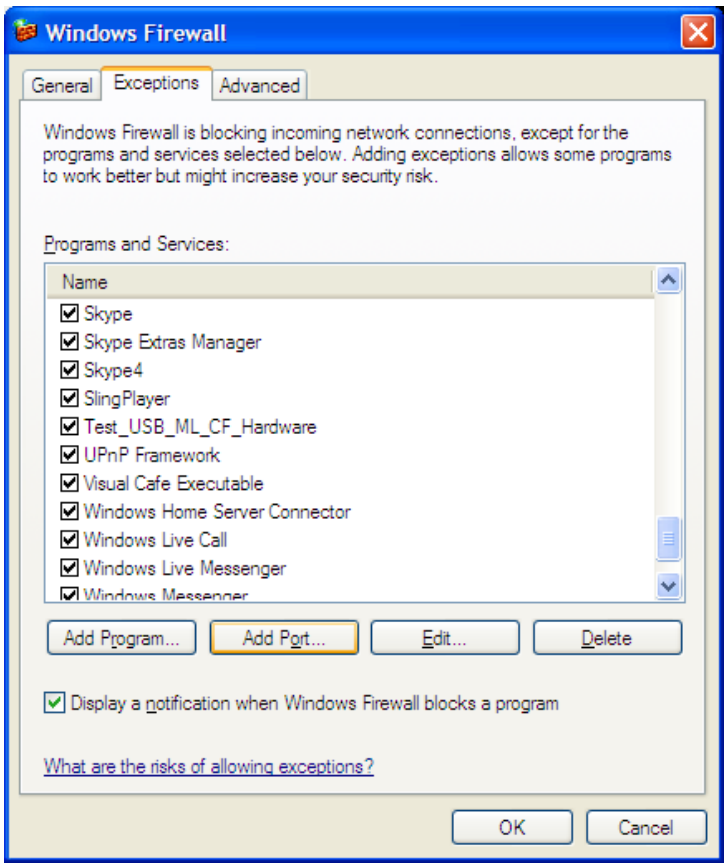
http://technet.microsoft.com/en-us/library/dd421710(WS.10).aspx

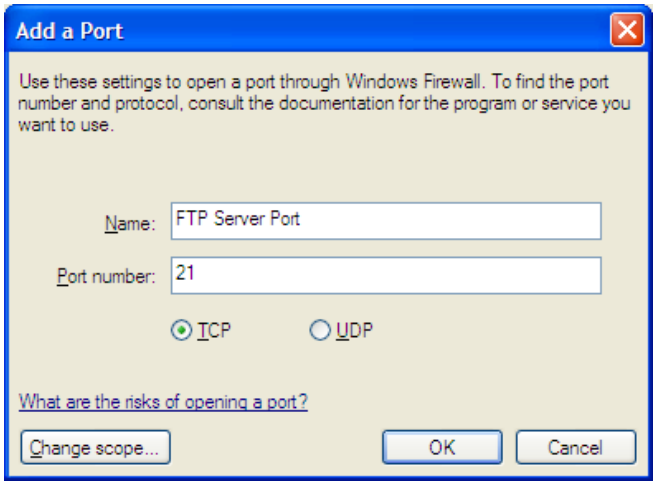
Microsoft focuses mainly on the FTP server in their IIS server (for obvious reasons) instead of Filezilla. If you are setting Filezilla as a permanent FTP server behind a software firewall you can try to make the following configuration setup:

1) You must specifically setup a range of port number for Passive mode use. These are the port number that Filezilla will assign to the PLC to make a data channel connection when it attempts to transfer a file using passive mode. The following is an example where two port numbers are assigned so that two PLCs may connect to the Filezilla simultaneously. You can add a larger port range if more PLCs may connect to the FTP Server simultaneously.



Next, open up Windows Firewall and add to the Exception list the port 21 (command port) and ports 41000 to 41001 (or whatever range limit you have set to in the FileZilla Setup screen). This should allow the FTP Server to receive connection from these ports that the PLC will be using to make the command and data connections.





20.8 Uploading File From PLC to FileZilla FTP Server Directory

20.8.1 Overview of The FTP Protocol

The FTP protocol requires **two** socket-connections between the devices performing the file transfer. One connection is the “command” channel where FTP commands such as STOR or DELE and the responses are sent as plain ASCII text strings between the FTP client and the FTP server. The second connection is the “data” channel where only the file content or the file directory data are being transferred.

There are two transfer modes: “Active” mode and “Passive” mode. Active mode requires that the **server** establish a data connection back to the **client.** Passive mode on the other hand, requires that the client also be the one to establish the data connection. i.e. For passive mode, both the command and the data connections are performed by the client (the PLC in this case).

The PLC has been designed to use only **passive** mode to transfer file to the FTP server. Passive mode is preferred because that is the only way to transfer file if the FTP Server is located on the Internet. The alternative active mode transfer requires the server to make a data connection back to the PLC that is sitting behind a router firewall, and that can be problematic unless the router is specifically configured to forward the data port to the PLC.

20.8.2 PLC FTP Upload Procedure

In order to upload file to the FTP Server, the PLC would use the PRINT #4 “<TCPConnect xxx.xxx.xxx.xxx:21>” command tag to connect to the FTP server port 21 to establish the “command” connection to the FTP Server. The PLC uses its PRINT #4 to send and INPUT$(4) to receive ASCII text strings from the FTP server via the command channel. The PLC would then send the command “PASV” to inform the FTP server that it wants to transfer a file in passive mode.

At this point you can use the PRINT #4 command to send any valid FTP command to the server, including changing directory (CWD command – make sure the directory exist), deleting a file (DELE command – beware of what you are deleting!) etc.

When the PLC is ready to start a file transfer to the FTP server, the server will in turn provide the PLC with the port number that it has opened for the PLC to connect to establish a data connection. Upon receiving this port number the PLC will make a second TCP connection to the given port and then the actual file transfer will begin.

A new, network service command tag named “FTPUPLD” handles the negotiation between the FTP Server and the PLC as well as handling of the data transfer from the PLC to the FTP server. The following is the syntax:

`PRINT #4 “<FTPUPLD Zxxx.yyy [destination file name]>”`

Where: Zxxx.yyy is the file name of the extended file that the PLC has access to. The [destination file name] can be any legal name acceptable to the server so you can attach a date or time stamp to the file name for easy identifications.

When the above **<FTPUPLD>** command is run, the PLC will send the actual “STOR” command in the background to the FTP server and then obtain the port number from the server and it will then make a data connection to it, and file transfer can then begin.

20.8.3 Monitoring The FTP Upload Progress

Once the file transfer begins the PLC firmware will handle the rest of the file transfer until either the file has been completely transferred or the transfer is aborted due to a network or server trouble. You can monitor the progress of the file transfer using either the **`STATUS(4)`** or **`STATUS(20)`** functions.

`STATUS(4)=` **`0`** **:** FTP client was idle or last FTP failed
`1` **:** FTP data transfer just started
`2` **:** 1st FTP segment transferred, now transferring the rest
`3` **:** FTP data transfer completed.

STATUS(20) > 0**:** Number of bytes uploaded to FTP Server. Transfer is in progress.
< 0**:** Total number of bytes uploaded. Transfer completed.

For example, If 2,345 bytes has been uploaded to the server and the transfer has ended, **STATUS(20)** will return the number = –2345.

Since file transfer can take substantial amount of time to complete, it is not wise to run a loop to wait for the file transfer to complete since this will block the PLC from processing any other part of the program. The demo "ExtendedFileSystem.PC6" shows you how to setup a monitoring function to periodically monitor the progress of the file transfer and report the transfer status on the LCD display.

Please refer to the comments in the custom function "fnConnFTP" and "fnMonFTP" of the "ExtendedFileSystem.PC6" program for more detailed descriptions of each command involved in the FTP file transfer.

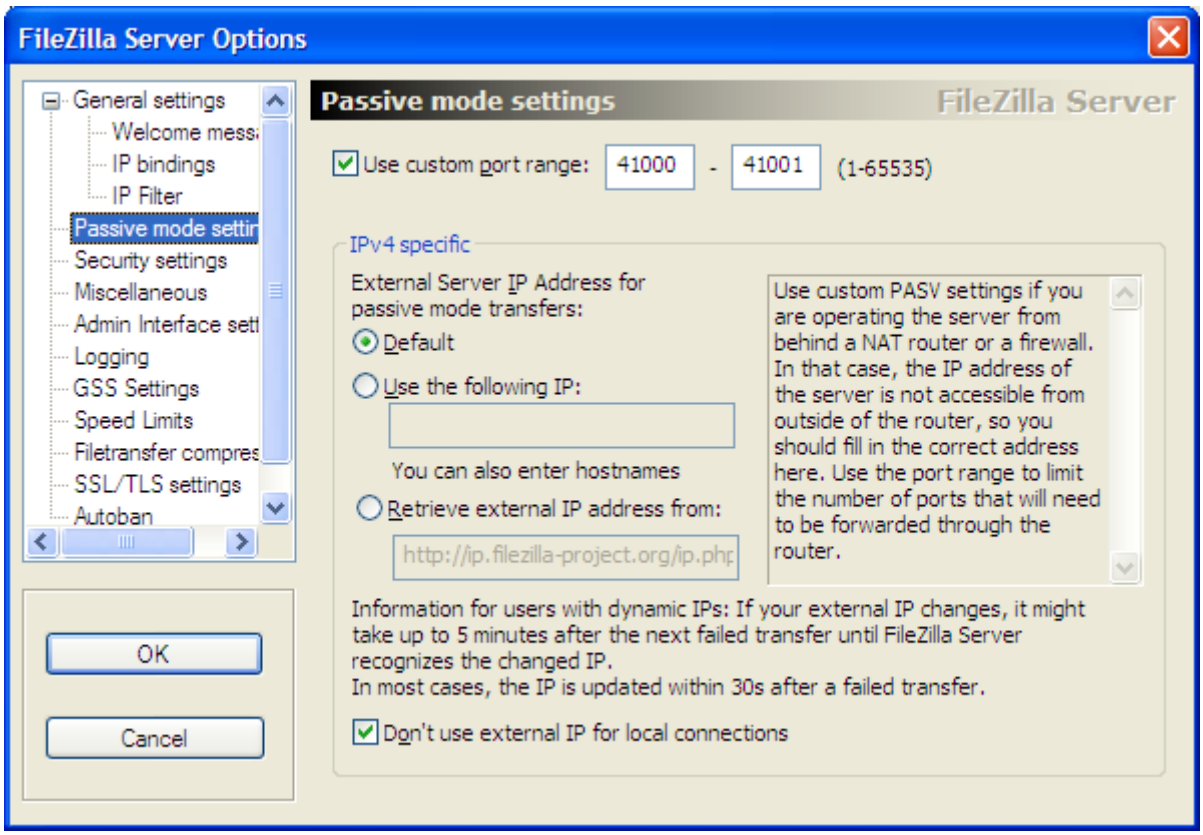
20.9 Setting Up A FTP Server Behind a Windows Firewall.

Please refer to the following Microsoft document describing issues and solutions related to FTP server behind the Windows Firewall.

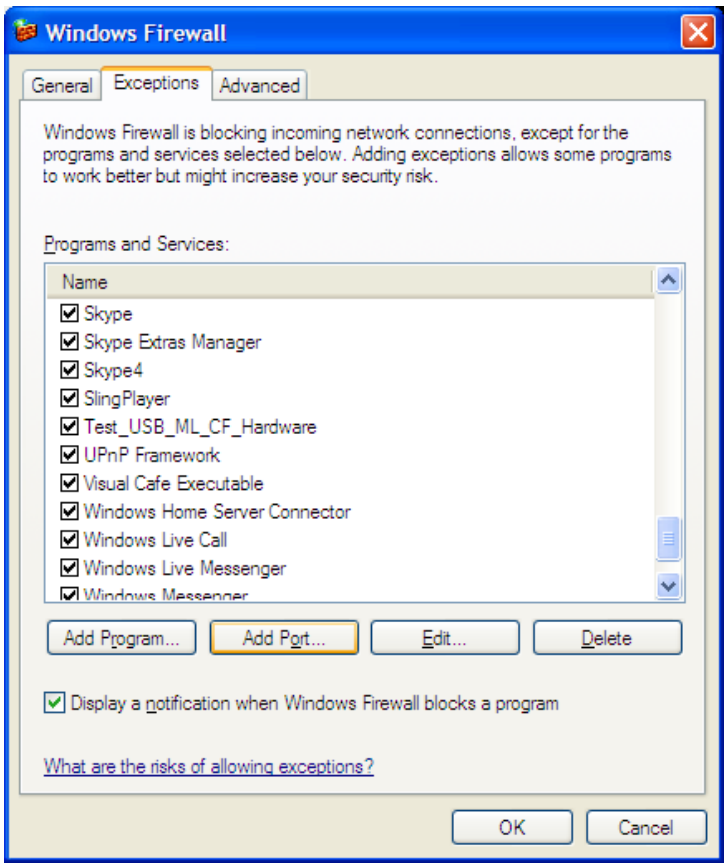
http://technet.microsoft.com/en-us/library/dd421710(WS.10).aspx

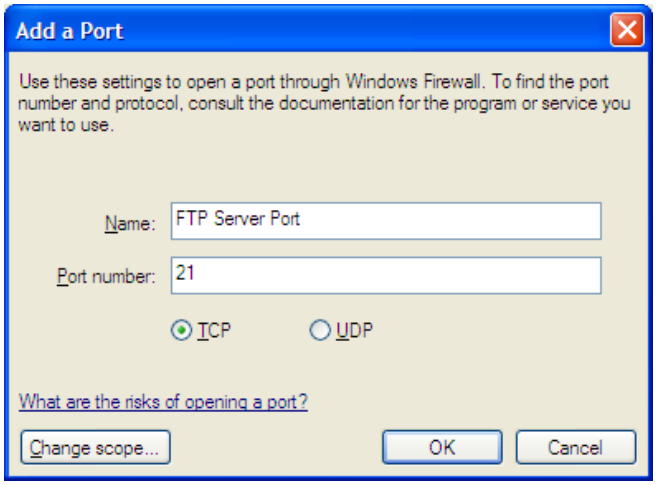
Microsoft focuses mainly on the FTP server in their IIS server (for obvious reasons) instead of Filezilla. If you are setting Filezilla as a permanent FTP server behind a software firewall you can try to make the following configuration setup:

1) You must specifically setup a range of port number for Passive mode use. These are the port number that Filezilla will assign to the PLC to make a data channel connection when it attempts to transfer a file using passive mode. The following is an example where two port numbers are assigned so that two PLCs may connect to the Filezilla simultaneously. You can add a larger port range if more PLCs may connect to the FTP Server simultaneously.



Next, open up Windows Firewall and add to the Exception list the port 21 (command port) and ports 41000 to 41001 (or whatever range limit you have set to in the FileZilla Setup screen). This should allow the FTP Server to receive connection from these ports that the PLC will be using to make the command and data connections.


Windows Firewall
General
Exceptions
Advanced
Windows Firewall is blocking incoming network connections, except for the programs and services selected below. Adding exceptions allows some programs to work better but might increase your security risk.
Programs and Services:
Name
Skype
Skype Extras Manager
Skype4
SlingPlayer
Test_USB_ML_CF_Hardware
UPnP Framework
Visual Cafe Executable
Windows Home Server Connector
Windows Live Call
Windows Live Messenger
Add Program...
Add Port...
Edit...
Delete
Display a notification when Windows Firewall blocks a program
What are the risks of allowing exceptions?
OK
Cancel


Add a Port
Use these settings to open a port through Windows Firewall. To find the port number and protocol, consult the documentation for the program or service you want to use.
Name:
FTP Server Port
Port number:
21
TCP
UDP
What are the risks of opening a port?
Change scope...
OK
Cancel

Chapter 21 PLC As A Modbus/TCP Gateway

21 USING PLC AS A MODBUS/TCP GATEWAY

21.1 Introduction

A Fx PLC or a Nano-10, FMD PLC with >= r77 firmware can be configured to act as the "MODBUS/TCP GATEWAY" for other Modbus serial devices while continue to function as a Super PLC!

A Modbus/TCP gateway essentially translates a Modbus/TCP command packet it receives from its Ethernet port into a Modbus RTU serial command and sends it out of its serial port (RS232 or RS485). If the Modbus RTU slave sends back a serial response, the gateway will in turn translate the RTU response data back to Modbus/TCP response packet and return to the client via the Ethernet port.

As such the Modbus/TCP gateway enables any non-Ethernet equipped, both TRi or 3rd party serial Modbus slave device to be directly accessible by a Modbus/TCP client via the Ethernet or the Internet. In addition, the user PLC program does not need to handle the gateway function at all as the CPU performs the translation functions automatically and transparently to the PLC's program.

Best of all, while acting as a gateway, the Nano-10, FMD or F-series PLC program can **simultaneously act as a Modbus master PLC** and read/write to any registers inside any of the attached Modbus RTU slave! The CPU firmware automatically schedules the order of the command/response packets whether it is originated from the client or from the PLC itself so that the Modbus/TCP command from the client will be responded to in the correct order and in a timely manner.

21.2 Application Ideas for Modbus/TCP Gateway

The Modbus/TCP gateway function can be very useful for many large area control systems, such as a building automation system. A master FMD or F-series PLC is linked to many Modbus RTU slave controllers via RS485 bus distributed across an entire building. The master PLC can perform sophisticated control functions since it has read/write access to **ALL** the Modbus RTU slaves it connects to. At the same time, a Modbus TCP client software such as a Building Management System (BMS) can access any registers in the master PLC or **ANY** of the slave Modbus RTU controller directly via the master PLC acting as a gateway for the slave PLCs.

21.3 Configuring The PLC As Modbus/TCP Gateway

The command used to define a serial port to become Modbus/TCP Gateway serial port is as follow:

```
SETSYSTEM 12, n
```

Where n = 1, 2 or 3 for COMM1, COMM2 and COMM3 port.

This statement must be run once (could be during start up via 1st.Scan pulse) to configure the serial port #n to be used to send out RTU commands and received RTU responses. If SETSYSTEM 12,n is not run then the Modbus gateway function will be disabled.

Here is how it works: A Modbus/TCP client (such as a SCADA, HMI etc) would send a Modbus/TCP request with a specific 8-bit Modbus slave ID to the gateway PLC. If the ID **matches** the native ID that is assigned to the gateway PLC then the PLC will react normally by sending its own Modbus/TCP response packet back to the client. The PLC will not send any RTU command out of the gateway serial port. This means that the client can access the master PLC's own register as per normal.

However, if the client send a Modbus/TCP packet with a different ID from that of the gateway PLC's ID, then the gateway PLC will translate the command into a Modbus RTU command and send it out of the gateway serial port #n defined by the "SETSYSTEM 12, n" statement. The PLC will also wait for a response from the RTU slave via the gateway serial port, and if it does receive a response it will translate it into Modbus/TCP response packet and return to the client.

Note:

1) If the Modbus/RTU slave being addressed does not respond to the RTU command, then the gateway PLC will wait till time-out (default is about 150ms) and then resend the Modbus RTU command and wait for a response again. By default the gateway PLC will repeat this procedure twice and if it still doesn't receive a response after repeated attempts, it will send back a "TARGET DEVICE FAILED TO RESPOND" error packet back to the Modbus/TCP client. (i.e. It will set the 7th bit of the function code to "1" and send back an exception code "0B" hex).

2) The CPU firmware is designed to handle only one Modbus gateway translation for each Modbus/TCP connection per ladder logic scan. This is to ensure that the CPU is not completely bogged down by its Modbus gateway job and will have time to run its own program. If the Modbus/TCP clients attempt to overload the PLC with more requests than it can handle the PLC will frequently return "SLAVE DEVICE BUSY" error response packets back to the client. (i.e. It will set the 7th bit of the function code to "1" and send back an exception code "06").

21.4 Fine-Tuning The Modbus/TCP Gateway Function

It is important to know that every time the PLC runs a Modbus/TCP to Modbus RTU translation its own program scan time increases due to the need for the CPU to wait for a response from the slave Modbus RTU devices. The delay becomes much more pronounced when a Modbus slave device fails to respond (e.g. not online). The system designer must therefore take this into consideration when designing the system:

1) The Modbus/TCP client should not overload the Modbus gateway with unnecessarily frequent Modbus requests. Whenever the PLC report a "TARGET DEVICED FAILED TO RESPOND" function the client program should back off for a while before re-trying communication with the Modbus/RTU slaves again.

2) You can change the number of wait states the gateway will wait for a response from the Modbus/RTU slave by running:

```
SETSYSTEM 1, w
```

Where w = number of serial port wait states for each COMM port as follow:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Ethernet		COMM3		COMM2		COMM1	

Thus each COMM port could be configured to wait from 0 to 3 wait states for a response. The default settings for w = &H55 (or 01010101 binary) which means each COMM port as well as the Ethernet port uses 1 wait state of about 150ms..

3) You can change the number of retries (default = 2) the gateway will attempt to get a response from the Modbus RTU slave by running:

```
SETSYSTEM 2, n
```

Where n = number of retries for each COMM port as follow:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Ethernet		COMM3		COMM2		COMM1	

Thus each COMM port could be configured to re-attempt to send the command from 0 to 3 times if it does not get a response from a Modbus RTU slave.

The default settings for n = &HAA (or 10101010 binary) which means each COMM port as well as the Ethernet port will retry twice (making a total of 3 attempts). E.g. If you run SETSYSTEM 2, 0 the gateway will not resend the RTU command on any of its COMM port if does not receive a response on the first attempt. This can help to reduce the CPU scan time if a Modbus RTC slave does not respond.

4) We strongly recommend that the serial port #n that is to be used as a gateway serial port be configured to be “No protocol” by running the SETPROTOCOL n, 10 statement once. This is to ensure that the serial data returned from the RTU slaves will never be interpreted wrongly by the master PLC as incoming commands, which could lead to errors. (The same advise applies to PLC programs that use the serial port to run READMODBUS, WRITEMODBUS, READMB2 and WRITEMB2 commands).

21.5 Modbus/TCP Gateway Sample Program

Please download the following self-explanatory I-TRiLOGI sample program that demonstrates the new Modbus/TCP Gateway capability via any of the selected COMM port:

http://www.tri-plc.com/trilogi/modbusgateway.zip