

# LADDER + BASIC

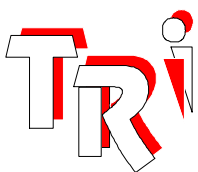
Editor

Simulator

Web-Server

# Internet TRiLOGI

Version 5.2



Triangle Research  
International, Inc.

**Programmer's  
Reference**

## Copyright Notice

TRiLOGI Version 5.x are trademarks and copyrights ©2001-2003 of TRIANGLE RESEARCH INTERNATIONAL, Inc. ("TRi").

All rights reserved. No parts of this manual may be reproduced, transmitted, transcribed, stored in retrieval system, or translated into any human or computer language, in any form or by any means, without the express written permission of TRIANGLE RESEARCH INTERNATIONAL PTE LTD, SINGAPORE. Please refer all inquiries to [info@tri-plc.com](mailto:info@tri-plc.com)

\* MSDOS and Windows 95/98, NT, 2000 and XP are a trademarks of Microsoft.

MODBUS is a trademark of Groupe Schneider.

All other trademarks belongs to their respective owners.

## Disclaimer

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 responsibilities for errors or omissions or any consequential damages resulting from the use of the information contained in this publication.

# Table of Contents

## Chapter 1 – Internet TRiLOGI 5.x Installation Guide

- I. Introduction to the TRiLOGI Program 1-1
- II. Installing TRiLOGI 5.x on Windows 98, Me, NT, 2000 or XP 1-1

## Chapter 2 – Introduction to Internet TRiLOGI Client/Server Architecture

2-1

## Chapter 3 – Using The TLServer – Web Server for TRiLOGI

- I. Overview 3-1
- II. Serial Port Setup 3-3
- III. Configure Users 3-7
- IV. Setup Emails 3-9
- V. Files and Email Services 3-12

## Chapter 4 – Running The Internet TRiLOGI Client

- I. Running The Internet TRiLOGI Application 4-1
- II. Running TRiLOGI Applet Using Web Browser 4-2

## Chapter 5 – Ladder Logic Programming Tutorial

- I. Your Assignment: Creating Your First Ladder Logic Program 5-1
- II. Testing Your Ladder Logic Program Using The Simulator 5-11
- III. Transferring Your First Ladder Program To The PLC 5-13

## Chapter 6 – TRiLOGI Ladder Logic Editor Reference

- I. The Browse Mode 6-1
- II. The Circuit-Editing Mode 6-3

## Chapter 7 - TRiLOGI Main Menu Reference

- I. File Menu 7-1
- II. Edit Menu 7-4
- III. Controller Menu 7-7
- IV. Simulate Menu 7-10
- V. Circuit Menu 7-12
- VI. Help Menu 7-13

## Chapter 8 - Ladder Logic Language Reference

- I. Ladder Logic Fundamentals: Contacts, Coils, Timers and Counters 8-1
- II. Special Bits 8-5

III. Special Functions	8-7
IV. Using TRILOGI Sequencers	8-12

## Chapter 9 - Introduction to TBASIC Custom Functions

I. Overview	9-1
II. Custom Function Editor	9-1
III. Custom Function Execution	9-2
IV. Simulation & Examination of TBASIC Variables	9-5
V. On-Line Monitoring of TBASIC Variables	9-8
VI. Error Handling	9-9

## Chapter 10 – TBASIC Statements, Functions, Operators And Variables

I. What are TBASIC Statement and Functions?	10-1
II. TBASIC Integer Constants, Variables & Operators	10-2
III. String Variables and Constants	10-7
IV. Special Variables – EMINT, EMLINT & EMEVENT	10-8

## Chapter 11 - TBASIC Keyword Reference

1. ABS( <i>x</i> )	11-1
2. ADC( <i>n</i> )	11-1
3. ASC( <i>x</i> )	11-1
4. CALL <i>n</i>	11-1
5. CHR\$( <i>n</i> )	11-2
6. CLRBIT <i>v,n</i>	11-2
7. CLRIO, SETIO, TOGGLEIO, TESTIO	11-2
8. CRC16	11-3
9. DELAY	11-3
10. FOR ... NEXT	11-4
11. GetCtrSV ( <i>n</i> ); GetTimerSV ( <i>n</i> )	11-4
12. GETHIGH16( <i>v</i> )	11-5
13. GOTO @ <i>n</i>	11-5
14. HEX\$( <i>n</i> ), HEX\$( <i>n,d</i> )	11-5
15. HEXVAL( <i>x</i> \$)	11-6
16. HSCDEF <i>ch, fn_num, value</i>	11-6
17. HSCOFF <i>ch</i>	11-7
18. HSTIMER <i>n</i> (High Speed Timers)	11-7
19. IF..THEN..ELSE..ENDIF	11-7
20. INCOMM( <i>ch</i> )	11-8
21. INPUT\$( <i>n</i> )	11-9
22. INTRDEF <i>ch, fn_num, edge</i>	11-9
23. INTROFF <i>ch</i>	11-9
24. LEN( <i>x</i> \$)	11-9

25. LET	11-9
26. LOAD_EEP( <i>addr</i> )	11-10
27. LOAD_EEP\$( <i>addr</i> )	11-10
28. LSHIFT <i>i,n</i>	11-11
29. MID\$( <i>x\$,n,m</i> )	11-11
30. NETCMD( <i>ch, x\$</i> )	11-11
31. OUTCOMM <i>n,x</i>	11-12
32. PAUSE	11-12
33. PIDcompute( <i>ch,E</i> )	11-13
34. PIDdef <i>ch, lmt, P,I,D</i>	11-14
35. PMON <i>ch</i> ; PMOFF <i>ch</i>	11-15
36. PRINT # <i>n x\$</i> ; <i>y</i> ; <i>z</i> ....	11-15
37. PULSEFREQUENCY( <i>ch</i> ); PULSEPERIOD( <i>ch</i> ); PULSEWIDTH( <i>ch</i> )	11-16
38. READMODBUS ( <i>ch, DeviceID, address</i> )	11-16
39. READMB2 <i>ch, ID, addr, var, count</i>	11-17
40. REFRESH	11-18
41. REM (or ')	11-18
42. RESET	11-18
43. RETURN	11-18
44. RSHIFT <i>i,n</i>	11-19
45. SAVE_EEP <i>data, addr</i>	11-19
46. SAVE_EEP\$ <i>data, addr</i>	11-20
47. SETBAUD <i>ch, baud_no</i>	11-21
48. SETBIT <i>v,n</i>	11-21
49. SetCtrSV <i>n, value</i> ; SetTimerSV <i>n, value</i>	11-22
50. SETDAC <i>n,x</i>	11-22
51. SETHIGH16 <i>v, data</i>	11-22
52. SETIO <i>labelname</i>	11-23
53. SETLCD <i>n,offset,x\$</i>	11-23
54. SETLED <i>n,m, value</i>	11-23
55. SETPASSWORD <i>string</i>	11-24
56. SETPROTOCOL <i>ch, mode</i>	11-25
57. SETPWM <i>n,x,y</i>	11-26
58. SETSYSTEM <i>n, data</i>	11-26
59. STATUS (n)	11-27
60. STEPCOUNT( <i>ch</i> )	11-27
61. STEPCOUNTABS( <i>ch</i> )	11-28
62. STEPHOME <i>ch</i>	11-28
63. STEPMOVE <i>ch, count, r</i>	11-28
64. STEPMOVEABS <i>ch, position, r</i>	11-29
65. STEPSTOP <i>ch</i>	11-30
66. STEPSPEED <i>ch, pps, acc</i>	11-30
67. STR\$( <i>n</i> ); STR\$( <i>n, d</i> )	11-31
68. STRCMP( <i>A\$,B\$</i> )	11-31
69. STRLWR\$( <i>A\$</i> )	11-31
70. STRUPR\$( <i>A\$</i> )	11-32

71. TESTBIT ( <i>v,n</i> )	11-32
72. TESTIO ( <i>labelname</i> )	11-32
73. TOGGLEIO <i>labelname</i>	11-32
74. VAL( <i>x\$</i> )	11-32
75. WHILE ... ENDWHILE	11-33
76. WRITEMODBUS <i>ch, DeviceID, address, data</i>	11-33
77. WRITEMB2 <i>ch, ID, addr, var, count</i>	11-34

## **Appendix 1 - Application Notes & Programming Examples**

I. Important Notes to Programmers of TRILOGI Version 5.x	A1-1
II. TRILOGI Sample programs	A1-3
1. Display Alphanumeric Messages on built-in LCD Display	A1-4
2. Setting Timer/Counter Set Values (S.V.) Using LCD Display	A1-5
3. Using a Potentiometer As An Analog Timer	A1-6
4. Motion Control of Stepper Motor	A1-8
5. Activate Events at Scheduled Date and Time	A1-9
6. HVAC (Heating, Ventilation and Air-Conditioning) Control	A1-10
7. Closed-Loop PID Control of Heating Process	A1-11

## **Appendix 2 - PLC & PC Hardware Setup and Configuration**

I. PLC to PC Connection	A2-1
II. Networking Issues	A2-1

## **Appendix 3 - PLC-to-Modem Communication Setup**

1. Modem Connection	A3-1
2. Communication Speed	A3-2
3. Software and Programming	A3-3

# Chapter 1: Internet TRiLOGI 5.x Installation Guide

---

## I. Introduction to the TRiLOGI Program

---

TRiLOGI is a trademark name used by Triangle Research International to describe its family of Ladder or Ladder+BASIC program editor, simulator and up-loader software.

The original TRiLOGI program was written to run under the Microsoft MS-DOS operating system and today is still available for programming TRI's range of PLCs such as the E10, the H-series and M-series PLCs. In fact, we have included all the DOS version of TRiLOGI software in the root directory of your CD-ROM free-of-charge within the zip file "DOS TRiLOGI V4.13.zip"

The newer Internet TRiLOGI 5.x client/server suite is written to run under all currently available 32-bit Microsoft Windows operating systems as Windows 98, Me, NT, 2000 and XP. Due to the fact that Internet TRiLOGI is written in 100% pure Java, we could someday port the software suite to other O/S platforms such as Apple OSX or Linux. However, at the moment only the following PC Windows O/S have been tested: Windows 98, ME, NT, 2000 and XP. The "SetupTL5.exe" file can only run under the PC based Windows O/S.

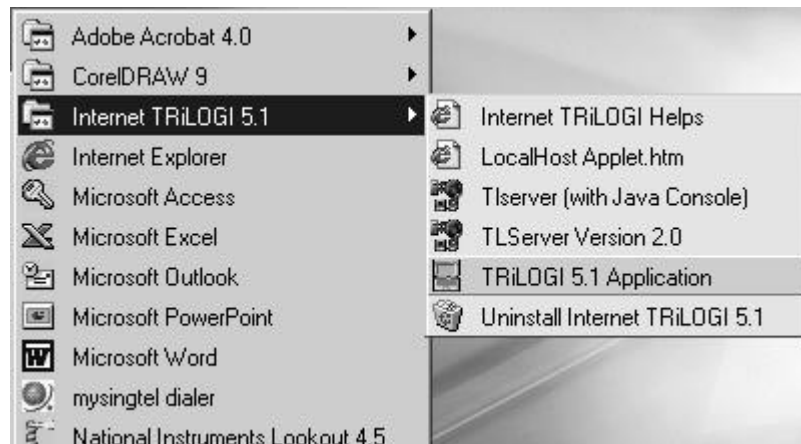
---

## II. Installing TRiLOGI 5.x on Windows 98, Me, NT, 2000 or XP

---

1. In the root directory on the Internet TRiLOGI 5.x CD-ROM, you should find a folder "x86-Windows" which is where all the setup files for PCs running MS Windows are located.
2. You should install Java Run Time Environment (**JRE**) Version 1.3.1 on your PC **BEFORE** installing the TRiLOGI Client/Server bundle. First, double-click on the file "j2re1\_3\_1-win.exe" to install Java. Please follow all instructions provided by the Install Shield program and install it in the given default path: "C:\Program Files\JavaSoft\JRE\1.3.1".
3. After you have installed JRE 1.3.1, double-click on the "SetupTL5.exe" to install Internet TRiLOGI.
4. All TRiLOGI Version 5.x files will be installed in the following default directory: "C:\TRiLOGI\TL5". You normally would not need to go directly to this directory to run TRiLOGI. This is because during installation of TRiLOGI, a program Group folder "Internet TRiLOGI 5.x" will be created to in the

Start Menu to provide short cuts to the TLServer program, the TRiLOGI application and the TL50Applet starter, as illustrated in the following picture.



**Note:** A short cut for TLServer is also created on the "Quick Launch" toolbar on some Windows platform that supports it.

TLServer shortcut



## Chapter 2: Introduction to Internet TRiLOGI Client/Server Architecture

---

1. Internet TRiLOGI is a **Client / Server** application suite. The entire program is broken into two parts: the **Server** and the **Client**.
2. **Server:** In order to run the complete TRiLOGI program, including access to the PLC, you must start the **TLServer** first. TLServer behaves like a typical web server and it is capable of serving HTML web pages as well as Java Applet to an Internet Browser such as the Microsoft Internet Explorer or Netscape Navigator. TLServer connects to the PLCs via the PC's serial communication port and it is the one responsible for conveying communication messages between the Internet TRiLOGI client and the M-series PLCs. (Note: TLServer is not included with Education version of TRiLOGI since there is no real PLC involved.)
3. **Client:** The TRiLOGI program is the one which you use to create your ladder logic + TBASIC program and is called the "Client" program. (If you are programming the PLC offline then you only need to run the client program without the TLServer.) The beauty of the client/server configuration is that it does not matter whether the server and client are located at the same computer or at 20,000 miles apart and they work exactly the same way. The client and the server can communicate via any form of network connection, including the Internet. This makes it possible for the user to program the PLCs either locally or remotely via the Internet or even wirelessly via mobile Internet.

Another important advantage of client/server architecture is that **multiple clients** may access the same server simultaneously. Hence you can run multiple copies of the TRiLOGI clients at different places around the world simultaneously for troubleshooting a single PLC. You can also run the TRiLOGI client AND the "Tri-Excellink" clients simultaneously!

4. TRiLOGI client software is available in two forms:
  - i. As a Local Java **Application** - The TRiLOGI program (as well as the JVM, see below) must be locally installed in the PC that it runs on.
  - ii. As a Java **Applet** - The client computer only needs to use a Java-enabled Web browser such as the Internet Explorer 5+ or Netscape Navigator 4.5+ to invoke the TRiLOGI applet. There is no need to install the TRiLOGI software in the local computer.

## TRiLOGI Application vs Applet: Which is Better?

	Pros	Cons
Application	<ul style="list-style-type: none"> <li>• Starts up immediately.</li> <li>• Can read/write TRiLOGI files to local hard disk or to TLServer.</li> <li>• Can access any TLServer on the network.</li> <li>• Program behavior predictable since the copy of JVM is local.</li> <li>• Printing is supported via Java 2 function calls to the JVM.</li> </ul>	<ul style="list-style-type: none"> <li>• Require local installation of TRiLOGI software at every client computer.</li> <li>• Require installation of JVM at every client computer.</li> <li>• Need to specify the proxy server IP address clearly if running behind a firewall.</li> <li>• Printing Service is not available to the applet.</li> </ul>
Applet	<ul style="list-style-type: none"> <li>• No need to install any software or JVM at the client computer.</li> <li>• Possible to control your PLC via any "Cyber Café".</li> <li>• Maintenance and Upgrading of software is simple since only one copy of the TL50Applet.jar file needs to be changed.</li> <li>• Centralized storage of program files only at the server. This is good for providing PLC program training.</li> </ul>	<ul style="list-style-type: none"> <li>• Can only read/write TRiLOGI files to the TLServer but not to the local hard disk.</li> <li>• Can only access the TLServer from which it was loaded.</li> <li>• May take a few minutes to load itself the first time. (Thereafter the browser should cache it for rapid start up.)</li> <li>• Program behavior may vary for different make or different versions of the browser.</li> </ul>

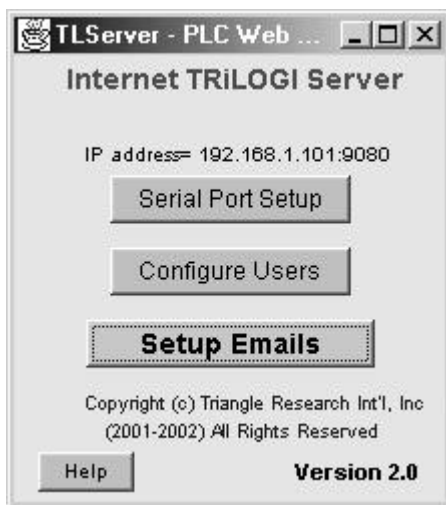
We shall describe how to run the TRiLOGI Application and Applet software in Chapter 4.

## Chapter 3: Using The TLServer – Web Server for TRiLOGI

### I. Overview

**To Start TLServer**, either click on its Icon on the Windows “Quick Launch bar” (shown below) or click on the “Start” button and select “Internet TRiLOGI 5.x”, then select “TLServer Version 2.x” and a TLServer panel will appear. You can minimize TLServer but it should be **actively running** in order to service network request from TRiLOGI via the Internet or local area network.

TLServer shortcut



TLServer acts as a gateway to connect the M-series PLC to the corporate LAN or the Internet so that they can be controlled and programmed by a TRiLOGI client from anywhere in the world.

When a client program such as TRiLOGI wants to read from or write to a PLC, it send a command to the TLServer using the TCP/IP protocol transported via the Intranet, the Internet or a local host connection.

The TLServer, upon receiving the command, will carry out the actual reading or writing to the PLC via the PC's RS232 or RS485 port. The data received from the PLC is then relayed back to the client program via TCP/IP protocol. TLServer is also a Web Server which serves up web pages that contain the TRiLOGI Java Applet to enable you to use any Java-enabled Web browser to access the PLC without the need to install a local copy of the TRiLOGI application software

**Note:** Starting from Version 2.0, TLServer also provides "**File and Email Services**" to the PLCs. That means that a PLC can send a command to the TLServer to open a file and save its data into the PC's hard disk. It can also command the TLServer to send out the data as an email to anybody in the world. File and Email Services are described in Section V in this chapter.

The new Email Service works differently from the original email function provided in TLServer 1.0 in that TLServer does not poll the PLC; instead it is the PLC that initiates an email request asynchronously. This makes it possible for a PLC to dial-in via a modem to request the TLServer to help it send out an email without demanding a constant connection the TLServer. However, the original email function is still supported in

Version 2.0 because that has the advantage of being able to service email requests for multiple PLCs linked via the RS485 network.

When TLServer is first started, it will query the operating systems for the IP addresses of the computer that it runs on. (It may take a while if the O/S is slow to return the IP address). It will then display the obtained IP addresses (maximum of two) on the TLServer front panel so that the user can quickly determine the IP addresses that they can use to access the PLC. The following are some possible **IP address** scenarios:

- If the computer is not linked to any network or the Internet and does not have any network adapter installed, then only the localhost IP address ("127.0.0.1:9080" where 9080 is the port number) will be displayed.

**Note:** regardless of whether your PC is networked or not, **the local host IP address: 127.0.0.1:9080 is always available** to the client program running on the **same PC** where the TLServer is running, even though it may not be displayed on the TLServer's front panel. (TRiLOGI and TRi-ExcelLink are all known as "client" programs). So whether your PC is networked or not, you can still use TLServer and TRiLOGI on a localhost connection. In that case the TRiLOGI and TLServer work together on the same PC just like a normal Window based programming software. **We recommend using the localhost IP address: 127.0.0.1:9080** if you are running both the client and the server on the same PC. If the computer has an 'always on' direct connection to the Internet directly then the IP address will be your Internet IP address.

- If the computer is networked to the corporate Intranet, or you have connected this computer to a router to share internet connection with a few other computers, then the IP address shown is an internal IP address, also known as the "**Intranet**" IP address. The intranet IP address is assigned by either the System Administrator or by the router (known as DHCP server). You can access this computer from other computers in the same LAN, but the intranet IP address is not accessible from outside of the LAN.

To access the TLServer from outside of the LAN, You will need to configure your router's internal settings to define the PC that runs the TLServer as a "Virtual Server". You can then access the TLServer using the router's **public** IP address and the routers does the job of translating the **public** IP address to the intranet IP address and route the messages to/from the PC that has been defined as the virtual server. This process is known as Network Address Translation (**NAT**).

- If you connect a computer in a LAN to the Internet via a dial up connection, you will see two IP addresses: one is the Intranet address and another is the Internet IP address. The Intranet address is only accessible from within the Intranet. The Internet IP address will be what you need to use if you are accessing TLServer from the Internet. See the PLC Setup & Configuration section of the installation guide for more detailed explanation of Intranet Installation and problem with Firewalls.

**Notes to Dial-Up Users:** If you are testing the Internet capability of TLServer using dial-up connection, you **must connect to the Internet first** before starting TLServer so that TLServer can report the correct Internet IP address to you. You will not see the local host IP address (127.0.0.1), only the Internet IP address will be shown.

The moment TLServer is running, it is ready to accept connection from the TRILOGI client. You can also configure TLServer's communication port setting, add/remove users from the system and set up TLServer to query the PLC for outgoing email requests and process them accordingly. The following sections explain the function of each button. (Remember, you can also call up their context-sensitive help by pressing <F1> key after pressing the relevant button on the TLServer front panel.)

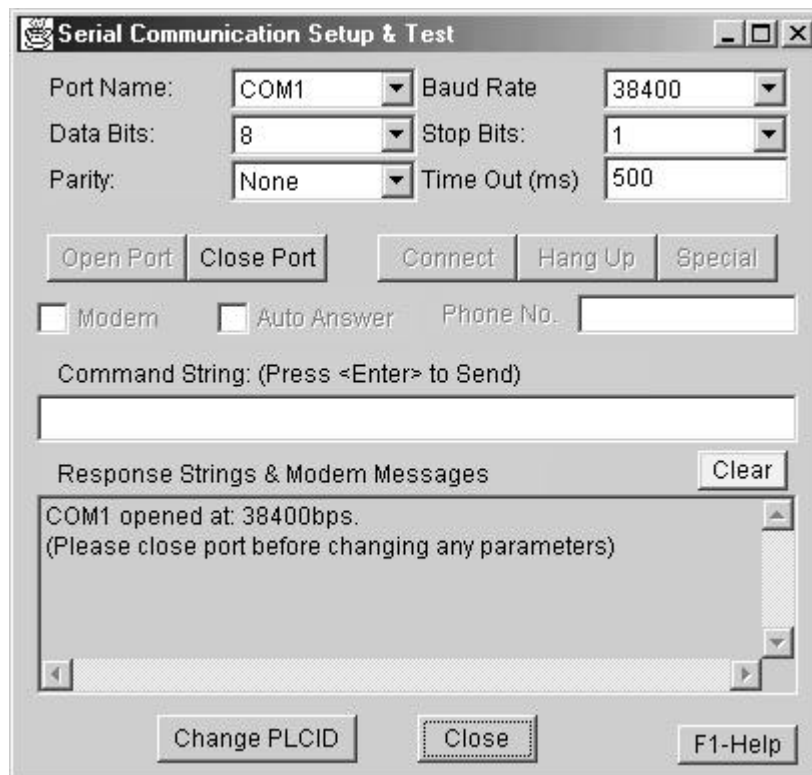
---

## II. Serial Port Setup

Serial Port Setup

---

### 1. Setting Up and Test Serial Communication



This dialog box allows you to configure the serial port of the host computer to match the setting on the PLC for proper communication. Most of the items here are self-explanatory. If you have more than one PLC connected to the host computer via RS485, all the PLCs must have the same serial port settings as the TLServer. The **Open Port** button allows you to test whether the communication port is available to TLServer. You can also click the **Close Port** button to temporarily relinquish the port to other applications


(such as TRiLOGI Version 3 or 4). Note that you will need to close an opened port before you can change its parameter.

The "Command String" text entry field allows you to test communication with the PLC using its native or MODBUS ASCII protocols. If you enter a string here and press <Enter>, the ASCII string will be sent to the PLC connected to the serial port and the response string will be displayed in the bottom text box. If the serial port is not yet opened this command will automatically open it. **Note that only multi-point host link commands are accepted here.** The only point-to-point command acceptable here is the "**IR\***" command which queries the ID address of the PLC.

If you have only **one** PLC connected to your TLServer computer, then you can test the communication now using the following command string:

Command String : **IR\***  
Response String : **IR01\***

The response string tells you that the ID address of this single PLC is 01. You can then try other host link commands using this ID address. (e.g. @01RI0000\* to query the states of inputs #1 to #8)

If you have more than one PLC connected you should not use the "IR\*" since all connected PLCs will try to respond simultaneously, thus resulting in a garbage return string. To change the ID of a PLC, e.g., from 01 to 05, you can send the command string "@01IW0500\*" to the PLC. In TLServer 2.0 and above there is also a  button that does this for you automatically. You can click on the "Detect ID" button to check the current ID and then the "Change ID" button to write the new ID to the PLC.

## 2. Changing Communication Settings

Most likely you may want to leave the comm port settings at their default values: **38,400 bps, 8 data bits, 1 stop bit, no parity**. Some reasons for changing the comm port settings may be due to the need to change the PLC's serial port to lower values (e.g. for communication via radio using 9600 bps). Changes to the comm settings are saved to the TLServer configuration file: "TLserver1.cfg" when you quit TLServer.

One other scenario is when you need to power cycle an M-series PLC with DIPswitch #4 turned ON (to halt the CPU in order to disable any execution by the "1st.Scan" pulse). Since the PLC's serial port is set to 9600 bps when power ON with DIP Switch #4 set, you will need to change the baud rate temporarily in order to communicate with the PLC (e.g. to blank out a program that causes trouble).

However, do remember to **change the baud rate setting back to 38400bps** after you have reset the PLC with the DIP switch OFF, otherwise you may have problems communicating with the PLC later on since changes to comm settings are automatically saved.

### 3. Modem Support

**a) Dial Modem:** TLServer 2.0 incorporates support for dialing a modem connected to the PC's COM port. This is useful if the PLC has to be located at a remote location yet still has access to the public telephone line or to a cellular phone. You can then connect the PLC to a standard analog modem such as the US Robotic 33.6Kbps or Hayes Acura smart modem. The TLServer can then dial the phone number of the remote modem and make a connection. Once a connection is established, the remote PLC is immediately accessible to client applications such as Internet TRiLOGI or TRi-ExcelLink, etc over the Internet, Intranet or localhost as if it were connected to the TLServer via the serial port directly.

#### Notes:

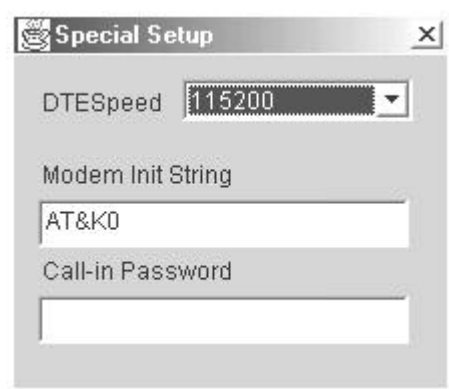
- Due to the time delay for modulation/demodulation process as well as transmission delay, two-way communications via modem tends to be noticeably slower than connection made by direct cable connection. This is quite normal and does not indicate a problem with the communication setup.
- The PC's modem must be able to emulate the COM port of the PC in order for the TLServer modem function to work. Some of the newer computers employ "win modem" or "soft modem" which may only work with Windows' dial-up networking. These kinds of modems are implemented in software and they do not necessarily emulate a standard PC COM port properly. They also demand quite a bit of CPU horsepower to process the communication. Therefore these types of modem may not work too well with the TLServer. If your built-in soft modem does not work properly with TLServer, you should get an external modem. These are quite inexpensive nowadays and they will work much better with the TLServer modem support function.
- To setup TLServer to dial a modem, first close the active COM port by clicking on the "Close Port" button. Select the COM port where the modem is connected (you can find out the which COM the modem is connected to by checking the "Control Panel -> Modems -> Properties"), then click to select the "Modem" checkbox. You will then be able to enter a telephone number to dial. The 3 buttons: "Connect", "Hang Up" and "Special" become enabled when you select the "Modem" mode. Note that the "Baud Rate" field now becomes the "DTE speed" which specify the line rate between the PC and the modem (this has nothing to do with the

actual baud rate between the modems which will be automatically negotiated based on the quality of connection). Normally you should leave the DTE speed set to the highest value (115200) unless your modem manufacturer specifies otherwise. The PLC can be operating at a different baud rate from the PC to modem-line-rate because of the modulation/demodulation action of the modem.

**Important:** The PLC-to-modem connection must be properly prepared before you can use TLServer to connect to the PLC's modem. **Please refer to Appendix 3 to read more details about the PLC-to-Modem Communication Setup.**

Once you have entered a proper phone number, click on the "Connect" button to start dialing the modem (make sure that the "Auto Answer" check box is not checked). If the remote modem is busy or does not answer the call you will see the corresponding error messages in the response box. Click on the "Hang Up" button anytime to abort the dialing operation.

If you click on the "Special" button a special dialog box will appear as follow:



- You can change the DTE speed by selecting a new value from the choice menu.
- You can specify a special AT command to be sent to the modem during modem initialization. Normally you can leave this field to its default value which is AT&K0.

You can also specify a special "Call-in Password" which is only used if the TLServer puts itself in auto-answer mode (see description later). Any incoming connection made by a remote modem must give the correct password upon connection; otherwise the connection will be immediately dropped. The Call-In password feature is disabled if the corresponding textbox is empty.

**b) Auto Answer:** If you select the "Auto Answer" checkbox and click on the "Connect" button, the TLServer will setup the modem to automatically answer the incoming call on the first ring. There are many uses of this capability:

Any number of PLCs in the field can periodically dial in to a single TLServer and write or append the values of their internal variables to data files on

the PC's hard disk using the PLC File Service commands. This is extremely useful for deploying the M-series PLC for data-acquisition purposes. You can format the data using CSV format so that the file can be readily imported into an MS Excel or Lotus 123 spreadsheet.

- The remote PLC can dial in and ask TLServer to send out its data to any email address immediately.
- The remote PLC can dial in and synchronize its real time clock with the TLServer.
- The TLServer can play the role of an ISP where the PLC can dial in and be accessible to the Internet.

Some ". PC5" sample programs that enable a PLC to dial in to the TLServer and request for file or email services are provided in the following folder:

"C: \TRi LOGI \TL5\usr\samples\FileService\_Modem"

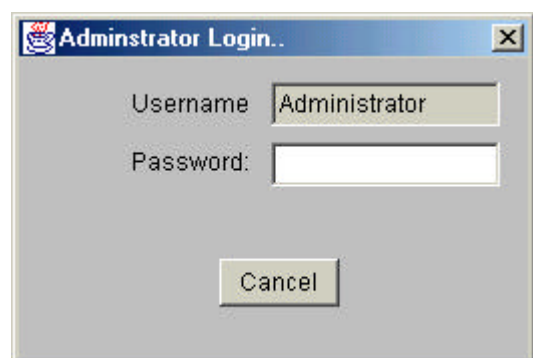
To prevent unauthorized access by any incoming call, you can specify a "Call-In Password" string as described above. If the "Call-In Password" contains any text other than an empty string, then the incoming caller, upon connection, must immediately send a CR-terminated string that matches the "Call-In Password" string in order to maintain the connection. If the password is incorrect the TLServer will disconnect the remote modem to prevent unauthorized access. If the call-in password is validated the TLServer will acknowledge it by sending a CR-terminated string "<OK>" to the remote PLC via the modem. It is the duty of the incoming caller to check the acknowledgement string to ensure that the TLServer does not drop the connection.

---

### III. Configure Users

Configure Users

By definition, only the "Administrator" is authorized to add/delete users and change password. Hence when you click on the "Configure Users" button, you are assumed to be the Administrator and is asked to enter the Administrator's password to gain entry. By default, no password has been defined for the Administrator, so you should just press <Enter> key to gain entry the first time.



Once you get through the Administrator Login screen, a dialog box will popup, which allows you to add new users who are allowed access to the

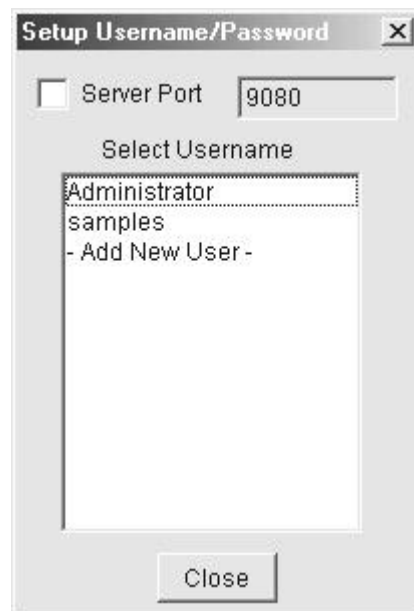
TLServer and the PLCs. You can also change the password, username or the access level of an existing user or delete an existing user. A new user defined here will be given his/her own exclusive subdirectory to store ladder programs. For PCs, this directory is located at:

" C:\TRi LOGI \TL5\usr\

where <username> is the same as the Username defined here.

- **"Select Username"** - Double-clicking on an existing username opens up the username/ password dialog. You can now add password to the Administrator if you wish to prevent unauthorized access to the predefined usernames and passwords. There is also a pre-defined user named "samples" with no password where many samples TRiLOGI files are stored.

If you select a username and then press the <DEL> key, you can delete the user provided its directory is empty. (You can use Window Explorer or TRiLOGI Application to delete the contents of the user's directory first before deleting him/her from TLServer).



- **"- Add New User -"** Clicking on this field allows you to add new users to the system. You can add as many users as you like subject to memory and hard disk limit.
- **Server Port:** If you click the check box to the left of the "Server Port" label, you can change the default "port" that the TLServer listens on. When the client accesses the TLServer. Whatever you define here must be matched by the same port number

E.g. if the port number = 8000, then localhost access must be:

http://127.0.0.1:8000/

However, if the port number is defined as 80 (default port for HTTP server), then you can access the server using just the IP address without the port number: http://127.0.0.1/

### **What Port Number Should TLServer Use?**

In most cases you can simply use the predefined port number "9080". However, you may like to read the explanation box below regarding definition of a "Port". You can see that the default port for most public web-

servers is port 80. You can define TLServer to listen at port 80 as well; in that case there is no need to specify the port number in the URL. However, there are reasons why you may or may not want to do that. It depends on whether you are installing TLServer on a corporate intranet or on the public Internet and whether the client (TRiLOGI) is to access TLServer within the intranet environment or from the public Internet. Please read Appendix 2: "PLC & PC Hardware Setup and Configuration" for an explanation of how to use the port number properly.

**Ports**, or addresses within a computer, are used to enable communication between programs. Port addresses are 16-bit addresses that are usually associated with a particular application protocol. An application server, such as a Web server or an FTP server, listens on a particular port for service requests, performs whatever service is requested of it, and returns information to the port used by the application program requesting the service.

Popular Internet application protocols are associated with *well-known ports*. The server programs that implement these protocols listen on these ports for service requests. The well-known ports for some common Internet application protocols are shown below:

<i>Port</i>	<i>Protocol</i>
21	File transfer protocol
23	Telnet protocol
25	Simple mail transfer protocol
80	Hypertext transfer protocol

---

## IV. Setup Emails

### Setup Emails

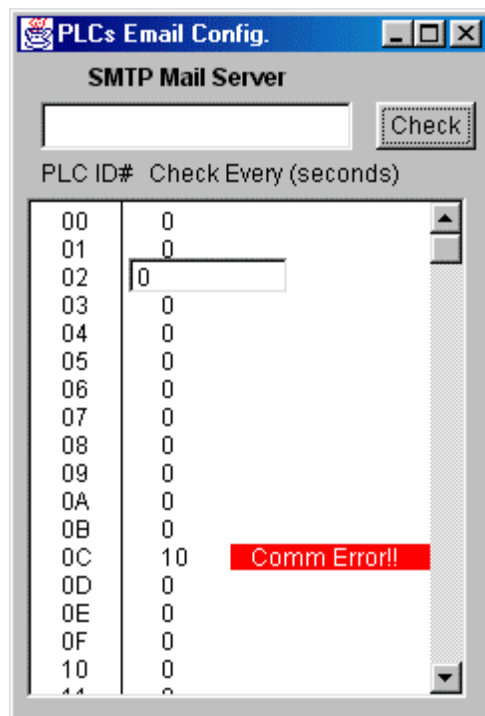
This TLServer Email capability is first implemented in TLServer 1.0 and is carried forward to Version 2.0 and above. This method depends on the TLServer to periodically scan each configured PLC for the state of their email request flags and hence require that the TLServer be constantly connected to the PLC(s) via the serial port. (Another email support function available only to TLServer version 2.0 and above is described in the next section: "File and Email Services")

A PLC program raises an email request flag by setting the variable emEVENT [1] to a non-negative value (see explanation on "Writing TRiLOGI Programs that Can Send Emails" in the later part of this section) whenever it needs to send an email. The TLServer, upon sensing that an email request

flag has been raised, will extract the sender, recipient and message strings from the PLC's internal variables and send them out using the pre-defined SMTP outgoing mail server.

A single TLServer can service the email requests for one or more (max. = 256) PLCs connected to it via RS232 or RS485. To setup the server to handle email requests, click on the "Setup Emails" button on the TLServer to open the following dialog box:

- **SMTP Mail Server:** This will be the same Outgoing Mail Server that you use in your email program such as the Eudora or Outlook Express. If in doubt, ask your ISP or System Administrator for help. This server must be setup properly before the TLServer can send any email. If your SMTP server requires authentication via POP3 you will have to use your email program to check your email once prior to using it to send emails.
- **PLC ID# column:** For you to select PLCs with ID from 00-FF (256 in total) to set the email service period.
- **Check Every (seconds)** - this allows you to define how often the TLServer should check the PLC (the email service period) for the state of the outgoing email request flag.



Simply click on field next to the PLC ID# of interest to open up a text entry field (as shown in the figure for ID=02). Enter a non-zero value (in seconds) to define its email-servicing period.

**Note:** Both SMTP Server and email service period definitions will be saved to the hard disk when you exit TLServer program. They will be reloaded when you start TLServer again. This email service period does not determine how often the PLC will send email, since email will only be sent when the email request flag is set even if you had set a very short email service period. It only affects how quickly the email will be sent whenever a PLC raises its email flag. You should set a short service period (say every 10 seconds) for urgent email (such as alarm condition). For non-urgent email such as hourly or daily production report you can set a much longer servicing period to reduce the communication loads on the PLCs.

## Inactive PLC

TLServer attempts to communicate with every PLC that has a non-zero email service period. However, if the PLC is inactive (e.g. It has not been turned ON or has been disconnected from the server) the communication will fail. Since communication failure takes considerable amount of CPU waiting time and could affect the normal communication with other active PLCs, inactive PLCs are internally marked by the TLServer (shown as **Comm Error!!** message in the email setup dialog) and will not be checked according to their defined service period to avoid repeated communication failure. However, TLServer will re-scan these inactive PLCs every two minutes to check whether they have come on-line. If an inactive PLC is found to respond to command it will be unmarked and put back in service for its email request.

You can also manually force the re-scanning of all PLCs by clicking on the **Check** button once. Then scroll to the PLC of interest to check if there is a **Comm Error!!** message. Check the PLC communication port wiring if there is an error.

## Writing TRiLOGI Programs that Can Send Emails

In order to send an email, the TRiLOGI program needs to use the string variables A\$, B\$, C\$ to store the headers and D\$ to Z\$ to store the messages. (Not all strings need to be used; unused strings are still available for normal program use) The special variable emEVENT[1] is used as an email request flag which should be initialized to -1 when the program is not requesting email service. When the TRiLOGI program wants to send an email, it first stores the sender, recipient and subject into the following variables:

A\$	Sender email address - which can be used to identify the source of the email.
B\$	Recipient email address - this one must be accurate
C\$	Subject of the message.
D\$	First line of Message
E\$	Second line of email message
...	.....
Z\$	The 23rd line of the email message
emEVENT[1]	<b>-1 = NOT</b> sending any email. <b>0 to 23 = number of lines</b> in the email message body which are contained in D\$ to Z\$.

The maximum number of lines in your email is limited by the number of string variables D\$ to Z\$ (23 in total) available in the M-series PLC.

For example, if the PLC needs to send email to `trilogi@yahoo.com` with a 1-line greeting, then the program needs to activate a custom function that contains the following statements:

```
A$ = "Demo1@PLC"           ' sender
B$ = "trilogi@yahoo.com"   ' recipient
C$ = "This is an email demonstration"
' subject
D$ = "The time is"
      +STR$(TIME[1]) + ": " + STR$(TIME[2]) +
      ". How are you doing?" ' Message body
emEVENT[1] = 1
```

You must also setup the email service period (say every 10 second) in the "Setup Emails" screen for this PLC. When the TLServer scans the PLC and found that its `emEVENT[1]` is set to 1, it will extract the headers and message body from the PLC's string variables. Only A\$ to D\$ will be extracted in this example since the message contains one line of body text only, as indicated in `emEVENT[1]`.

TLServer will then contact the SMTP server to send out the email. In addition, **after processing** the email request, the **TLServer will set the `emEVENT[1]` variable to a value of "-1"** (no email). Hence there is no need for the TRiLOGI program to worry about clearing the email request flag after the email has been sent. In addition, this provides a way for the PLC program to know whether the TLServer is functioning properly and whether the email request has already been processed. However, do take note that even if the `emEVENT[1]` has been reset it does not guarantee that the email has been correctly dispatched to the recipient. Success of emailing is subject to the proper configuration of the TLServer, the network quality and availability of the SMTP server at the moment when TLServer tries to send out the email. For urgent situations you may consider sending out multiple emails periodically until the user has attended to the machine.

---

## V. Files and Email Services

---

Starting from version 2.0, the TLServer provides a number of "File and Email Services" to the PLCs via the serial comm port. Basically a PLC can send service requests to the TLServer using "tags" (which are ASCII characters enclosed between the '<' and '>' characters) and the TLServer will perform

the service requests upon receiving valid commands. All data between the <command [parameter]> tag and the </> tag will be treated as data for the requested service.

Since the PLC is the one that initiates the service request, it does not need to be linked to the TLServer all the time unless it needs to request a service from the TLServer. This makes it possible for a remote PLC to connect to the TLServer, via the telephone line and modem to perform the required file or email services, then disconnects itself from the TLServer so that other PLCs can take turns to connect to the TLServer to request for services.

Note: All the files created or used in the write/append/read actions are located in the directory: <trilogi base directory> /FileService. (Hence the default path is **C:\TRI LOGI\TL5\FileService**). You may also read/write files that are located in the sub-directory below the ".../FileService" directory provided that the sub-directory already exists.

The currently supported files and emails services are described below:

<p><b>1. Write data to file.</b></p> <p><b>Format:</b></p> <pre>&lt;WRITE [filename]&gt; data data data... data .... &lt;/&gt;</pre>	<p>E.g. To save data of DM[1] to DM[10] to a file name "testWrite.txt", execute the following statement from a custom function:</p> <pre>PRINT #1 "&lt;WRITE testWrite.txt&gt;" ' Write data request FOR I = 1 TO 10   PRINT #1 DM[I];" "; REM delimited by space characters. NEXT PRINT #1 ' send a CR character. PRINT #1 "&lt;/&gt;" ' End of Service request</pre> <p>The TLServer will close the file after it receives the end-of-service tag "&lt;/&gt;" from the PLC and it will in turn send a "&lt;OK&gt;" string to the PLC to acknowledge that the <b>WRITE</b> request has been completed successfully. It is up to your PLC program to check for the "&lt;OK&gt;" tag to determine if the service it requested have been completed successfully.</p>
<p><b>2. Append data to file</b></p> <p><b>Format:</b></p> <pre>&lt;APPEND [filename]&gt; data data data... data.... &lt;/&gt;</pre>	<p>E.g. To append the time of an event to a file name "testAppend.txt", execute the following statements in a custom function when the event take place:</p> <pre>PRINT #1 "&lt;APPEND testAppend.txt&gt;" ' Append data request PRINT #1 "Event Time = ";TIME[1];";";TIME[2];";";TIME[3] PRINT #1 "A=";A PRINT #1 "&lt;/&gt;" ' End of Service request</pre> <ul style="list-style-type: none"> <li>• If the file does not exist a new file will be created.</li> </ul>

	<p>Otherwise, the PLC's real time clock data in the format "hh:mm:ss" and the value of A will be appended at the end of the file "testAppend.txt" every time the above statements are executed.</p> <ul style="list-style-type: none"> <li>The TLServer will close the file after it receives the end-of-service tag "&lt;/&gt;" from the PLC and it will in turn send a "&lt;OK&gt;" tag to the PLC to acknowledge that the APPEND request has been successfully completed. It is up to your PLC program to check for the "&lt;OK&gt;" tag to determine if it the service it requested have been completed successfully.</li> </ul>
<p><b>3. Email data to recipient</b></p> <p><b>Format:</b></p> <pre>&lt;EMAIL [email address]&gt;  Sender: [sender email] Subject: [subject text] data data data... data .... &lt;/&gt;</pre>	<p>E.g. To send data to an email address: whoever@yahoo.com with the subject "PLC Email Test", execute the following statements:</p> <pre>PRINT #1 "&lt;EMAIL whoever@yahoo.com&gt;" ' change it to your own email. PRINT #1 "Sender: triuser@hotmail.com" ' it can be anything. PRINT #1 "Subject: PLC Email Test" PRINT #1 "Hello, this email is sent by your friendly TRiLOGI PLC" PRINT #1 "Don't worry, everyting is working out great today!" PRINT #1 &lt;/&gt;</pre> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>"Sender:" field should be in email format such as xxx@yyy.zzz, but it does not need be a valid email address.</li> <li>"Subject:" field is optional and may be omitted totally</li> <li>The TLServer will first save all the data it received into a temporary file named "Email.txt" in the default directory. After the TLServer receives the end-of-service tag "&lt;/&gt;" from the PLC and it will then send out the email to the recipient email address. This email service will make use of the SMTP server defined in the "Setup Emails" portion of the TLServer configuration, so make sure that you have defined a correct SMTP server before testing the email service function.</li> </ul> <p>When the email has been successfully sent via the SMTP server, the TLServer will send an "&lt;OK&gt;" tag to the PLC to acknowledge that the <b>EMAIL</b> request has been successfully completed. It is up to your PLC program to check for the "&lt;OK&gt;" tag to determine if it the service it requested has been processed.</p>

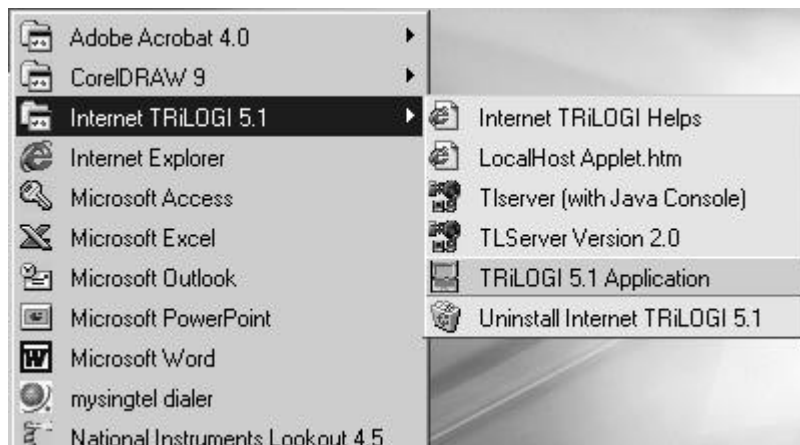
<p><b>4. Read Data from File.</b></p> <p><b>Format:</b></p> <pre>&lt;READ [filename]&gt; &lt;/&gt;</pre>	<p>This service allows the PLC to request the TLServer to open a text file and upload its content to the PLC. This may be useful for loading some previously saved parameters.</p> <p>Upon receiving this command and if the specified [filename] is successfully opened, the TLServer will begin sending all the ASCII characters contained in the text file to the PLC. Note that line breaks in a text file are sent to the PLC as CR character only and not as a CR+LF pair. As such, your PLC program can easily use the INPUT\$(1) command to read in all the CR-terminated text string one string at a time and then interpret or convert the data as necessary. After sending out the last byte in the data file to the PLC, the TLServer will send a CR-terminated acknowledgement string "&lt;OK&gt;" to the PLC to signal that the READ command has been properly completed.</p>
<p><b>5. Read Real Tim Clock From TLServer</b></p> <p><b>Format:</b></p> <pre>&lt;READ RTC[]&gt; &lt;/&gt; &lt;READ Date[n]&gt; &lt;/&gt; where n = 1,2,3 or 4.</pre> <pre>&lt;READ Time[m]&gt; &lt;/&gt;</pre> <p>where m=1,2,3.</p>	<p>This service allows the PLC to get the Real Time Clock data of the TLServer (i.e. the PC in which the TLServer runs on). The type of data is indicated in the Date[n] and Time[n] parameter which correspond to the DATE[n] and TIME[n] system variables in TBASIC:</p> <p>i.e. Date[1] = year; Date[2]=month; Date[3]=day; Date[4]=DayofWeek;</p> <p>Time[1]=hour; Time[2]=minute; Time[3]=second.</p> <p>For full synchronization, use the &lt;READ RTC[]&gt; tag which returns the values of the Date[1], Date[2], Date[3], Date[4], Time[1], Time[2], Time[3] in 7 CR-terminated ASCII strings.</p> <p>Upon receiving this command the TLServer will immediately send the relevant clock/calendar data as CR-terminated ASCII string(s) to the PLC. Your PLC program can easily use the INPUT\$(1) command to read in the data and convert them into integers using the VAL command. Note that unlike the "READ file" service, the TLServer does not send "&lt;OK&gt;" string after performing the "READ RTC" service.</p>

## Chapter 4: Running The Internet TRiLOGI Client

### I. Running The Internet TRiLOGI Application

Basically there are 3 methods in which you can start the TRiLOGI application, as follow:

- 1) If the Internet TRiLOGI and JRE have been properly installed on your PC, you can just double-click on the short-cut "TRiLOGI 5.1 Application" in the "Start Menu" to start the TRiLOGI application.



- 2) You can also open My Computer and open the folder: C:\TRiLOGI\TL5\, then double click on the file "TL51.jar" to start TRiLOGI application. If JRE has been properly installed the TL50.jar file will be recognized by Windows to represent executable Java program and it will run immediately. (Note: In the same folder you will also find the file "TLServer20.jar" which is the actual jar file for TLServer).
- 3) The third alternative is to run the program from DOS command line: First, run the MS-DOS prompt and then navigate to the directory "C:\TRiLOGI\TL5". At the directory, enter the following command line:

```
C: \TRi LOGI \TL5> j ava -j ar TL50. j ar
```

This procedure is actually encapsulated by the "TL5.BAT" batch file located at the " C: \TRi LOGI \TL5" folder that you can double-click from the same folder to execute. This method of starting TRiLOGI application has an advantage in that it opens the Java Console window, which can be useful because system errors and exceptions are normally reported via the Java Console. This can give a clue to reason of failure. (You can also start TLServer by running the "tlserver.bat" file.)

## H E L P !!!

When running TRiLOGI, you can get on-line help any time by pressing the <F1>. A Help window will open to show you the typical key/mouse actions. You can also click on the <More Help> button to get context-sensitive help loaded into your web-browser. It is assumed that you have Internet Explorer installed in the following directory:

C: \Program Files\Internet Explorer\IEXPLORE.EXE

However, If your PC does not come with this browser installed, then TRiLOGI Application will report problem opening the web-browser. If that is the case you'll need to use the "Notepad" program to manually edit the "config.tl5" file in the "C: \TRi LOGI \TL5\" directory. Modify the first line:

Browser Path=C: /Program Files/Internet Explorer/IEXPLORE.EXE

to match the correct browser path info. This problem does not occur to the applet since the TL5Applet automatically uses the same browser in whom it was loaded to open the help files, hence the applet does not need to know the browser path.

---

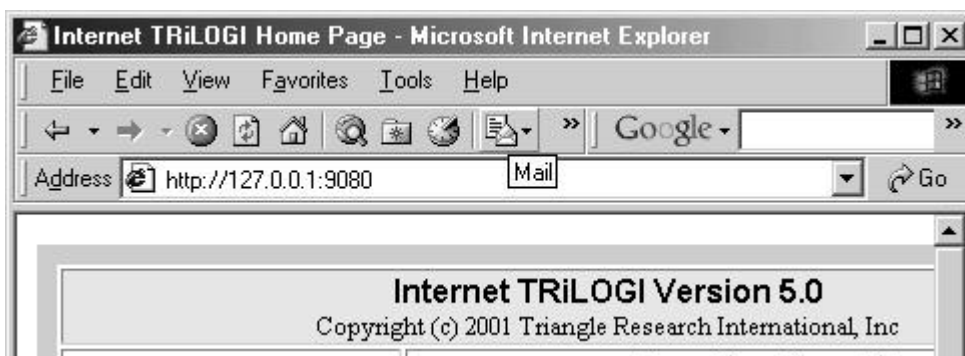
## II. Running TRiLOGI Applet Using Web Browser

---

- 1) Before you could run the TRiLOGI Applet in a web browser, make sure that the TLServer is already running.
- 2) Next, start up your Internet Browser. It should be either an Internet Explorer version 5.0 or later, or Netscape Navigator/Communicator Version 4.5 or later. Earlier versions of browser have some bugs in their JVM implementation and hence may not work well with TRiLOGI.
- 3) Next, check the TLServer front panel for its IP Address. If you are running TLServer on a PC without network connection it will probably show: IP Address = 127.0.0.1:9080. If you have an Internet connection before you start up TLServer, then you will see the Internet IP address of your PC. If your PC has both a local area network connection as well as a direct Internet connection, you will see two IP addresses being reported. (Although localhost address 127.0.0.1 may not be reported but **it is always there** as long as both the Client and the Server reside in the same computer. **Always use the localhost IP address 127.0.0.1** if both the Client and the TLServer are running on the same computer.)

- 4) Now, simply key in the IP Address, including the port number in your browser's "Address" (for IE5) or "URL" (for Netscape) text entry area. For localhost connection, enter:

**http://127.0.0.1:9080**



- 5) The browser will now issue a HTTP request to the TLServer. Since no filename has been specified, the default file in the web-server root directory "index.html" is loaded. This HTML file is written in Javascript to provide some other options. To start the TRiLOGI as an applet, select the appropriate option and the TL5Applet.jar file will be loaded from the TLServer into your browser for execution.

**Note:** The TLServer's root directory is not the same as the PC's root directory. In TLServer, the root directory is actually at "**C:\TRi LOGI\TL5\publ i c\**". This is the directory where the index.html and TL5Applet.jar file are stored and these files are served to the web browser when you enter the TLServer's IP Address as mentioned above. Visitors have no access to the PC's file directory above the server's root directory so the content of your other PC files will not be at risk of being exposed to visitors to TLServer.

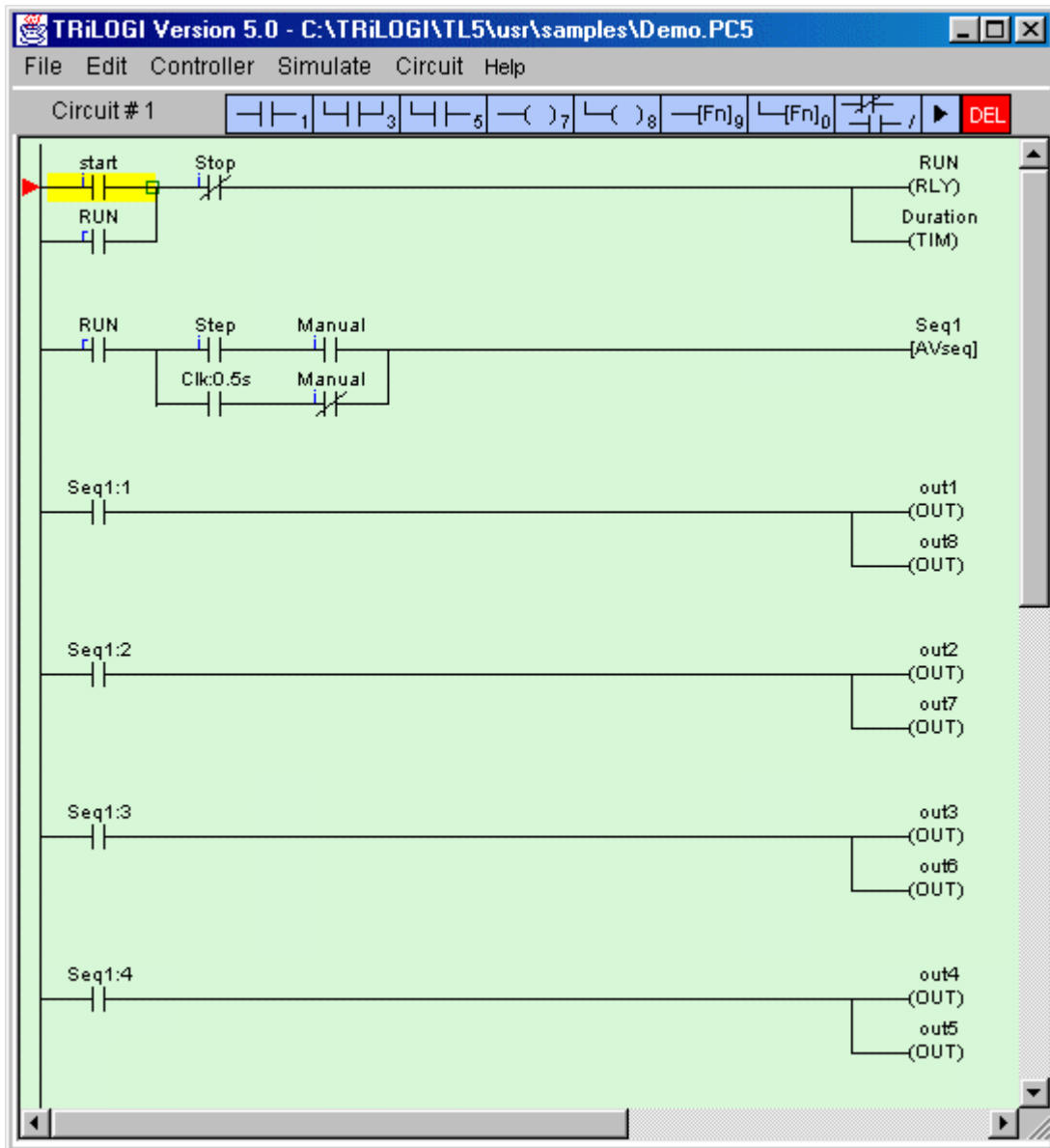
### **Disabling TRiLOGI Applet**

If you want to prevent visitors to TLServer from loading TRiLOGI Applet at all, just remove the "TLApplet.jar" file from the directory: "**C:\TRi LOGI\TL5\publ i c\**". In that case you can only access the PLC using the TL5 application program.

# Chapter 5: Ladder Logic Programming Tutorial

## I. Your Assignment: Creating Your First Ladder Logic Program

In this tutorial, we would like to create a simple program as shown below:



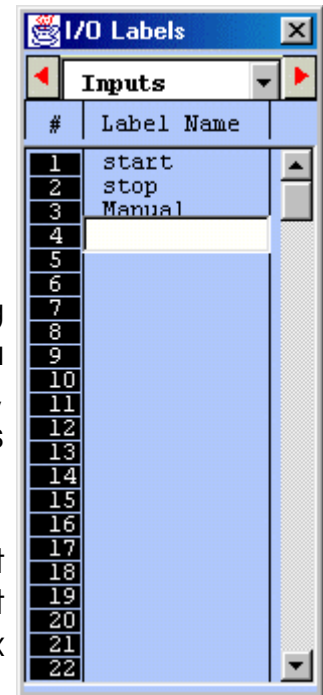
Simply follow the steps below to create your first ladder logic circuit.

1. Open pull-down "File" menu and select "New".
2. You should now be in the "Browse" mode of the logic editor. The vertical line on the left end of the screen is the "power" line. The cursor is at the position where you can key in your very first ladder logic.

- Before we commence the circuit creation, let us define the I/Os to be used for this program. The following I/Os are required:

Inputs : Start, Stop, Manual, Step  
Outputs : Out1, Out2,.... Out8  
Relays : Run  
Timers : Duration  
Sequencer : Seq1

- Open up the I/O label editing Window by pressing <F2>. (Although you can also click on "Edit" menu and select the item "I/O Table" to achieve the same, we strongly recommend learning the hot key F2 as it is often much more convenient to use).
- Scroll to the "Inputs" window by using the left/right cursor keys or by clicking on the red color left/right arrow buttons or simply select it from the choice box between the left/right arrow buttons.



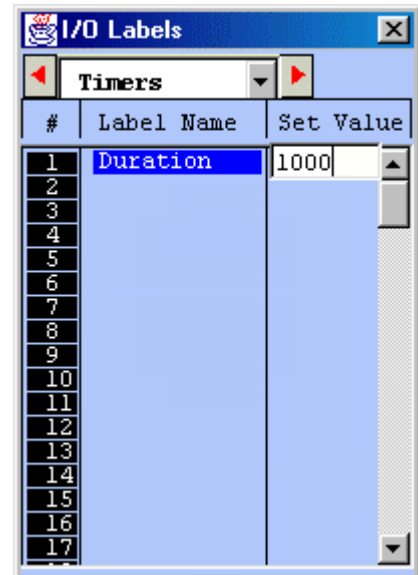
- Move the deep blue color highlight bar to Input #1 position by clicking on it. Click again to open up a text field for entering the name for Input #1. Enter the name "Start" for Input #1. Press <Enter> key to accept the name. The text field will be closed and the name "Start" is now assigned to Input #1. If you made a mistake, simply press the "spacebar" or click on the input location again to edit it.
- Press <Enter> key again and the highlight bar will be moved to Input #2.
- Without using the mouse button, simply start typing the name "Stop" at Input #2. The text field will be automatically opened up at Input #2 for entry. Press <Enter> after typing in the name for "Stop" input.
- Complete entry of the other two input label names "Manual" and "Step" as above. Note that if you enter more than 10 characters in the text field, only the first 10 characters are accepted. Also, white spaces between names are not acceptable and will be automatically converted to the underscore character ( '\_ ' ). e.g. If you enter the name: "M series PLC" for an I/O, it will be accepted as "M\_series\_P".
- After entering label names for Inputs #1 to #4, move to the "Output" table by pressing the right cursor key or by clicking on the right arrow button. Enter all the output and relay label names in their respective I/O tables. We will discuss the "Timer" table in the next step.

## Important Notes

- a) You can shift the Items in the I/O table up or down or insert a new label between two adjacent, pre-defined labels. Simply press the <Ins> key or Right-Click the mouse button to pop up the "Shift I/O" menu that allows you to shift the selected I/O. However, please note that if you shift the I/O down, the last entry in the I/O table (e.g. Input #256) will be lost.
- b) TRiLOGI Version 5 allows I/O label names of up to 10 characters. However, if you wish to keep compatibility with DOS TRiLOGI Version 4.x you should use no more than 8 characters to define the I/O names.

11. Timer table has an extra column "Set Value" located to the right of the "Label Name" column.

12. After you have entered the label name "Duration" for Timer #1, a text entry box is opened up at the "Set Value" location of Timer #1 for you to enter the SV for the timer. SV range is between 0 and 9999. Enter the value 1000 at this location.



13. For a normal timer with 0.1s time base, the value 1000 represents 100.0 seconds, which means that the "Duration" timer will time-out after 100.0 seconds. If the timer had been configured as "High Speed Timer" using the TBASIC "HSTimer" command, then the time-base would become 0.01s, meaning the value 1000 represents only 10.00 seconds.

14. We are now left to define the sequencer, "Seq1". The sequencer is an extremely useful device for implementing sequencing logic found in many automated equipment. TRiLOGI supports 8 sequencers of 32 steps each. Each sequencer requires a "Step counter" to keep track of the current step sequence.

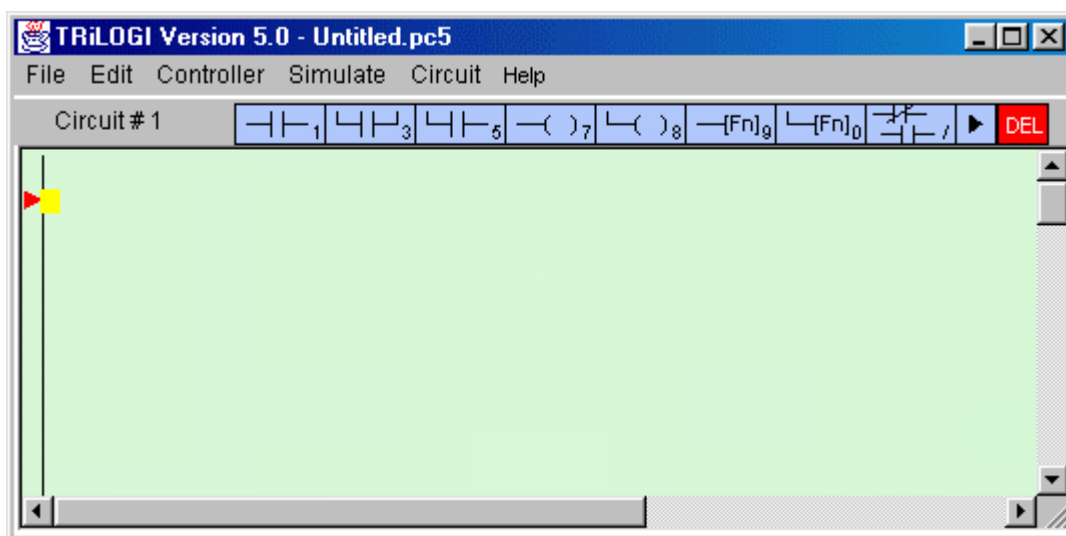
The first 8 counters in the counter table also double as the step counters for the 8 sequencers. These sequencers must be named "Seq1" to "Seq8" if they are to be used, i.e. Counter #1 to be named as "Seq1", Counter #2 as "Seq2", etc. However, any counter not used as sequencer may

assume any other name (up to a maximum of 10 characters) if they are used as ordinary counters.











15. If you are at the "Timers" table, pressing the right cursor key again will bring up the "Counters" table. Enter the name: "Seq1" at the label column for Counter #1. Press <Enter> and the text entry field will be opened at the "Set Value" column. For now, let's enter a preset value of "4" for "Seq1".
16. We have now completed defining the I/Os, timers and counters. Press the <ESC> key to close the counter or other tables. Note that not all labels need to be defined before programming. You may create the label names any timer during circuit creation by pressing hotkeys <F2>.
17. We are ready to create Circuit #1 as shown below:




18. With the circuit pointer (red color triangle) at Circuit #1, press the <Spacebar> to enter the "Ladder Edit" mode. You can also enter the circuit edit mode by double clicking at Circuit #1.




Once you enter the "Ladder Edit" mode, a row of ladder icons appear along the top of the main TRiLOGI window just below the pull down menu. The following is a description of each item. A yellow color highlight bar, which you can move to select an element in the ladder circuit, will appear.

	<1> - Left click to insert a normally-open series contact. <2> - Right click to insert a normally-closed series contact.
	<3> - Left click to insert a N.O. parallel contact to highlighted element <4> - Right click to insert a N.C. parallel contact to highlighted element
	<5> - Left click to insert a N.O. parallel contact to enclose one or more elements. <6> - Right click to insert a N.C. parallel contact to enclose one or more elements.
	<7> - Insert a normal coil which may be an output, relay, timer or counter.
	<8> - Insert a parallel output coil (not an entire branch) to the current coil.
	<9> - Insert a special function coil which includes execution of CusFn
	<0> - Insert a parallel special function coil to the current coil.
	</> - Invert the element from N.O. to N.C. or from N.C. to N.O.
	Click to move the highlight bar to the right (same effect as pressing the right arrow key). This can be used to move cursor to a junction that cannot be selected by mouse click.
	<b>Double-click</b> to delete a highlighted element.

19. Now insert the first element by left clicking on the  icon. The icon will change to a bright yellow color to show you the element type that you are creating. At the same time, an I/O table should appear on the screen with a light beige-color background instead of the normal light blue background. The I/O table now acts like a pop-up menu for you to pick any of the pre-defined label names for this contact.
20. The contents in the table are not normally meant to be edited at this moment. Scroll to the "Input" table and click on the label name "Start" and a normally open contact will be created at Circuit #1.



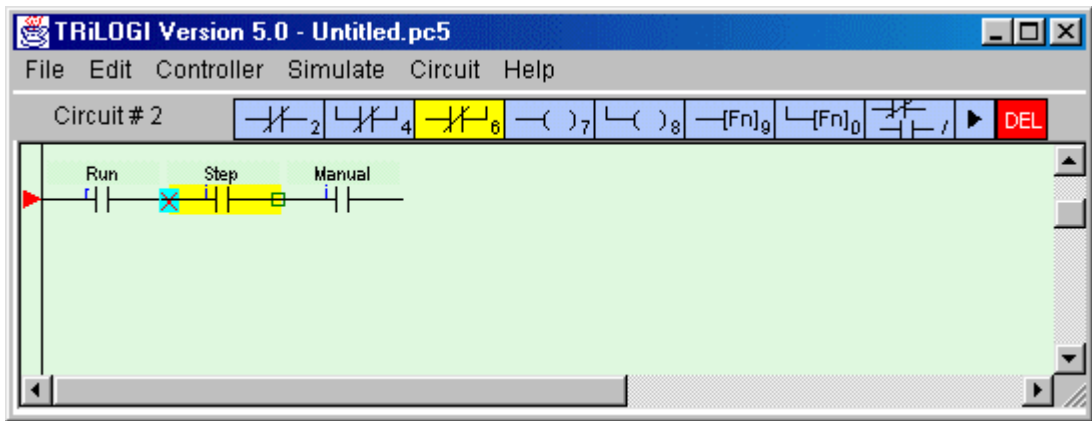
25. Notice that the coil symbol ---(RLY) indicates that this is a relay coil, which is helpful in identifying the function of the coil. TRiLOGI automatically places the coil at the extreme right end of the screen and completes the connection with an extended wire.
26. Right below the relay coil is a parallel timer coil with label name "Duration". To create this coil, click on the  icon. This allows you to connect a parallel coil to the existing coil. The "I/O" table will pop up for selection again. Since we want to choose a timer, scroll to the "Timer" table and pick the first timer with the label "Duration" to complete the circuit.
27. **Press the <Enter> key once to complete Circuit #1**


**Congratulation! You have just successfully created you very own ladder logic circuit. It is that simple!**

28. We will now create Circuit #2 as shown below.






29. Follow the steps #20 to #23 to create the following circuit fragment:

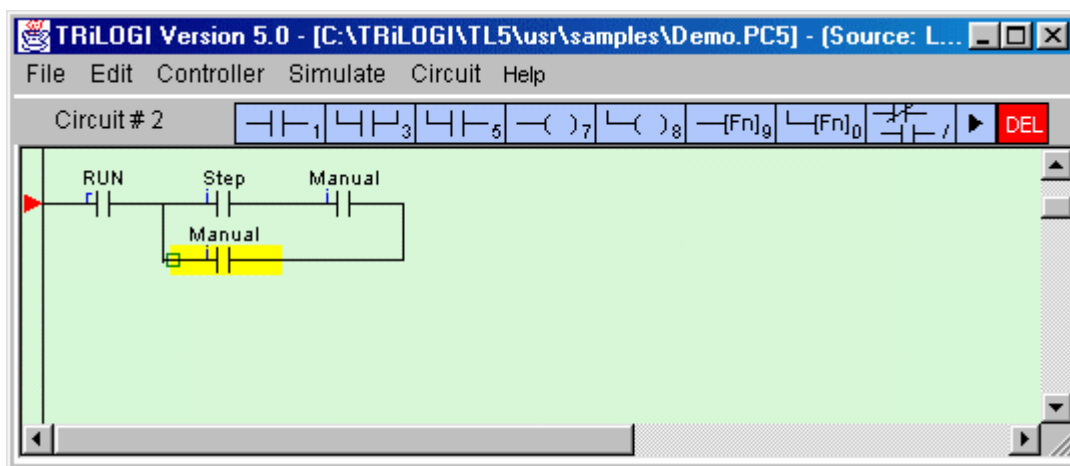



30. We want to enclose the two series contacts "Step" and Manual" with a parallel branch that contains two elements. First, we will create the branch for the N.C. "Manual" contact.
31. Click on the element "Step" to highlight it. Then right-click on the  icon to create a N.C. parallel circuit that encloses both the "Step" and the "Manual" contacts. A cross will appear at the left hand end of the "Step" contact, indicating that this is the starting location of the parallel circuit. You should now click on the "Manual" contact to select the ending

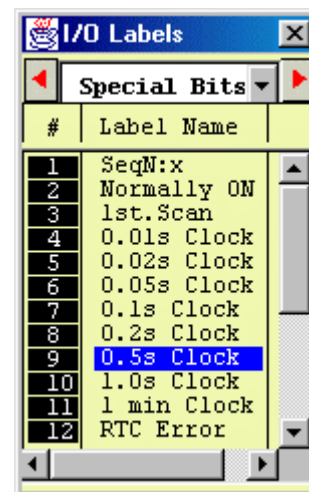
location for the parallel circuit. The yellow highlight bar will be positioned at "Manual" contact now.

32. You will notice that the  icon has now changed into a yellow color N.C. contact  with an opposite connection arm. You should now click on the  symbol to close the parallel branch. (One possible short-cut method is to double-click at the ending location to close the branch).

As usual an I/O table will be opened for you to select the I/O. For now, select the "Manual" label from the "input" table to create the following circuit:

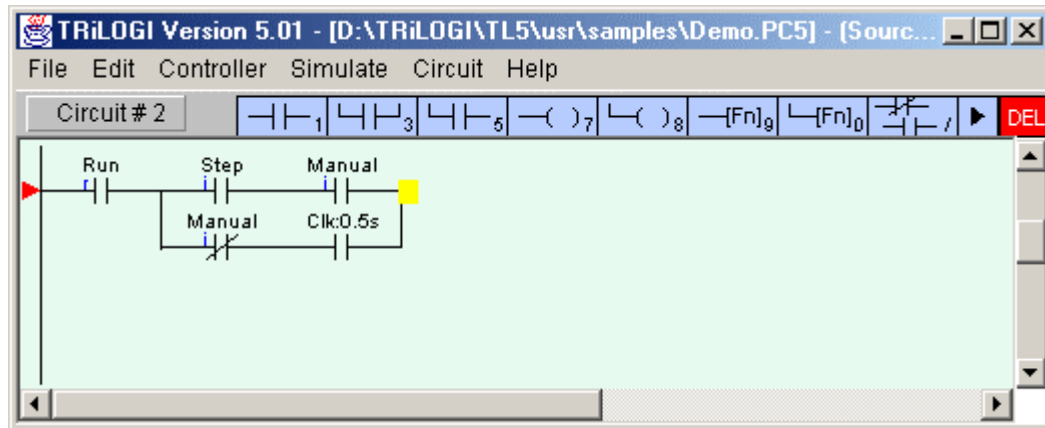



33. Next, we want to insert the special bit "Clk:0.5s" to the left of the "Manual" contact. Press the <TAB> key to move the insertion point to the left end of the highlight bar as shown above. Then left-click on the  icon to create a normally open contact. Scroll the I/O table to the "Special Bits" table and select the item: "0.5s Clock". The parallel branch would have been completed by now.

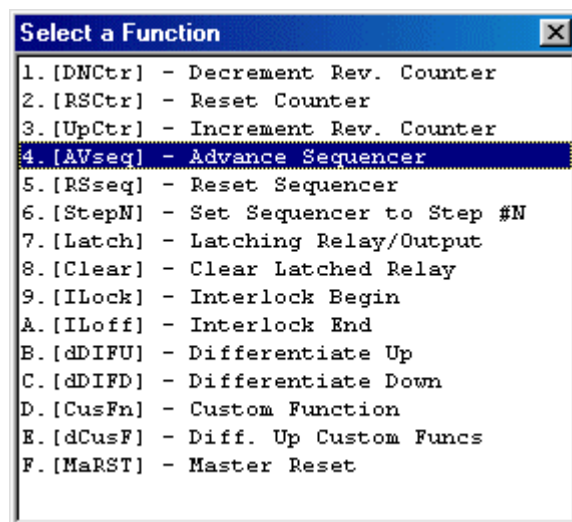


34. **Note:** The "Special Bit" table comprises some clock pulses and some other special purpose bits. These include the eight built-in clock pulses in the system with periods ranging from 0.01s to 1 minute. Built-in clock pulses are useful if you need a time base to create, for example a "flashing light". A contact such as "Clk:0.1s" will automatically turn itself ON for 0.05s and then OFF for another 0.05s and then ON again, resulting in a series of clock pulses of period = 0.1 second.

35. Next, move the highlight bar to the right end junction of the parallel circuits as follow:



36. Now, click on the  icon to insert a special function coil. A popup menu will appear for you to select the desired special function. Click on the item "4.[AVseq]-Advance Sequencer" to insert the Advance Sequencer function [AVseq].




37. This function is one that will increment the step counter of Sequencer #1 each time its execution condition goes from OFF to ON.

**Again, remember to press the <Enter> key to complete Circuit #2**

38. Circuits #3 to #6 are similar to one another. They make use of the Sequencer to turn on the Outputs 1 to 8 to create a pattern of "running lights" when executed. The label "Seq1:1" of the contact in Circuit #3 represents Step #1 of Sequencer 1. Remember that each sequencer can have up to 32 steps (Step #0 to 31), with each step individually accessible as a contact. A normally-open contact "Seq1:1" will be closed whenever the step counter of Sequencer 1 reaches number 1. Likewise a normally-closed contact "Seq5:20" will be opened when the step counter of Sequencer 5 reaches number 20.

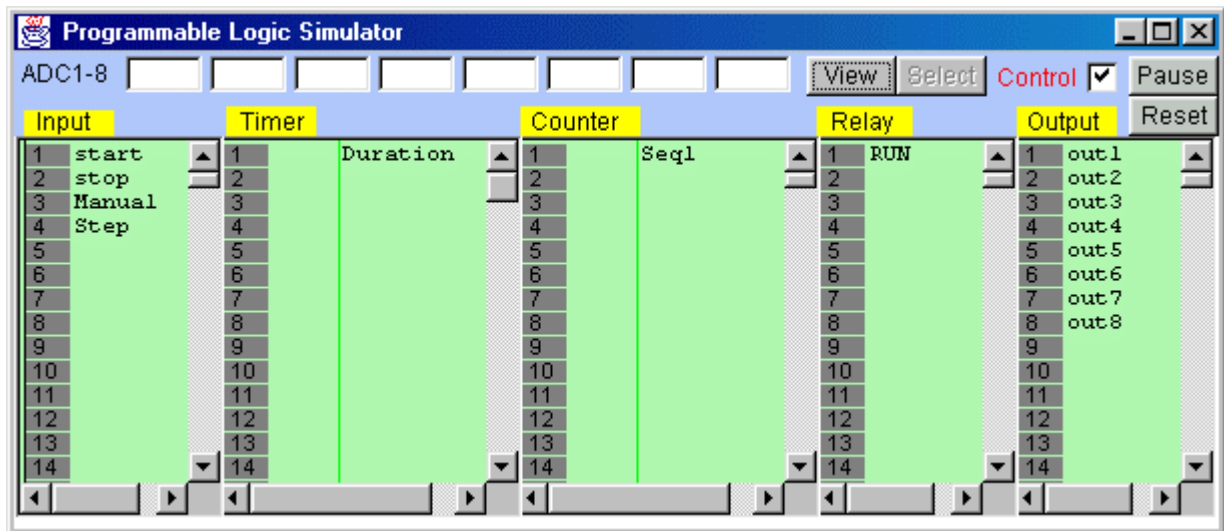


39. To create the normally-open contact "Seq1:1", left-click on the  icon. When the I/O table pops up, scroll to the "Special Bit" table and select the item #1 "SeqN:x". When prompted to select a sequencer choose "Sequencer 1" and another dialog box will open up for you to enter the specific step number for this sequencer.
40. We have thus far been creating ladder circuits only by clicking on the ladder icons. **A short-cut method** of choosing elements to be created without using the mouse does exist. Observe the icon carefully and you will notice a small numeral at the lower right hand corner of each icon that corresponds to the shortcut key. You may wish to try this short cut for the remaining part of Circuit #3. Press the <7> key and the Output table will immediately pop up for selection of a coil. Pick "Out1" from the "Output" table and the "Out1" coil will be connected.
41. Circuits #4, 5 and 6 are very similar to Circuit #3 and you should have little problem creating them. Complete these circuits and we are ready for some interesting simulation exercises. When you have created all the circuits, press <Enter> key or <ESC> key at the last blank circuit to end "Ladder Edit" mode.
42. We can make our program more comprehensive to other users by utilizing the "Comments" feature of TRILOGI. Open the "Circuit" menu and select "Insert Comment". A comment editor window will be opened up to allow you to add your comments to any part of the circuit. When you are done with your comments, just press <ESC> key or close the comment editor window and the comments you just entered will be inserted between the circuits. Each comment occupies a circuit position and there is no limit to the number of lines a comment circuit may have. (However, if you wish to keep data file compatibility with the old DOS TRILOGI Version 4.x you should limit the comment to no more than 4 lines per comment and each line should contain no more than 70 characters.)

A comment circuit may be moved around or deleted just like any other ladder circuits. If you wish to edit the comment, just double-click on it or press the <Spacebar> to open up the comment editor window. You can use the normal text editing keys such as left, right, up, down cursor keys, and <Ctrl-Left>, <Ctrl-Right>, <Del> and <Backspace> keys for editing the comment text.

## II. Testing Your Ladder Logic Program Using The Simulator

The stage has been set and the show is ready! Having completed the demo program, it is time to test if it works as intended using the built-in real-time programmable controller simulation engine. Open the "Simulate" pull-down menu and activate the command "Run (All I/O reset) - Ctrl+F9". TRiLOGI will immediately compile the ladder program and if no error is detected, it will instantly proceed to open up the "Programmable Logic Simulator" screen, as shown below:

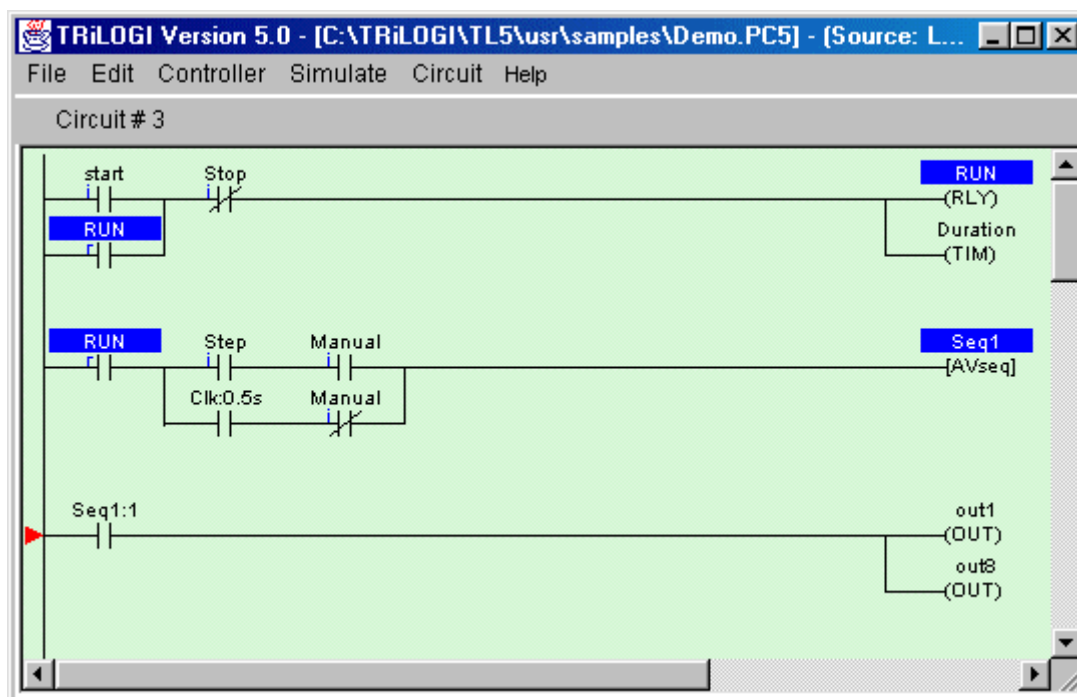


1. If you have followed closely all the instructions during the creation of the demo program, you should not encounter any compilation error. However, if you do receive an error message, then please check your circuit against the picture shown in the [assignment](#) page, then make all the necessary corrections and then try again.
2. The simulator screen comprises 5 columns: Input, Timer, Counter/ Sequencer, Relay, and Output. With the exception of the Relay table that contains up to 512 elements, and the Timer table that contains up to 128 timers, all other columns contain 256 elements each. Every column has its own vertical scroll bar. You can use the mouse to scroll each column independently to locate the desired I/O.
3. The label names for the inputs, outputs, relays, timers and counters defined earlier in the I/O tables automatically appear in their respective columns. To the left of each label name column is an "LED" lamp column that indicates the ON/OFF state of respective I/O. A red color lamp represents the ON state of an I/O, whereas a dark gray color lamp represents an OFF state. The I/O number is indicated in the middle of the lamp.

- The simulator require the use of the mouse to work properly so it is important to remember the mouse button actions as follow:

Left Mouse Button	Turn ON the I/O when pressed. Turn OFF when button is released.
Right Mouse Button	Toggle the I/O when pressed once. (i.e. OFF becomes ON and ON become OFF)

- Our ladder program requires us to "push" the "Start" button momentarily. You can simulate this action by moving the mouse pointer to the "Start" label (or the LED lamp) and press the **LEFT** mouse button once and then release the button. The action starts!
- At this time, notice that the relay "RUN" is latched ON and the timer "Duration" begins to count down from the value of 1000 every 0.1sec, and the Output #1-#8 are turning ON/OFF sequentially in a "running light" pattern. Sequencer "Seq1" in the "Ctr/Seq" column begins to count upward from 1 to 3 and then overflows to 0 and repeats continuously. For each step of the Sequencer, the corresponding Output will be turned ON. Our demo program will show a running light pattern starting from Outputs 1 & 8, then 2 & 7, 3 & 6 and 4 & 5 and then back to 1 & 8, 2 & 7.....
- Now you should verify that the logic works as intended by observing the ladder diagram. You should notice that the "Run" labels in all circuits are highlighted as shown below:



8. The logic states of any I/O can be displayed on the ladder diagram directly. An Input, Output, Relay, Timer or Counter contact that is turned ON will have its label name highlighted in the ladder diagram. This feature helps greatly in debugging and understanding the logical relationship between each I/O. For example, from the above figure, we can see clearly that the "Self-latching" circuit for relay "Run" works as intended: when we first turn ON the "Start" input, "Run" will be energized and its contact which is parallel to "Start" will hold itself in the ON state, even if we subsequently turn OFF the "Start" input by releasing the button.
9. The timer coil "Duration", being connected in parallel to "Run" relay, will also be energized. However, its contact will only be closed after 100 seconds (when its present value counted to 0). To break the latched On "Run" relay, we must energize the "Stop" input momentarily to break the "power" flow. Try it now.
10. Let's restart the system by turning ON the "Start" input momentarily again. Next, we want to turn ON the "Manual" input. Move the mouse pointer to the "Manual" input and then press the **right mouse button**. "Manual" input will be "stuck at "ON" state even after you have released the right mouse button. Click on "Manual" button using the right mouse button again and it will be turned to OFF.
11. When "Manual" input has been turned ON, the running lights should stop. This is because the normally closed contact of the "Manual" input in Circuit #2 is now turned OFF and the 0.5s clock pulse could not trigger the [AVseq] function anymore.
12. If you now left-click on the "Step" input, the running lights will move one step at a time in response to your mouse click. Observe the Seq1:x contact with respect to the counter value of Seq1 and the logic of this circuit become very clear instantly.
13. Observe that the timer "Duration" continues to count down every 0.1 second, and when it reaches 0, the "Duration" output coil label will be highlighted. You can use this timer to stop the program by connecting a N.C. "Duration" contact to Circuit #1. This is left as an exercise for you!

---

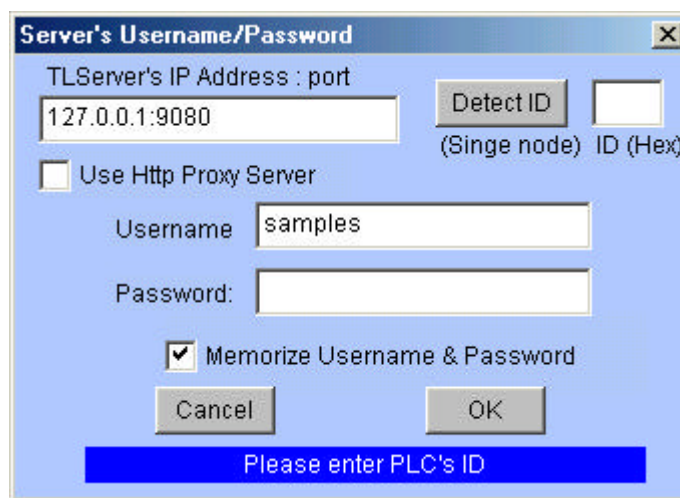
### III. Transferring Your First Ladder Program To The PLC

---

After having tested your ladder logic program on the simulator, you are probably eager to try out the program on the actual PLC! Here are the quick steps:

1. Connect the DB9 programming cable from the PC COM port to the PLC's COMM1 port.
2. Connect power to the PLC

3. Run the "TLServer" as described in Chapter 3. TLServer must be running before you can have any kind of communications with the PLC.
4. Click on the "Setup Serial Port" button and test the communication with the PLC by entering the "IR\*" string into the "Command" box. You should receive a "IR01\*" response from the PLC (provided the default ID=01 has not been changed). If you face any communication problems with the PLC then you will have to troubleshoot it first by making sure that (a) the correct COM port is used, and (b) no other program (e.g. a PDA cradle) is currently controlling the COM port that is used with the PLC.
5. Next, click on the "Controller" menu and select "Program Transfer to PLC". You will be asked to login to the TLServer. If no one has changed the default info you should see the following screen:



6. The default IP address:port is 127.0.0.1:9080. This is the localhost IP address and should be the one used when both TLServer and TRiLOGI client are running on the same PC. There is a default user defined in TLServer with the username "samples" and no password which is what we will use for now.
7. Next, click on the "Detect ID" button. If all go well the ID '01' will appear in the box next to the "Detect ID" button. Otherwise you will receive an error message that explains what did not go right.
8. If you have received an ID correctly, you can then click the "OK" button. TRiLOGI will compile the program and then begin transferring the compiled codes into the PLC. Just follow the steps on the screen until the entire program transfer procedure is completed. Then click on the "Yes" button when you are asked if you wish to "Reset all I/Os?".
9. Before you actually turn on the physical inputs to the PLC to test the program, we would like to show you how you can actually control the PLC's I/O from TRiLOGI software. First, click on "Controller" menu and select "On-Line Monitoring". You should see the "Full Screen Monitoring" window

that looks identical to the “Programmable Logic Simulator” screen that you have seen earlier while testing your program on the simulator.

10. The logic states of the inputs, outputs, relays, timers and counters that you see on the “Full Screen Monitoring” screen are linked to the PLC’s actual inputs, outputs, relays, timers and counters. When performing on-line monitoring, TRiLOGI software continuously sends out serial communication commands to retrieve the data from the PLC and display them on the screen.
11. Next, ensure that the “Control” check box on the monitoring screen is checked (selected). You will now be able to remotely trigger any of the PLC’s input by clicking on its label name. Try to click on the “Start” input once and you should see the running lights on the PLC outputs. Click on the “Stop” input and the running light should pause. The running light will also be reflected on the “Output” column of the “Full Screen Monitoring” window.
12. Please note that when you click on an input label, TRiLOGI actually only manages to change the input bit for only one scan time. Thereafter the PLC will update the “Input” bit using the actual logic states of the physical input.
13. Next, pause the PLC by pressing the “Pause” button. You should see that the “Pause” light on the PLC being turned ON. Now, you can actually turn ON the physical outputs or internal relays of the PLC by clicking on any output label using the LEFT mouse button. Releasing the LEFT mouse button on the output will turn it OFF. You can even latch ON or OFF an output by clicking on the label using the RIGHT mouse button. Try it – it can be fun!

## **Summary**

We have completed this tutorial and have successfully created a simple ladder program. We have also performed real time simulation to test the program's functionality and have transferred the demo program into the PLC via the TLServer. By now you would probably have a good appreciation of TRiLOGI's superb capability and ease of use and are ready to include TRiLOGI as an integral part of your programming needs!

## Chapter 6: TRiLOGI Ladder Logic Editor Reference

TRiLOGI's ladder logic editor window lies between the main menu bar along the top of the screen and the help message line along the bottom of the screen. The cursor will appear in the window whenever you are in the logic editor. The ladder logic editor comprises two modes: the **Browse** mode and the **Circuit Editing** mode. We shall explain the operation of both modes

---

### I. The Browse Mode

---

You are normally in the browser mode when you start up the program. The browse mode allows you to manipulate the whole ladder logic circuit as a single entity: you can view any circuit, make copies of it, move it to another location or delete it entirely. Each complete ladder logic "circuit" is given a circuit number. You should see a small red color marker showing you the currently selected circuit. The circuit number of the selected circuit is shown on the upper status line as "Circuit # xxx".

#### 1. Mouse Actions

Since TRiLOGI Version 5.x runs under windowing environment, all usual mouse action applies. You can grab the vertical scroll bar to scroll to your desired circuit and click on it to select it. Double click on a circuit enters the Circuit-Editing Mode, which will be described later.

#### 2. Keyboard Actions

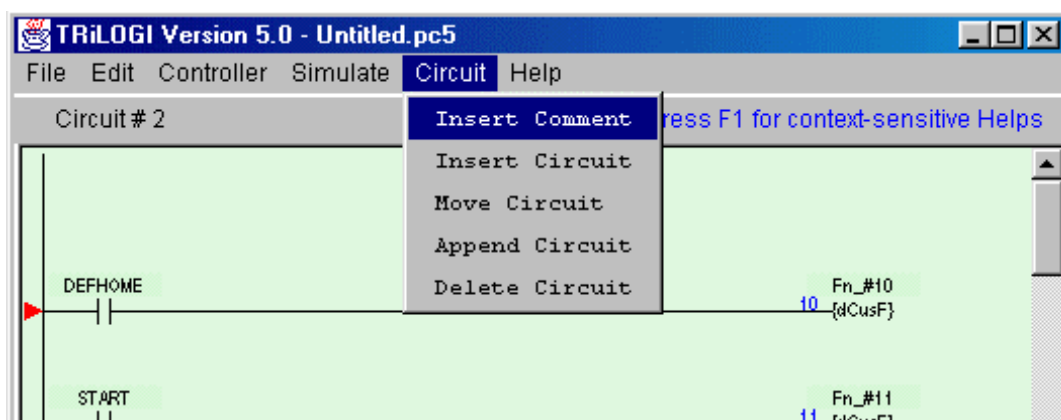
The functions of various keys in the browse mode are explained below:

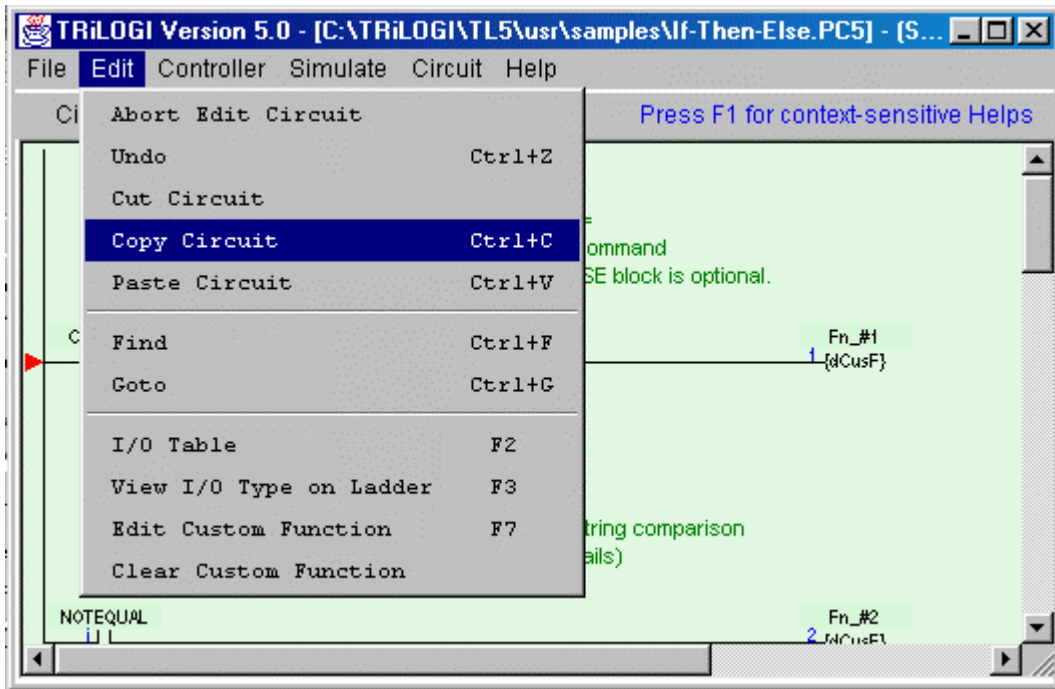
<Spacebar>	Allows you to enter circuit editing mode for the currently selected circuit. If the selected circuit is a comment circuit, the comment editor will be opened automatically.
<F1>	Activates the help function to display on-line help.
<F2>	Opens the I/O Table to create the I/O Label Name
<F3>	Turns ON/OFF display of the I/O type for ladder logic contacts on the screen. All ladder logic contact symbols are normally identified by their label names. However, you can also display an optional small literal to indicate the I/O types. e.g. i=input, o=output, r=relay, t= timer and c=counter.

<F5>	Refreshes the display. If for some reason the screen is garbled by incomplete circuit display, you can just press the <F5> key to redraw the screen.
<F7>	Opens any custom function. If the currently selected circuit contains a custom function, then it will be opened for editing. Otherwise TRiLOGI will ask you to select a custom function # from a menu.
<F8>	Compiles the TRiLOGI program to show the compilation statistics.
<F9>	Runs the simulator without resetting any I/O
<Ctrl-F9>	Resets all I/Os and then runs the simulator.
<Ctrl-F8>	Resets all I/Os except inputs and then runs the simulator.
<Up>/<Dn> <PgUp> <PgDn>	Use the up/down cursor keys to move the marker to other circuits and the "Circuit #" display at the upper status line will simultaneously reflect the change. If you attempt to venture beyond the screen, the logic editor screen will scroll. The <PgUp> and <PgDn> keys can be used to scroll one page at a time.

### 3. Using the Circuit & Edit Menu

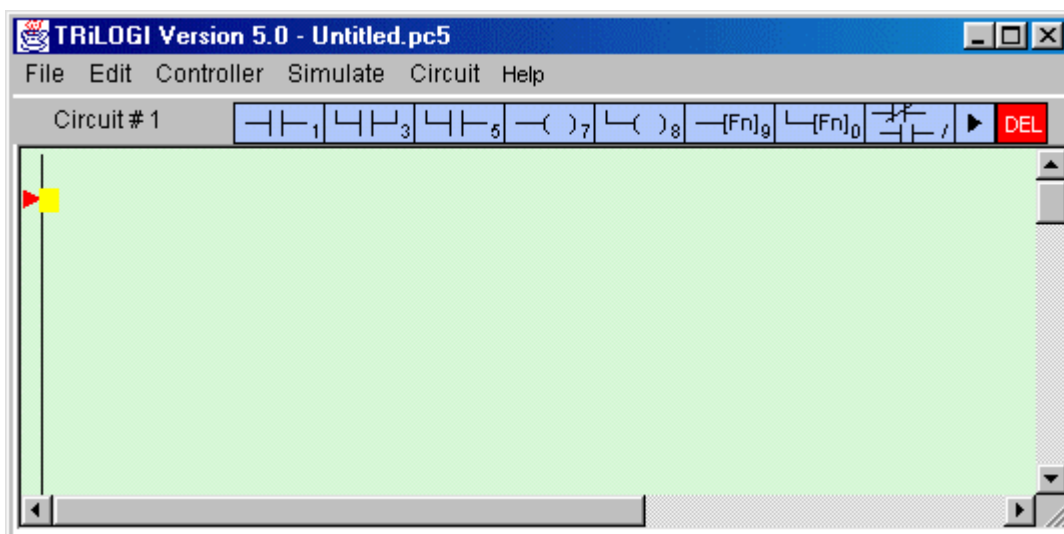
The "Circuit" and Edit menus contain various commands that you may need for adding comments, copying or delete circuit as well as and for re-arranging the order of the ladder circuits. Please refer to Chapter 7 for descriptions of the actions associated with each item in this menu.





## II. The Circuit-Editing Mode

TRiLOGI comes with a smart editor that allows you to insert or delete a single element within a circuit easily. The editor interprets your circuit immediately upon entry and prevents you from creating illegal circuit connections. The functions of various keys in the circuit-editing mode are detailed below. You know that you are in the circuit editing mode when a row of ladder logic icons appears along the upper status line next to the circuit number and a yellow color highlight bar appears and you can move it to select an element in the ladder circuit, as shown below:

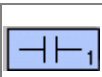
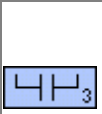


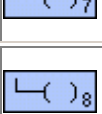
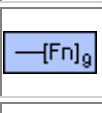
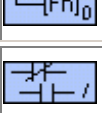






## 1. Mouse Actions

**Left Click** - When you click on an element using the left mouse button, the element is selected and highlighted by the yellow color highlight bar.

**Right Click** - When you click on an element using the right mouse button, you are allowed to directly edit the label name of the element. This can be a convenient feature if you need to change one or two characters in the name only. However, if the element is a custom function [dCusFn], or [CusFn], then the custom function editor will be opened for you to edit the function directly.

**Insert Ladder Element** - You create the ladder circuit element simply by moving the mouse pointer to the icon and pressing either the left or the right mouse button to insert a ladder logic element to the currently highlighted element. The following is a description of the functions of each icon. A yellow color highlight bar will appear which you can move to select an element in the ladder circuit.

	<1> - Left click to insert a normally open series contact. <2> - Right click to insert a normally closed series contact.
	<3> - Left click to insert a N.O. parallel contact to highlighted element <4> - Right click to insert a N.C. parallel contact to highlighted element
	<5> - Left click to insert a N.O. parallel contact to enclose one or more elements. <6> - Right click to insert a N.C. parallel contact to enclose one or more elements.
	<7> - Insert a normal coil, which may be an output, relay, timer or counter.
	<8> - Insert a parallel output coil (not an entire branch) to the current coil.
	<9> - Insert a special function coil which includes execution of CusFn
	<0> - Insert a parallel special function coil to the current coil.
	</> - Invert the element from N.O. to N.C. or from N.C. to N.O.
	Click to move the highlight bar to the right (same effect as pressing the right arrow key). This can be used to move cursor to a junction that cannot be selected by mouse click.
	<b>Double-click</b> to delete a highlighted element.

When you click on an icon, for example, the , the icon will change to bright yellow color to show you the element type that you are creating. At the same time, an I/O table should appear on the screen with a light beige-color background. The I/O table acts like a pop-up menu for you to pick any of the pre-defined label names for this contact. This saves you a lot of typing and at the same time eliminates typo errors that could result in a compilation failure. You should spend a few minutes to follow the "Chapter 5: Ladder Logic Programming Tutorial" on the steps needed to create a ladder program.

As mentioned previously, the ladder editor is intelligent and will only accept an action that can result in the creation of a correct ladder element. Otherwise it will simply beep and ignore the command.

### **UNDO Circuit Editing**

If you have wrongly inserted or deleted an element and wish to undo the mistake, you can either select "Undo" from the "Edit" menu or press <Ctrl-Z> key to undo the last step. The undo buffer stores the last 10 editing steps. You can also choose to abort all the operations on the current circuit by selecting "Abort Edit Circuit" to abort all changes made to the current circuit.

## **2. Create Ladder Circuit Using The Keyboard**

Users of existing TRiLOGI version 3.x or 4.x who are familiar with creating ladder programs using the keyboard will be delighted to know that they can still create their ladder programs using the keyboard. The keyboard actions are described below:

### **Left/Right/Up/Down cursor keys**

The cursor keys are for moving the highlight bar from one element to another in their four respective directions. You can only move in a direction that will end up with an element.

### **<ESC>**

Press <ESC> key to end the circuit-editing mode and return to the browse mode of the logic editor.

### **<Enter>**

When you are done with editing the current circuit, hit <Enter> to proceed to the next circuit.

### **<Tab>**

If you observe the highlight bar carefully, you will notice a dark green color square at the right end of the highlight. This indicates the insertion location where a series contact will be attached. You can change the insertion location to the left or the right of the highlight bar by pressing the <TAB> key.

The position of the cursor has no effect when you connect a parallel contact to the highlighted element. The left terminal of the element will always be connected to the left side of the parallel branch.

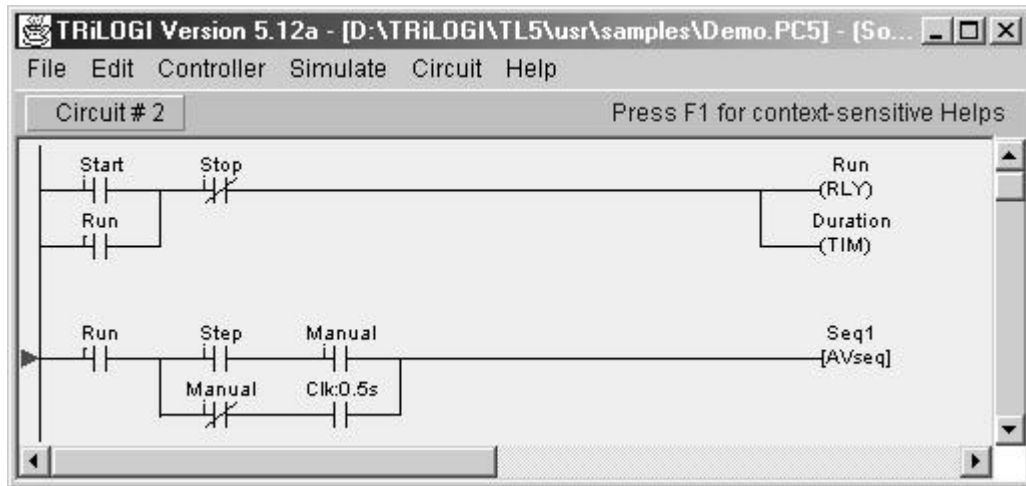
### **<0> to <9> , </> & <E> keys**

Pressing the key <0> to <9> and </> is equivalent to clicking on the icon shown in the table. The equivalent keyboard number is shown as a small numeral at the lower right corner of the icon. The </> key is the quickest way of converting a normally open contact to a normally closed one (and vice versa).

Pressing the <E> key when a contact or coil is selected allows you to edit the label name directly. Note that it is the user's responsibility to ensure that the label is valid.

## Chapter 7: TRiLOGI Main Menu Reference

Both TRiLOGI application and applet programs have nearly identical look and feel (as shown below), with the exception that the Applet can't save to or load from local drive and it does not support the "Printing" function.



The main body of the program window is for displaying and editing your ladder logic program. A ladder logic program is made up of many ladder "rungs". In TRiLOGI we call each ladder rung a "circuit" with an associated "circuit number". The currently selected circuit is marked by a little red triangle pointed to the circuit's intersection with the left vertical line (a.k.a. the "power rail" in ladder logic terminology)

The circuit number of the selected circuit is displayed on a button located just above the top left corner of the ladder editor window. If you happen to click on this button, a dialog box will popup that prompts you to enter the circuit number that you wish to go to and the editor will bring you there immediately.

---

### I. File Menu

---

The File menu provides commands for the opening/saving of TRiLOGI files either on the local hard disk or on the TLServer's storage space.

#### 1. **New** <Ctrl+N>

Activate this command when you want to create a new ladder logic program. All current ladder circuits and custom functions will be cleared from the screen and the default filename is "Untitled.pc5".

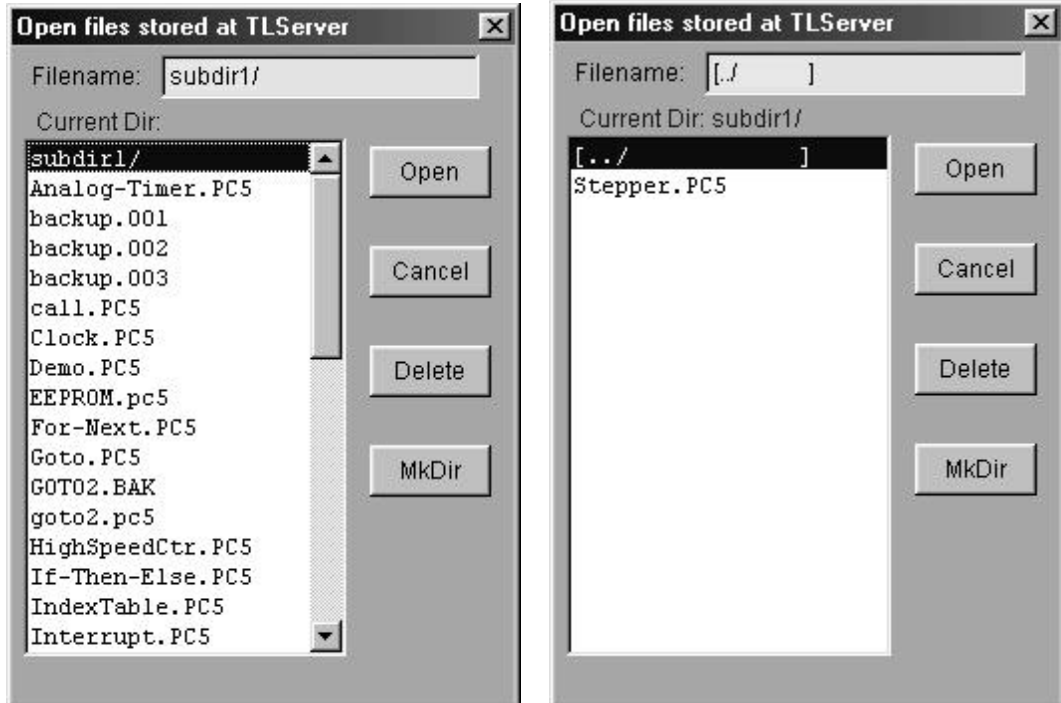
## 2. Save <Ctrl+S>

This command saves the whole ladder logic program, all I/O tables and all the custom functions to the disk. The current file will be saved to the same source from which it was opened from, i.e. If a file has been previously opened from the TLServer via the network, this command will save the file back to the TLServer. Likewise, a file opened from the local hard disk will be saved automatically to the local hard disk.

## 3. Open (TLServer) - <Ctrl+O>

This command is for loading a TRiLOGI file from the TLServer. When executed, you will be prompted to enter the Username and the Password to gain access to the TLServer. (The same Username and Password must have already been defined in TLServer for this to work). If you are running TRiLOGI as a local application instead of as an applet, you may be required to enter the "IP Address: port" of TLServer in order to connect to TLServer. (Note that last entry of IP Address: port is saved to the TL5 configuration file and will be loaded when the TL5 application re-starts).

Each user has his/her own exclusive directory for storing his/her TRiLOGI files. Once authenticated, a network file dialog will be opened for you to select a file, delete a file or create a subdirectory, as shown below:



Simply double-click on the desired file or select the file you wish to open and click the "Open" button to open the TRiLOGI file.

**Sub-directory:** The Mkdir button allows you to create a sub-directory on the server to organize your files. Subdirectory names always end with a "/" character. If you open a subdirectory its contents will be displayed in the file window. To return to the parent directory from a sub-directory, you simply double-click on the [../] symbol.

#### 4. **Save As (TLServer)**

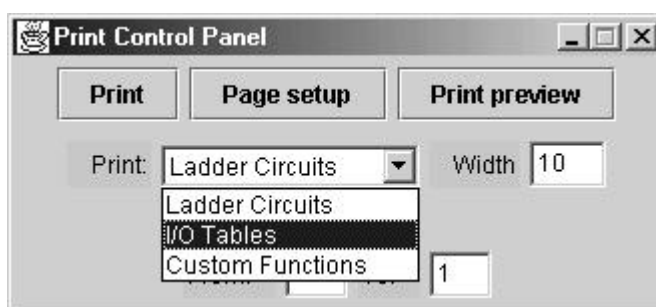
Use this command if you wish to save the currently edited TRiLOGI file to the TLServer using a different filename. You will be prompted to enter the Username/Password (and IP address if it is a TL5 application) to gain access to TLServer. Once authenticated, the network file dialog similar to that described in "Open (TLServer)" will be opened for you to enter a file name or select a filename to overwrite.

#### 5. **Open (Local Drive) / Save (Local Drive)**

For TL5 Application (not Applet) you can open or save a file from/to the local hard disk. You will be presented with the typical file dialog provided by your O/S. This command however is not available to the TL5 Applet since an applet does not have the right to access local hard disk resources.

#### 6. **Print**

For TL5 Application (not Applet), you may use all the printing resources supported by your O/S to print a selectable range of the ladder diagram, the I/O Tables or the custom functions. When executed the following "Print Control Panel" will appear:



To print, first select the item from the choice box and define the range you wish to print and then click on the "Print" button. For "Ladder Circuits", the range indicates the circuit numbers. For "I/O Tables", the range indicates the I/O number (up to 256) and for "Custom Functions", the range is the function number.

You can use the "Print preview" button to check the pagination of the printing on screen. You can select paper size and print orientation, etc. by clicking the "Page setup" button. Empty custom functions will be

automatically skipped to save paper. When you select to print the "Ladder Circuits" a special "Width" textbox appears. This textbox is for you to enter the maximum number of series element that can be printed on the paper width. Changing this number affects the scaling of the ladder diagram when printed. The smallest number is 5 and largest number is 13. Use a smaller number if you wish to have a larger printout. However, please note that if your ladder program contains circuits with more elements than that indicated by the "Width" parameter the "out-of-page" part of those ladder circuits will not be printed.

**Note:** The "Print" function requires the support of Java 2 JVM (which is provided by Java Runtime Environment version 1.3.1 when you installed it) but most browsers to-date do not yet not support Java 2 unless with a special Java plug-in. It is for this reason that the "Print" function is disabled when you run TRiLOGI as an Applet.

## 7. **Exit**

Execute this command to exit orderly from the TRiLOGI program. You will be prompted to save the current file if the contents have been edited and the changes have not yet been saved.

---

## II. **Edit Menu**

---

### 1. **Abort Edit Circuit**

Changes made to the current ladder circuit can be aborted if you execute this command before pressing <Enter> to accept changes made to the current circuit. If changes have already been accepted by pressing the <Enter> key, then this command will have no effect. This command is useful if you wish to completely abandon changes you have made to a circuit without going through all the undo steps.

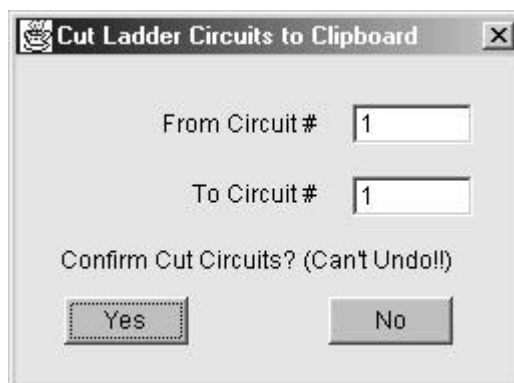
### 2. **Undo <Ctrl+Z>**

Undo the last changes made to a ladder circuit. TRiLOGI automatically stores the last 10 edited steps so you could execute undo several times to restore the circuit back to its original shape.

### 3. **Cut Circuit - <Ctrl+X>**

You can remove a number of circuits from the current ladder program and store them temporarily in the clipboard for pasting into another part of this ladder program or into another file altogether. In other words, it lets you move a block of circuits from one part of the ladder program to another part or into another file. Once you execute the "Cut Circuit"

command, a prompt box as shown below will appear. You have to specify the range of the circuits you wish to cut and press the "Yes" button to remove them from the ladder program.



\* Please note that you can't UNDO a Cut Circuit operation.

#### 4. **Copy Circuit (Ctrl+C)**

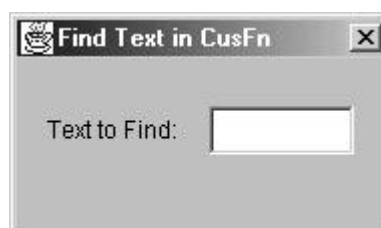
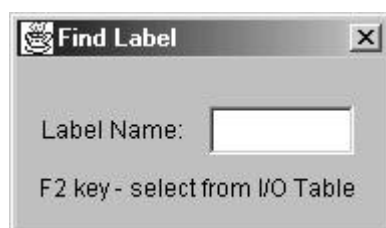
You can copy a block of circuits from the current ladder program and store them into the clipboard for pasting into another part of this ladder program or into another ladder program file altogether. The range dialog box similar to "Cut Circuit" will appear for you to enter the range of circuit to copy.

#### 5. **Paste Circuit <Ctrl+V>**

When you execute this command, the block of ladder circuit that you "Cut" or "Copy" into the clipboard will be pasted just before the currently selected circuit. The current circuit number will be adjusted to reflect the change.

#### 6. **Find <Ctrl+F>**

The Find command allows you to quickly locate a ladder logic circuit that contains a particular label name. This is useful for searching for the activity of a particular I/O in the program. The Find command can also be used to search for a keyword in a TBASIC program. When this command is executed you will be further prompted to select the options of either searching for a ladder logic label or finding a text in a Custom Function.



**Find Ladder element:** you can enter into the text field a string that partially or fully matches the label name you wish to locate. You can also press the <F2> key to open up the I/O table and pick the label name from the I/O table.

**Find Text in CusFn:** TRiLOGI will search through all the custom functions to locate the text that matches your entered text. The first CusFn that contains a match will be opened up for you to read. You will then be prompted to indicate if you wish to continue the search. Note that the text window in the custom function editor window is read-only during execution of the "Find Text" command.

If you click "No" at the prompt dialog, the last opened CusFn will stay open. However, at this stage the content in the Custom function editor is still **read-only**. If you wish to edit text in the CusFn you will need to click on the text window and it will be enabled for editing. This feature is implemented to prevent accidental changes to the custom function during the search process.

#### 7. **Goto <Ctrl+G>**

Use this command to move towards a specific circuit number. The "Goto" command is particularly useful if your program contains many circuits, and it is inconvenient to search for a particular circuit using the mouse or the cursor keys.

#### 8. **I/O Table <F2>**

Open up the I/O Table for defining label names for the PLC's I/O. For detailed explanation of I/O tables, please click on the following link: [I/O Definition Table](#)

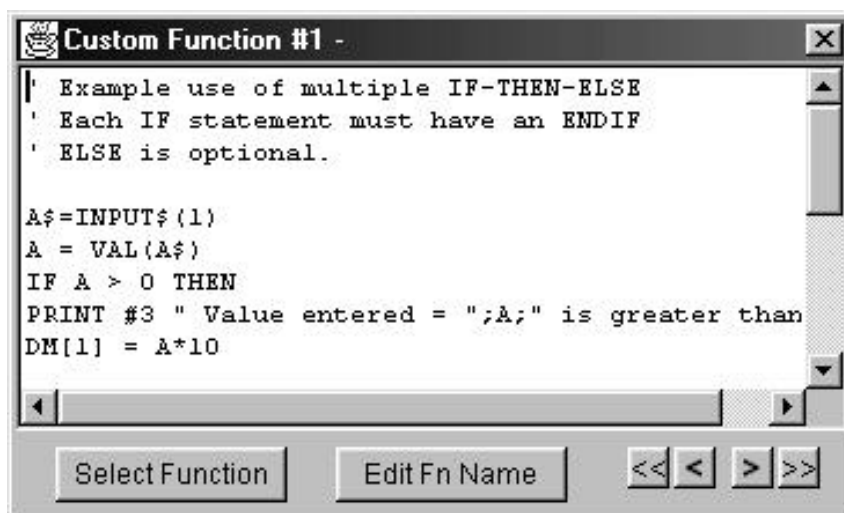
#### 9. **View I/O Type on Ladder <F3>**

Toggle between display or no display of the I/O type for ladder logic contacts on the screen. All ladder logic contact symbols are normally identified by their label names. However, you can also choose to display an optional small literal to indicate the I/O types. E.g. i=input, o=output, r= relay, t= timer and c=counter. When TRiLOGI first starts, the display is enabled but you have the option of turning it off if you find it distracting.

#### 10. **Edit Custom Function <F7>**

Opens up the Custom Function Editor window for you to enter the TBASIC program. You will be required to select the custom function number or a label name from the CusFn table (which is part of the I/O Table) . Each TRiLOGI file can contain a maximum of 256 custom

functions. Each custom function will be opened in its own window. The custom function number and the optional label name will be displayed on the Title of the Custom Function editor window:



You can scroll from one custom function to the next one using the **<** **>** keys. However, clicking on the **<<** and **>>** buttons allows you to scroll to the previous or the next **non-empty** CusFn. All empty functions will be skipped. This is useful if you need to browse through all the custom functions to locate something.

If you wish to copy/cut text from one CusFn and paste to another you will have to use the <Ctrl-C>, <Ctrl-X> and <Ctrl-V> keys.

### 11. Clear Custom Functions

This command allows you to select a range of custom functions whose content you want completely cleared. You will be prompted to select the range of custom functions to erase. Note that this action is not undoable.

---

## III. Controller Menu

---

All commands in this menu are for communication with the PLCs via the TLServer. Hence the TLServer must be actively running and connected to the PLC(s) via its serial port before the commands here can be successfully executed. Note that TLServer can be running on the same computer that TRILOGI is running on (using localhost IP 127.0.0.1), or on another computer in the same local area network, or anywhere in the world with an Internet connection. The experience is identical regardless of where the TLServer (and hence the PLC) is situated.

If there is no existing connection made to the TLServer, then execution of any command in this menu will always bring up the password dialog for you to enter the Username/Password as well as the IP address;port of the TLServer. You must be positively authenticated before you are able to log-in to the TLServer. See "Log In to TLServer" for detailed explanation of the Username/Password Dialog box. Once you have login to the TLServer, see explanation of each function below:

**1. Select Controller <Ctrl-I>**

The only editable field is the ID field. You have to enter the ID address in hexadecimal notation (00 to FF). This command allows you to select another PLC that is connected to the same TLServer but with a different ID for on-line monitoring or program transfer.

**2. Connect/Disconnect to Server**

Use this command to log-in to the TLServer only if you have no intention to perform other controller commands. You may find that you seldom need to use this command since running the On-Line Monitoring or Program Transfer commands will also let you log-in to the TLServer if you have not yet done so. However, once you are connected, this command changes into "Disconnect from Server" and this is the only way for you to log out of the currently connected TLServer so that you can use the Username/Password dialog box to log-in as a different user, or to log-in to another TLServer of a different IP Address/port number.

**3. On-Line Monitoring <Ctrl+M>**

See On-Line Monitoring help document for details.

**4. Program Transfer to PLC <Ctrl+T>**

This command is only available if your log-in username is assigned the access level of a "Programmer". If you login as a "User" or "Visitor", this command is disabled from the Controller's menu. (It will be enabled again after you disconnect from the server)

You can use this command to transfer your TRILOGI ladder +TBASIC program into the PLC. You will be prompted to confirm your action to prevent accidentally affecting the target PLC. The ladder program must be compiled successfully before the program transfer process can take place. The progress of the transfer process will be clearly shown on the program transfer dialog box.

## 5. Open Matching Source File

You can use this command to query the connected PLC for the filename of the last TRILOGI program transferred to it and it will attempt to match it to a file stored in the log-in user's directory at TLServer. If the file is found, it will be opened. Otherwise it will report that the file is not found. Note that this command only opens the source file based on file name matching. It does not verify whether the file has been modified. It is the user's responsibility to ensure that the file stored in the server is the same one that has been compiled and transferred to the PLC.

Note that if you have created a new file (i.e. the file name is "Untitled") and then attempt to perform on-line monitoring, this command will be called automatically to try to open a file that matches the PLC. The command is also invoked when you select a PLC with a different ID either from the "Controller" menu or from within the "Full-Screen Monitoring" window.

## 6. Get PLC's Hardware Info

You can find out the PLC's firmware version number, the maximum of input, outputs, relays, timers and counters supported on this PLC as well as the total amount of program memory available. The same info will be displayed when you try to transfer a program to the PLC.

## 7. Set PLC's Real Time Clock

The PLC's real time clock (RTC, which includes both date and time) can be set quickly using this command. When you execute this command, a dialog box that contains the year, month, day, hour, min, sec and day of week are displayed for you to enter the value. The dialog box is initially filled with value taken from the client's computer's own calendar and clock. You can change any of the field to the desired values and then click on the "Set PLC's Clock" button:



The dialog box will be closed after the TRiLOGI has transferred all the data to the PLC. You should use on-line monitoring to verify that the data has indeed been properly written into the PLC.

Note that the "Year" field is restricted to only between 1996 and 2096, "Month" is between 1 and 12, "Day" is between 1 and 31, "Hour" is between 0 and 23, "Min" and "Sec" are between 0 and 59. If you enter an illegal value TRiLOGI will beep and the cursor will be put at the offending text field. Correct the mistake and then click on the "Set PLC's Clock" button again to transfer the values to the

---

## IV. Simulate Menu

---

TRiLOGI allows you to perform almost 100% simulation of your PLC's program off-line on your PC. This is a great tool for testing a program quickly before a machine has been manufactured. It is also a wonderful tool for all new PLC programmers to practice their ladder logic programming skill without the need to purchase a PLC test station.

TRiLOGI automatically compiles a ladder program before activating the simulator. If an error is found during compilation, the error will be highlighted where it occurs and the type of error is clearly reported so that you can make a quick correction.

### 1. **Run (All I/O Reset) <Ctrl+F9>**

This should be the option to use when you first begin to test your TRiLOGI program. When executed, all I/O bits (including inputs) are cleared to OFF state, all integer data are set to 0 and all string data are set to empty string. Then the "Programmable Logic Simulator" window will open for you to conduct the simulation test of your TRiLOGI ladder program.

### 2. **Run (reset Except i/p) <Ctrl+F8>**

Very often you may wish to keep the input settings "as is" when you reset the simulator. This situation is quite realistic in the sense that when a PLC is powered-on, some of its inputs may already be in the ON state. (e.g. sensors may detect the end positions of a cylinder rod, etc). This command allows you to preserve the logic states of all "Inputs" while resetting all other data. Note that the <Ctrl-F8> key also works in the "Simulator" screen so that at any time you can reset the simulator without affecting the logic states of the inputs.

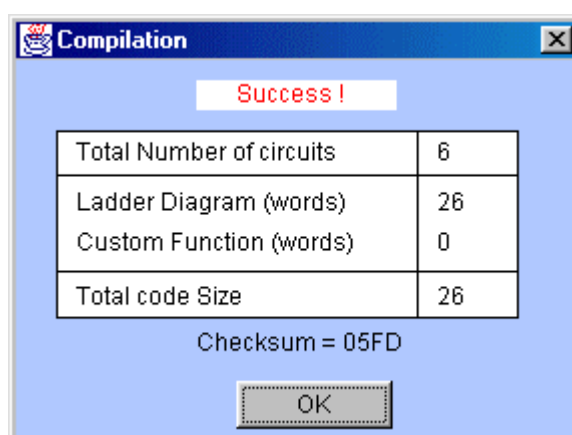
### 3. Continue Run (no reset) <F9>

Use this command to continue simulating the program since you last closed the simulator. All previous data are kept intact and will be used by the simulator to continue execution of the ladder program. If you have made changes to the ladder program or custom functions, the whole program will be recompiled before running.

Note that first scan pulse (1st.Scan) will not be activated when this command is executed since this is supposedly a continuation from the previous simulation run. This command can be useful if you have discovered a simple bug in your software during simulation, you can fix it immediately and test the effect of the change on the simulator instantly without restarting the entire simulation session from the beginning again.

### 4. Compile Only <F8>

Allows you to compile the TRiLOGI file only in order to view the compilation statistics:



### 5. Reset All I/Os <Ctrl-R>

Clears all I/Os in the simulation engine without invoking the simulator. Since all I/Os whose logic states are turned ON in the simulator will also be shown as highlight on the ladder diagram, this offers a way to clear the I/Os if it hinders your viewing of the ladder program.

---

## V. Circuit Menu

---

### 1. Insert Comments

Comments are specific remarks used by a programmer to explain various characteristics of a program segment and are ignored by the compiler. TRiLOGI Version 5.0 allows comments to be freely inserted between circuits. Execute this command and the Comment Editor will be opened. The comment editor allows you to enter any text you like that best describe the working of the circuit. All standard text editing keys, including cut and paste are applicable to the Comment Editor. When you have finished editing the comment, press <ESC> key to close it.

Once a comment has been created, it is assigned a circuit number and is treated like any other circuits. You can edit it by pressing the <spacebar> when you are in Browse mode, alternatively, you can move it around, copy it to another destination or delete it entirely using commands in the "Circuit" menu.

### 2. Insert Circuits

This command enables you to insert a new circuit just before the currently selected circuit. The current circuit number will be increased by one while the new circuit will assume the current circuit number. You will be placed in the [circuit editing mode](#) for immediate circuit creation.

### 3. Move Circuit

You can rearrange the order of the circuits by using this command. Select the circuit you wish to move and execute the "Move Circuit" command, then select a destination circuit location and press <Enter>. The selected circuit will be moved to the new location before the destination circuit.

Note that if you wish to move a block of circuits to a new location, you may find it more productive to use the "Cut Circuit" and "Paste Circuit" commands in the "Edit" menu.

### 4. Append Circuit

Execute this to add a new circuit to the ladder logic program. This new addition will be positioned immediately after the last circuit in the entire program.

## 5. Delete Circuit

This command allows you to delete the one or more circuits. You will be prompted to enter the range of circuits that you wish to delete. Please note that you can't UNDO a delete circuit operation.

---

## VI. Help Menu

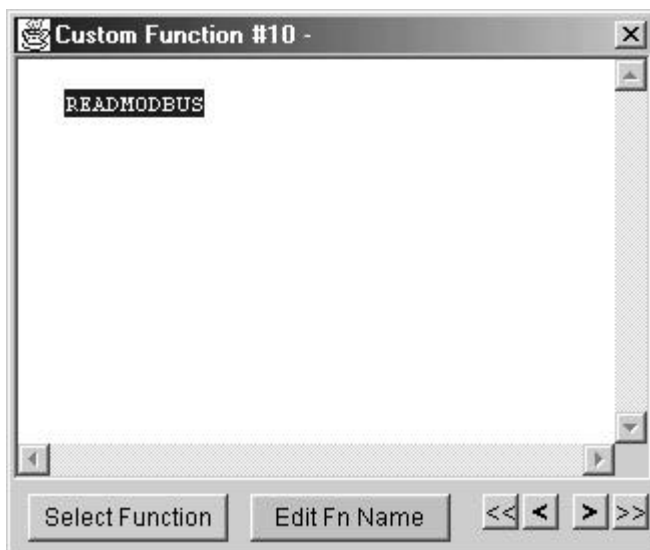
---

All contents in this manual are available for instant reference on the computer where the TRiLOGI program is running. This includes running the "apple" version of TRiLOGI program on a remote browser because the relevant help files will be retrieved from the TLServer automatically.

You can call up the help files anytime by pressing the <F1> key. You can also select the "Content" item from this "Help" menu to bring up the content page of the entire on-line TRiLOGI help files. On the content page you can find the links to the Ladder Logic Editor and the entire TBASIC language reference.

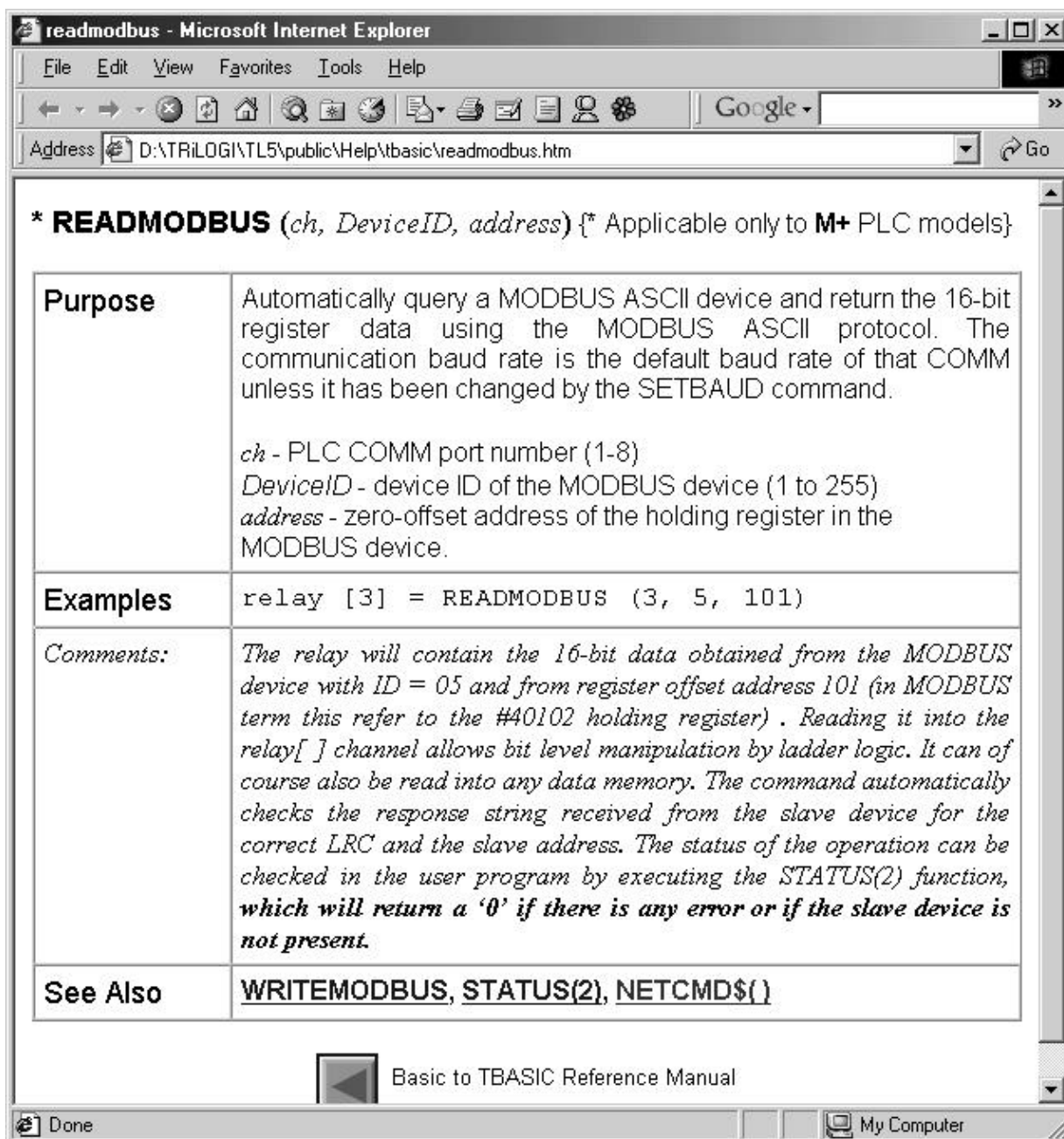
### Instant Help for TBASIC Keywords

One convenient feature implemented in TRiLOGI Version 5.x is the ease of getting help for the syntax of a known TBASIC keyword. E.g., if you want to find the syntax for the keyword "READMODBUS", instead of navigating through the help file links, you can simply select the "READMODBUS" keyword in the custom function editor and then press the <F1> key. You will be immediately presented with the help document for "READMODBUS" command as illustrated in the following screen shots.



Press the <F1> key once after you have selected the keyword of interest.

=> The help file for the keyword will be immediately loaded by Internet Explorer, as follow:



The screenshot shows a Microsoft Internet Explorer window titled "readmodbus - Microsoft Internet Explorer". The address bar displays "D:\TRILOGI\TL5\public\Help\tbasic\readmodbus.htm". The main content area contains the following information:

**\* READMODBUS** (*ch*, *DeviceID*, *address*) {*\* Applicable only to M+ PLC models*}

<b>Purpose</b>	Automatically query a MODBUS ASCII device and return the 16-bit register data using the MODBUS ASCII protocol. The communication baud rate is the default baud rate of that COMM unless it has been changed by the SETBAUD command.  <i>ch</i> - PLC COMM port number (1-8) <i>DeviceID</i> - device ID of the MODBUS device (1 to 255) <i>address</i> - zero-offset address of the holding register in the MODBUS device.
<b>Examples</b>	<code>relay [3] = READMODBUS (3, 5, 101)</code>
<b>Comments:</b>	<i>The relay will contain the 16-bit data obtained from the MODBUS device with ID = 05 and from register offset address 101 (in MODBUS term this refer to the #40102 holding register) . Reading it into the relay[ ] channel allows bit level manipulation by ladder logic. It can of course also be read into any data memory. The command automatically checks the response string received from the slave device for the correct LRC and the slave address. The status of the operation can be checked in the user program by executing the STATUS(2) function, which will return a '0' if there is any error or if the slave device is not present.</i>
<b>See Also</b>	<a href="#">WRITEMODBUS</a> , <a href="#">STATUS(2)</a> , <a href="#">NETCMD\$( )</a>

Basic to TBASIC Reference Manual

Done My Computer

# Chapter 8: Ladder Logic Language Reference

---

## I. Ladder Logic Fundamentals: Contacts, Coils, Timers and Counters

---

### 1. Contacts

Ladder logic programs mimic the electrical circuit diagrams used for wiring control systems in the electrical industry. The basic purpose of an electrical control system is to determine whether a load should be turned ON or turned OFF, under what circumstances and when it should happen. To understand a ladder program, just remember the concept of current flow - a load is turned ON when the current can flow to it and is turned OFF when the current could not flow to it.

The fundamental element of a ladder diagram is a "Contact". A contact has only two states: open or closed. An open contact breaks the current flow whereas a closed contact allows current to flow through it to the next element. The simplest contact is an On/OFF switch, which requires external force (e.g. the human hand) to activate it. Limit switches are those small switches that are placed at certain location so that when a mechanical device moves towards it, the contact will be closed and when the device moves away from it, the contact will be open.

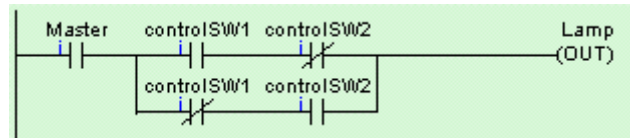
If a contact is connected to a load and the contact is closed, the load will be turned ON. This simple concept can be illustrated by the most basic ladder diagram as follow:



The vertical line on the left is the "Power" line; current must flow through the "Switch" contact in order to turn ON the load "Lamp". (In fact, there should be a second vertical line on the right end of the ladder diagram to provide a return path for the current flow, but this is omitted to simplify the circuit diagram). Now, if instead of wiring the switch to the lamp directly as suggested in the above diagram, you could connect the switch to the PLC's **input** and connect the lamp to the PLC's **output**, and then write the above ladder program to perform the same job. Of course it makes little sense to use a PLC if that is all you want to do. We will see how a PLC can simplify wiring shortly.

**Note:** The contact "Switch" shown in the above diagram is termed a Normally-open (N.O.) contact.

Now, let's say if there are 3 switches that must work together to control the lamp. A Master switch must be ON, and one of the two control switches "controlsw1" and "controlsw2" must be ON while the other must be OFF in order to turn ON the lamp (think of two-way switches in your house and you will get the idea). We can wire all 3 switches to 3 inputs of the PLC and the lamp to the output of the PLC. We can write the following ladder program to perform this task:



A contact with a "/" across its body is a Normally-Closed (N.C.) contact. What it means is that the ladder program is using the "inverse" of the logic state of the input to interpret the diagram.

Hence in the above ladder diagram, if "Master" and "controlSW1" are turned ON but "controlSW2" is turned OFF, the lamp will be turned ON since the inverse logic state of an OFF state "controlSW2" is true. Think of an imaginary current flowing through the "Master" contact, then through the "controlSW1" and finally through the normally-closed "controlSW2" contact to turn ON the lamp.

On the other hand, if "controlSW1" is OFF but "controlSW2" is ON, the Lamp is also turned ON because the current could flow via "Master" and then through the lower parallel branch via N.C. "controlSW1" and the N.O. "controlSW2".

**Note:** As you can see, although the switch "controlSW1" is connected to only 1 physical input to the PLC, but it appears twice in the ladder diagram. If you actually try to connect physical wires to implement the above circuits, both "controlSW1" and "controlSW2" will have to be of multiple poles type. But if you use a PLC, then these two switches only need to be of single-pole type since there is only one physical connection, which is to the input terminal of the PLC. But in the ladder diagram the same contact may appear as many times as you wish as if it has unlimited number of poles.

The above example may be simple but it illustrates the basic concept of logical "AND" and "OR" very clearly. "controlSW1" and "controlSW2" are connected in series and both must be TRUE for the outcome to be TRUE. Hence, this is a logical AND connection. On the other hand, either one of the two parallel branches may be used to conduct current, hence this is a logical OR connection.

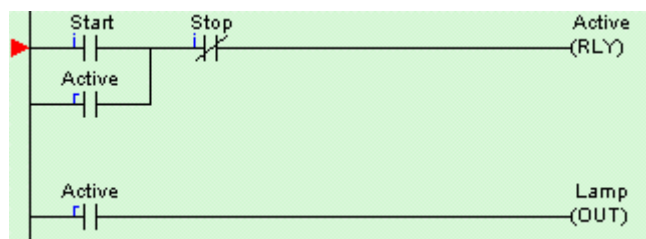
Once you understand this fundamental principle of interpreting a ladder diagram, everything should become clearer and simpler. Ladder diagram programming can be used to create a rather sophisticated control system. However, In TRiLOGI we augment its power further by allowing a ladder program to activate customized functions created in TBASIC.

## 2. Relay Coils

A contact can also be activated by the presence of an electrical current. This makes it possible for a control system to control the turning ON or OFF of a large load by using electrical current to activate a switch that can conduct high current. The simplest form of this type of contact is a relay.

In traditional electromagnetic relay, a coil of wire is wound around an iron core that turns it into an electromagnet. When current passes through the "coil" the magnet is "energized" and the force is used to either close a contact (that makes it a normally-open contact, closed only when energized) or open it (that will be a normally-closed contact since it is closed when not energized).

Ladder Logic programming language borrows some of those terms used to describe the electromagnetic relay for its own use. You connect a relay coil to the right end of the ladder diagram just like an output, as follow:



In a PLC, there are hundreds of internal "relays" which are supposed to behave like the typical electromagnetic relay. Unlike an output (e.g. the "Lamp" output) which has a physical connection out of the PLC, when an internal relay is turned ON, it is said to be "energized" but you will not see any changes in the PLC's physical I/Os. The logic state is kept internally in the PLC. The contact of the relay can then be used in the ladder diagram for turning ON or OFF of other relays or outputs. A relay contact in the ladder diagram can be Normally-Open (NO) or Normally Closed (NC) and there is no limit to the number of contacts a relay can have.

### **3. Out Coils**

A PLC output is really just an internal relay with a physical connection that can supply electrical power to control an external load. Thus, like a relay, an output can also have unlimited number of contacts that can be used in the ladder program.

### **4. 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 a last aborted timer is re-energized, the delay timing will restart afresh using the SV of the timer and not continue from the last aborted timing operation.

### **5. Counter Coils**

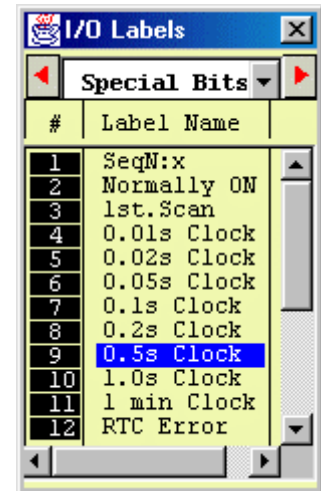
A counter is also a special kind of relay that has a programmable Set Value (SV). When a counter coil is energized for the first time after a reset, it will load the value of SV-1 into its count register. From there on, every time the counter coil is energized from OFF to ON, the counter decrements its count register value by 1. Note that the coil must go through OFF to ON cycle in order to decrement the counter. If the coil remains energized all the time, the counter will not decrement. Hence counter is suitable for counting the number of cycles an operation has gone through.

When the count register hits zero, all 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].

## II. Special Bits

TRiLOGI contains a number of special purpose bits that are useful for certain applications. These include 8 clock pulses ranging from periods of 0.01 second to 1 minute, a "Normally-ON" flag and a "First Scan Pulse", etc.

To use any of these bits, enter the ladder editor and create a "contact"; when the I/O table pops up, scroll the windows until a "Special Bits" menu pops up. *This menu is located after the "Counter Table" and before the "Input" table.* as shown below:

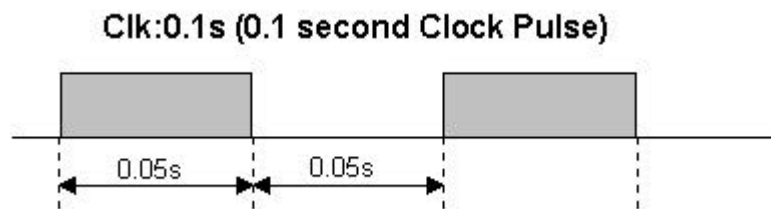


### 1. Clock pulse bits

The 8 clock pulses supported by TRiLOGI are:

Clock Pulse Period	Ladder Symbol
0.01 second	Clk:.01s
0.02 second	Clk:.02s
0.05 second	Clk:.05s
0.1 second	Clk:0.1s
0.2 second	Clk:0.2s
0.5 second	Clk:0.5s
1.0 second	Clk:1.0s
1 minute	Clk:1min

A clock pulse bit is ON for the first half of the rated period, then OFF for the second half. Duty cycles for these clock pulse bits are therefore 50%, as follow:



The clock pulse bits are often used with counter instructions to create timers. Additionally, they can be used as timing source for "Flasher" circuit. A reversible counter can also work with a clock pulse bit to

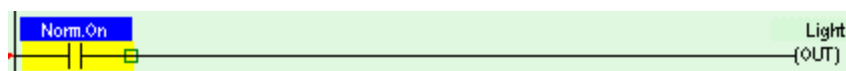
create secondary clock pulses of periods that are multiples of the basic clock pulse rate.

## 2. SeqN:X

These are special "Sequencer" contacts which are activated only when the step counter of a Sequencer N reaches step #X. E.g. a Normally Open contact Seq2:6 is closed only when Sequencer #2 reaches Step #6. At any other step, this contact is opened.

## 3. Normally ON Flag - Norm.ON

You can make use of this flag if you need to keep something permanently ON regardless of any input conditions. This is because with the exception of Interlock Off function ———[ILOff], a coil or a special function is not allowed to connect directly to the power line (the vertical line on the left end of the ladder diagram). If you need to permanently enable a coil, consider using the "Normally-ON" bit from the "Special Bits" menu, as follow:



## 4. First Scan Pulse - 1st.Scan

This special bit will only be turned ON in the very first scan time of the ladder program. After that it will be permanently turned OFF. This is useful if you need to initialize certain conditions at the beginning. When the program is transferred to the PLC, this bit will only be ON when the PLC is first powered up or after it has been reset.

## 5. Real Time Clock Error - RTC.Err

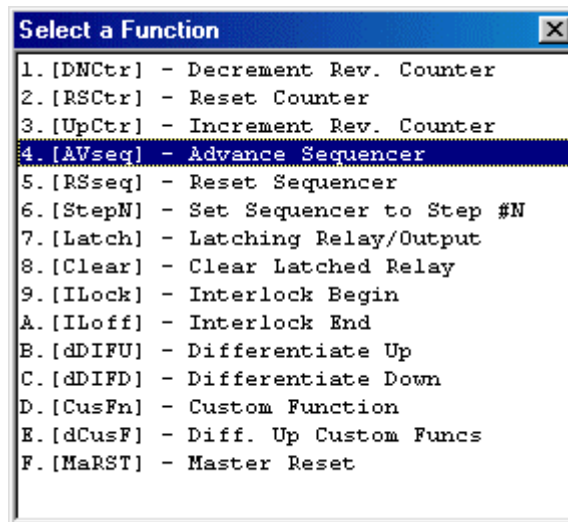
This bit is turned ON if the M-series PLC does not have battery-backed MX-RTC option and the clock has been reset due to power failure or watchdog timer reset. This gives warning to applications that require a correct real world time (such as scheduled ON/OFF operation) that the clock data is incorrect, hence enabling corrective action to be taken.

---

### III. Special Functions

---

During ladder circuit editing, when you click on the or icon to create a special function coil, a special function menu will pop up as shown below:



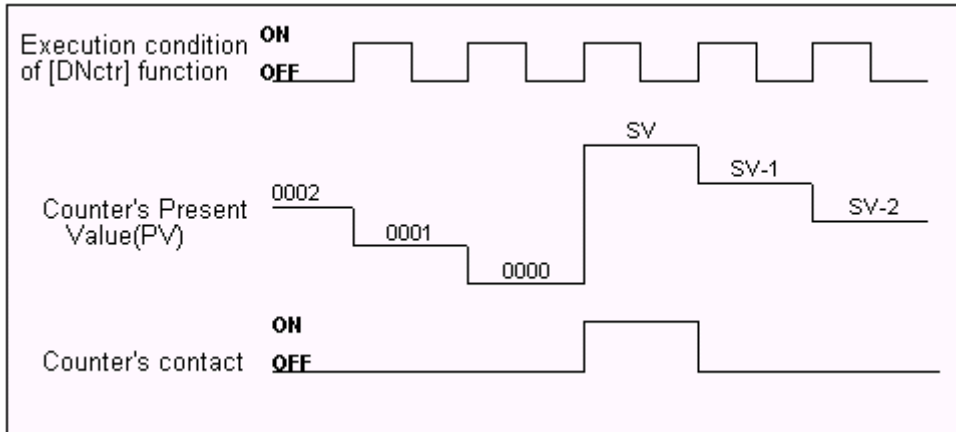
#### 1. Reversible Counter Functions: [DNctr], [Upctr] and [RSctr]

The [DNctr], [UPctr] and [RSctr] functions work together to implement reversible counter functions on any of the 128 counters supported by TRILOGI.

The ordinary down counter (created by clicking on the icon) essentially decrements the counter value by 1 from the "Set Value" (SV) and will stop when its count becomes zero. Unlike the ordinary down counter, a reversible counter is a circular counter that changes the counter present value (PV) between 0 and the SV. When you try to increment the counter past the "Set Value", it will **overflow** to become '0'. Likewise if you try to decrement the counter beyond '0', it will **underflow** to become the "Set Value".

All three counter functions [DNctr], [UPctr] and [RSctr] can operate on the same counter (i.e. assigned to the same counter label) on different circuits. Although these circuits may be located anywhere within the ladder program, it is recommended that the two or three functions which operate on the same counter be grouped together in the following order: DNctr], [Upctr] and [RSctr]. Note that NOT all three functions need to be used to implement the reversible counter.

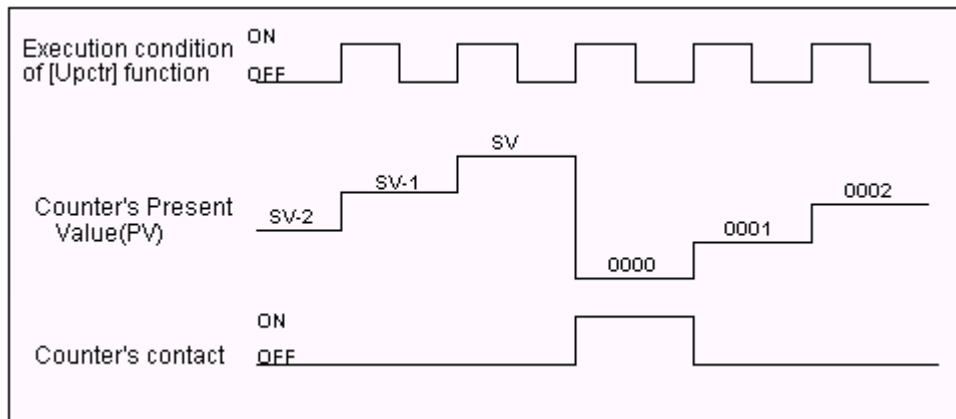
## Decrement Counter [DNctr]



Each time when the execution condition of a [DNctr] function changes from OFF to ON, the present value of the designated counter is changed as follow:

- If the counter's present value (PV) is inactive, load the counter register with the "Set Value" (SV, defined in the Counter table) minus 1.
- If the counter's present value (PV) is already '0', then load the counter's PV with the SV defined in the counter table and turn on the counter's contact (also known as the completion flag).
- Otherwise, decrement the counter PV register by 1.

## Increment Counter [Upctr]



Each time when the execution condition of an [Upctr] function changes from OFF to ON, the present value of the designated counter is affected as follow:

- If the counter is inactive, load the counter register with the number '0001'.

- b. If the counter's present value (PV) is equal to the Set Value (SV, defined in the Counter table), load the counter register with number '0000' and turn on the counter's contact (also known as the completion flag).
- c. Otherwise, increment the counter PV register by 1.

### Reset Counter [RSctr]

When the execution condition of this function changes from OFF to ON, the counter will reset to inactive state. This function is used to reset both a reversible counter and an ordinary down-counter coil.

## **2. Sequencer Functions: [AVseq], [RSseq] and [StepN]**

Please refer to the later section in the chapter on "Using TRILOGI Sequencers"

## **3. Latch Relay Function [Latch]**

Latching relay is convenient for keeping the status of an execution condition even if the condition is subsequently removed. The program elements that are assigned as Latching Relays will remain ON once they are energized. Only Relays and Outputs may be assigned as Latching Relays.

On selecting [Latch] function, you can use the left/right cursor keys or click on the left/right arrow keys to move between the Relay and Output tables. The selected relay or output will now be assigned as a Latching Relay. You will be able to see the label name of the program element above the [Latch] symbol in the ladder diagram.

Although latch-relay can be used in place of self-latching (Seal) circuits, a latch-relay in an interlock section will not be cleared when the interlock occurs. Only a self-latching circuit as shown in the following will be cleared in an interlock section:



## **4. Clear Relay Function [Clear]**

To de-energize a program element that has been latched by the [Latch] function, it is necessary to use [Clear] function. On selecting [Clear], choose the output or relay to be de-energized. When the execution condition for that circuit is ON, the designated output or

relay will be reset. In the ladder diagram, the program element label name will be shown above the [Clear] symbol.

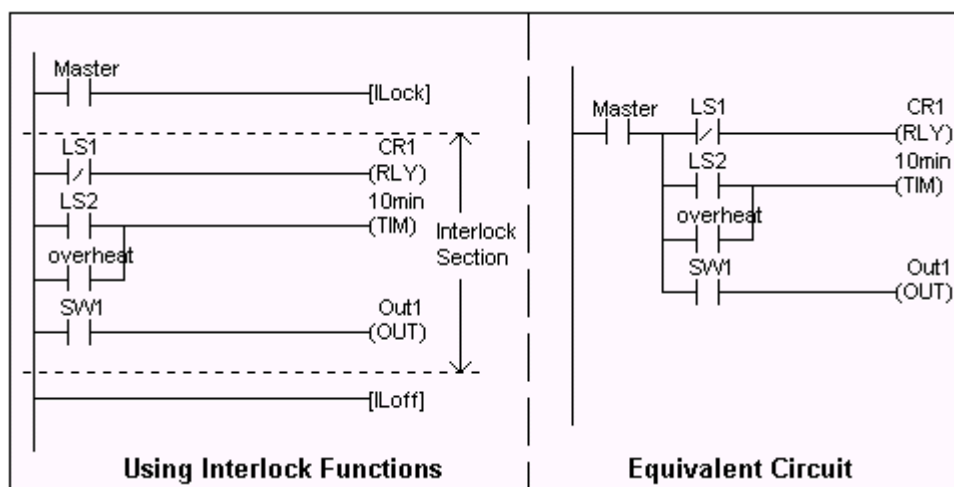
If the execution condition for [Latch] and [Clear] functions are both ON at the same time, then the effect of the designated bit depends on the relative locations of these two functions. Remember that an output or relay bit energized by [Latch] will remain ON until it is turned OFF by [Clear]. It is recommended that [Clear] circuit be placed just after the [Latch] circuit for the same output or relay controlled by these two functions. This ensures that [Clear] function has higher priority over [Latch] function, which is normally so in hardware latch-relay or other industrial PLCs.

## 5. Interlock [ILock]

The "Interlock" [ILock] and "Interlock Off" [LOff] functions work together to control an entire section of ladder circuits. If the execution condition of a [ILock] function is ON, the program will be executed as normal. If the execution condition of [ILock] is OFF, the program elements between the [ILock] and [LOff] will behave as follow:

- All output coils are turned OFF.
- All timers are reset to inactive.
- All counters retain their present values.
- Latched relays by [Latch] function are not affected.
- [dDIFU] and [dDIFD] functions are not executed.
- All other functions are not executed.

An Interlock section is equivalent to a master control relay controlling a number of sub-branches as follow:



Note that [LOff] is the only function that does not need to be energized by other program elements. When you use one or more [ILock] functions, there must be at least one [LOff] function before the

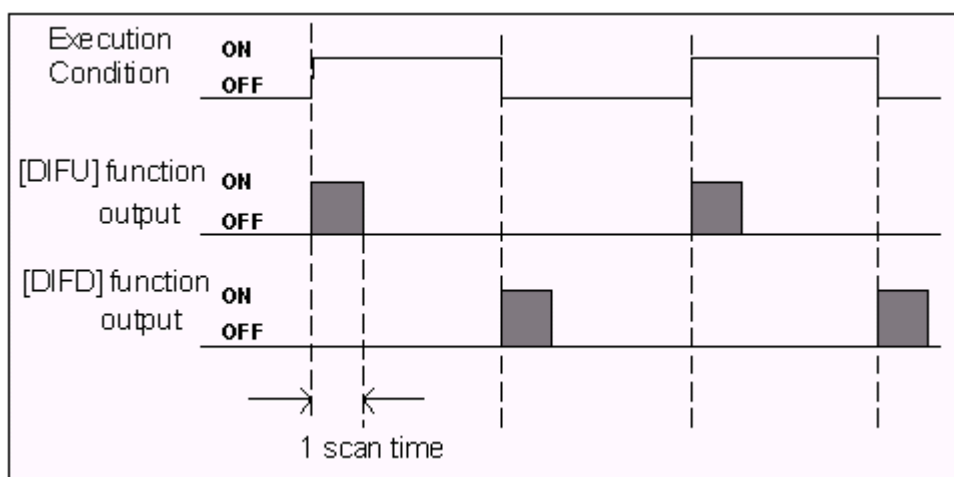
end of the program. Otherwise the compiler will warn you for the missing [LOff]. The logic simulator always clears the Interlock at the end of the scan if you omit the [LOff] function.

You can program a second or third level Interlock within an Interlock section using a few [Llock] functions. However, you only need to program one [LOff] function for the outermost Interlock section, i.e. [LOff] need not be a matching pair for an [Llock] function.

## 6. Differentiate Up and Down [dDIFU] and [dDIFD]

When the execution condition for [dDIFU] goes from OFF to ON, the designated output or relay will be turned ON for one scan time only. After that it will be turned OFF. This means that the function generates a single pulse for one scan time in response to the rising-edge of its execution condition. When its execution condition goes from ON to OFF nothing happens to the output or relay that it controls.

On the other hand, when the execution condition for [dDIFD] goes from ON to OFF, the designated output or relay will be turned ON for one scan time only. After that it will be turned OFF. This means that the function generates a single pulse for one scan time in response to the trailing edge of its execution condition. When its execution condition goes from ON to OFF, nothing happens to the output or relay that it controls.



## 7. Custom Functions: [CusFn] and [dCusF]

These two functions allow you to connect a user-defined custom function (CusFn) to the ladder logic as if it is a relay coil. Custom functions are created using the integrated text editor provided by TRILOGI Version 5.x.

## 8. Master Reset

An ON condition to this function clears all mailbox inputs, outputs, relays, timers and counter bits to OFF, resets all timers counters/sequencers to inactive state, and clears all latched relay bits. All integer variables will be cleared to zeros and all string variables will be assigned to empty string.

---

## IV. Using TRiLOGI Sequencers

---

A sequencer is a highly convenient feature for programming machines or processes that operate in fixed sequences. These machines operate in fixed, clearly distinguishable step-by-step order, starting from an initial step and progressing to the final step and then restart 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

TRiLOGI Version 5 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".

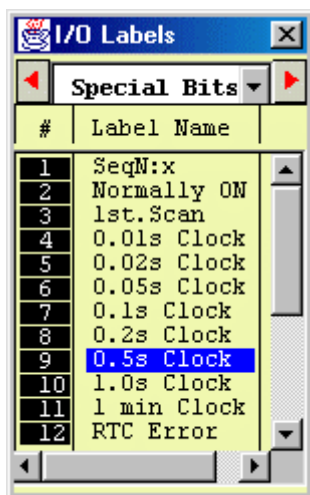
To use a sequencer, first define the sequencer name in the Counter table by pressing the <F2> key and scroll to the Counter Table. Any counter to be used as sequencer can only assume label names "Seq1" to "Seq8" corresponding to the counter numbers. For e.g. if Sequencer #5 is to be

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.

Construct a circuit that uses the special function "Advance Sequencer" [AVSeq]. 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 change 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.

Then 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. 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.

## 1. Special Sequencer Functions

Quite a few of the ladder logic special functions are related to the use of the sequencer. These are described below:

### Advance Sequencer - [AVseq]

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

### 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.

### 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.

### 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.

## 2. Other Applications

### *a. Driving Stepper Motor*

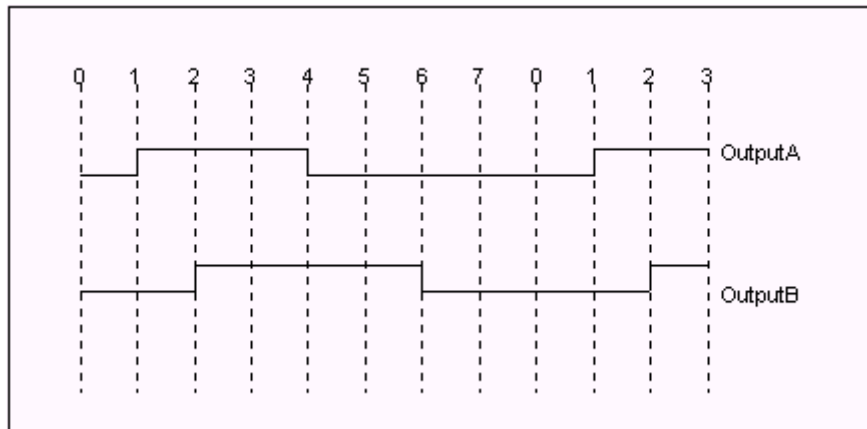
A sequencer may be used to drive a stepper motor directly. A two-phase stepper motor can be driven by four transistor outputs of the controller directly (for small motors with phase current < 0.5A) or via solid-state relays. The stepper motor can be driven using a sequencer that cycles through Step#0 to Step#3 (full-step mode) or Step#0 through Step#7 (half-step mode). Each step of the sequencer is used to energize different phases of the stepper motor. A clock source is needed to drive the stepper motor through

its stepping sequence. The stepping rate is determined by the frequency (which is equal to 1/period) of the clock source.

Clock pulses with periods in multiples of 0.01 second can be generated easily using the "Clk:.01s" bit and an [Upctr] function. For e.g., to generate a clock source of period = 0.05s, use "Clk.01s" to feed to an [Upctr] counter with Set Value = 4. The counter's contact (completion flag) will be turned ON once every 5 counts (0,1,2,3,4), which is equivalent to a 0.05 sec. clock source.

*b. Replacing a Drum Controller*

A drum controller can be replaced easily by a sequencer if the timing of the drum's outputs can be divided into discrete steps. Assuming a drum controls two outputs with the timing diagram shown in the following figure:



This can be replaced by an 8-step sequencer. Step 1 (e.g "Seq1:1") turns ON and latch Output A using [Latch] function, Step 2 turns ON and latch Output B, Step 4 turns OFF Output A using the [Clear] function, and Step 6 turns OFF Output B. All other steps (3,5,7,0) have no connection.

### 3. 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 6.9.

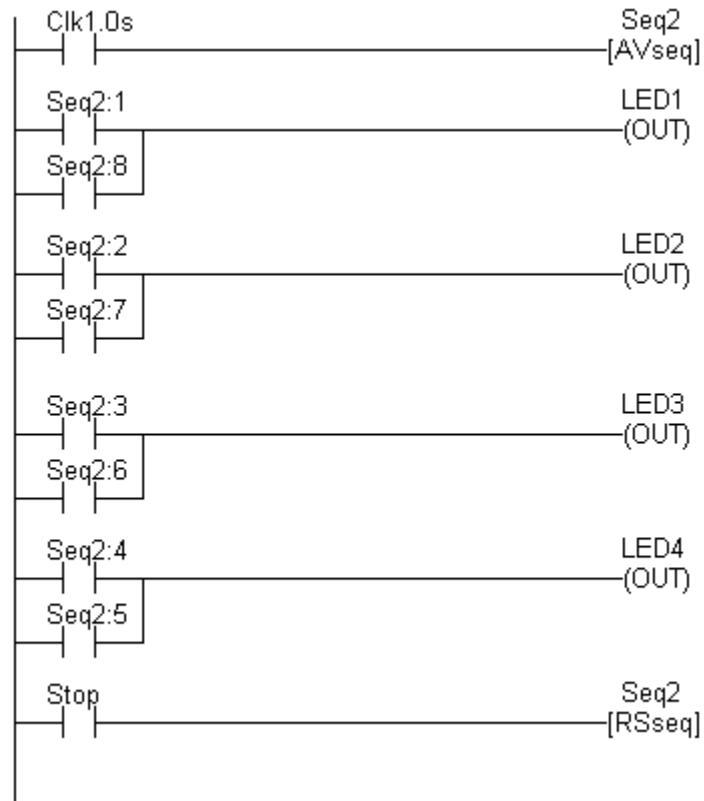


Figure 6.9

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 LED is turned ON by Step 0, all LEDs will be OFF.

# Chapter 9: Introduction to TBASIC Custom Functions

---

## I. Overview

---

TRiLOGI Version 5 supports user-created special functions, known as Custom Functions (the symbol CusFn will be used throughout this manual to mean Custom Functions). Up to 256 CusFns can be programmed using a special language: TBASIC.

TBASIC is derived from the popular BASIC computer language widely used by microcomputer programmers. Some enhancements as well as simplifications have been made to the language to make it more suitable for use in PLC applications.

There are three simple ways to create a new CusFn:

1. From the "Edit" pull-down menu, select the item "Edit Custom Function" and select the function number from a pop-up CusFn selection table that may range from 1 to 256. You may also use the hotkey <F7> to open up the selection table. The selection table allows you to define unique and easily identifiable names for each custom function. Once you have selected the custom function the editor window will open up with the contents of that particular custom function.
2. If you have already created a ladder circuit which connects to either a [CusFn] or [dCusF] function (both appear as menu-items within the "Special Function" pop-up menu), then you can easily open up that particular CusFn by clicking the right mouse button while the highlight bar is at the [CusFn] or [dCusFn].

---

## II. Custom Function Editor

---

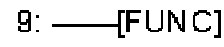

The custom function editor window allows creation of any number of lines of TBASIC program statements. Since this is a standard text editor, you should have no problem using the key and mouse controls to edit the text. Pressing <F1> at the text editor window opens up a Help screen that will show you the common keys and mouse actions. E.g. To copy a paragraph of text, select it using the mouse and the press <Ctrl-C>. Move the text editing cursor to the destination and press <Ctrl-V> to paste it to the new location.

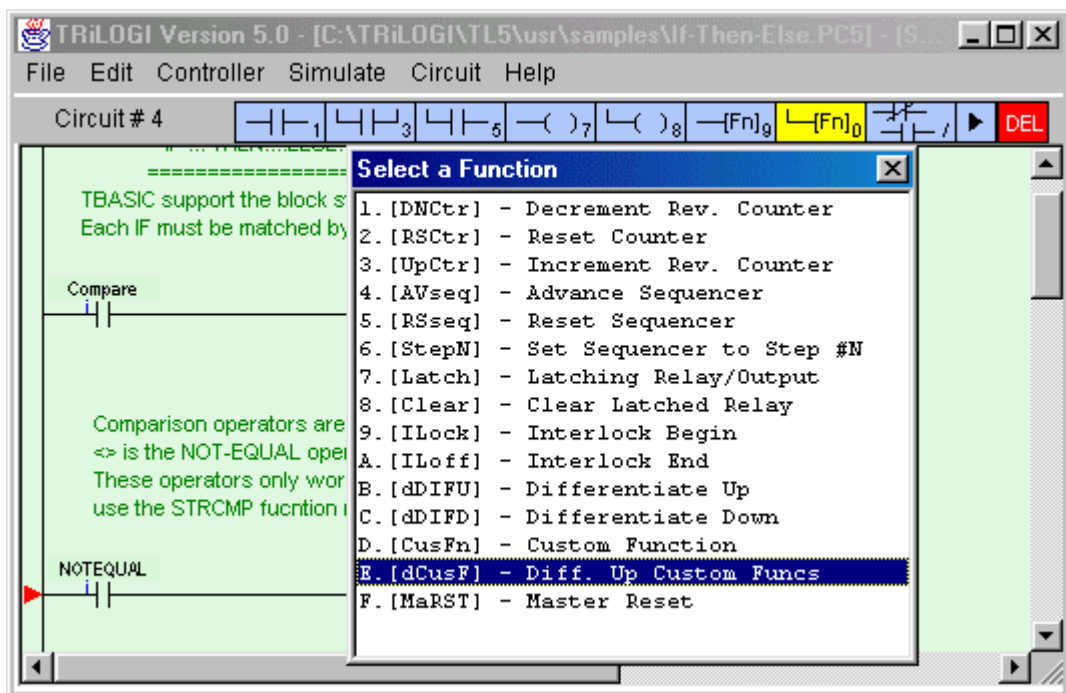
### III. Custom Function Execution

It is important to understand when and how a TBASIC-based Custom Function is executed with respect to the rest of the program. There are basically two ways in which a CusFn will be executed:

#### 1. Triggered by Ladder Logic Special function coil [CusFn]

A custom function may work the same way as any other special functions in the TRiLOGI ladder diagram programming environment. When you are in ladder circuit editing mode, press <Ins> key to open the "Ins Element" menu.

Select the item 9:  [FUNC] or 0:  [FUNC] to create a special function output. A pop-up "Select a Function" menu will appear.



Select either item:

" D : [CusFn] - Custom created Function" or

" E : [dCusF] - Diff. Up Custom Functn"

to create a CusFn. You will be required to enter the selected custom function number from 1 to 256. Note that CusFn created using

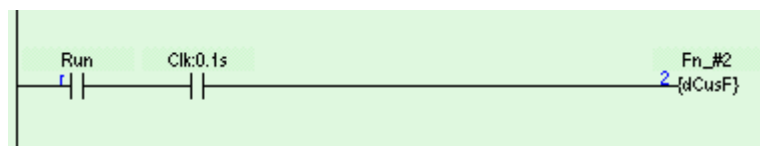
" E :Diff. Up Custom Functn [dCusF]"

is a "Differentiated Up" instruction. This means that the function will be executed only once every time when its execution condition goes from OFF to ON. Nothing will happen when its execution condition goes from ON to OFF.

On the other hand, using "D: Custom created Function [CusFn]" will mean that the CusFn will be executed every scan as long as its execution condition is ON. This is often not desirable and the coil created using this menu item will be highlighted in RED color to serve as an alarm to programmer. You will probably find that you will use the differentiated version [dCusF] far more frequently.

## 2. Periodic Execution of a Custom Function

There are many situations when you need the PLC to periodically monitor an event or perform an operation. For example, to monitor the temperature reading from a probe or check the real time clock for the scheduled time, and to continuously display changing variables on the LCD display. It is not efficient to use the continuous [CusFn] function for such purposes. It is far better to use the built-in clock pulses to trigger a differentiated Custom function [dCusF]. You can choose a suitable period from 0.01s, 0.02s, 0.05s, 0.1s, 0.2s, 0.5s, 1.0s and 1 minute for the application. Other periods can also be constructed with a self-reset timer. The custom function will only be executed once every period controlled by the system clock pulse or the timer, as follow:



For example, you don't need to update the value of a variable displayed on the LCD screen any faster than the human eye can read them. So using a 0.5s clock pulse may be sufficient and this will not take up too much CPU time for the display. For slow processes such as heating, a 1.0s clock pulse to monitor temperature change is more than sufficient.

### IMPORTANT

- i. When the CPU scans the ladder logic to a circuit which contains a CusFn, and the execution condition of the circuit is TRUE, the corresponding CusFn will be immediately executed. This means that the CPU will not execute the remaining ladder circuits until it has completed execution of the current CusFn. Hence if the CusFn modifies a certain I/O or variable, it is possible to affect the running of the remaining ladder program.
- ii. Note that the INPUT[n] variables contain data obtained at the beginning of the ladder logic scan and not the actual state of the physical input at the time of the CusFn execution. Thus, it will be futile to wait for the INPUT[n] variable to change inside a CusFn unless you execute the REFRESH statement to refresh the physical I/O before you examine the INPUT[n] variable again.

- iii. Likewise, any changes to the OUTPUT[n] variable using the SETBIT or CLRBIT statement will not be transferred to the physical outputs until the end of the current ladder logic scan. Hence do not wait for an event to happen immediately after executing a SETBIT or CLRBIT statement on an OUTPUT[n] because nothing will happen to the physical output until the current ladder logic scan is completed.
- iv. If you want to force the output to change immediately you will need to execute the REFRESH statement. Consideration must be given to how such an act may affect the other parts of the ladder program since not the entire ladder program has been executed.
- v. Like all ladder circuits, the relative position of the circuit that triggers the CusFn may affect the way the program works. It is important to consider this fact carefully when writing your ladder program and TBASIC CusFns. Always remember that the CPU executes the ladder logic and CusFn sequentially, even though the equivalent circuits in hard-wired relay may seem to suggest that the different rungs of ladder circuits were to work simultaneously.
- vi. In line with the typical Ladder Logic programming rules, a CusFn may appear only once within the ladder diagram, regardless of whether it appears in the normal or differentiated form. A compilation error will occur if a CusFn appears in more than one circuit.
- vii. However, a CusFn may be "CALLED" as a subroutine by any other CusFn and there is no restriction placed on the number of repeated CALL of a CusFn by more than one CusFn. A CusFn may also modify the logic states of an I/O element or the value of internal timers and counters using its powerful TBASIC commands (such as SetBit, ClrBit). The compiler however will not alarm the user that a CusFn may inadvertently alter the logic state of an I/O already controlled by some other ladder circuit.

This power and flexibility offered by the TBASIC-based custom functions must therefore be handled with greater care by the programmer. It is important to prevent conflicting output conditions due to an I/O being controlled or modified at more than one place within a logic scan. The net result is that the logic state of the I/O appears to be in different states at different parts of the ladder circuit. This could lead to bizarre outcomes that may be difficult to trace and debug.

### 3. Interrupt Service CusFn

A CusFn may also serve as an "Interrupt Service Routine" which is executed asynchronously from the normal ladder logic execution. An interrupt-driven CusFn is run when the condition that causes the interrupt occurs. The response time to execution is very short compared to the scan time of the ladder program. There are several interrupt sources that can trigger a CusFn:

#### a) Special Interrupt inputs

An M-series PLC contains some special "Interrupt" inputs which, when enabled by the INTRDEF statement, will trigger a particular CusFn defined in the INTRDEF statement when the logic level at the interrupt pin changes state (either from OFF to ON or from ON to OFF).

#### b) High Speed Counters (HSC) Reach Target Count

An M-series PLC contains some "High Speed Counter" inputs which, when enabled by the HSCDEF statement, will trigger a particular CusFn defined in the HSCDEF statement when the counter reaches a preset target count value. This enables the CPU to carry out immediate action such as stopping a motor or performing some computation.

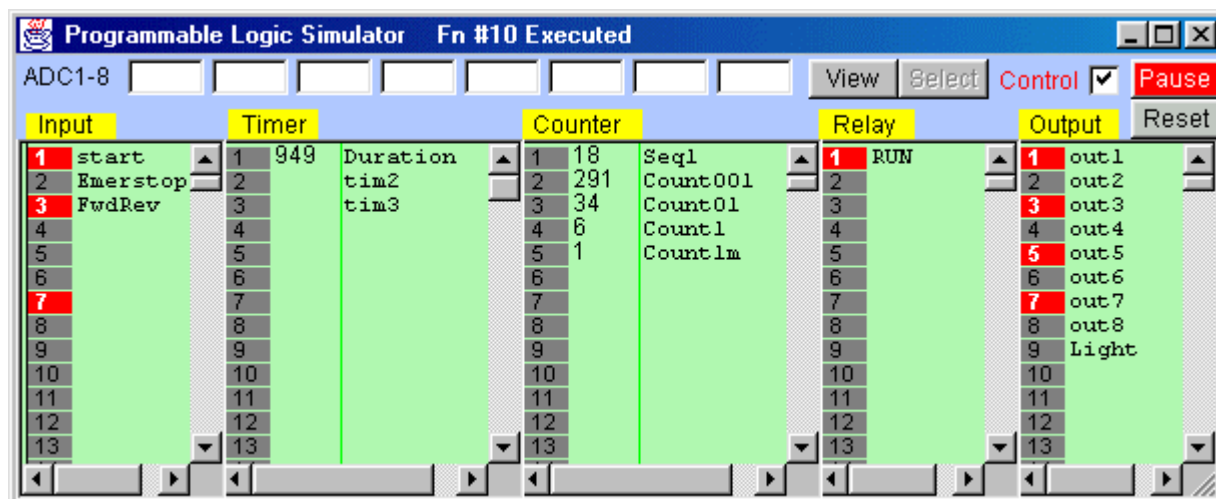
---

## IV. Simulation & Examination of TBASIC Variables

---

### 1. Simulation Run of CusFn.

TRiLOGI fully supports simulation of all TBASIC commands. After you have completed coding a CusFn, test the effect of the function by connecting it to an unused input. Run the simulator by pressing <F9> or <Ctrl-F9> key. Execute the CusFn by turning ON its control input. If your CusFn executes a command that affects the logic state of any I/O, the effect can be viewed on the simulator screen immediately.



However, if the computation affects only the variables, than you may need to examine the internal variables.

An I/O or internal relay bit that has been turned ON is indicated by a RED color rectangular lamp, which simulates a LED being turned ON. You can pause the logic simulator at any time by pressing the <Ctrl-P> key or clicking on the [Pause] button. Likewise the simulator engine can be reset by clicking on the [Reset] button.

## 2. Simulation of ADC Inputs

Along the top edge of the Programmable Logic Simulator screen, you will find 8 text fields adjacent to the label "ADC1-8". The programmer can enter the expected ADC values for ADC#1 to #8 in these text fields. In effect, these simulate the potential signal strength at their respective ADC input pins. These values will be captured by the TBASIC program when an ADC(n) command is executed in a custom function for ADC #n.

Note: values entered at the ADC input text field will only be updated when the user press the <Enter> key or the <TAB> key to ensure that only finalized entries are used by the TBASIC program. (Otherwise, imagine if you try to enter the value 123 at ADC #1, the program would first be receiving "1", then "12" and then "123" which was not the intention).

## 3. Viewing TBASIC Variables

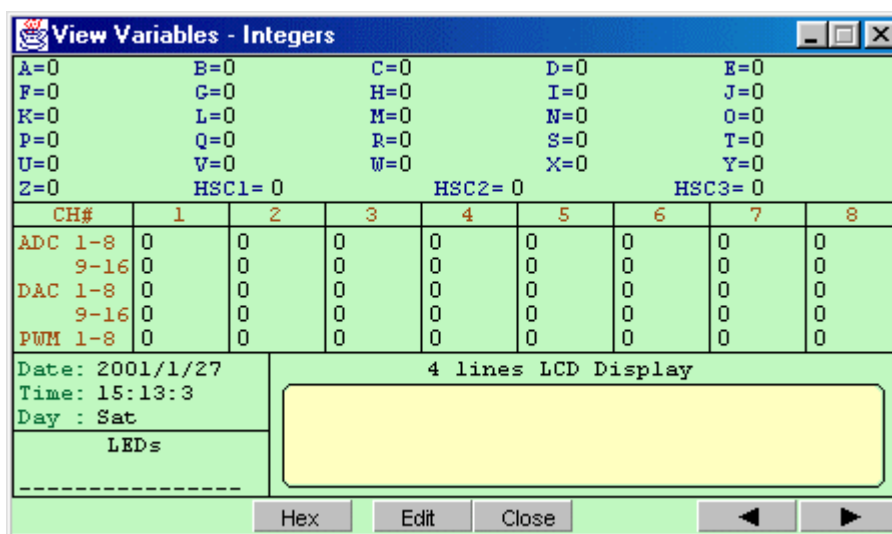
The values of the internal variables as a result of the simulation run can be viewed by pressing the <V> (which stand for "View") key or by clicking on the [View] button while in the simulation screen. A pop-up window will appear with the values of all the variables as well as special peripheral devices supported by TBASIC. The variables are organized into 4 screens. You can move from screen to screen using the left/right cursor keys or by clicking on the navigation buttons:

### Integer variables Screen

The first screen comprises all 26 32-bit integer variables A-Z, the system DATE and TIME, ADC, DAC, PWM and the resulting values of setLED and setLCD commands. The initial DATE and TIME figures shown during simulation are taken from the PC's internal real-time clock values. However, subsequent values can be affected by the values assigned to the variable DATE[n] and TIME[n].

The present values of the first 3 high-speed counters: HSC1 to HSC3 are also shown on this page. Note that ADC data for any particular A/D channel #n will only be shown if an ADC(n) function has been

executed. Otherwise the ADC value shown on screen will not reflect the true current value of the ADC port.



### Data Memory Screen

The second screen displays, in 25 pages, the values of the 16-bit DM variables from DM[1] to DM[4000]. Each page displays 16 rows x 10 columns = 160 DM variables. You can scroll up and down the pages by clicking on the [PgUp] or [PgDn] buttons or using the corresponding keys on the keyboard.

### String Variable Screen

The third screen displays the value of the 26 string variables A\$ to Z\$ in 4 pages, depending on the length of each string. If the execution condition is ON and the CusFn is not of the differentiated type, then the CusFn will be continuously executed. The result of the variable will be continuously updated on the viewing window.

### System Variable Screen

System variables such as INPUT[n] , RELAY[n] and emINT[n] are visible in this screen. You may wish to click on the [Hex] button to view the values in hexadecimal notation as this is more commonly used by programmers to identify the bit patterns in these variables.

## **4. Changing The Contents of Variables**

While the "View Special Variables" window is open, you may change the contents of the following variables by clicking on the [Edit] button:

A-Z, A\$ to Z\$, DM[n], DATE[n], TIME[n], INPUT[n], OUTPUT[n], RELAY[n], TIMERBIT[n], CTRBIT[n], TIMERPV[n], CTRPV[n] and HSCPV[n], emINT[n], emLINT[n].

A text entry window will pop up and you will have to enter the values in the form of assignment statements, such as:

```
e.g. A = 5000;  
DM[99]=5678;  
OUTPUT[2]=&H01AB  
B$ = "Welcome to TBASIC"
```

The variable will take up the new value as soon as it is entered, and if the execution condition for any CusFn is ON, the simulator will process the newly entered data immediately and produce the new outcomes. This gives you greater flexibility in controlling the simulation process.

## 5. Decimal and Hexadecimal Representation

All the numeric data shown in the "Special Variables" window are by default displayed in decimal notation. You can display the number in hexadecimal format by clicking on the [Hex] button or by pressing the <H> key. Press the <D> key if you wish to switch back to the decimal format. This feature is very useful for programmers who are familiar with hexadecimal representation of a binary number. The [Hex] button will become the [Dec] button when you enter the Hex display mode.

---

## V. On-Line Monitoring of TBASIC Variables

---

If you execute the "On-Line Monitoring/Control" command from the "Controller" pull-down menu, TRiLOGI Version 5 will continuously query the PLC for the values of all their internal variables. These variables' values will be updated in real time in the "View Special Variables" window. You may also alter the value of any variables in the PLC using the "Edit Variable" window (by clicking on the "Edit" button at the "View xxx Variables" window.

This ability of TRiLOGI to provide instant and full visibility of all the PLC's internal variables greatly facilitates the programmers' debugging process. The ease of programming offered by the TRiLOGI programming environment is really what really sets the M-series PLCs far ahead of many other PLCs where both programming and debugging are really painstaking tasks. (This is assuming they have been fully equipped with all the expensive "options" to match the M-series built-in capability!)

### 1. PAUSE and RESET of Target PLC

During On-Line Monitoring, if the "View Special Variables" window is opened, you can still reset the PLC's internal data by pressing the <Ctrl-R> key. Pressing the <P> key can halt the PLC. A halted PLC can

subsequently be released from the halted mode by pressing the <P> key again.

## 2. Using LCD Display for Debugging

You should take advantage of the built-in LCD display port of the T100MD to display internal data at the location where you want to track their values, especially if the value changes rapidly which may not be constantly captured by on-line monitoring screen.

---

## VI. Error Handling

---

Since the CusFn text editor does not restrict the type of text that may be entered into its editor, the TRiLOGI compiler will have to check the syntax of the user's TBASIC program to look out for misspelling, missing parameters, invalid commands, etc. Such errors, which can be tracked down during compilation process, are known as "Syntax Errors".

### 1. Syntax Error

TRiLOGI employs a sophisticated yet extremely user-friendly syntax error tracking system: When a syntax error is encountered, the compilation will be aborted immediately and the CusFn, which contains the error, is automatically opened in the text editor. The location of the offending word is also highlighted and a pop-up message window reports to you the cause of the error. You can then immediately fix the error and re-compile until all the errors have been corrected.

Error Message	Cause / Action
Undefined symbol found	Only TBASIC commands and legal variable names are allowed. See Chapter 3.
Compiler internal error	Serious trouble, please email to the manufacturer support@tri-plc.com to inform us.
" ) " found without matching " ( "	-
Integer expected	Expect to see either an integer variable or integer constant.
Value is out-of-range	Check the language reference for allowable range of values for the command.
Duplicate line label number	Label for GOTO must be unique within the same CusFn.
Undefined GOTO destination:	Put a matching label at the place where the GOTO statement is supposed to go.

Invalid GOTO label	@# must be in the range 0-255
Type mismatch (numeric and string types may not mix)	In an expression, strings and integers may not be mixed unless converted using the conversion function. e.g. STR\$, VAL, etc.
String is too long	A string is limited to 70 characters
Too many line labels	There should not be more than 20 GOTO labels within the same CusFn.
Unknown Keyword	Most likely wrong spelling for TBASIC statement or function.
WHILE without ENDWHILE	Every WHILE statement must be ended with a matching ENDWHILE statement. Nested WHILE loop must have proper matching ENDWHILE for each WHILE.
IF without ENDIF	Every IF statement must be ended with a matching ENDIF statement to define the boundaries for the block controlled by the IF statement. For multiple IF THEN statement, each IF must be matched by a corresponding ENDIF.
FOR without NEXT	Every FOR statement must be ended with a matching NEXT statement to define the boundaries for the block controlled by the FOR statement. For nested FOR loops, each FOR must be matched by a corresponding NEXT.
Expect keyword "TO"	Required by FOR statement.
Must be an integer	String variable or constant not allowed.
Must be an integer variable only	Integer constant not allowed.
Must be an integer constant only	Integer variable not allowed.
Must be a string	Integer constant or variable not allowed.
Must be a string variable only	String constant not allowed.
Must be a string constant only	String variable not allowed.
Incomplete Expression	Expression not ended properly.
String constant missing closing "	String constants must be enclosed between a pair of opening and closing quotation character (")
Must be Integer A to Z only	index for FOR..NEXT loop must be A-Z.

## 2. Run-Time Errors

Certain errors only become apparent during the execution of the program, e.g.  $A = B/C$ . This expression is perfectly OK except when  $C = 0$ , then you would have attempted to divide a number by zero, which does not yield any meaningful result. In this case a "run-time error" is said to have occurred. Since run-time errors cannot be identified during compilation, TRiLOGI also checks the validity of a command during simulation run and if a run-time error is encountered, a pop-up message window will report to the programmer the cause and the CusFn where the run-time error took place. This helps the programmer locate the cause of the run-time errors to enable debugging. The possible run-time errors are listed in the following table and they are generally self-explanatory.

Run-Time Error Message
Divide by zero
Call stack overflow! Circular CALL suspected!
FOR-NEXT loop with STEP = 0!
SET_BIT position out-of-range!
CLR_BIT position out-of-range!
TEST_BIT position out-of-range!
STEPSPEED channel out-of-range!
Illegal Pulse Rate for STEPMOVE!
Illegal acceleration for STEPMOVE!
STEPSMOVE channel out-of-range!
STEPSTOP channel out-of-range!
ADC channel out-of-range
DAC channel out-of-range
LED Digit # within (1-12) Only!
PWM Channel out-of-range!
LCD Line # must be (1-4) Only!
PM channel out-of-range!
System Variable Index Out-of-range!
Shifting of (A-Z) Out-of-range!
Illegal Opcode - Please Inform Manufacturer!
Timer or Counter # Out-of-Range!

# Chapter 10: TBASIC Statements, Functions, Operators and Variables

---

## I. What are TBASIC Statement and Functions?

---

### 1. STATEMENT

A STATEMENT is a group of keywords used by TBASIC to perform certain action. A statement may take 0,1,2 or more arguments. The following are some TBASIC statements: PRINT, LET, IF, WHILE, SETLED ...etc.

### 2. FUNCTION

A FUNCTION acts on its supplied arguments and return a value. The returned value may be an integer or a string. A function can usually be embedded within an expression as if it is a variable or a constant, since its content will be evaluated before being used in the expression. E.g.

A\$ = "Total is \$" + STR\$(B+C)

STR\$(n) is a function which returns a string and therefore can be used directly in the above string assignment statement.

The most distinguishable feature of a FUNCTION is that its arguments are enclosed within parenthesis "(" and ")". e.g. ABS(n), ADC(n), MID\$(A\$,n,m), STRCMP(A\$,B\$).

**Note:** Statements or functions and their arguments are **NOT** case-sensitive. This means that commands such as PRINT and PriNt are identical. However, for clarity seeks we use a mix of upper and lower case characters in this manual.

### 3. DELIMITER

A TBASIC program consists of many statements. Each statement usually falls on a different line. The new line therefore acts as a "delimiter" which separates one statement from another. Some statements such as IF..THEN..ELSE..ENDIF span multiple statements and should be separated by proper delimiters.

To make a program visually more compact, the colon symbol ":" may be used to act as delimiter. E.g.

```
IF A > B THEN
  C = D*5
ELSE
  C = D/5
ENDIF
```

may be written more compactly as

```
IF A > B : C=D*5:ELSE:C=D/5:ENDIF
```

---

## II. TBASIC Integer Constants, Variables & Operators

---

The TBASIC compiler in TRiLOGI Version 5 supports full 32-bit integer computations. However, only variable A to Z are 32 bits in length which allow them to represent number between  $-2^{31}$  to  $2^{31}$ , the remaining system variables and data memory DM[n] are all 16-bit variables which means that they can only store number between -32768 to +32767. However, all numerical computations and comparisons in TBASIC are carried out in 32-bit signed integer, regardless of the bit-length of the variables involved in the numerical expression.

### 1. Integer Constants

These may be entered directly in decimal form, or in *hexadecimal* form by prefixing the number with the symbol "&H". e.g.

```
12345678
&H3EF =1007 (decimal)
```

If the result of an expression is outside the 32-bit limits, it will overflow and change sign. Care must therefore be exercised to prevent unexpected result from an integer-overflow condition.

A constant may be used in an assignment statement or in an expression as follow:

```
A = 12345
IF A*30 + 2345/123 > 100
THEN ....ENDIF
```

## IMPORTANT (16-bit variables comparison)

When entering an integer constant using the hexadecimal prefix "&H", it is important to note the sign of the intended value and extend the signs to most significant bit of the 32 bit expression. E.g. to represent a decimal number "-1234", the hexadecimal representation must be "&HFFFFFFB2E" and not "&HFB2E".

Assuming that a 16-bit variable DM[1] contains the number -1234 and a comparison statement is made to check if the number is -1234. The 32-bit hexadecimal representation of constant -1234 is &HFFFFFFB2E. If you enter the constant as 16-bit representation "&HFB2E" as follow:

```
IF DM[1] <> &HFB2E CALL 5
```

TBASIC translates the number "&HFB2E" into a 32-bit decimal number 64302, which when compared to the number "-1234" contained in DM[1] will yield a "False" result which is an error. The following are the correct representation:

- a) IF DM[1] <> -1234 CALL 5 : ENDF
- b) IF DM[1] <> &HFFFFFFB2E" CALL 5: ENDF

## 2. Integer Variables

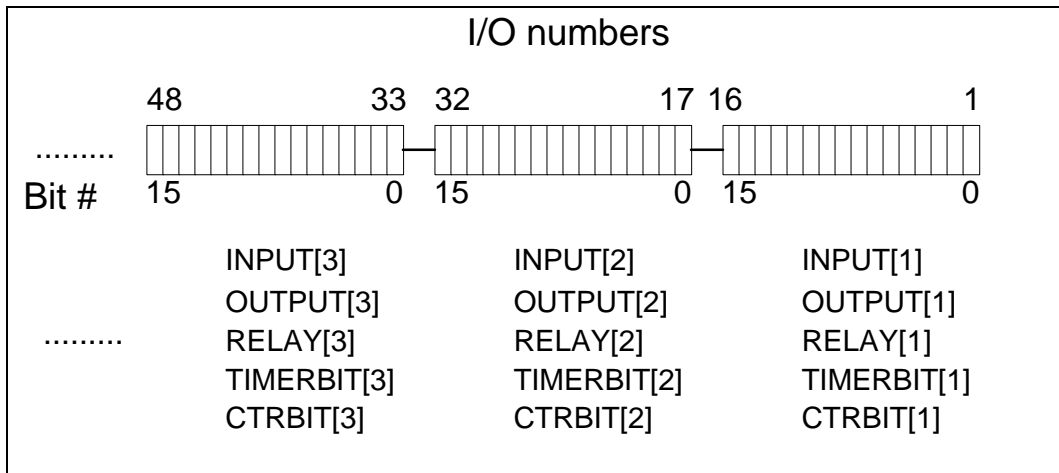
Variables are memory locations used for storing data for later use. *All Integer variables used in TBASIC are **GLOBAL** variables* - this means that all these variables are shared and accessible from every custom function.

TBASIC supports the following integer variables:

- i. 26 Integer variables A, B, C....Z which are 32-bit variables. Note that the variable name must be a single character.
- ii. A large, one-dimensional 16-bit integer array from DM[1] to DM[4000], where DM stands for Data Memory. A DM is addressed by its index enclosed between the two square brackets "[" and "]". e.g. DM[3], DM[A+B\*5], where A and B are integer variables.
- iii. System variables. These are special integer variables which relates to the PLC hardware, as follow:

## Inputs, Outputs, Relays, Timers and Counters Contacts

The bit addressable I/Os elements are organized into 16-bit integer variables INPUT[n], OUTPUT[n], RELAY[n], 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:



## Timers and Counters Present Values

The present values (PV) of the 128 timers and 128 counters in the PLC can be accessed directly as system variables:

timerPV[1] to timerPV[256], for timers' present value  
 ctrPV[1] to ctrPV[256], for counters' present value

## DATE and TIME Variables

The PLC's Real-Time-Clock (RTC) derived date and time can be accessed via variables DATE[1] to DATE[3] and TIME[1] to TIME[3], respectively as shown in the following table:

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]		

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

## High Speed Counters

The M-series PLC support High Speed Counters (HSC) that can be used to capture high frequency incoming pulses from positional feedback encoder. These high speed counters are accessible by CusFn using the variables HSCPV[1] to HSCPV[8]. All HSCPV[n] are 32-bit integer variables.

### **3. Integer operators:**

Operators" perform mathematical or logical operations on data. TBASIC supports the following integer operators:

#### i) Assignment Operator:

An integer variable (A to Z, DM and system variables, etc) may be assigned a value using the assignment statement:

```
A = 1000
X = H*I+J + len(A$)
```

#### ii) Arithmetic Operators:

Symbol	Operation	Example
+	Addition	A = B+C+25
-	Subtraction	Z = TIME[3]-10
*	Multiplication	PRINT #1 X*Y
/	Division	X = A/(100+B)
MOD	Modulus	Y = Y MOD 10

#### iii) Bitwise Logical Operators: logical operations is perform bit-for-bit between two 16-bit integer data.

Symbol	Operation	Example
&	logical AND	IF input[1] & &H02 ...
	logical OR	output[1] = A   &H08
^	Exclusive OR	A = RELAY[2] ^ B
~	logical NOT	A = ~timerPV[1]

#### iv) Relational Operators : Used exclusively for decision making expression in statement such as IF *expression* THEN ..... and WHILE *expression* ....

Symbol	Operation	Example
=	Equal To	IF A = 100
<>	Not Equal To	WHILE CTR_PV[0]<> 0
>	Greater Than	IF B > C/(D+10)
<	Less Than	IF TIME[3] < 59
>=	Greater Than or Equal To	WHILE X >= 10
<=	Less Than or Equal To	IF DM[I] <= 5678
AND	Relational AND	IF A>B AND C<=D
OR	Relational OR	IF A<>0 OR B=1000

v) Functional Operators : TBASIC supports a number of built in functions which operate on integer parameters as shown below:

*ABS(n), ADC(n), CHR\$(n), HEX\$(n), STR\$(n)*

#### 4. Hierarchy of Operators

The hierarchy of operators represents the priority of computation. E.g.  $X = 3 + 40*(5 - 2)$ . The compiler will generate codes to compute  $5 - 2$  first because the parentheses have the higher hierarchy. The result is then multiplied by 40 because multiplication has a higher priority than addition. Finally 3 will be added to the result. If two operators are of the same hierarchy, then compiler will evaluate from left to right. e.g.  $X = 5 + 4 - 3$ .  $5+4$  is first computed and then 3 will be subtracted. The following table list the hierarchy of various operator used.

Hierarchy	Symbol	Descriptions
Highest	( )	Parentheses
	*, / , MOD	Multiplication/Division
	+, -	Add/Subtract
	-	Negate
	&,  , ^, ~	Logical AND,OR,XOR,NOT
Lowest	=,<>,>,>=,<,<=	Relational operators

---

### III. String Variables and Constant

---

A string is a sequence of alphanumeric characters (8-bit ASCII codes) which collectively form an entity.

#### 1. String Constants

A string constant may contain from 0 to 70 characters enclosed in double quotation marks. e.g.

```
"TBASIC made PLC numeric processing a piece of cake!"  
"$102,345.00"
```

#### 2. String Variables

TBASIC supports a maximum of 26 string variables A\$, B\$ ... Z\$. Each string variable may contain from 0 (null string) up to a maximum of 70 characters.

**Note:** For M-series PLC with firmware version r45 and above, you can access the 26 string variables using an index: \$\$[1] to \$\$[26]. I.e. A\$ is the same as \$\$[1], Z\$ is the same as \$\$[26]. Note that \$\$[1] to \$\$[26] are not additional string variables, it just give you a way to index the string variables not possible on previous firmware version. Also, only TRiLOGI version 5.2 and above properly support these variable names. **Caution:** Do not try to transfer a program using \$\$[n] variable to a PLC with firmware earlier than r45 as it can cause the PLC operating system to crash.

#### 3. String Operators

i) Assignment Operator: A string variable (A to Z, DM and system variables, etc) may be assigned a string expression using the assignment statement:

```
A$ = "Hello, Welcome To TBASIC"  
Z$ = MID$(A$,3,5)
```

ii) Concatenation Operators: Two or more strings can be concatenated (joined together) simply by using the "+" operator. e.g.

```
M$ = "Hello " + A$ + ", welcome to " + B$
```

If A\$ contains "James", and B\$ contains "TBASIC", M\$ will contain the string: "Hello James, welcome to TBASIC".

iii) Comparison Operator: Two strings may be compared for equality by using the function STRCMP(A\$,B\$). However, the integer comparator such as "=", "<>", etc cannot be used for string comparison.

iv) Functional Operators: **TBASIC** supports a number of statement and functions which take one or more string arguments and return either an integer or a string value. e.g.

LEN(x\$),                      MID\$(A\$,x,y),                      PRINT #1 A\$,....  
SETLCD 1, x\$VAL(x\$)

Please refer to the next chapter for detailed descriptions of these operators.

---

## IV. Special Variables – EMINT, EMLINT & EMEVENT

---

The T100MD+ and T100MX+ PLCs are originally designed to be used with a third-party server software named "EMIT 3.0", made by emWare Inc. of Salt Lake City, USA. However, we have decided to depart from the EMIT platform for all future PLC design and we no longer support the later version of EMIT server.

Anyway, there are some special variables set aside for data exchange with the EMIT and these may now be freely used as additional memory in your TBASIC program:

- a) emInt[1] to emInt[16]: These are 16 bit unsigned integer variables.
- b) emEvent[1] to emEVENT[16]: These are 16 bit unsigned integer variables. EmEVENT[1] is also used for email purpose.
- c) emLInt[1] to emLInt[16]: These are 32-bit unsigned integer variables.

Note that since EMLINT[1] to EMLINT[16] form an integer array, these **32-bit integer variables** can be easily addressed by an index such as EMLINT[n] which is not possible on the 32-bit integer variables A to Z.

## Chapter 11: TBASIC Keyword Reference

### **ABS(x)**

Purpose : To return the absolute value of the numeric expression *x*

Examples : **A = ABS(2\*16- 100)**

Comments : *A should contain the value 68.*

---

### **ADC(n)**

Purpose : To return the value from the Analog-To-Digital Converter channel #*n*. *n* should be between 1 and 16.

Examples : **A = ADC(2)**

Comments : *n may be a numeric expression which returns a value between 1 and 16. If it is out-of-range, a run-time error will be reported and the function will be aborted.*

*TRiLOGI software is able to support up to 16 channels of 16-bit bipolar ADC (which may has a range between -32768 to 32767. The actual number of ADC channels and the resolution will depend on the target PLC. On the T100MD+ and T100MX+, all the A/D are normalized to 12-bit with a range of between 0 and 4096.*

---

### **ASC(x\$, n)**

Purpose : To return the numeric value that is the ASCII code for the *n*th character of the string *x\$*. If *x\$* is a null string, ASC(*x\$,n*) returns value 0. *n* may start from 1 up to the length of the string.

Examples : **B = ASC("Test String", 6)**

Comments : *B should contain the value 83 (which is ASCII value of 'S'). If n is less than 1 or greater than string length, ASC(x\$, n) returns a 0.*

See Also : **CHR\$(n)**

---

### **CALL n**

Purpose : To call another Custom Function CusFn #*n* as subroutine. When the called function returns, execution will continue from the following statement. *n* must be either an integer constant between 1 and 256, or the label name of the Custom Function defined in the Custom Function table.

Examples : **IF B > 5 THEN CALL 8 : ENDIF**  
**CALL Addition** "Addition" must be a defined name.

See Also : **RETURN**

---

## **CHR\$(n)**

Purpose : To convert a number *n* into its corresponding ASCII character. *n* must be a numeric constant (0 to 255)

Examples : **C\$ = "This is Message #" + CHR\$(&H35)**

Comments : *C\$ should contain: "This is Message #5", since CHR\$(&H35) returns the character '5'.*

See Also : **ASC( )**

---

## **CLRBIT v, n**

Purpose : To clear the Bit #*n* of the integer variable *v* to '0'. *n* is an integer constant or variable of value between 0 and 15. *v* may be any integer variable or a system variable such as relay[*n*], output[*n*], etc. If *v* is a 32-bit integer, CLRBIT will only operate on the lower 16 bits.

Following digital electronics convention, bit 0 refers to the least significant bit (right most bit) and bit 15 the most significant bit (left most bit) of the 16-bit integer variable. A quick way to find out the bit position and index of an I/O variable is to open their I/O table and check the "CH:BIT" column. Bit positions beyond 9 are represented by hexadecimal number A to F.

Examples : **CLRBIT output[2], 11**

Comments : *Physical output #28 will be turned OFF.  
(Output channel #2 bit #11 = Output #17 + 11 = 28)*

See Also : **SETBIT, TESTBIT, SETIO, CLRIO, TOGGLEIO & TESTIO**

---

**CLRIO** *labelname*

**SETIO** *labelname*

**TOGGLEIO** *labelname*

**TESTIO** (*labelname*)

Purpose : Manipulate the logic states of any input, output, relay, timer or counter contact bit within a CusFn. The *labelname* refers to the label names defined in the input, output, relay, timer or counter tables.

**SETIO** set a bit to ON, **CLRIO** clear the bit to OFF, and **TOGGLEIO** flip the current logic state of that I/O bit. **TESTIO** function returns a 1 if the bit is ON and a 0 if the bit is OFF.

E.g. **SETBIT alarm**  
**IF TESTBIT(alarm) THEN ... ELSE ...ENDIF**

Comments *This function offers a more efficient way of manipulating the I/O bits compared to the SETBIT and CLRBIT function. However, SETBIT and CLRBIT*

functions have the advantage that they can use variables to indicate the index and bit position of the bit to be affected, whereas the I/O bit that affected by the commands here are fixed during compile time. **Note** that output bit changed in custom function will only be updated at the physical output at the end of the ladder logic scan unless a “REFRESH” command is being executed.

See Also : **SETBIT, CLRBIT**

---

\* **CRC16** (*var, count*) { \* Applicable only to PLC with firmware **r44** or higher }

Purpose : This function returns the computed CRC16 for a range of integers starting from variable “*var*” with the range indicated in the parameter “*count*”. CRC16 is a 16-bit version of “Cyclic Redundancy Check” – a popular mathematical formula for checking error in a data stream.

Examples : **DM[ 100 ] = CRC16(DM[ 5 ], 8)**  
**X = CRC16(RELAY[ 2 ], 4)**

Comments : CRC16 for DM[5], DM[6]....DM[12] will be assigned to DM[100]  
CRC16 for RELAY[2], RELAY[3], RELAY[4] will be assigned to X.

---

## **DELAY** *n*

Purpose : To provide a time delay of *n* millisecond to the process.

Example : **DELAY 100**

Comments : Provide a 100 ms (0.1s) delay to the current custom function.

It is important to note that this is a “brute force” delay method and only to be used with caution. When a DELAY function is executed the CPU waits at the statement until the period specified by the “delay” is over. This means that all the remaining ladder programs and other custom functions will stop responding to changing input conditions, only system services (serial input, countdown timers and host link commands etc) as well as interrupt driven CusFns will work during the period of delay. This may not be desirable if the rest of the process must respond to fast changing inputs. For delays longer than 0.1s a much better way is to invoke the regular PLC timer and use the timer contact to trigger another custom function at the end of the delay.

For the T100MD+ and T100MX+, the minimum delay provided by this function is 10ms, and the resolution of the time delay is 10ms. This means if you execute **DELAY 155** the actual delay will be rounded to 160ms, and for **DELAY 154** the actual time delay will be 150ms.

---

## FOR ... NEXT

Purpose : To execute a series of instructions for a specified number of times in a loop.

Syntax : **FOR** *variable* = *x* **TO** *y* [**STEP** *z*]

. . . .

### **NEXT**

where *variable* may be any integer variable A to Z only and is used as a counter. *x*, *y* and *z* are numeric expressions. STEP *z* is an optional part of the statement.

*x* is the initial value of the counter, *y* is the final value of the counter.

Program lines following the **FOR** statement are executed until the **NEXT** statement is encountered. Then the counter is incremented by the amount specified by **STEP**. If **STEP** is not specified, the increment is assumed to be 1.

A check is performed to see if the value of the counter is greater than the final value *y* if **STEP** is positive (or smaller than the *y* if **STEP** is negative). If it is not greater, the program branches back to the statement after the **FOR** statement, and the process is repeated. If it is greater, execution continues with the statement following the **NEXT** statement. This is called a **FOR-NEXT** loop.

A run-time error will result if **STEP** is evaluated to be 0.

Examples :       FOR I=1 TO 10  
                  FOR J = 100 to 1 STEP -10  
                  DM[ I ] = DM[ J ]  
                  NEXT  
                  NEXT

Comments : *FOR-NEXT loops may be nested; i.e. a FOR-NEXT loop may be placed within the context of another FOR-NEXT loop. When loops are nested, each loop must have a unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop. Each Loop must have a separate NEXT statement to mark the end of the loop.*

See Also : **WHILE ... ENDWHILE**

### **GetCtrSV (n)**

### **GetTimerSV (n)**

Purpose : Return the **Set Value (S.V.)** of the Counter #*n* or Timer #*n*.  
*n* should be between 1 and the maximum number of timers and counters in your PLC.

Note : Although the present values (P.V.) of timers and counters #n can be accessed directly as variables "TimerPV[n]" & "CtrPV[n]", the Set Values however can only be obtained by these two functions.

See Also : **SetCtrSV, SetTimerSV**

---

## **GETHIGH16(v)**

Purpose : This function returns the upper 16-bit of a 32-bit integer variable *v*. This can be used to break the value of a 32-bit integer data or variable into two 16-bit values so that they can be saved to the EEPROM or to the DM[n].

Examples : **DM[ 1] = GetHIGH16(A)**  
**save\_EEP GetHIGH16(&H12345678), 10**

See Also : **SETHIGH16**

---

## **GOTO @n**

Purpose : To branch unconditionally out of the normal program sequence to a specified line with label @*n* within the present Custom Function.

The destination line must have a corresponding line label marked as "@*n*", where *n* must be a constant within 0-255. Note that the label is local only to the present CusFn. i.e. another CusFn may have a label with the same *n* but the GOTO @*n* will only branch to the line label within the same CusFn.

Examples : **@156 SETBIT 0, 3**  
**. . .**  
**GOTO @156**

Comments : *An error message will appear during compilation if the destination label is undefined.*

---

## **HEX\$(n)**

### **HEX\$(n, d)**

Purpose : To return a string that represents the hexadecimal value of the numeric argument *n*. If the second format is used then this function will return a string of '*d*' number of characters.

Examples : **A\$ = HEX\$(1234)**  
**B\$ = HEX\$(1234, 7)**

Comments : *A\$ will contain the string : "4D2", B\$ will contain the string "00004D2".*

See Also : **HEXVAL(), STR\$( ), VAL( )**

## HEXVAL(x\$)

Purpose : To return the value of a hexadecimal number contained in the argument x\$.

Examples : **B = HEXVAL("123") \* 100**

Comments : *B should contain the value 29100 (&H123 = 291)*

See Also : **HEX\$( ), STR\$( ), VAL( )**

---

## HSCDEF *ch, fn\_num, value*

Purpose : Enable and set up parameters for the High Speed Counters channel *ch*. These counters operate independently of the ladder logic scan time and can capture high speed input pulses generated by position encoders.

*ch* = channel number (1-8)

*fn\_num* = Custom Function # to trigger when value is reached.

*value* = trigger when HSC reach this (32-bit) integer value.

If the PLC supports quadrature encoder inputs, then the HSC counter variable **HSCPV[*ch*]** will increment/decrement according to direction of rotation. When *value* is reached, the specified custom function activates immediately.

Important : All High Speed Counters are disabled automatically when the PLC is reset unless they are enabled by the **HSCDEF** statement. However, if more than one HSCDEF for the same channel *ch* is executed, only the last executed HSCDEF statement will take effect. Hence you should put the next HSCDEF statement within the CusFn triggered by the first HSCDEF. By chaining the HSCDEF statement from one CusFn to another, you can control the motion of the machine using the HSC value to execute a series of CusFn one by one. Within these CusFn you can program what to do to control the motion. E.g. changing the speed, putting on the brake, change direction of motion, etc. You can use the SETIO, CLRIO for digital ON/OFF control and setDAC, setPWM for proportional control.

Example : **HSCPV[1] = 0**  
**HSCDEF 1, 19, -3310003**  
.  
.  
.  
**SETLCD 1, 1, STR\$(HSCPV[1], 6)**

Comments : *Enable High-Speed Counter #1 and make it activate function #19 when the counter reaches -33,100,003. Present value of HSC#1 was cleared to 0 before activating it. Note that TRiLOGI Version 5.x does not*

*perform simulation of the High Speed counter operation since there is no High Speed Counter inputs on the simulator screen.*

See Also : **HSCOFF**

---

### **HSCOFF** *ch*

Purpose : Disable High Speed Counter #*ch* (*ch* = 1 to 8)

If you no longer need the high speed counter, it should be disabled in order not to waste the CPU's time to service the interrupt generated by the change of state at the HSC input..

---

### **HSTIMER** *n*

Purpose : To define PLC Timer #1 to #*n* as "High Speed Timers" (HST). A HST counts down every 0.01s instead of every 0.1s for normal timer, and their other properties are identical to normal timer. Those Timers whose number are above *n* are not affected and remain ordinary timers.

---

### **IF .. THEN .. ELSE .. ENDIF**

Purpose : To make a decision regarding program flow based on the result returned by an expression.

Syntax :           **IF** *expression* **[THEN]**  
                          .....  
                          **[ELSE]**  
                          .....  
                          **ENDIF**

If the result of the expression is non-zero (logical true), the block of program lines between the **THEN** and the **ELSE** statements will be executed. If the result of the expression is zero (false), the block between the **IF** and **ELSE** will be ignored, and the block between the **ELSE** and **ENDIF** statements will be executed instead.

If there is no ELSE statement, and if the result of the expression is false, the block of program lines between the THEN and the ENDIF statement will be ignored, but execution will continue right after the ENDIF statement.

#### Nesting of IF statement

Statement blocks within the IF..THEN..ELSE statement may contain other IF..THEN..ELSE blocks (nesting). Note that each IF statement must be ended with the ENDIF statement. Otherwise an error message "IF without ENDIF" will be reported during compilation.

Testing Equality: Special comparison operators may be used in the expression of the IF statement. Only integer expression may be compared. For comparison of strings, please refer to the "STRCMP(A\$, B\$)" function.

Equal	=
Not Equal	<>
Greater than	>
Less than	<
Greater than or Equal to	>=
Less than or Equal to	<=

Examples : **IF A >= B\*5- 20\*C OR C=20**  
**B = B- 1**  
**ELSE**  
**B = B\*3**  
**ENDIF**

Comments : *A few comparison expressions may be linked with logical-AND (AND statement) or logical-OR (OR statement) operator as shown in the above examples.*

## **INCOMM(ch)**

Purpose : To return a single 8-bit binary data obtained from Comm. channel # *ch*.

*ch* must be a numeric constant between 1 and 8. The actual target hardware determines the valid port #. This function returns -1 if there is no data waiting at serial port.

Example : **FOR I=1 to 100**  
**DM[I] = INCOMM(2) :**  
**IF DM[I] < 0 RETURN : ENDIF**  
**NEXT**

Comments : *Usually the PLC buffers the serial data arriving at its COMM port so that the program does not need to continuously check the COMM port for data. When the program is ready to process the data it can use the FOR..NEXT loop shown in the above example to read in all the data in the COMM buffer until it encounters a -1, which indicates that the buffer is empty.*

**Note:** **INCOMM is now supported on all COMM ports of T100MD1616+ and T100MX+ families of PLCs.**

See Also : **OUTCOMM, INPUT\$( ), PRINT #**

## **INPUT\$(ch)**

Purpose : To return a string obtained from communication port # *ch*.  
*ch* must be a numeric constant between 1 and 8. The actual target hardware determines the valid port #. This function returns f0 if there is no valid string waiting at serial port.

Example : **D\$ = INPUT\$(2)**

Comments : A Carriage Return (CR) or ASCII code 13 marks the end of the input string from the communication port. The returned string however will exclude the CR character. In TRILOGI simulator, the user will be prompted to enter the string in a pop-up window.

See Also : **INCOMM( ), PRINT #, OUTCOMM**

---

## **INTRDEF ch, fn\_num, edge**

Purpose : Enable Interrupt Input channel #*ch*.  
*ch* = channel number (1-8)  
*fn\_num* = Custom Function number to execute when interrupt pin changes according to the defined edge. This is the Interrupt Service Routine ISR.  
*edge* = Positive number means rising edge-triggered.  
0 or negative number means falling-edge triggered.

See Also : **INTROFF**

---

## **INTROFF ch**

Purpose: Disable Interrupt Input channel # *ch*.

See Also : **INTRDEF**

---

## **LEN (x\$)**

Purpose : To return the number of characters in *x\$*.

Examples : **L = LEN("This is a test string"+CHR\$(13))**

Comments : *L = 22* because blanks and non-printing characters are counted.

---

## **LET**

Purpose : To assign the value of an expression to a variable

Syntax : **[LET] variable = expression**

Examples : **LET D = 11**  
**A\$ = "Welcome to TBASIC"**

Comments : *LET* statement is optional: i.e. the equal sign is sufficient when assigning an expression to a variable name. The variable type on both sides of the

*equal side must be the same. i.e. string variable may not be assigned to a numeric expression and vice-versa.*

Important : a) When assigning a 16-bit variable to a 32-bit integer, only the lower 16 bits of the 32-bit integer will be assigned. Hence the programmer must take special care if the 32-bit number is out of the range of a 16-bit number (which is between -32768 to 32767).

b) If a negative 16-bit number is assigned to 32-bit integer variable, then the sign bit will be extended to 32 bits.

e.g. **DM[ 1 ] = -123.**

**A = DM[ 1 ]**

The 16-bit hexadecimal value of -123 is &HFF85, but A will be assigned the hexadecimal value &HFFFFFF85. Their decimal representation are however the same.

---

### **LOAD\_EEP**(*addr*)

Purpose : To return a 16-bit integer value saved in the EEPROM by the **SAVE\_EEP** statement.

*addr* - EEPROM address in TRiLOGI version 5.x. Actual PLC may have less EEPROM space. Please refer to your PLC's reference manual for the upper limit.

Examples : **relay[1] = LOAD\_EEP(10): A = LOAD\_EEP(2)**

See Also : **SAVE\_EEP**

---

\* **LOAD\_EEP\$** (*addr*) { \* Applicable only to PLC with firmware **r44** or higher }

Purpose : This function returns a string previously saved into the PLC's internal data EEPROM using the "SAVE\_EEP\$" command.

Examples : **X\$ = Load\_EEP\$(5)**  
**FOR I = 1 to 5**  
**\$\$[I] = Load\_EEP\$(I+10)**  
**NEXT**

Comments : 1. String saved in EEPROM string location #5 is loaded into X\$  
2. Strings saved in EEPROM string locations #11 to #15 are loaded into A\$ to E\$ (\$\$[1] to \$\$[5] represents A\$ to E\$)

See Also : **SAVE\_EEPR\$** for explanation of how the data EEPROM area are organized in the M-series PLC's to provide storage area for both integers and strings.

---

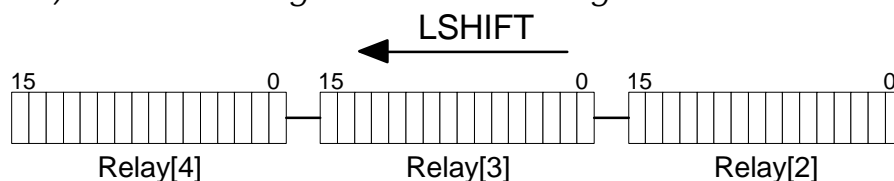
## LSHIFT $i, n$

Purpose : To shift 1 bit to the left the integer variable  $i$  which must be either an integer variable, a DM[n] or a system variable such as relay[n], output[n], etc.

LSHIFT instruction permits more than one variable to be chained together before performing a bit shift. The parameter  $n$  indicates the number of channels to be chained starting from  $i$  upward.  $n = 1$  if only one variable is involved.

Examples : **LSHIFT relay[2], 3**

Comments : The relay channels #2, #3, and #4 (which represent relays number #17 to #64) are chained together in the following manner:



Bits are shifted from the lower channel towards the upper channel. Bit #15 of Relay[2] will be shifted into Bit #0 of Relay[3] and so on. Bit #15 of the highest channel Relay[4] will be lost.

See Also : **RSHIFT**

## MID\$( $x$, $n, m$ )$

Purpose : This function returns a sub-string of  $m$  characters from  $x$, beginning with the  $n$ th character.$

$x$$  - any string expression, variable or constant.

$n$  - any numeric expression producing a result of between 1 to 255

$m$  - any numeric expression producing a result of between 0 to 255.

Examples : **A\$ = MID\$("Welcome to TBASIC", 4, 7)**

Comments : A\$ should contain the string : "come to".

## NETCMD\$( $ch, x$$ )

Purpose : This function sends a multi-point host link command string specified in the  $x$$  via serial port # $ch$  to another M-series or H-series PLC. It will then wait for a specified amount of time for a response string from the other PLC and this response string is then returned.

$ch$  - This refer to the communication port #. Please refer to the target PLC for details.

$x$$  - contains a valid host link command in multi-point format, excluding the Frame Check Sequence (FCS) and the terminator characters (\*

and CR). NETCMD\$ function will automatically compute the FCS and append to the end of x\$ and together with the terminator characters will be sent to the other PLC via COMM #ch.

- Note:
- 1) If the target PLC does not respond then this function returns an empty string.
  - 2) This function checks the FCS of the response string, and if the FCS is wrong it indicates an error in the serial reception and it will return an empty string.

Examples : **A\$ = NETCMD\$(3, "@05RI00")**

Comments : *To read the Input channel #0 of the PLC with ID = 05 connected to COMM #3 of this PLC. The response string will be assigned to A\$.*

**Special** : If the last character of x\$ is a "~" character, NETCMD\$ will send out the string without the '~' character. It will not append the FCS and '\*' to the outgoing string and it will not send out the carriage return (ASCII 13) character. It will also NOT check the response string for FCS. This allow NETCMD\$ to be used to interface to third-party ASCII devices with different command/response formats.

E.g. **A\$ = NETCMD\$(3, "Hello World~")**

The string "Hello World" will be sent out of serial COMM port #3. A\$ will receive the full returned string without applying any FCS check on the return string.

---

## **OUTCOMM** n, x

Purpose : This statement can be used to send an 8-bit byte of data ' x ' via Comm port #n. This command is added because PRINT#n command cannot be used to send out CHR\$(0). Zero is treated as the end of a string in TBASIC and will be ignored if you use PRINT #n statement to send out CHR\$(0).

Examples : **OUTCOMM 2, 225**

---

## **PAUSE**

Purpose: To set a breakpoint for executing the CusFn. This is used mainly for debugging a CusFn. By Inserting a **PAUSE** statement at the place of interest, you can suspend the program execution when **PAUSE** is encountered, after which you may examine the values of the relevant variables. You can continue to perform on-line monitoring of the PLC that has been paused. Program execution can also be continued by pressing the <P> key during Simulation or On-line Monitoring.

## PIDcompute(*ch*, *E*)

Purpose: This function computes the output for the PID compensator/ controller, using the *P*, *I*, and *D* Gains defined in the PIDdef statement for the same channel *ch*. The integral and differential values are stored within the channel's internal data space and will be automatically used by the PID computation routine. The PIDcompute( ) function uses the *lmt* (max. limit) term of **PIDdef** statement to limit the results of its computation. If the absolute value of the computed result is greater than "*lmt*", then the result will be set equal to "*lmt*" for +ve number and to "*-lmt*" for negative value. When this happens, the integral term will not accumulate the current error to prevent an "integrator windup" which is very undesirable for the system.

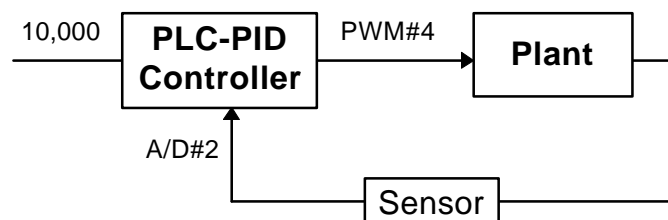
*ch* = channel number (1-16)

*Err* = Closed-loop Error.

(i.e. Set point value - Feedback Value)

The controller may obtain feedback from ADC, High Speed Counters, **PULSEFREQUENCY** or other means. The obtained result is then scaled and subtracted from the desired (set point) value to get "Err ". All computations are performed in 32-bit integers and the function returns a 32-bit integer that can be assigned to any variable. Any scaling for actual output (DAC or PWM) will be computed by the user within the same CusFn and sent to the output.

Example :



E.g. Implementing Closed-loop Digital Control with PID computation function

$$E = 10000 - \text{ADC}(2) * 20$$

$$A = \text{PIDcompute}(5, E)$$

$$\text{setPWM } 4, (A + 8000) / 100$$

Comments: The set point value is 10000 units, the feedback value is read from ADC channel #2 and then multiplied by 20 to convert (scale) it to the same unit as the parameter to be controlled. PID computation channel #5 (assume somewhere in the program a PIDdef for channel #5 has been executed before) is then used to compute the desired controller output value using the error signal (= set point - feedback value ADC(2) x 20).

The desired output (stored in variable *A*) is then added to the offset value 8000 and then scaled down by a factor of 100 before being sent out physically via PWM Channel #4.

**Important:** In actual implementation, use a clock pulse such as 0.1s, 0.5s or 1s etc to periodically activate the **PIDcompute( )** function so that digital control in discrete-time can be implemented. The PID sampling period depends on the time constant of the system. For very slow response processes such as the cooking temperature of a large body of water, the time constant is very large and even slower than 1.0 seconds clock may be sufficient. Do not use unnecessarily short sampling time because it increases computation time and slows down overall performance of the system.

**PIDdef** *ch, lmt, P, I, D,*

Purpose: To set up the parameters for a Proportional, Integral and Derivative (PID) Controller function. The function **PIDcompute( )** will make use of the parameters defined here for the corresponding channel #*ch*.

*ch* = channel number (1-16)

*lmt* = Maximum (saturation) limit for the computed result.

*P* = Proportional Gain ( $K_P$ )

*I* = Integral Gain ( $K_I$ )

*D* = Differential Gain ( $K_D$ )

Transfer Function of a PID Controller are defined as follow:

$$G(s) = K_P + \frac{K_I}{s} + K_D s$$

$$K_P = \text{Proportional Gain} = \frac{1}{\text{Porportional Band}}$$

$$K_I = \text{Integral Gain} = \frac{1}{\text{Integral Time Constant}}$$

All four parameters: *lmt*, *P*, *I* & *D* can be either 16 or 32-bit integer constants or integer variables. For the *lmt* term, the computed controller output value by the **PIDcompute( )** function is not allowed beyond the  $\pm$  *lmt* value (i.e. *lmt* represents the saturation point of the computed controller output). **PIDcompute( )** function implements "Integrator anti-windup" feature, which will avoid integrating the error signal when output is already saturated .

**Important:** When this statement is run, the integral and differential terms of channel *ch* is set to zero. Hence **PIDdef** should be run only once during initialization and not repeatedly executed. Otherwise the

**PIDcompute( )** function will not run properly because of the loss of integral and differential data.

See Also : **PIDcompute( )**

---

**PMON** *ch*

**PMOFF** *ch*

Purpose: **PMON** enables Pulse Measurement Function at channel #*ch*, whereas **PMOFF** disables the channel. After enabling the channel, you may then use the functions **PULSEWIDTH(*ch*)** and **PULSEPERIOD(*ch*)** to obtain the width and period of the input pulses arriving at the pulse measurement input pin. You must call PMON once during initialization to enable the pulse measurement hardware. Otherwise the two functions will only return 0. You should avoid repeatedly executing PMON function, otherwise the pulse measurement hardware will be reset repeatedly as well, and accurate measurement cannot be obtained.

If you no longer need to measure the pulse-width or period for a particular channel which has been PMON before, you should disable it using PMOFF to save CPU time because pulse measurement is interrupt driven and consumes CPU time.

Example: **PMON 1 : PMOFF 5**

See Also : **PULSEWIDTH( ), PULSEPERIOD( )**

---

**PRINT#** *n x\$; y; z....* Statement

Purpose : To send a string of ASCII characters formed by its parameter list (*x\$; y; z*) out of the PLC to other devices via the communication port #*n*.

Parameters: *n* must be an integer constant of between 1 and 8. Integer value in the parameter list (*y; z..*) will be converted into the equivalent ASCII representation. Each parameter must be separated by the semicolon(;).

Action : The ASCII string is first formed by the **PRINT** statement using all the arguments in the argument list and the completed string is then sent out of the serial channel #*n* at one go. The **PRINT** statement automatically sends a Carriage Return (CR-ASCII 13) out of the specified serial port after sending out the last character in the argument list. A **PRINT** statement that ends with a semi-colon ";", will not send the CR character.

If you have a long string to send than you can use ";" to break the whole command into several lines, with each line ending with a ";" except the last lines.

Examples :       **PRINT #2 "The value of A+B = "; A+B;**  
                  **PRINT #2 "Units"**

Comments : *IF A=5 and B=100, the string "The value of A+B = 105 Units" and a CR character will be sent out via Comm. port #2. In TRiLOGI simulation mode, the ASCII string will be displayed on a pop-up window to simulate PRINT action.*

See Also :   **INPUT\$( )**

---

### **PULSEFREQUENCY(*ch*)**

### **PULSEPERIOD(*ch*)**

### **PULSEWIDTH(*ch*)**

Purpose:       Return in Hz the frequency of the last input pulse; Return in microseconds the width or period of the input pulses arriving at channel #*ch* of the pulse-measurement pin. The pulse-measurement channel #*ch* must have been enabled by the **PMON** statement already. If the pulses stop coming in then PULSEFREQUENCY will return a zero while the other two functions will saturate at a certain maximum value (for T100MD+ it is equivalent to about 3.28 seconds)

*ch* = channel # (1-8)

Example:       **A = PULSEWIDTH(1)**

See Also :   **PMON, PMOFF**

---

### **READMODBUS (*ch, ID, addr*)**

Purpose :   Use the MODBUS ASCII or RTU protocol to automatically query a MODBUS ASCII or RTU slave device and obtain the desired 16-bit register data. The communication baud rate is the default baud rate of that Comm port unless it has been changed by the SETBAUD command.

*ch*   - PLC Comm port number  
          (1 to 8 using Modbus ASCII or 11 to 18 using Modbus RTU).

*ID*   - device ID of the MODBUS device (1 to 255)

*addr* - zero-offset address of the holding register in the MODBUS device.

Example :   **relay [3] = READMODBUS(3, 5, 101)**

*Comments* : The relay will contain the 16-bit data obtained from the MODBUS device with ID = 05 and from register offset address 101 (in MODBUS term this refer to the #40102 holding register) . Reading it into the relay[ ] channel allows bit level manipulation by ladder logic. It can of course also be read into any data memory.

This command automatically checks the response string received from the slave device for the correct LRC and the slave address. The status of the operation can be checked by the user program by testing the STATUS(2) function, **which will return a '0' if there is any error or if the slave device is not present.**

See Also : **WRITEMODBUS, STATUS(2), NETCMD\$( )**

---

**\* READMB2** *ch, ID, addr, var, count* { \* Applicable only to **M+** firmware **r44** or higher }

*Purpose* : Think of this as the multi-word version of READMODBUS command. Unlike the READMODBUS command which is a function that returns a single 16-bit word, this command is implemented as a statement so that multiple words of data can be stored into the PLC internal memory .

*Parameters:*

- ch* - PLC COMM port number (1 to 8 using Modbus ASCII or 11 to 18 using Modbus RTU).
- ID* - Device ID of the MODBUS slave device (1 to 255)
- addr* - Zero-offset address of the holding register in the MODBUS slave device starting from 0 = 40001.
- var* - the starting variable in the master for storing the returned data. (may be a DM or any system variable)
- count*- number of variables to read (max. = 16 in M+ PLC).

*Example* : **READMB2 3, 5, 101, DM[ 10], 8**

*Comments* : The PLC will use MODBUS ASCII protocol, via its Comm port #3, to query the slave MODBUS device with ID = 05 and ask for 8 words of data starting from register offset address 101 (in MODBUS term this refer to the #40102 holding register) . Once it receives the returned data these 8 words will be stored in the memory locations: DM[10], DM[11],.....DM[17].

This command automatically checks the response string received from the slave device for the correct slave address and LRC (or CRC16 RTU protocol is used). Like READMODBUS command, the status of this operation can be checked by the user program by testing the STATUS(2) function.

See Also : **WRITEMB2, STATUS(2)**

---

## REFRESH

Purpose : To Force immediate refresh of the physical inputs and outputs. This can be used after executing a **SETBIT** or **CLRBIT** command on an output[n] variable and to force the physical output to change immediately (subject to I/O refresh time delay). Otherwise, the physical output will only be updated during the normal refresh cycle which will occur only at the end of every ladder logic scan.

This is useful for situations which require immediately action such as shutting down a load during an emergency. This command is likely to be used mainly by an Interrupt CusFn.

---

## REM (or ') Statement

Purpose : To allow explanatory remarks to be inserted in a program. The text after the REM statement until the end of the line will be ignored by the compiler. An abbreviation for the REM statement is the apostrophe ( ' )

Examples :       **REM Waiting for the right time to turn on**  
                  '   **This is also a remark line.**

---

## RESET

Purpose : To perform a software reset of the PLC from within a CusFn. All the variables will be reset to zero or inactive and all the hardware outputs such as DAC and PWM will be turned OFF. The effect is the same as the Master Reset [MaRST] function in the ladder logic. The first scan bit (1st.Scan) will also be turned ON for one scan time.

However, if the program is stuck at some dead loop (such as WHILE, FOR-NEXT) in a CusFn, then [MaRST] would not be executed since the ladder program would not have a chance to scan the ladder rung containing the [MaRST] function. If this command is used by an interrupt service function, then it is possible to get the system out of the dead loop since the interrupt function can interrupt the dead loop and reset the PLC.

---

## RETURN

Purpose : Unconditionally ends the execution of the current CusFn and return to the caller (which is either the ladder program or another CusFn which has executed a CALL command).

Use of the RETURN statement is optional if there is no conditional ending required. After executing the last statement the CusFn will return to the caller automatically.

See Also : **CALL**

---

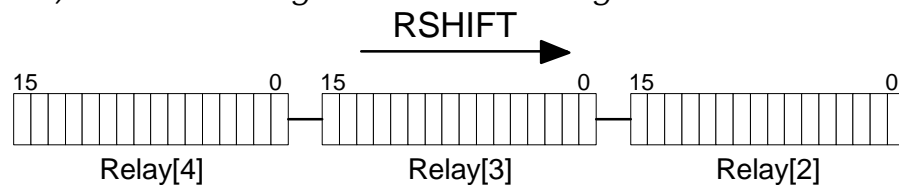
### **RSHIFT** *i,n* Statement

Purpose : To shift the integer variable *i* 1 bit to the right. *i* must be either an integer variable, a DM[n] or a system variable such as relay[n], output[n], etc.

RSHIFT instruction permits more than one variable to be chained together before performing a bit shift. The parameter *n* indicate the number of channels to be chained starting from *i* upward. *n* = 1 if only one variable is involved.

Examples : **RSHIFT relay[2], 3**

Comments : The relay channels #2,#3, and #4 (which represent relays number #17 to #64 ) are chained together in the following manner:



Bits are shifted from the upper channel towards the lower channel. Bit #0 of Relay[4] will be shifted into Bit #15 of Relay[3] and so on. Bit #0 of the lowest channel Relay[2] will be lost.

See Also : **LSHIFT**

---

### **SAVE\_EEP** *data, addr*

Purpose : To store a 16-bit integer *data* in the user's definable EEPROM address *addr* for non-volatile storage. If you attempt to save a 32-bit data, only the lower 16-bit will be saved. To save the entire 32-bit data, save the upper 16-bit using the **GETHIGH16( )** function and the lower 16-bit directly in two separate locations.

*data* - may be a 16-bit integer constant or variable.

*addr* - EEPROM address. Actual PLC may have less EEPROM space. Please refer to your PLC's reference manual for the upper limit.

Example : `save_EEP relay[1],100`

See Also : **LOAD\_EEP( ), GETHIGH16( ), SETHIGH16, LOAD\_EEP\$( ) and SAVE\_EEP\$**

---

\* **SAVE\_EEP\$** *strdata, addr* { \* Applicable only to PLC with firmware **r44** or higher }

Purpose : To store a string *strdata* in the user's definable EEPROM address *addr* for non-volatile storage.

*stringdata* - may be any string constant or string variable.

*addr* - EEPROM address (1,2,3...). Please refer to your PLC's reference manual for the upper limit of EEPROM space.

Example : **save\_EEP\$ A\$, 3**

Comments : *The content of A\$ will be stored at string space #3 of data EEPROM.*

See Also : **LOAD\_EEP\$()**

### Save\_EEP\$ Implementation on M+ PLC

Save\_EEP\$ and Load\_EEP\$ are two new TBASIC commands available only to the newest M+ PLC with firmware revision r44 and above. These commands allow you to save "strings" into the non-volatile data EEPROM area of the PLC. The EEPROM space is divided into 40-byte chunks for string storage. I.e. regardless of the length of the string, each string storage location will occupy a fixed 40-character length. Hence if "*stringdata*" parameter is longer 40 characters then only the first 40 characters will be stored in the EEPROM, the remaining characters will be discarded.

The string and integer data actually share the same pool of data EEPROM space. However, the string spaces are allocated from the top of the data EEPROM space downward, while the integer spaces are allocated from the bottom of the data EEPROM space and grow upward. This implementation allows say both SAVE\_EEP n, 1 and SAVE\_EEP\$ x\$, 1 to be executed in the same program without the string and integer data writing over each other space.

However, when the addresses grow larger up to a certain point, the integer and string data space will cross path and overwrite each other's space. It is therefore the programmer's responsibility to check that this does not happen. Here is how:

Assume the total EEPROM space for integer data = N words (16 bit).

Total number of data EEPROM space = 2N bytes

=> Maximum number of string EEPROM space = 2N/40 (rounded down).

To determine the upper limit of one type of storage, you have to first decide how much space you want to allocate to the other type.

E.g. 1:  $N = 1700$ , and you want use the first 510 location for integer data, that means the maximum number of string space available =  $(1700 - 500) * 2 / 40 = 59$ .

E.g. 2:  $N = 7700$ , and you want to store 200 strings. The maximum number of integer space available =  $(7700 * 2 - 200 * 40) / 2 = 3700$ .

---

### **SETBAUD** *ch, baud\_no*

**Purpose** : To set the communication "Baud Rate" of the PLC's serial channel #*ch*. All the M series PLC serial ports are defined as 8 data bit, 1 stop bit, and no parity and each has been preset to a certain default baud rate, which the PLC will assume every time its powers up. The baud rate may or may not be changed, depends on the PLC model. Please refer to the PLC's User's manual for the *baud\_no* that represent the baud rate of each serial channel and the range of *baud\_no* each of these serial ports may assume.

Caution should be taken when programming the baud rate of the "Host link" port because if a wrong baud value is set the host PC may not be able to communicate with it. If this happens suspend the PLC using its hardware switch and reset the PLC and re-load the program with correct setting.

**Examples** :        **SETBAUD 3, 3**    ' Set serial port #3 to 9600.

---

### **SETBIT** *v,n*

**Purpose** : To set the bit #*n* of the integer variable *v* to '1'. *n* is an integer constant or variable of value between 0 and 15. *v* may be any integer variable or a system variable such as relay[*n*], output[*n*], etc. However, if *v* is a 32-bit integer, **SETBIT** will only operate on the lower 16 bits.

Following digital electronics convention, bit 0 refers to the least significant bit (rightmost bit) and bit 15 the most significant bit. (leftmost bit) of the 16-bit integer variable. A quick way to find out the bit position and index of an I/O variable is to open their I/O table and check the "CH:BIT" column. Bit position beyond 9 are represented by hexadecimal number A to F.

**Examples** :        **SETBIT output[2], 11**

Comments : *output #28 will be turned ON.  
(Output channel #2 bit #11 = Output #17 +11 = 28)*

See Also : **CLRBIT, TESTBIT( )**

---

**SetCtrSV** *n, value*

**SetTimerSV** *n, value*

Purpose : Change the **Set Value (S.V.)** of the Counter #*n* or Timer #*n* to *value*. This statement to allow the user to modify the S.V. of the PLC internal timers and counters without changing the source program. A TBASIC function can be written easily to make use of a few digital or analog inputs to modify the SV of these internals timers/counters. The new S.V is also stored in the program EEPROM and hence is non-volatile. (See sample program "set\_TCSV.PC4")

*n* should be between 1 and 128.

*value* should be between 0 and 9999.

Examples : **SetCtrSV 10, 1234**  
**SetTimerSV 3, GetTimerSV(3)+10**

Comments : *Counter #10 will assume a S.V. of 1234..  
: S.V of Timer #3 will be increased by 10.*

Related : The present values (P.V.) of timers and counters can be read or written directly as integer variables "TimerPV[*n*]" & "CtrPV[*n*]". But the Set Values can only be changed by these two functions.

See Also : **GetCtrSV( ), GetTimerSV( )**

---

**SETDAC** *n, x* Statement

Purpose : To set channel #*n* of the PLC's Digital-to-Analog Converter (DAC) with the 16-bit integer result of the expression *x*. *n* must range between 1 and 16. Once set, the DAC channel will latch the set value until the next SETDAC statement on the same channel is executed.

Examples : **SETDAC 5, A+B\*16**

Comments : *DAC channel #5 will be set with the value of A+B\*16. A run- time error will result if n is less than 1 or is greater than 16. The actual number of DAC channels depends on the PLC model in use.*

---

**SETHIGH16** *v, data*

Purpose : To assign the upper 16-bit of a 32-bit integer variable *v* to *data*. The lower 16-bit of *v* is unaffected. This can be used to construct the

value of a 32-bit integer data using two 16-bit data obtained from either the EEPROM or the DM[n].

Examples :        **A = DM[ 2 ]**  
                  **SETHIGH16 A, DM[ 1 ]**

See Also :        **GETHIGH16( )**

---

**SETIO** *labelname* -- Please refer to the definition of **CLRIO** command

---

### **SETLCD** *n, offset, x\$*

Purpose : To display the string expression *x\$* on Line #*n* on built-in alphanumeric Liquid Crystal Display (LCD) or compatible Vacuum Fluorescent Display (VFD). *x\$* may be formed by concatenation of various strings using the '+' operator (e.g. "Temp =" + STR\$(A,3) + CHR\$(223) + " C"). Integers must be converted to string using the STR\$( ) or HEX\$( ) function to be accepted by this function.

**Special case:** if *n* = 0 the string *x\$* will be sent to the LCD's "Instruction-Register" which allows hardware-specific LCD configuration such as clear screen, set cursor ON/OFF etc. (please refer to LCD's manual for details)

The parameter *offset* = 1 to 40 allows you to send the string *x\$* beginning from the *offset*<sup>th</sup> position. Only the characters position to be occupied by *x\$* will be written to the display, other characters of the display remain unaffected.

The PLC may support LCD display modules capable of displaying up to 4 lines x 40 characters per line of alphanumeric characters. If the display has fewer lines or fewer characters per line, the unavailable lines or characters will be ignored by the PLC. Once set, the LCD display will latch the set value until the next SETLCD statement on the same line and same offset is executed. On the TRiLOGI simulator, the result of the SETLCD is displayed together with the Special Variables view screen.

Examples :        **SETLCD 1, 1, "This is a 1x20 LCD Display"**

---

### **SETLED** *n, m, value*

Purpose : To display the integer *value* on the PLC's built-in 7-segment LED displays, starting from the *n*<sup>th</sup> digit and occupying *m* number of digits. Leading zeros will be added to the left of the display if *value* occupies less digit than that specified by *m*.

However, if  $m$  is less than 1 (e.g.  $m = 0$ ) then *value* is treated as a single 8-bit ASCII character to be displayed rather than as a numeric value. Special symbols may be displayed on the LED panel if the LED driver is able to display the corresponding ASCII character.

$n$  must be between 1 to 16. The digit position is counted from left to right. i.e. the leftmost LED digit is digit #1. TRiLOGI supports up to 16 LED digits. The actual number of LED on the PLC may vary from 0 to 16, in this case only the available digits will be effective. *Value* may be a 16- or 32-bit integer number. Once set, the LED display will latch the set value until the next SETLED statement on the same digit is executed. On the TRiLOGI simulator, the result of the SETLED is displayed together with the Special Variables screen, which may be viewed by pressing the <V> key while in the simulation mode.

Examples :       **SETLED 5, 4, 89**

Comments : LED digit #5 to #8 (counting from left to right) displays 0089.

---

## **SETPASSWORD** *string*

Purpose : When this statement is executed, the PLC will not properly respond to any host link commands sent to it except the command "PWxxx...xx" which must contains the same string "xxx...xx" (not more than 19 characters) as defined in the SETPASSWORD command. All other commands will receive a "PWER" response indicating a "password error" state. Once the correct password has been accepted the PLC will work as normal and respond to all host link commands. Execution of "PW" host link command without any string will put the password lock back in force to prevent unauthorized access.

Example :       **SETPASSWORD "I love TRi LOGI"**

When using TRiLOGI the software will automatically prompt you to enter the password string if it encounters a PLC which has been password-locked. Note that the password is case sensitive. Password locked PLC cannot be accessed by older version of TRiLOGI.

Comments : *This feature is mainly used to protect an unattended PLC which is linked to an auto-answer modem. Without password protection anybody can dial in with a TLServer or TL41.exe and have full control of the PLC, which may be a serious security problem. Within the PLC software you may also use a timer to periodically re-arm the PLC with this command for maximum protection. You can also use different passwords for different time of the day or a set of rotating passwords to provide greater security.*

## SETPROTOCOL *ch, mode*

Purpose: A T100M+ series PLC automatically senses the type of communication protocols sent to it and responds accordingly. You may however fix the protocol type so that it does need to check the protocol type before responding. This command also allows the PLC to be defined as "No Protocol" so that it will not automatically respond to data that it receives which appears like one of the supported protocols. This may be important in some applications in which the PLC serial port is used purely to receive incoming data via INCOMM and INPUT\$ and you do not want it to respond to some data that appears to it as a valid communication protocols. This is also useful for implementing user's own communication protocol without worrying about conflict with the existing protocols.

*ch* = 1, 2 or 3 (COMM port number)

*mode* = 0 - Auto sensing (default mode)

1 - Fixed at RTU mode

2 - Fixed at EMIT mode

3 - Fixed at MODBUS ASCII mode

4 - Fixed at OMRON C20H protocol mode

5 - Fixed at NATIVE host link command mode

10- No protocol. (For creating user own custom protocol.

**IMPORTANT:** Please note that if you set the protocol to other than the "Native" (mode=5) or "Auto" (mode=0) that serial port will no longer respond to commands from TRiLOGI and you will encounter "Communication Errors" when you try to perform any communication using TRiLOGI with that serial port. You can still use the other unaffected serial port (e.g. COMM3, RS485) that support host link commands.

To regain communication with the serial port you will have to execute another SETPROTOCOL function that set it to mode 0 or 5 (assuming it has been written into the program), or you must reset the controller by turning OFF the power and then ON again. If you execute a SETPROTOCOL using the 1ST.Scan then you must turn on DIP switch #4 before powering up the PLC so that the SETPROTOCOL command will not be executed and you can regain control of the PLC using TRiLOGI.

## SETPWM *n, x, y*

Purpose : To set channel #*n* of the PLC's Pulse-Width Modulation (PWM) output with duty cycle represented by (*x*/100 %) and at a frequency (in Hz) given by parameter *y*.

*n* must range between 1 and 8. Once set, the PWM channel will latch the set value until the next **SETPWM** statement on the same channel is executed. *x* should range between 0 and 10000. If *x* is more than 10000, the duty cycle will be set to 100%

Examples : **SETPWM 1, 4995, 2000**

Comments : *PWM channel #1 will be set to operate at 49.95% duty cycle for PWM that can resolve up to 0.01%. The actual resolution will depend on the PLC's PWM resolution. The PWM frequency is set to 2000 Hz or nearest. For a 10-bit PWM the best resolution is about 1/1024 = 0.1 %. This means that in the above example the PWM will be rounded to 50%. Please check the target PLC's manual for the actual resolution.*

## SETSYSTEM *n, data*

Purpose: Allow changing of certain default system's parameters. Currently the only data defined are those that affect the serial communication commands. More parameters may be defined in future.

<i>n</i>	<i>data</i>
1	# of wait states (multiple of 0.15s) while waiting for a response from a slave controller after executing a NETCMD\$ or a READMODBUS/ WRITEMODBUS command. Default number of wait state = 1. e.g. SETSYSTEM 1, 3 The PLC will wait 3 x 150ms = 450ms for a valid response from the slave controller.
2	# of retry if NETCMD\$ or READMODBUS/ WRITEMODBUS failed to get a valid response from slave controller. Default = 2. (a total of 3 tries) e.g. SETSYSTEM 2,5 The PLC will retry up to 5 times if it failed to communicate with the slave. Note longer waiting time when failure occur if you increase the number of retries.

3	<p>0 - Respond as fast as possible to hostlink or MODBUS commands received from the host computer or another PLC.</p> <p>1 - (default) to allow at least a 0.01s (10ms) to elapse before responding to host link commands received from the host computer or another PLC. This delay is needed for auto-switch type RS485 converter to allow time for the hardware transceiver to switch direction.</p>
---	---

## STATUS (*n*)

Purpose : Return the status of various system operations.

Function	Returned value
STATUS (1)	0 - Normal power on reset 1 - Reset by Watch Dog Timer (WDT)
STATUS (2)	0 - READMODBUS or WRITEMODBUS failure 1 - READMODBUS or WRITEMODBUS successful
STATUS(8)	PLC's ID address stored in EEPROM for host communication

Examples :     **IF STATUS(2)        ' MODBUS READ/WRITE OK**  
                   **...**  
                   **ELSE                    ' MODBUS READ/WRITE failed**  
                   **...**  
                   **ENDIF**

## STEPCOUNT (*ch*)

Purpose : While the stepper motor controller is sending out pulses, this function can be used to monitor the number of stepper pulses sent to the Stepper Motor Channel *#ch* since the execution of the last "STEPMOVE" command. Hence this function returns the relative number of step moves.

This function can also be used to "measure" the physical size of a part if we use the stepper motor to drive a sensor and use the STEPSTOP command and the interrupt input to halt the stepper motor when the edges are detected. The physical size is then computed using the number of steps the stepper motor travels from one edge to another edge. The center position can be easily determined using such data too.

## STEPCOUNTABS (*ch*)

Purpose : Returns the absolute position of the stepper motor #*ch*. This function returns a zero if a **STEPHOME** command had just been executed and the stepper has not been moved since.

---

## STEPHOME *ch*

Purpose : Set the current position counter of stepper # *ch* to zero. This indicates a new "Home" position of that stepper motor. This command should be executed only when the stepper has reached a particular position to be regarded as the home position. All STEPMOVEABS command executed subsequently will be relative to the defined home position.

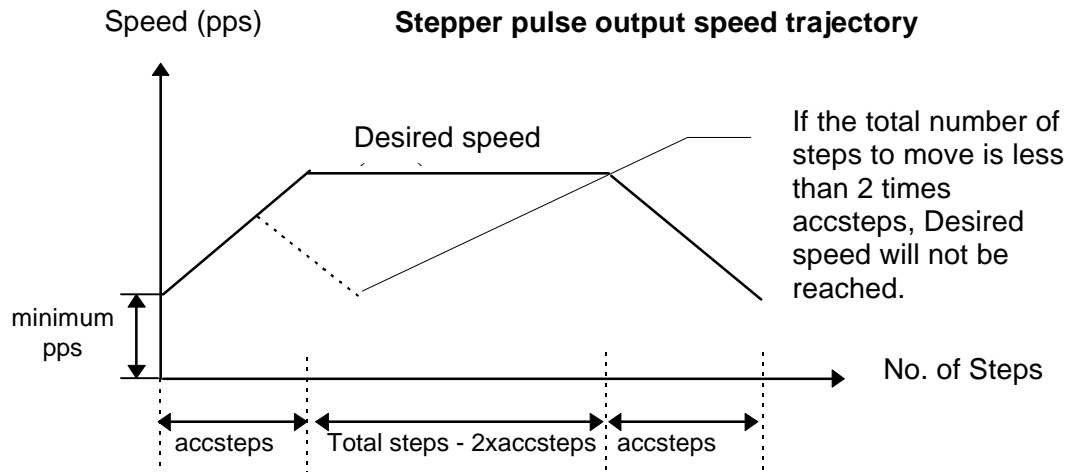
---

## STEPMOVE *ch, count, r*

Purpose : To activate the PLC's built-in stepper motor pulse generator channel #*ch* to output *count* number of pulses. The speed and acceleration parameters for the motion is defined by the STEPSPEED statement on the same channel # *ch*, which must be executed at least once before the first STEPMOVE command is issued. After executing the STEPMOVE command the PLC hardware will take over the actual pulse generation operation. The user's program will continue to execute even though the pulse generation is not yet completed. The internal relay #*r* can be used to signal to the other parts of the ladder program regarding the status of pulse generation, as follow:

When STEPMOVE command is first executed, the internal relay #*r* will be cleared before the first pulse is sent. After the completion of the movement (i.e. when all the pulses have already been sent), the relay #*r* will be set.

*ch* should be between 1 and 8. *Count* is a 32-bit integer number which allows you to program the stepper motor to move from 1 to  $+2^{31}$  (i.e. 2,147,483,647) steps. *Count* can also be an integer variable A-Z. However, If you use a 16-bit variable such as DM[*n*] for *count* then the range of movement can only be between 1 to 32,767.



Pulse generation can be interrupted by issuing a **STEPSTOP** command in another CusFn, which may occur say, in cases when the hardware hits a limit-switch and must stop the motor immediately.

**Important:** When a stepper channel is already activated (i.e. mid-way through its pulse generation) repeat execution of STEPMOVE command on the same channel will be ignored by the PLC. Re-execution of the STEPMOVE command on this channel can only take effect after the channel's pulsing operation has been completed by itself or aborted by the STEPSTOP command.

When in TRiLOGI simulation mode, execution of the STEPMOVE command will bring up a pop-up window that displays all the parameters of the motion path.

**Examples :**        **STEPMOVE 1, 5000, 10**

**Comments :** *Send out 5000 pulses on channel 1 and at the end of motion turn ON relay #10.*

**See Also :**    **STEPMOVEABS, STEPCOUNT( ), STEPCOUNTABS( ), STEPSPEED, STEPSTOP, STEPHOME**

## **STEPMOVEABS** *ch, position, r*

**Purpose :** This new 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  $\pm 2 \times 10^9$ ). 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 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 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.

See Also : **STEPCOUNTABS, STEPHOME , STEPSPEED, STEPMOVE, STEPSTOP, STEP COUNT**

---

### **STEPSTOP** *ch*

Purpose : To abort a stepper channel *#ch* which is in motion due to exceptional circumstances.

Examples : **STEPSTOP 2**

Important : Motion aborted by STEPSTOP command will not trigger the end-motion relay *#r* specified in the STEPMOVE command.

See Also : **STEPCOUNT( ), STEPSPEED, STEPMOVE**

---

### **STEPSPEED** *ch, pps, acc*

Purpose : To set the speed *pps* and acceleration/retardation *acc* parameters for the PLC's stepper motor motion controller (pulse-generator) channel *#ch*.

*ch* should return a value of between 1 and 8. Speed *pps* is based on no. of pulse per second (pps) output by the pulse generator. The acceleration *acc* determines the total number of steps taken to reach full acceleration from standstill and the number of steps from full speed to a complete stop. The stepper motor calculates and performs the speed trajectory according to these parameters when the command STEPMOVE is executed.

STEPSPEED command should be executed at least once before executing any subsequent STEPMOVE command to control the pulse generation. The defined parameters will be remembered until another STEPSPEED statement operating on the same stepper channel is executed again.

Examples :       **STEPSPEED 2, 2000, 20**

Comments : *The PLC's Stepper motor controller channel #2 is configured to send out pulses at 2000 pulses per second when STEPMOVE instruction is executed. It follows a linear acceleration trajectory which takes 20 pulses to reach the full speed of 2000 pps. This is equivalent to an acceleration of*

$$a = \frac{V^2}{2S} = \frac{2000^2}{2 \times 20} = 100,000 \text{ pulse/s}^2$$

---

**STR\$(n)**

**STR\$(n, d)**

Purpose : To return a string that represents the decimal value of the numeric argument *n*. If the second format is used then this function will return a string of '*d*' number of characters.

Examples :       **A\$ = STR\$(-1234)**  
                  **B\$ = STR\$(-1234, 7)**

Comments : *A\$ will contain the string : "-1234" , B\$ will contain the string "-001234"*

---

**STRCMP(A\$, B\$)**

Purpose : Perform a comparison between its two string expressions A\$ and B\$. IF A\$ and B\$ are equals, **STRCMP** will return a 0, if A\$ is of lower order (in ASCII table order) than B\$ the function will return a negative value. Otherwise it returns a positive value.

Examples :       **IF STRCMP(A\$, B\$)=0 THEN**  
                          **STEPMOVE 1, 1000, 1**  
                  **ENDIF**

Comments : *IF A\$ and B\$ are the same then turn on the stepper motor #1.*

---

**STRLWR\$(A\$)**

Purpose : To return a string which is an all-lowercase copy of A\$.

Examples :       **B\$ = STRLWR\$(A\$)+Z\$**  
                  **C\$ = STRLWR\$(C\$)**

Comments : *The second example shows how to convert a string to all lower case.*

---

## **STRUPR\$(A\$)**

Purpose : To return a string which is an all-uppercase copy of A\$.

Examples :       **B\$ = STRUPR\$(A\$)**  
              **C\$ = STRUPR\$(C\$)**

Comments : *The second example shows how to convert a string to upper case.*

---

## **TESTBIT (v, n)**

Purpose : To return the logic state of bit #n of the variable v. The function returns 1 if the bit is '1', otherwise it returns 0.

n is an integer of value between 0 and 15. v may be any integer variable, however, if v is a 32-bit integer **TESTBIT** will only test the lower significant 16 bits. A quick way to find out the bit position and index of an I/O variable is to open their I/O table and check the "CH:BIT" column. Bit position beyond 9 are represented by hexadecimal number A to F.

Examples :       **TESTBIT (Input[2], 3)**

Comments : *To test whether input #20 is ON*  
              *(Input channel #2 bit #3 = Input 17 + 3 = 20)*

See Also :       **SETBIT, CLRBIT**

---

**TESTIO (labelname)** -- Please refer to the definition of **CLRIO** command

---

**TOGGLEIO labelname** -- Please refer to the definition of **CLRIO** command

---

## **VAL(x\$)**

Purpose : To return a value of a decimal number contained in the argument x\$.

Examples :       **B = VAL("123") \* 100**

Comments : *B should contain the value 12300*

---

## **WHILE** *expression* .... **ENDWHILE**

Purpose : To execute a series of statements in a loop as long as a given condition is true.

Syntax : **WHILE** *expression*

...  
...

### **ENDWHILE**

When **WHILE** statement is encountered, the expression will be evaluated and if the result is true, the statements following the expression will be executed until the **ENDWHILE** statement. Thereafter, execution branches back to the **WHILE** statement and the expression is evaluated again. The loop statements will be executed repeatedly until the expression becomes false.

Warning: Be careful that the **WHILE** loop will not be an endless loop as the PLC will appear to freeze up, being trapped in an endless-loop execution. TRILOGI simulator attempts to detect this situation by giving a warning message if a loop is executed for an unduly large number of loops.

Examples :       **WHILE S = 1**  
                  **IF INPUT[1] & &H0002: S = 0 : ENDWHILE**  
                  **ENDWHILE**

*Comments : Execution will only be terminated when input #2 is ON. WHILE loops may be nested; i.e. a WHILE loop may be placed within the context of another WHILE loop. Each Loop must have a separate ENDWHILE statement to mark the end of the loop.*

---

## **WRITEMODBUS** *ch, DeviceID, address, data*

Purpose : Automatically write the 16-bit *data* to a MODBUS ASCII device using the MODBUS ASCII protocol. The communication baud rate is the default baud rate of that COMM port unless it has been changed by the SETBAUD command.

*ch*           - PLC COMM port number (1-8)  
*DeviceID*   - Device ID of the MODBUS device (1 to 255)  
*address*     - Zero-offset address of the holding register in the MODBUS device.  
*data*         - the 16-bit data to be written to the MODBUS device

Example :       **WRITEMODBUS 3, 8, 1000, 1234**

*Comments: The data 1234 will be written to the MODBUS device with ID=08 at the holding register offset address 1000 (in MODBUS convention this refer to holding register #41001).*

The command automatically checks the response string received from the slave device for the correct LRC and the slave address. The status of the operation can be checked by the user program by testing the STATUS(2) function, which will return a '0' if there is any error or if the slave device is not present.

See Also : **READMODBUS( ), STATUS(2), NETCMD\$( )**

---

**\*WRITEMB2** *ch, ID, addr, var, count* { \* Applicable only to **M+** firmware **r44** or higher }

Purpose : Think of this as the multi-word version of WRITEMODBUS command.

Parameters : *ch* - PLC COMM port number  
(1 to 8 using Modbus ASCII or 11 to 18 using Modbus RTU).  
*ID* - Device ID of the MODBUS slave device (1 to 255)  
*addr* - Zero-offset address of the holding register in the MODBUS slave device starting from 0 = 40001.  
*var* - the starting variable in the master whose data is to be sent out (may be a DM or any system variable)  
*count* - number of variables to send (max = 16 in M+ PLC).

Example : **WRITEMB2 13, 5, 101, DM[10], 8**

*Comments : The PLC will use MODBUS RTU protocol, via its Comm port #3, to write 8 words of data from DM[11] to DM[17] to the slave MODBUS device with ID = 05 and into its register offset address 101 to 108 (in MODBUS term this refer to the #40102 to #40109 holding register) .*

The command automatically checks the response string received from the slave device for the correct slave address CRC16. Like READMODBUS command, the status of this operation can be checked by the user program by testing the STATUS(2) function.

See Also : **READMB2, WRITEMODBUS, STATUS(2)**

---

# Appendix 1: Application Notes & Programming Examples

---

## I. Important Notes to Programmers of TRILOGI Version 5.x

---

### 1. Understanding Ladder Logic Execution Process

Like all industrial PLCs, the CPU of the M-series PLC first checks the logic states of the physical inputs and copies them into memory. During the ladder logic scan the actual logic states of the physical Inputs (except for interrupt inputs) are ignored by the PLC. The CPU uses the memory copy of the inputs to execute the ladder program.

The CPU executes its ladder logic program starting from the top rung of the program to the bottom rung. When the CPU reaches a ladder rung that activates a {CusFn} or {dCusF} that custom function will be executed. The CPU will only continue to scan the rest of the ladder program when the current custom function ends normally. Hence the order in which a ladder rung is placed within a ladder program can have an effect on the behavior of the program.

Output bits which are changed as a result of the program execution will only be updated to the **physical** outputs at the end of the ladder logic scan. One scan time is defined as the time it takes to execute the 3 steps (read physical inputs, execute program, update physical outputs). The CPU repeats these 3 steps continuously all the time, known as "Ladder Logic Scanning".

Hence, it is important to note that the variables INPUT[n] s and OUTPUT[n] in TBASIC are not the actual physical I/Os of the PLC, but only a memory representation of the actual I/Os which will be updated only during the I/O update cycles. The logic states of physical inputs are copied into the INPUT[n] variables during input scan and the physical outputs are set to the logic states contained in the OUTPUT[n] variables during output updates.

Therefore, one potential error that traditional BASIC programmers tend to commit is to attempt to poll for a change in the variable INPUT[n] within TBASIC such as the following:

```

      WHILE INPUT[1] = 0
      . .
      ENDWHILE

```

This will result in an endless loop since the value of the variable INPUT[1] will never change during execution of the custom function regardless of the actual logic states of physical input #1 to #8. The only way to force upon a physical I/O update is to use the **REFRESH** command, but it is

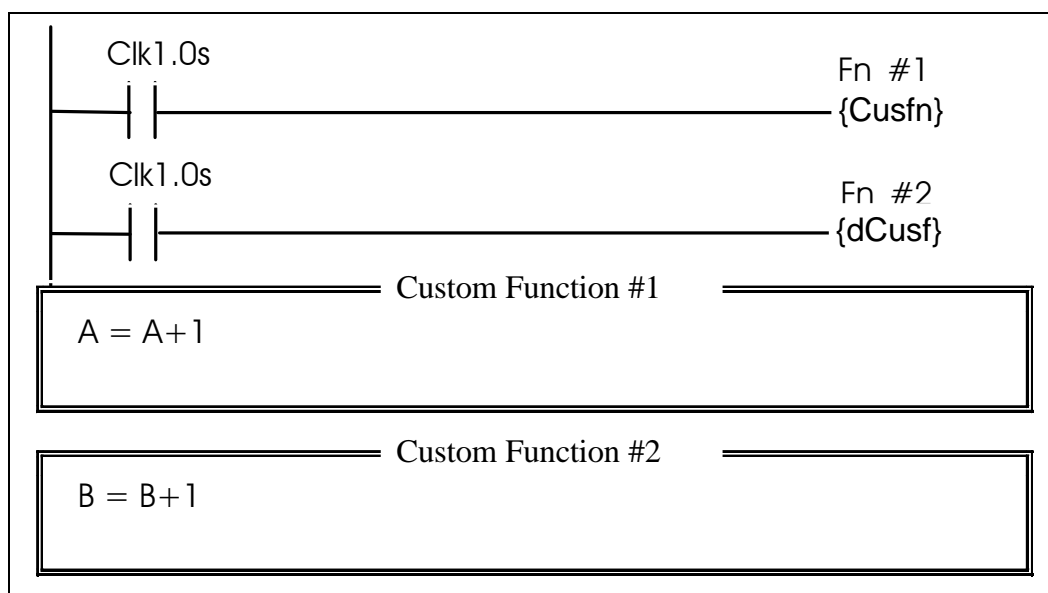
not a good practice for ladder logic programming to update physical I/Os in the midst of a program execution. The REFRESH command is meant more for forcing an immediate output to be turned ON or OFF during time-critical situations.

Hence it is important to allow a ladder logic program to finish its scan so that the physical I/Os can be updated. You should never hog the CPU within a particular custom function as this will mean the rest of the ladder program don't have a chance to be executed in a timely manner.

## 2. The Difference Between {CusFn} and {dCusF}

It is very important to understand the difference between the two formats of the custom functions once you understand how the ladder logic scanning process works as described in the last section. If you use the {CusFn}, the custom function will be executed **EVERY SCAN** of the ladder logic program as long as its execution condition is ON.

On the other hand, the {dCusF} (known as the differentiated format) is executed only **ONCE** when its execution condition goes from OFF to ON. The execution condition must go OFF and then ON again for the function to be executed again. It is not difficult to see that the differentiated format is used far more frequently than the other one since most custom functions involve arithmetic and when a condition is ON you most likely want the computation to be performed **ONCE** and not repeatedly in every scan of the ladder logic. You can easily understand the difference between the two formats if you run the following sample program:



Run the program in simulator and press the <V> key to view the changes in the variables A and B. You will see that B is incremented by

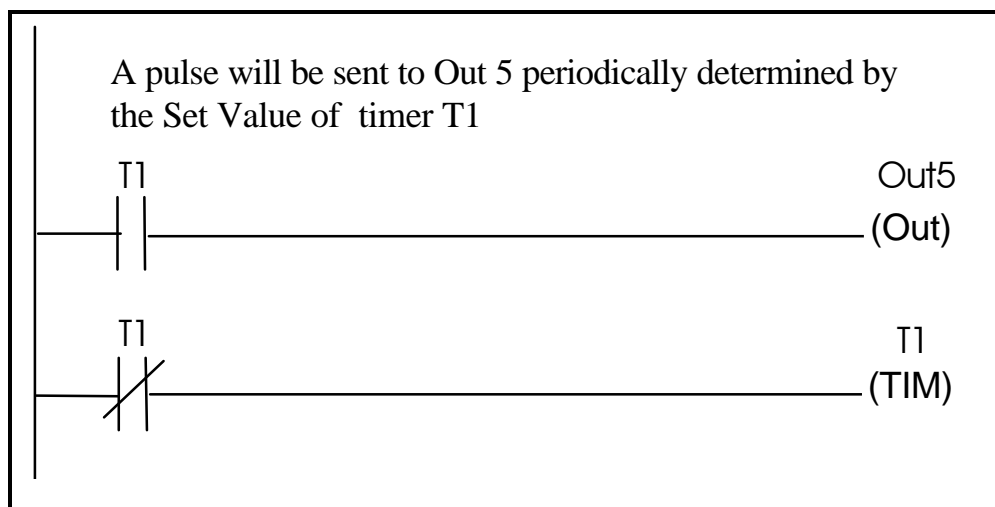
one every second, while A is incremented wildly for 0.5s and then stops for 0.5s. Try it! It can be very educational!

If you want to periodically check the status of an analog input or the real time clock, you should use a clock pulse (0.1s, 1.0s etc as shown in the example) and connect to a **{dCusF}**. Connecting to non-differentiated version would mean checking thousands of times for half the period and not at all for the other half period -- certainly not the intended outcome.

### 3. Timers Contact Updating Process

All the timers' contacts of the PLC, like the inputs and outputs, are updated simultaneously at the beginning of every ladder logic scan and not at the rung that contains the (TIM) coil. So if you are using self-reset timer, please note that if a timer times out its contact will be ON from the beginning of the ladder logic rung until the rung that contains the self-reset circuit. Thereafter the timer contact will be OPEN since the coil has been self-reset.

Hence please note that you should place the self-reset timer rung after all the ladder rungs that utilize the said timer contact. This allows those ladder rungs which use the timer contact to have a chance of being executed before the self-resetting rung clears the timer.



---

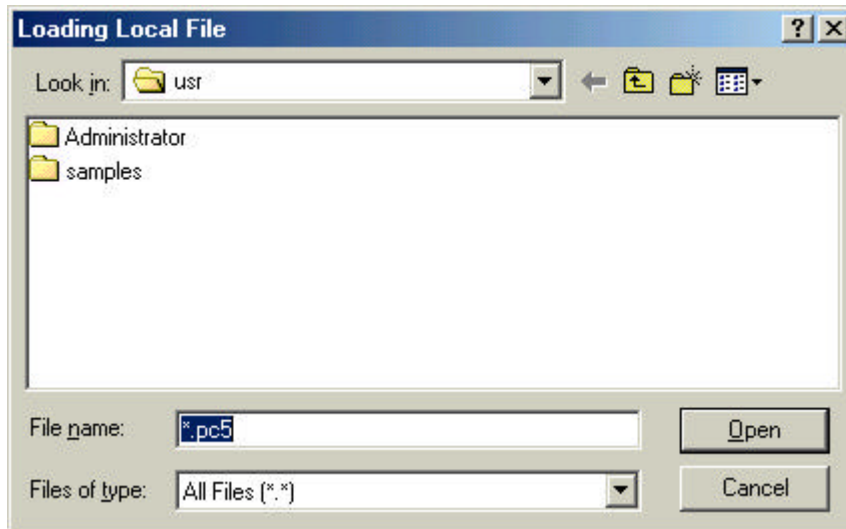
## II. TRiLOGI Sample programs

---

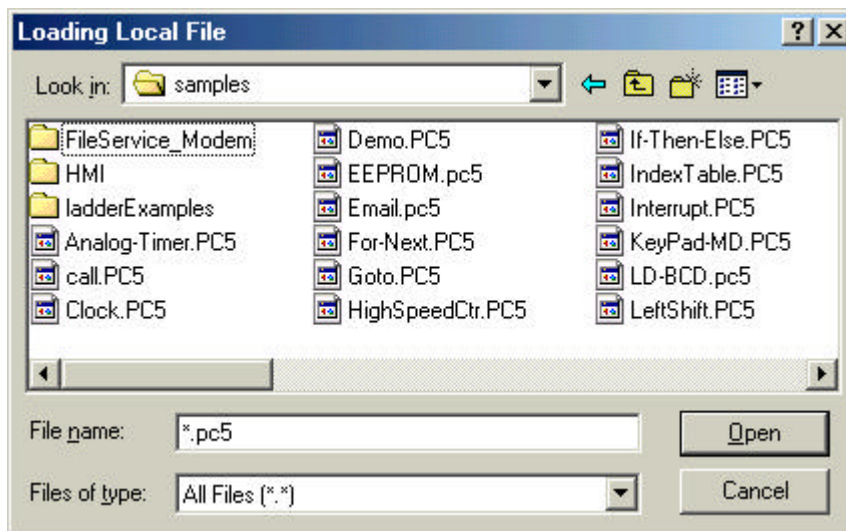
There are many well-documented demo as well as practical TRiLOGI program examples included in the following TRiLOGI installation folder:

<TRiLOGI installation folder>\usr\samples

When you click on TRiLOGI's "File -> Open (Local Drive)" command, you will be able select the user's folders where program files are stored. By default, only two users are defined: "Administrator" and "samples" as follow:



You should open the “samples” folder and select any files with “.PC5” extension for viewing.



There are also sub-folders within the “samples” folder where sample programs that relate to a particular topic or device are stored, such as those relates to using the MD-HMI. We strongly encourage you to open these example programs to see how these programs are structured. Most of these programs can be run in the simulator except those that involve communication with other devices.

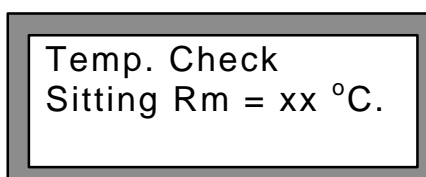
---

## 1. Display Alphanumeric Messages on built-in LCD Display

M-series PLC such as the T100MD-1616 supports built-in LCD display port that allows low cost connection to industry standard LCD display module. For such PLC, programming of the LCD display is via the SETLCD statement supported by TBASIC language.

### Assignment:

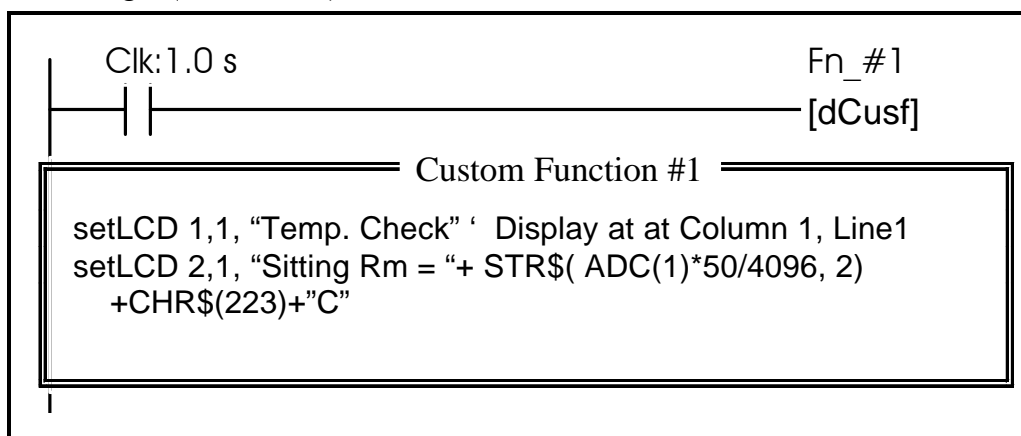
Every 1 second, display a message as follow:



Where  $xx$  depends on reading of A/D #1 which is returned by function `ADC(1)`.

Full scale A/D is 4096.

A/D range (0 to 4096)  $\Rightarrow$  Temperature 0 to 50°C



### Comments:

*Every one second, the special bit Clk:1.0s closes and activates Function #1. Within the Custom Function #1, `ADC(1)` reads the A/D converter #1 and converts it into degrees. The integer value is then converted into a two-digit string using the `STR$` function and concatenated to the rest of the text string for display using the `SETLCD` command.*

*Simulation of the display string to built-in LCD is supported on TRiLOGI Version 5.x. When in Simulation mode, press `<V>` key to view the Special Variables and the messages will appear in an LCD Simulation window.*

---

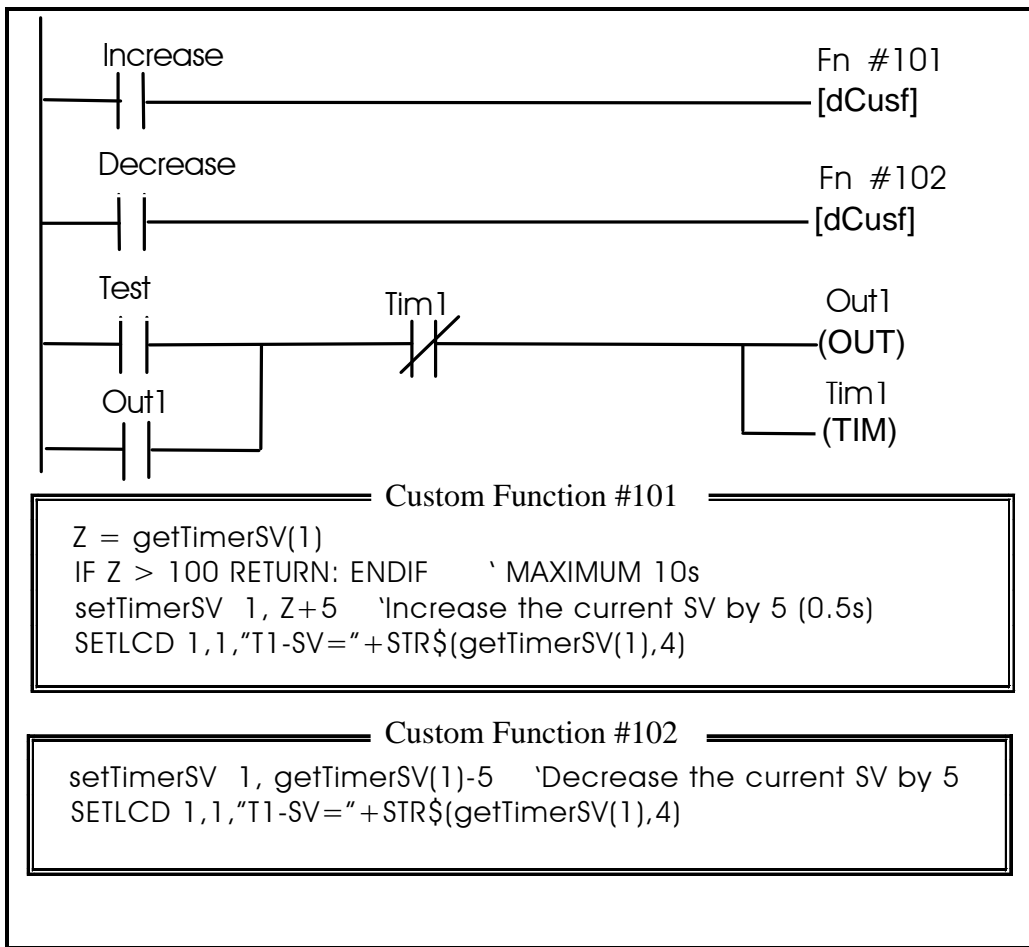
## 2. Setting Timer/Counter Set Values (S.V.) Using LCD Display

If you have an LCD display, then you can use two push-buttons inputs to change the Set Values (SV) of any selected timers or counters with visual feedback.

### Assignment:

- Press push-button "Increase" increment the SV of timer #1 by 0.5s. The upper limit for timer #1 SV is 10s (SV  $\leq$  100)

- Press push-button “Decrease” decrement the SV of timer #1 by 0.5s
- Press “test” button turns ON output #1 for a duration given by timer #1 and then turns it OFF.



### Comments:

The `getTimerSV(1)` function returns the current set value of the Timer #1. This value is read into variable `Z` in `CusFn #101` but used directly in `CusFn #102` for changing the Set Value of Timer #1. The `setTimerSV` statement uses the value of its second argument to update Timer #1's SV accordingly.

Note that changes to the set value SV will be updated in the program EEPROM memory and is non-volatile. However, EEPROM has a typical life-span of about 100,000 to 1,000,000 erase-write cycle. Exceeding this limit will “wear out” the EEPROM and resulting in a read error when the PLC operates. Hence, you should **NEVER** write a program that excessively changes the set value of the timer or counter (e.g. put it in a non-differentiated form of `[CusFn]` which executes every scan of the ladder program and continuously changes the content of the EEPROM).

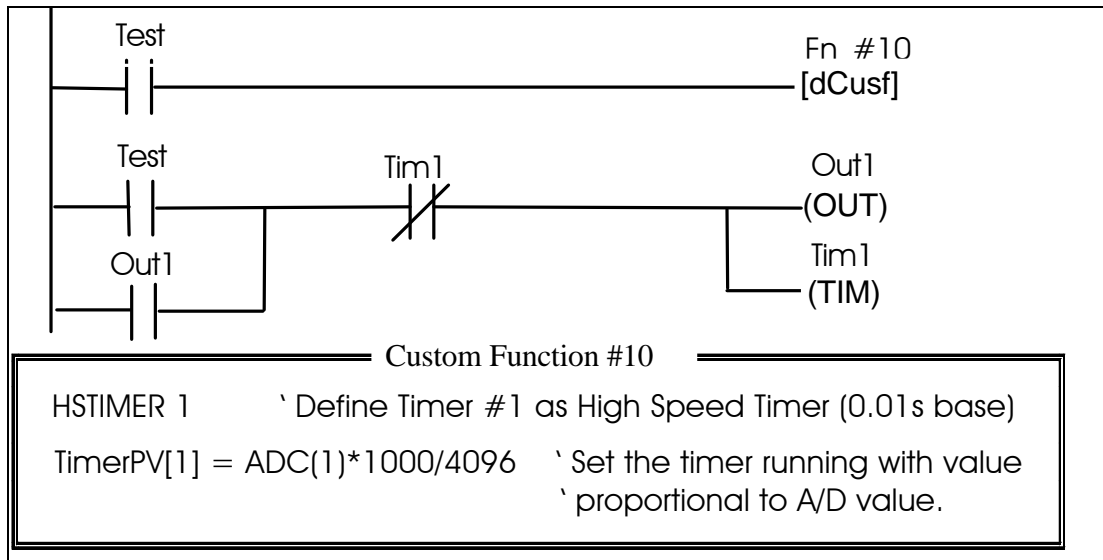
### 3. Using a Potentiometer As An Analog Timer

A cheap potentiometer can be connected to the PLC A/D input and provide a user-adjustable “knob” as an analog “Set Point” input device. A

scale can be drawn around the potentiometer to provide visual indication of set point value.

### Assignment:

- A potentiometer is connected to A/D #5. Use it to provide a timing range of 0 to 10.00 seconds.
- Pressing the “test” input turns ON output #1 for a duration determined by the potentiometer reading, after that turns output #1 OFF.



### Comments:

*To take full advantage of the resolution of the A/D converter, the timing range of 0-10 seconds is more finely divided when timer is defined as high-speed timer using the HSTIMER command. The time base is now 0.01s. This means that for maximum value of 10.00s, the timer should count down from 1000.*

*The next statement in CusFn #10 computes the ratio of the A/D input with respect to its full scale value of 4096 and multiplies it to the maximum timing value of 1000. I.e., if the potentiometer wiper is at half way, the A/D reading will be around 2048, the computation will result in a timing value =  $2048 * 1000 / 4096 = 500$ , or 5.00 second. Note that TRiLOGI 5.x does not support floating-point arithmetic. Hence the multiplication must be carried out before the division. Otherwise, if you compute  $2048 / 4096 * 1000$ , the result of the integer division of  $2048 / 4096 = 0$  and the whole expression yields a '0', which is clearly wrong!*

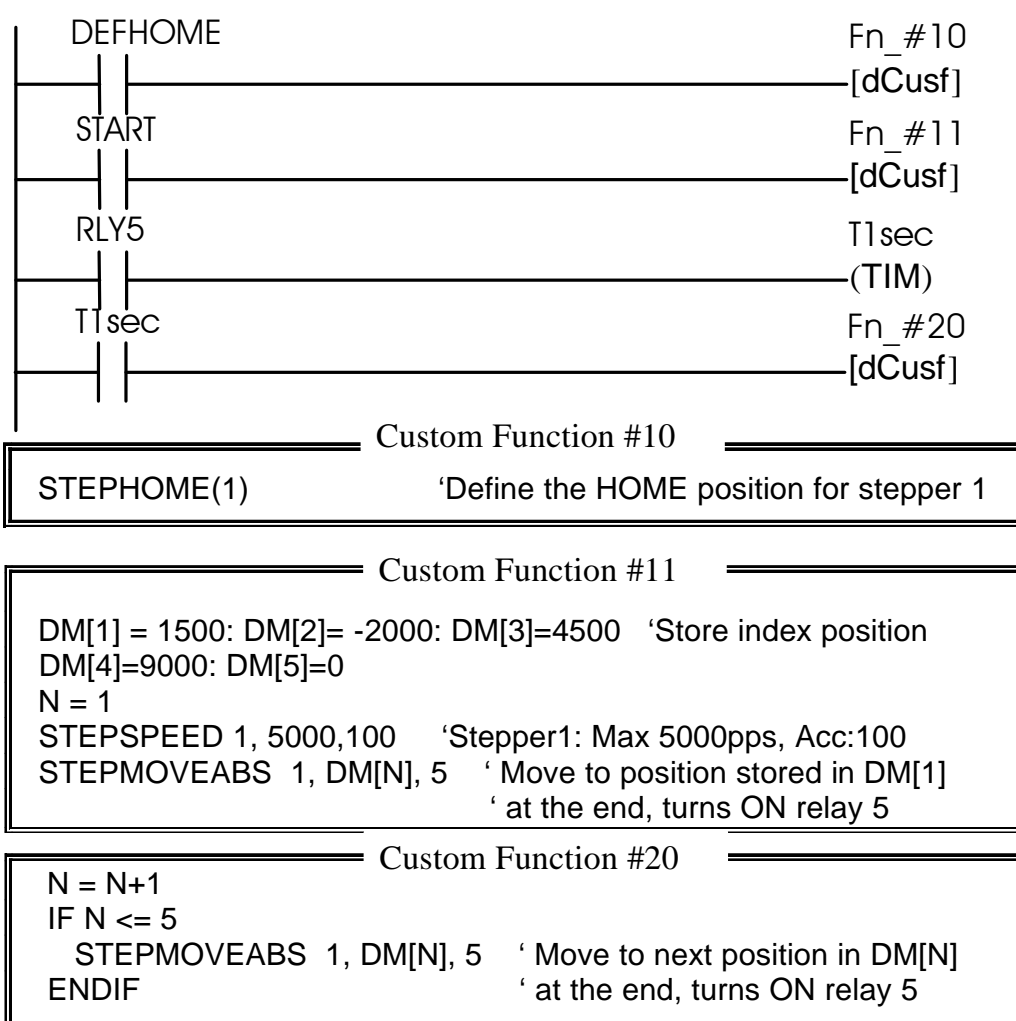
*The timer #1's Present Value (P.V) register is loaded with this number, which will start the timer countdown. In the next logic rung, the timer coil connected to the latched “OUT1” is necessary to prevent the timer from resetting itself. But It will not overwrite the PV with its own Set Value (SV), which will not be used at all in this case. This is because the previous ladder program has already started the timer with a value determined by the position of the potentiometer “knob”.*

## 4. Motion Control of Stepper Motor

The M-series PLC can generate pulses to feed to stepper motor driver. The maximum speed, acceleration, deceleration and total number of pulses to generate are definable using TBASIC. Both absolute positioning commands and relative move commands are supported.

### Assignment:

- A "DEFHOME" input define the current location as home position.
- Press the "START" input to begin indexing the stepper motor to position at 1500, -2000, 4500 and 9000 steps with respect to the HOME position. Pause for 1 seconds at each position. Return to home at the end of the cycle.
- Maximum speed = 5000 pps, Acceleration= 100 steps to full speed.



### Comments:

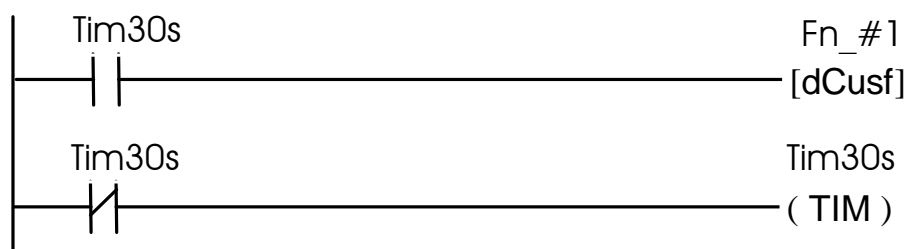
RLY5 is the label for internal relay #5. T1sec is a timer with preset value of 10. At the end of the pulse generation, RLY5 will be activated. Ladder logic senses RLY5 and executes the T1sec timer to cause a 1 second delay, after which custom function #20 is executed which triggers another STEPMOVEABS command and the process repeats for the other four indexing positions.

## 5. Activate Events at Scheduled Date and Time

All M-series PLCs have built-in Real Time Clock which keeps track of Date and Time and can be used to activate events at scheduled time.

### Assignment:

- Every day turn on output #1 (label name: Out1) at 19:00.
- Turn OFF output #1 at 7:00
- On 1st Jan 2000 at 12:00 turn ON output #5.
- On the same day at 18:00 turn OFF output #5



```
Custom Function #1
IF TIME[1]=19 AND TIME[2]=0 ` Hour hand at 19
  SETIO OUT1 ` Minute hand at 00
ELSE
  IF TIME[1]=7 AND TIME[2]=0
    CLRIO OUT1
  ENDIF
ENDIF

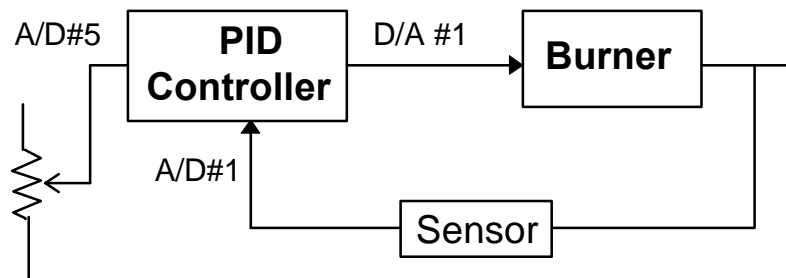
IF DATE[1]=2000 AND DATE[2]=1 ` Jan, year 2000
  IF DATE[3]=1
    IF TIME[1]=12 SETBIT OUTPUT[1],4:ENDIF
    IF TIME[1]=18 CLRBIT OUTPUT[1],4:ENDIF
  ENDIF
ENDIF
```

### Comments:

1. *Tim30s should have a Set Value = 300 and it activates Function #1 every 30 seconds. It is not necessary to check the clock too often as checking consume CPU execution cycles.*
2. *We used SETIO to control Output #1, but as a demonstration we use SETBIT to control Output #5 which is bit #4 of the variable OUTPUT[1]. The statement SETBIT output[1],4 turns ON output 5.*
3. *Actually it may not be necessary to check the minute hand since when the RTC turns from 18:59 to 19:00, the output will be turned ON as long as TIME[1]=19. Only when TIME[1]=7, then output #1 needs to be changed.*



## 7. Closed-Loop PID Control of Heating Process



E.g. Implementing Closed-loop Digital Control with PID computation function

### PID Controller Transfer Function:

$$G(s) = K_P + \frac{K_I}{s} + K_D s$$

$$K_P = \text{Proportional Gain} = \frac{1}{\text{Proportional Band}}$$

$$K_I = \text{Integral Gain} = \frac{1}{\text{Integral Time Constant}}$$

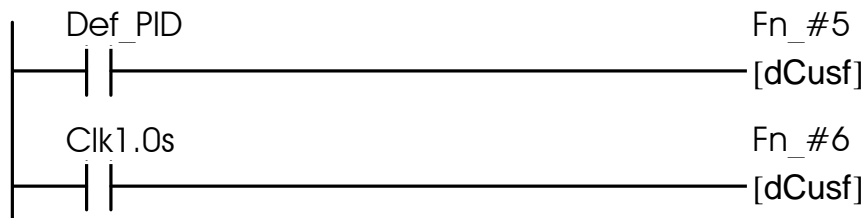
### **Assignment:**

- Read desired set-point temperature from a potentiometer connected to A/D #5 (S) with temperature range between 50 °C - 200 °C
- Measure the process temperature from a thermocouple + signal conditioner attached to A/D #1(T)
- Compute the Error = S - T. Apply Proportional + Integral + Derivative (P.I.D) algorithm to compute output X.
- Apply output X to Digital-to-Analog converter D/A #1 to control a variable position valve that feed fuel to the flame.
- Sample and compute every 1 second.

Full scale A/D range is 4096.

Range of Set Point: A/D #5 = 0 ⇒ 50 °C  
A/D #5 = 4096 ⇒ 200 °C

Range of Sensor: ADC#1 = 0 ⇒ 0 °C  
ADC#1 = 4096 ⇒ 300 °C



**Custom Function #5**

```

P = 500; I = 50; D = 0
PIDDEF 1, 2048*100 ,P,I,D    ` Use PID Engine #1, maximum limit
                               ` = +/- 50% of full scale

```

**Custom Function #6**

```

S = ADC(5) * (200-50)/4096 + 50    `Convert to °C
T = ADC(1) * (300 - 0)/4096

X = PIDcompute(1, S - T)/100 + 2048 ` X can vary within ± 50%
setDAC 1, X                        ` Write to analog D/A output #1

```

**Comments:**

1. *We use two decimal places to represent the gains  $K_P$ ,  $K_I$  and  $K_D$ . Each integer unit represents 0.01. Proportional gain  $K_P = 5$  is represented by variable  $P = 500$ . Likewise, Integral gains  $K_I = 0.5$  is represented by  $I = 50$  and Differential gains = 0 means Differential term is not used (P.I. only). The integrator limits of  $\pm 2048$  for the PIDDEF statement must be multiplied by 100 to be put on the same scale as the P,I and D parameters.*

*Note that since TRiLOGI does not support floating point arithmetic, the multiplication must be carried out before the division. Otherwise, if you compute  $150/4096 * ADC(5)$ , the result of the integer division of  $150/4096 = 0$  and the whole expression yields a '0', which is clearly wrong!*

2. *The value returned by PIDcompute() function is then divided by 100 to get the real value of controller output. PIDcompute() returns a signed value which can vary from -limit to + limit. We choose the 50% D/A output ( $4096/2 = 2048$ ) as the mean control point so that negative values from PIDcompute() means D/A output will be  $< 2048$ , positive values means D/A output will be  $> 2048$ .*

## Appendix 2: PLC & PC Hardware Setup and Configuration

### I. PLC to PC Connection

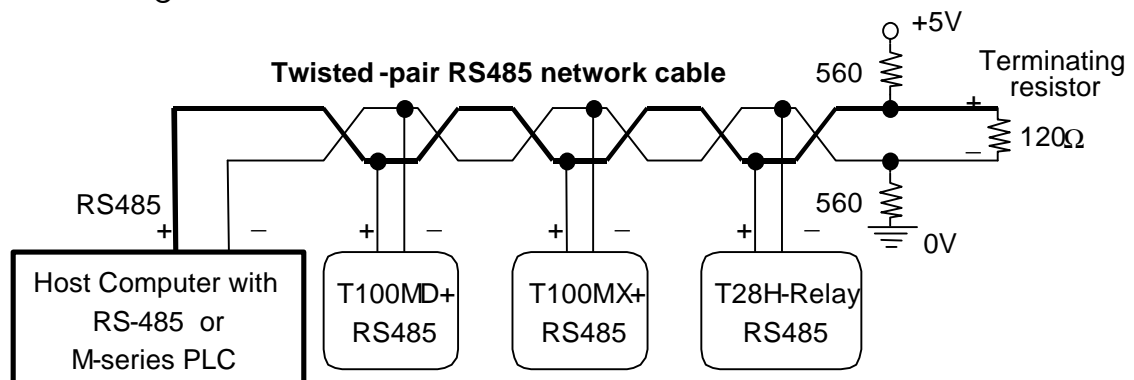
#### 1. Single PLC to One PC Running TLServer

The simplest configuration will be when there is only one PLC and one PC involved. You simply connect the PLC's RS232 port to the any of the RS232 serial port (COM1: to COM8:) of a PC and run the TLServer on it. If you use other than COM1: on your PC, you will need to configure TLServer's serial port to match the communication port number.

#### 2. Multiple PLCs to One PC Running TLServer

You can connect multiple M-series PLCs to a single PC running TLServer by connecting every PLC's RS485 in a daisy-chain manner to the PC's RS232 port. You do need to purchase a **RS232-to-RS485 converter** (such as the **Auto485** adapter) to connect the PC's RS232 port to the RS485 network. Please refer to the PLC's User Manual for details on installation issues regarding electrical specifications and termination requirements when connecting the PLCs on an RS485 network.

Internet TRiLOGI can log-in to the TLServer and have immediate access to all the PLCs on the RS485 network just by specifying the ID address of the PLC concerned. Up to 32 standard M-series PLCs can be networked to a TLServer. If you replace the RS485 driver IC by a 1/8 power type you can link up to 256 PLCs to a single TLServer for programming and monitoring!



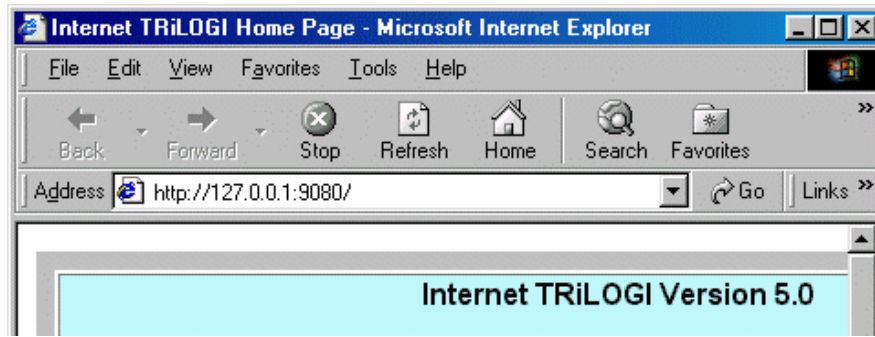
### II. Networking Issues

The networking method used by the PC running the TLServer, as well as how the TRiLOGI client software accesses the TLServer will have impact on the configuration of both the TLServer and the TRiLOGI client. We will consider various scenarios below.

## 1. TLServer and TRiLOGI Client On Same PC.

When both TRiLOGI client and TLServer runs on the same PC, we call this a "localhost" access and you can use the IP Address: 127.0.0.1:9080 to login to TLServer. Localhost access is always available regardless of whether this PC has any network connection to LAN or the Internet.

Note that If your PC has no network connection, then TLServer will report this localhost IP address on its front panel. However, if the PC is connected to the Internet or a LAN you will see different IP addresses. Remember that even if TLServer does not report 127.0.0.1, it is always available for localhost programming. You can either run the TRiLOGI Application directly or you can open up your web-browser and key in the following URL: <http://127.0.0.1:9080>. In the latter case, you are running TRiLOGI as an applet that is loaded from the TLServer.



## 2. TLServer has Direct Connection to the Internet

If the PC running the TLServer enjoys a **direct** connection to the Internet via dial-up, DSL, T1 line or cable modem, then TLServer will be accessible to any client on the Internet. Note that if you connect to the Internet via dial-up connection, then do remember to connect to the Internet before you run TLServer so that it can report the actual IP address on its front panel.

If the TRiLOGI client software also accesses the Internet via direct connection, you will have the least problem. However, if the TRiLOGI client is sitting behind a corporate **firewall**, then the situation is much more complex and it has impact on both the TLServer and the TRiLOGI settings, as shown in the following table:

<b>TRiLOGI Client has Direct Internet Connection</b>	<b>TRiLOGI runs on a PC protected by Corporate Firewall</b>
<ul style="list-style-type: none"><li>• TLServer:: port settings=80 or any value above 1024</li><li>• TRiLOGI: "Use HttpProxy" setting is optional</li></ul>	<ul style="list-style-type: none"><li>• TLServer: Port Settings = 80</li><li>• TRiLOGI: "Use HttpProxy" = true. May need to obtain proxy server's IP address.</li></ul>

### **3. What Happens when a Client is behind a firewall but the TLServer has direct link to the Internet?**

If the client PC is located within a corporate Intranet and protected by a firewall, then there are very limited means for the client to access the Internet outside of the firewall. Note that NOT all PCs within the LAN have access to the Internet. Whether a PC has access to the Internet or not is decided by your company's System Administrator.

Even if a particular client PC does have access to the Internet (because you are able to use a browser to visit Yahoo!), it doesn't mean that the PC has a direct connection to the Internet. What may actually happen is that the network administrator has setup a "Proxy Server" that will intercept your client PC's request to link to the Internet. The proxy server is the one that actually has a direct link to the Internet. It acts on behalf of the client within the Intranet to process HTTP connection to the Internet and passes the response data to back to the client.

So how does the proxy server determine whether a network packet is meant for the Internet and not meant for another workstation within the Intranet? It determines it by examining the port number that the packet attempts to connect to. If a connection is made to the well-known HTTP port which is = 80, it will be handled by the proxy server. If you use a port number such as 9080 (default TLServer port address) it may think that the connection is to be made to some local workstation and hence will not direct the packet via the proxy server to the Internet.

Therefore, in order for the client to make connection to the Internet via the proxy server, the TLServer port address should be set to 80. In addition, the TRiLOGI client should also be configured to access the TLServer via a "Http Proxy Server" as described in the document "Log-In to TLServer".

### **4. TLServer and TRiLOGI On The Same Local Area Network**

If the TLServer is running on a workstation that is part of a local area network, it is unlikely that the workstation will have a direct connection to the Internet (unless the System Administrator has deliberately configured it for that purpose). When the TLServer starts, it will report the IP Address of the workstation which is the Intranet IP address and NOT the Internet IP address.

Now if the TRiLOGI client is running on another workstation which is also part of the same local area network, then it is quite simple: TLServer can be assigned any unused port number above 1024 and the TRiLOGI client can access TLServer from any other workstations. However, avoid

setting TLServer to port 80 since by default, port 80 is for accessing the Internet via the HTTP proxy server.

## **5. How to access TLServer running on a Private LAN from the Public Internet?**

If the workstation that TLServer is running on does not have a direct Internet connection to the Internet, then it will normally not be possible to access the TLServer via the public Internet since the firewall will block any attempts to access a PC inside the LAN. There are two possible ways to overcome this:

1. Consult your System Administrator to configure a Network Address Translator that will assign you a public IP Address that will be mapped to the local workstation that runs the TLServer.
2. If your need to provide connectivity from the Internet is only temporary (e.g. allowing your contractor to fix a software bug) it may be easier to use a modem and dial-up to an ISP when the need arises. Once the connection is no longer needed, just hang up the modem. However, before you do this, please check with your company's System Administrator to make sure that you are not violating the security policy. If that is a problem, you may consider using a standalone notebook computer or PC (i.e. not connected to the LAN) to make the dial-up connection, which provides temporary Internet connectivity for the TLServer (and hence the PLCs) but will not compromise the security of your corporate Intranet.

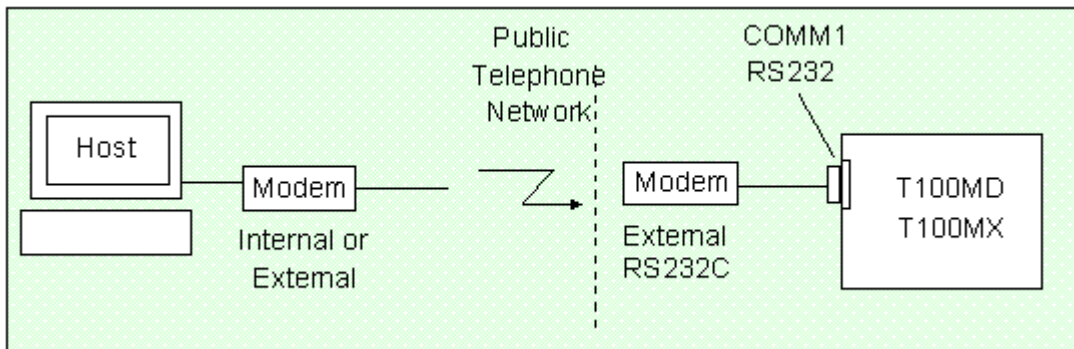
## **6. Home Networking Type Routers**

If your workstation shares an Internet connection via a low cost NAT router (these are getting very popular nowadays with home networking suppliers such as Linksys, NetGear, etc), your TRiLOGI client should not have much problem accessing a remote TLServer since these routers typically do not forbid your workstation from making direct outgoing connection to the Internet.

However, by default most NAT router's built-in Firewall will block any incoming attempt to access the TLServer. Fortunately, you should be able to configure the router to perform what is known as "Port Forwarding" – i.e. to forward any external TCP/IP packets that are destined to a certain port number to a designated workstation on the home network. In that case, you should configure your router to forward port number 9080 to the PC that runs the TLServer (assuming the TLServer is configured for port 9080). Please refer to your router's help manual for details.

## Appendix 3: PLC-to-Modem Communication Setup

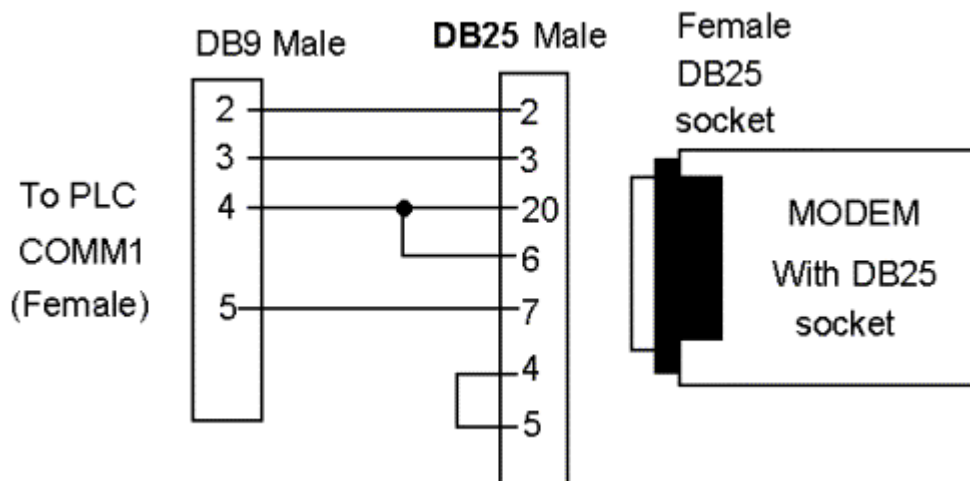
A remotely located M-series PLC can be connected to a host PC via public-switch telephone network (PSTN), radio or cellular phone network. This can be accomplished by using two analog modems, one connected to the PLC's RS232 serial port, and another modem connected to the remote host PC as follow:



There are a some technical issues that need to be handled carefully in order to successfully implement the modem-linked host communications as described in the following sections.

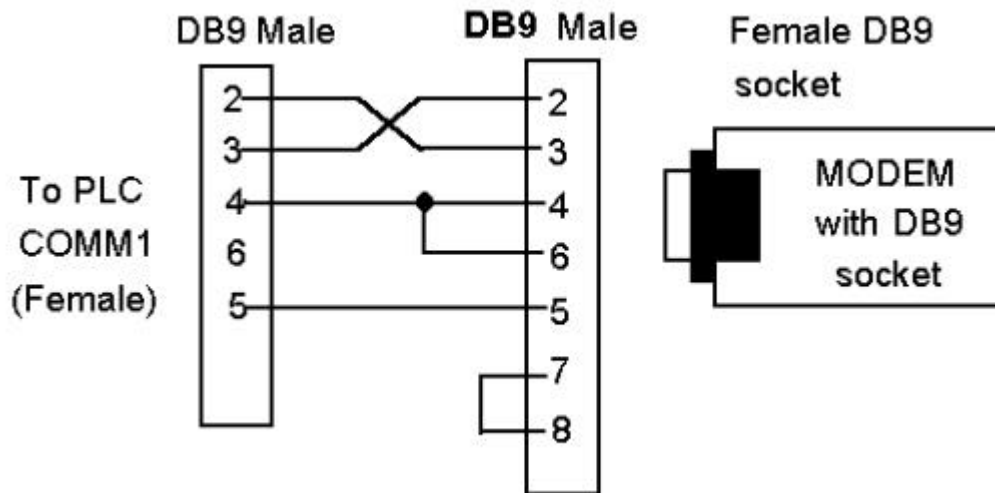
### 1. Modem Connection

**Modem 1:** The host PC may use any internal or external modem that can communicate at 2400 bps or faster. Connect the modem to the PC as instructed in the modem's manual and connect the phone line to the phone jack on the back of the modem marked "WALL" or "Line".



**Modem 2:** The modem to be attached to the PLC (modem2) must be an external modem with an RS232 connection port. Since modem are DCE type device, they most likely come with a female type DB25 or

DB9 socket meant for plugging into the PC's RS232 port. Since the PLC's host link port is also a female DB9, we need to construct a DB9-male-to-DB25-male cable or DB9-male-to-DB9-male cable to link the PLC to the modem, as follow:



## 2. Communication Speed

When communicating via modems, there are two different definitions of communication speeds that you should be aware of:

- The "DTE Speed" or "line rate" is the serial communication speed between the modem and the device connected to its RS232 port. Most modems can automatically detect the RS232 speed of the device and can assume any speed from 1200, 2400 all the way to 115,200 bps. The first ASCII character they receive from the device will determine the DTE speed that the modem will use to communicate with the device.
- The "modem-to-modem communication speed" is what you read on the modem specifications, such as 33.6Kbps, 56Kbps etc. When two modems are connected, they automatically negotiate for the best speed to communicate between the two of them based on the quality of the phone connection and the maximum speed that both modems are able to achieve. We usually have no control of what speed they choose to communicate. But one thing is for sure, which is that the modem-to-modem speed is always lower than the DTE speed.

Since the default communication baud rate of the M-series PLC's RS232 serial port is 38,400 bps, the PLC should send a modem initialization string to the modem on the first scan pulse so that the modem can recognize its default DTE speed (i.e. 38,400 bps) in order to talk to the

PLC. E.g. To reset the modem, you just have to send an ASCII string "ATZ" to the modem using the following TBASIC command:

```
PRINT #1 "ATZ"
```

If you want your modem to automatically answer to an incoming call (e.g. using TLServer 2.0 modem dialing capability), then you should execute the following TBASIC statement:

```
PRINT #1 "ATS0=1"
```

The above statement will tell the modem to answer on first ring, you can also change the number 1 to other numbers, E.g. if ATS0=3 it will answer on the 3<sup>rd</sup> ring of the phone.

### 3. Software and Programming

The TLServer 2.0, which is part of the Internet TRiLOGI software suite, already includes built-in support for dialing a modem. Hence if you are using the PLC in passive answer mode only, all the PLC needs to do is to send a modem initialization string "ATS0=1" using the "1st.Scan" pulse to put the modem in auto-answer mode whenever the PLC is powered up. The PLC does not need to issue any more commands to the modem. Whenever a user wants to communicate with the PLC, he/she will first use the TLServer to dial and connect to the PLC's modem and when the connection is established, he/she will then be able to use the TRiLOGI client or the TRi-Excellink program to communicate with the PLC. The fact that the PLC is connected via modem and not via direct RS232 is totally transparent to the client programs. To prevent unauthorized access to the PLC, you may need to use the TBASIC command "SETPASSWORD" to set a protective password.

The great flexibility of the M-series PLCs becomes even more apparent when you realize that you can easily program the PLC to automatically dial in to the TLServer to perform a number of tasks, such as using the PLC's File Service to save or append data to hard disk files, send email to anyone via the Internet or even synchronize its real time clock with the host PC!

A number of examples have been included in the "C:\TRi LOGI\TL5\usr\samples\FileService\_Modem" folder in TRiLOGI version 5.1 and above. All these examples make use of a powerful yet easy to use custom function that was written entirely using the standard TBASIC commands (see source code listing in the text box below). You only need to create the following simple ladder circuit to use this function (assuming it is function #10):

```

D$ = "ATDT*802" ' store the phone number
IF TESTIO(Connected) THEN ' already connected.
  IF TESTIO(Dial Modem)=0 ' connection no longer needed
    IF DM[3991]=0 ' used as timer for modem attention.
      PRINT #1 ' clear serial-out buffer.
      WHILE INCOMM(1) <> -1 ' clear whatever data in serial-in
buffer
        ENDWHILE
      ELSE
        IF DM[3991]=5
          PRINT #1 "+++"; ' get modem attention
        ELSE
          IF DM[3991]>=10 ' Wait 5 second to gain attention.
            PRINT #1 "ATH" ' hang up modem command.
            CLRI O CONNECTED
            DM[3991]=0
          ENDIF
        ENDIF
      ENDIF
    ENDIF
    DM[3991]=DM[3991]+1 'increment the timer
  ENDIF
  RETURN
ENDIF
IF TESTIO(dial Modem)=0 RETURN: ENDIF
' If DM[3990] > 0 it means a dialing action has started.
' If DM[3990] > 30 it means more than 30 seconds has passed
' and connection still not established, then retry.
IF DM[3990]=0 ' Use this DM as a flag
  WHILE INCOMM(1) <> -1 ' clear whatever data in serial buffer first.
    ENDWHILE
  PRINT #1 D$ ' Dial the number
  DM[3990]=1
  RETURN
ENDIF
AS = INPUT$(1)
IF LEN(AS) = 0
  DM[3990]=DM[3990]+1 ' also use it to track the time-out
  IF DM[3990] = 28 ' 28 seconds has lapsed.
    PRINT #1 "ATH"

```

```

ENDIF
IF DM[ 3990] >=30: DM[ 3990]=0: ENDF
RETURN
ENDIF
SETLCD 4, 1, AS
IF STRCMP(MIDS(AS, 2, 7), "CONNECT")=0 ' is connected
    DM[ 3990] = 0 ' for next round of connection
    DM[ 3991] = 0 ' reset timer for hang-up modem use
    SETIO Connected ' set an I/O bit to indicate connection
ENDIF

```

All you need to do is to copy and paste this custom function to your own Ladder+BASIC program, then create an I/O with label name "DialModem" – this may be an input, output, relay, timer or counter contact. The moment this I/O bit "DialModem" is turned on, the PLC will begin to execute the sequence of dialing the remote modem, waiting for a successful connection and then turning on an I/O bit with the label name "Connected". If the dialing cannot be completed within 30 seconds, this custom function will hang up and then re-dial. The process will be repeated indefinitely until either a successful connection is made or if the "DialModem" i/o has been turned OFF.

To disconnect from the modem (hang up), your PLC program just have to turn off the "DialModem" I/O bit and the abovementioned custom function will automatically perform the action of hanging up the modem.

**Note:** Since the PLC does not have a carrier detect (CD) connection to the modem, therefore if the connection is lost after a successful initial connection, the PLC would have no way of knowing it immediately. Your program would have to detect this condition (e.g. if it sends a file service command and does not receive a "<OK>" acknowledgement string from the host). Once the PLC notes that the connection is lost, it can re-establish the connection by simply turning off the I/O bit with label name "Connected". (say, by executing the "CLRIO Connected" statement). As long as the "DialModem" I/O bit is on, the custom function will re-dial and attempt to make another connection if it notices that the "Connected" bit has been turned OFF for whatever reason.

