I-TRiLOGI: Editor, Compiler, Simulator

i-TRiLOGI Reference Manual

Chapter 1 - Setup i-TRiLOGI

Please follow the download link provided in your i-TRiLOGI License Card to go to triplc.com website to download the installation guide.

We recommend that you install the i-TRiLOGI software in the default installation location as follow:

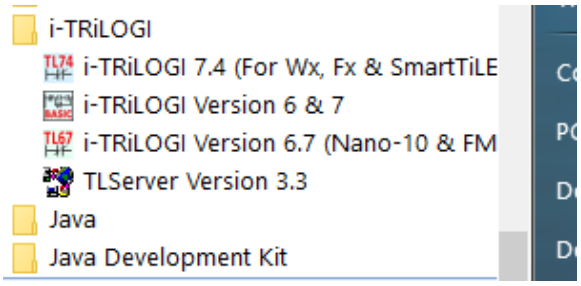
```
C:\TRiLOGI
```

The setup program will install two version of i-TRiLOGI:

1. Version 6.x – An integer version of i-TRiLOGI only for Nano-10, FMD88-10 and FMD1616-10.
2. Version 7.x – A floating point version of i-TRiLOGI for Wx100, Fx2424, Fx1616-BA and any SmartTILE-Fx based custom PLCs.

The two programs versions are stored in the subfolder TL6x and TL7x within the base TRiLOGI folder.

To start up i-TRiLOGI after installation, click the “Start” menu button in Windows, scroll to the i-TRiLOGI folder icon to select the program version you wish to run, as shown below:



Alternatively, you can open the C:\TRiLOGI folder and run the “TL67Launcher.exe” program. The launcher will present you with a menu to select the program version you want to run (depending on your PLC version).

Alternatively, you can open C:\TRiLOGI\TL7x folder and run the TL7x.exe file to start up i-TRiLOGI version 7.x directly.

Likewise, you can also open C:\TRiLOGI\TL6x folder and run the TL6x.exe file to start up i-TRiLOGI version 6.x directly.

What Is The Difference between Version 6.x and 7.x ?

Both version 6.x and 7.x have identical user interface and are often interchangeable. The main difference is that version 7.x supports many additional commands, especially the commands that have to deal with floating point variables and operations. In the case of Version >= 7.4 many additional commands have been added to support the new HMI and WiFi found only on the Wx100 PLC. Hence if you need to use these new capability you must use i-TRiLOGI 7.4 or later to program the Wx100 PLCs.

i-TRiLOGI Version 7.5 – From firmware version >= F94.2, Wx100 also supports new commands to enable it to publish and/or subscribe MQTT messages to any MQTT broker. In addition, two new functions are added to simplify JSON message decoding and encoding for applications that require you to encode or decode MQTT messages in JSON format. You will need to install new i-TRiLOGI version 7.5 or later to compile program that contains these MQTT and JSON related commands. (**Download MQTT Quick Start Guide (https://triplc.com/documents/MQTT_Quickstart1-Connect_Wx100_to_HiveMQ.pdf)**).

Note: Version 6.x only support integer math operation. If you intend to write programs that will run unmodified on both Fx and FMD PLC then you should only use Version 6.x since it is a subset of Version 7.x. You can even transfer any program written using Version 6.x into any PLCs that do support version 7.x. Likewise, if you write a PLC program using version 7.x, but did not use any features that are not available in Version 6, then it is possible to transfer the same program to the lower end of the PLCs as well.

Chapter 2 - Introduction to i-TRiLOGI

a) i-TRiLOGI Client / Server Architecture and TCP/IP Network Communication

i-TRiLOGI is the software which you use to create your ladder logic + TBASIC program . When you want to transfer your program to the PLC, i-TRiLOGI will be the party to initiate a network connection with the PLC. Hence i-TRiLOGI is called a network “Client” program.

The TRi Super PLCs such as the Wx100, Fx FMD or Nano-10 PLC have a built-in TCP/IP web server called the "F-Server". The i-TRiLOGI client can connect to the F-Server directly for programming and online monitoring over any TCP/IP network, including WiFi and the Ethernet. F-Server also provides file space that lets you install a control webpage written in Javascript for controlling the PLC operation via any web browser. This is known as Web Application.

The beauty of the client/server configuration is that it does not matter whether the server and client are located next to each other or half a world away 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.

Another important advantage of client/server architecture is that multiple clients may access the same server simultaneously. Hence you can run several copies of the i-TRiLOGI clients at different places around the world simultaneously for troubleshooting a single PLC. You can also run the i-TRiLOGI client AND the TRi-ExcelLink (a separate data collection software) clients simultaneously to access the same PLC.

b) Direct Serial Port Communication

Besides being able to connect directly to a network capable PLC such as Wx100 via the TCP/IP network, i-TRiLOGI 6.7 and 7.4/7.5 also directly supports serial port communication with the PLC via either the RS232 or the RS485 interface. However, this requires that the PC running the i-TRiLOGI software to have an RS232 or RS485 interface. You can easily add a RS232 or a RS485 port to a PC via a USB-to-serial adapter (e.g. the U-485 (https://www.triplc.com/u485.htm) adapter supplied by TRi).

Note that when i-TRiLOGI connects to the PLC via serial port the program will take exclusive possession of the serial port so no other program may use the same serial port until i-TRiLOGI disconnects from the PLC. As a result, only a single instance of iTRiLOGI can communicate with the PLC via direct serial port connection at any time.

c) Using The TLServer Software

Besides the i-TRiLOGI client software, the i-TRiLOGI setup program will also install a "TLServer" software which might be mentioned frequently throughout this manual.

TLServer is a legacy TCP/IP server program that runs on a PC and serves as a TCP/IP gateway for earlier generations of TRi Super PLC (those built between year 2000-2010) that did not have built-in Ethernet or WiFi network interface. TLServer will receive TCP/IP packets sent by the i-TRiLOGI client and then relays them out of the PC's serial port (RS232 or RS485) to the PLC. The response from the PLC will also be received by the TLServer software via the serial port and TLServer in turn package the serial response into TCP/IP packets and return them to the i-TRiLOGI client via the TCP/IP network.

Earlier version of i-TRiLOGI software (version 5 to 6.5x and version 7 to 7.1x) did not directly support serial interface communication with the PLC. So the TLServer software is still required for i-TRiLOGI to communicate with the PLC via serial port. Basically i-TRiLOGI client will connect to the TLServer as if it is a PLC and TLServer

is the one that interface to the PLC via the serial port.

Since i-TRiLOGI 6.7 and 7.4/7.5 now supports direct serial communication with the PLC, it is no longer quite necessary to run the TLServer software unless you want to make use of some of the file services it can provide to the PLC. Note that if you use TLServer as the interface to the PLC' serial port then multiple instances of the i-TRiLOGI can connect to the TLServer simultaneously, which is an advantage over the direct serial connection from the i-TRiLOGI software to the PLC.

Chapter 3 - i-TRiLOGI to PLC Interface

This chapter will discuss how to interface to the 'Super' PLC in terms of the most direct physical connection and the necessary software components for the purpose of programming and monitoring the PLC.

These procedures will be covered in a bit of depth to explain certain aspects, so you may wish to reference the quick-connect guides instead, which are in the appendix of this document, if you would like simple connection directions without explanations.

Physical Connection

There are a few direct ways to physically connect to the Super PLCs for programming and monitoring from the i-TRiLOGI software:

- SERIAL PORT (RS232 / RS485)
- ETHERNET (Wired, RJ45 terminated) – Fx, FMD and Nano-10 PLCs
- WiFi (wireless) – Wx100 PLC

Software Interface

The i-TRiLOGI programming environment (client) is required to interface to the PLC for programming and online monitoring whether the physical connection is serial, Ethernet or WiFi, but the server could be either TLServer or the F-Server depending on the PLC model and type of connection.

3.1 Communication Interface

Four methods of Communication interface to the PLC are described here: Serial RS232, Serial RS485, Ethernet and WiFi

3.1.1 Serial Port (RS232/RS485)

All TRi Super PLCs have at least one RS485 port and all but the Nano-10 and Wx100 also have at least one RS232 port. Either of these ports can be used to connect a PC to the PLC for programming, monitoring or setup.

RS232

TRi Super PLCs with a DB9 serial port (9-pin connector) can interface via RS232 directly to a PC if it has the same type of port or indirectly through a USB port. Most modern PCs now only have USB serial ports, so this would be the only option.

For USB connection, you will need a USB to RS232 adapter that plugs into the USB port of a PC and the DB9 serial port of the PLC. Typically a straight through DB9 cable will need to be added to the adapter to extend the connection since USB to RS232 adapters are fairly short. USB to RS232 port can easily be purchased online from a number of outlets.

RS485

All TRi Super PLCs have at least one two-wire RS485 serial port (screw terminal connector) and can only interface to a PC via RS485 indirectly through a USB port.

For USB connection, you will need a USB to RS485 adapter that plugs into the USB port of a PC and has a two wire connection to the PLC (usually labeled A & B or D+ & D-). Twisted pair cabling should be added to the adapter to make a connection since USB to RS485 adapters would not typically provide wire connection.

3.1.2 Ethernet

Introduction

All TRi Nano-10, FMD series, Fx-series, and any SmartTILE-Fx based Super PLCs all have an Ethernet port built in, which can be used for programming and monitoring with the i-TRiLOGI software, as well as communication with other Ethernet devices and software.

These PLCs can connect to the PC running TRiLOGI many ways as follows:

1. Wired connection to a router that the PC is also connected to (PC connection can be wired or wireless).
2. Direct connection to the PCs wired Ethernet port via crossover cable

Only the first option, which is most common, will be described here. The second option (direct connection via crossover cable) is described in the each PLC's User Manual.

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

Before You Begin

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

- Find out what your routers gateway address is (typically 192.168.1.1 or 192.168.0.1) and what static IP addresses are free to use with your PLC.*NOTE: if the routers gateway address is "192.168.1.1", the default PLC IP address (192.168.1.5) will most likely work unless it is already used by another device on the same network. If it is free to use, the next two steps can be skipped as the PLC will already be able to connect to the LAN.*
- Connect to the PLCs serial port from the PC with TRiLOGI and TLServer.
- Edit the PLC network settings using the Ethernet & ADC Configuration tool from the "Controller" menu in TRiLOGI. Only the IP address is necessary to configure for basic connection to the LAN.

Network Wiring

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

In this case, all that needs to be done is connect the PLC to the router using a standard Ethernet cable.

If you are on a corporate network, then you will need to consult the IT administrator to get the PLC connected to the network. If the IT department does not allow you to connect the PLC to the corporate network then you can simply acquire a simple Ethernet switch so that both your PC and your PLC can be plugged into.

3.1.3 WiFi

The latest generation of TRi Super PLC such as the Wx100 has built-in WiFi which allow the PLC program to be transferred over-the-air. The most important step for a WiFi device is to connect to the WiFi network and two important parameters must be entered into the PLC to enable it to connect to the WiFi netowrk:

a) WiFi SSID
b) WiFi Passkey

When shipped from the factory the WiFi-equipped Super PLC will be programmed to a known SSID and passkey used in the factory for testing. The user could also setup a WiFi router that uses the same SSID and passkey for immediate connection. Alternatively, the WiFi equipped PLC provides other means of programming the WiFi parameters, such as setting the PLC as a WiFi software AP, using the built-in HMI (Wx100) to program the parameters, or using the serial port and i-TRiLOGI to configure the WiFi parameters.

The various connection methods are described in details in the PLC's User Manual. E.g For Wx100 PLC please refer to the Wx100 User's Manual Chapter 2.1 (https://docs.triplc.com/#2334) for details.

3.2 Software Interface

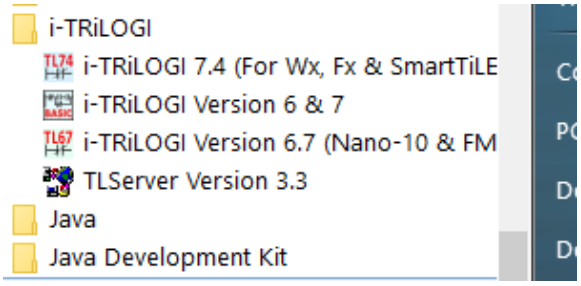
i-TRiLOGI is the client program that is used as the PLC programming environment. It is also able to communicate with the PLC to perform two important functions:

1. Program Transfer
2. Online Monitoring

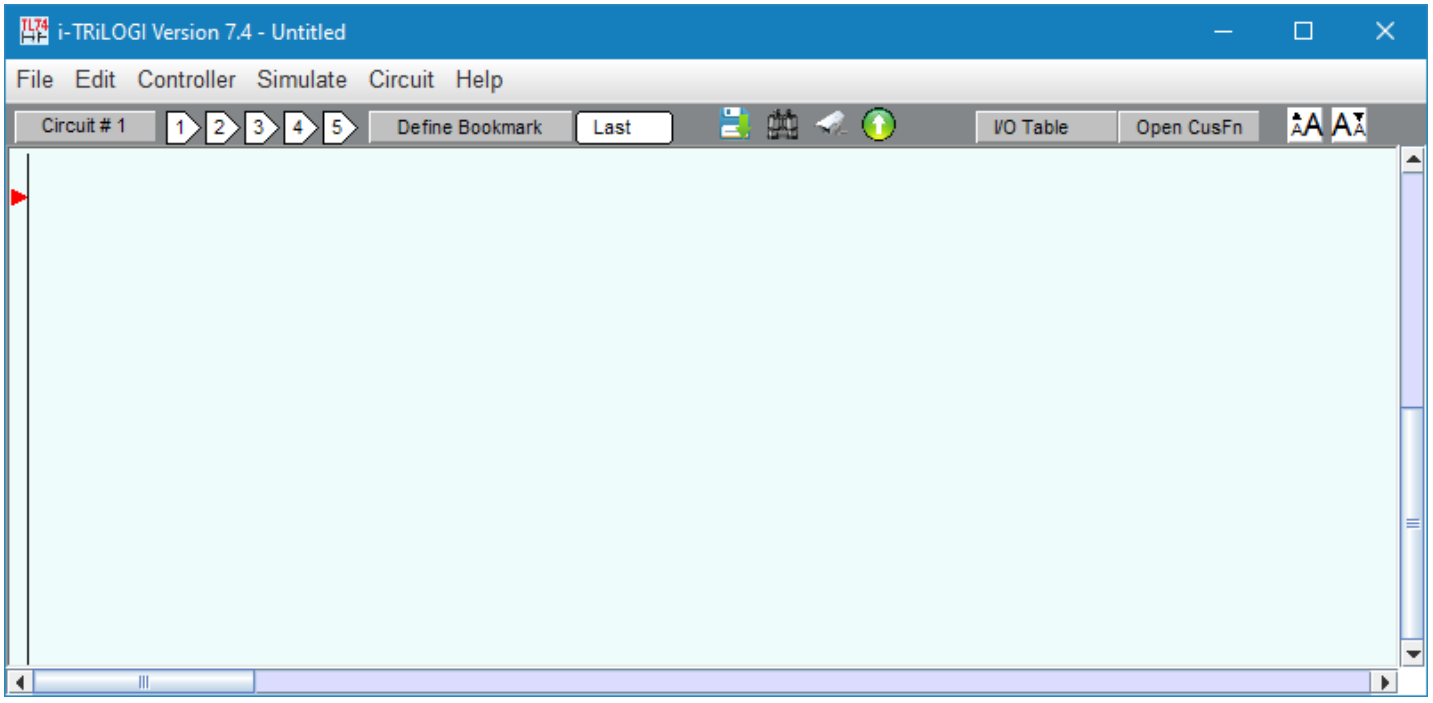
i-TRiLOGI can communicate with the PLC via either the serial port (RS232 or RS485) or via Ethernet or WiFi using TCP/IP networking protocols.

In this section we will only discuss how to use i-TRiLOGI software to perform these two communication functions over the various communication media. We will leave the discussion of how to write Ladder+BASIC program using i-TRiLOGI in the next few Chapters.

Now lets start i-**TRiLOGI** version 7.x by selecting from the Start menu as shown below:



The i-TRiLOGI editor window should open as shown below:

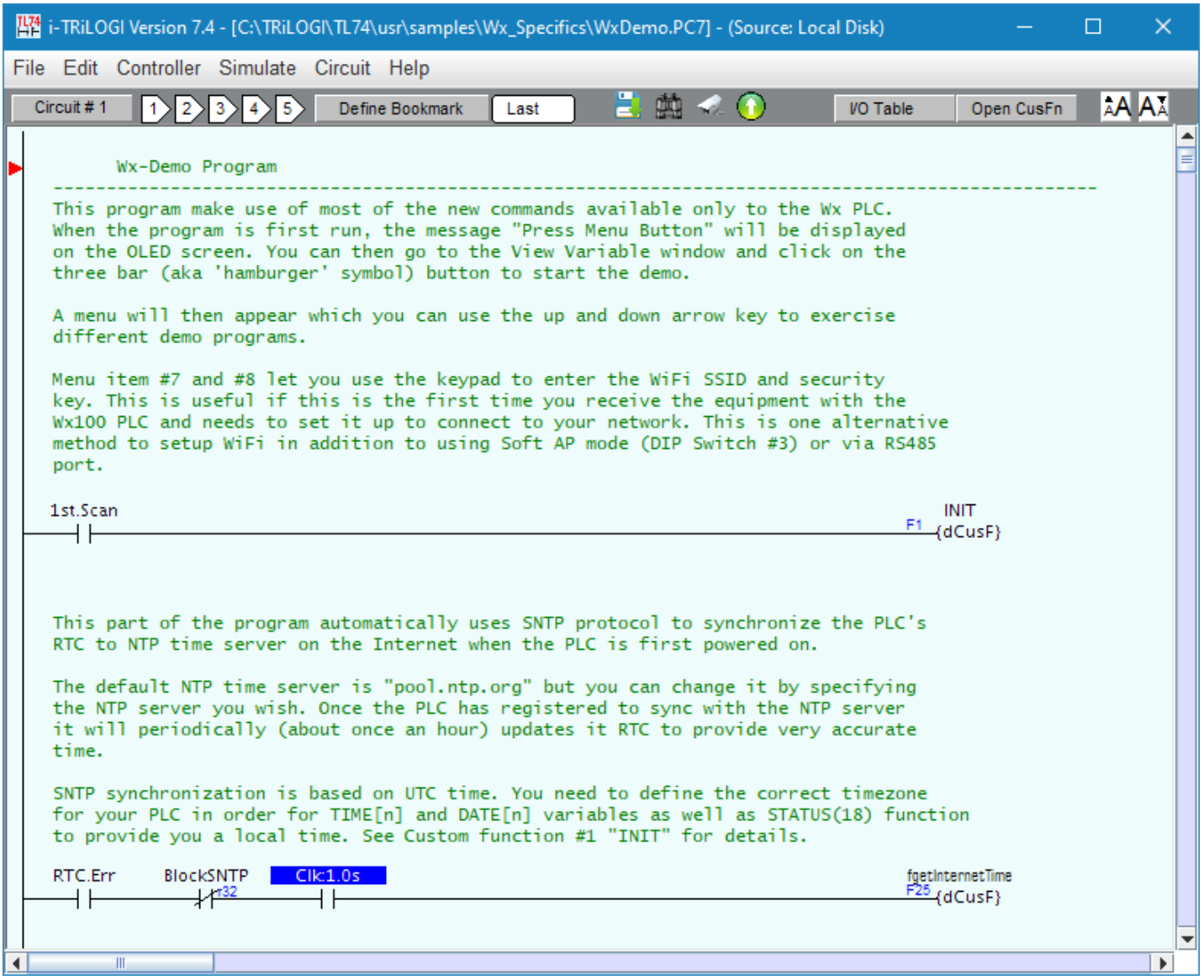


For the purpose of this tutorial, we will load the “WxDemo.PC7” program. This is a sample program that works only on the Wx100 PLC. This is included in the i-TRiLOGI installation at the following folder:

C:\TRiLOGI\TL74\usr\samples\Wx_Specifics

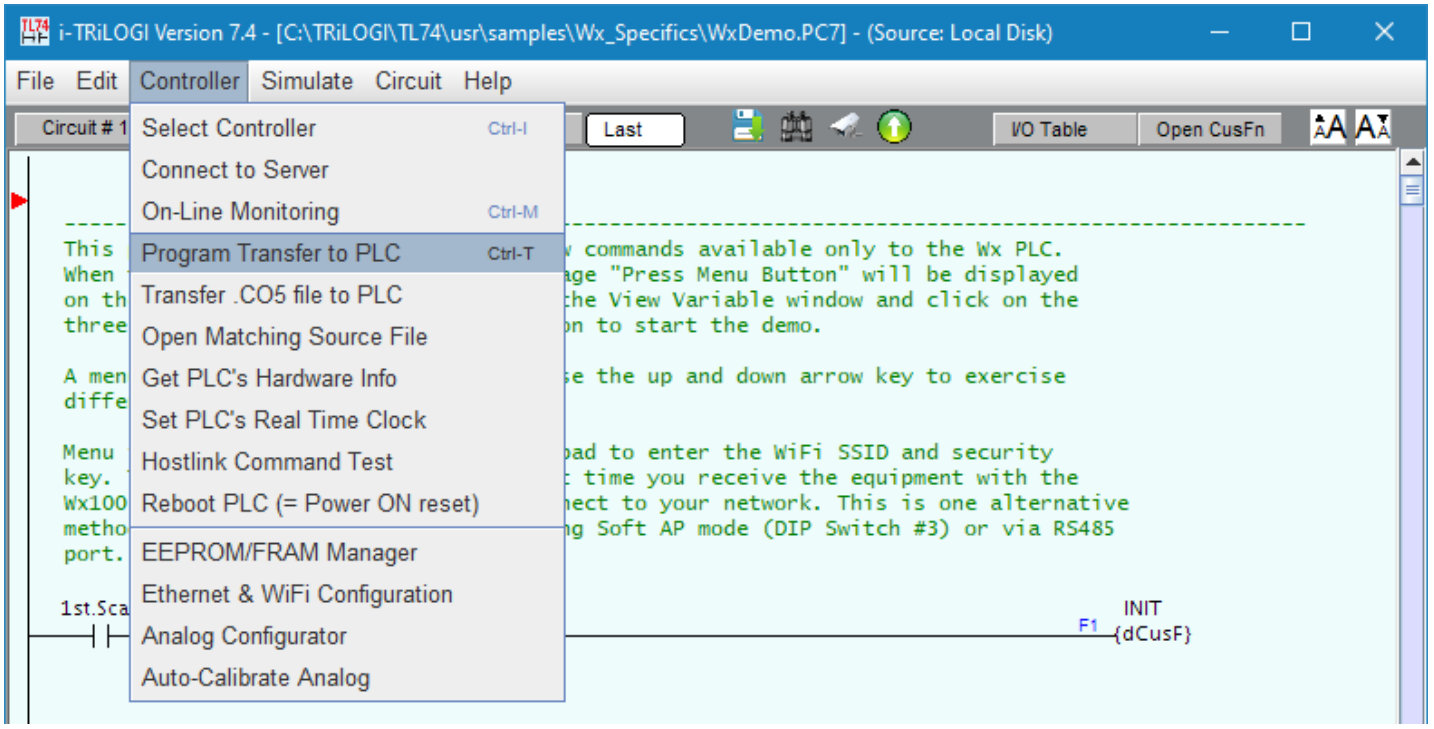
If you are using i-TRiLOGI with other PLC models such as the Fx2424 and Fx1616BA, you can load in any other working demo program that can be found in the folder: “C:\TRiLOGI\TL74\usr\samples\” and continue the same steps as below.

The WxDemo.PC7 should look like the following after you have loaded it from the “\samples\Wx_Specifics” sub-folder.

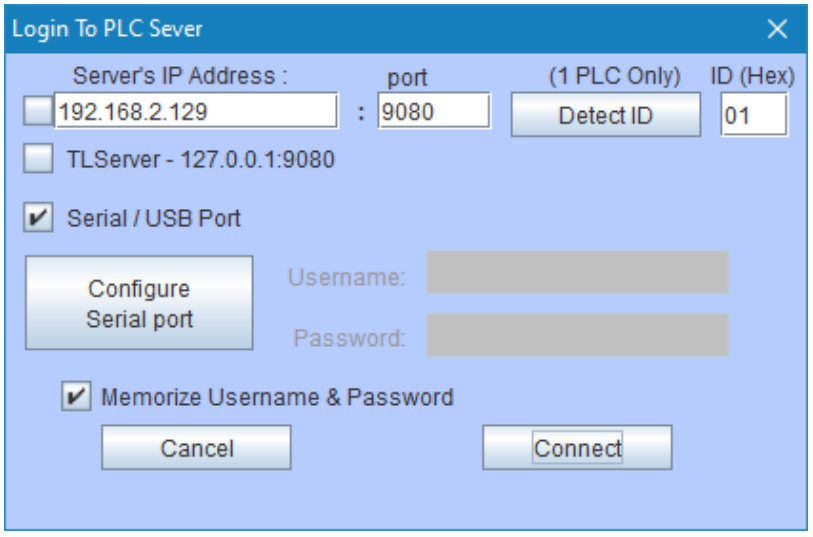


3.2.1 Program Transfer

Select “Program Transfer To PLC” from the “Controller” menu, or press CTRL+T on the keyboard to begin program transfer to the PLC:



You will be presented with the “Log In To PLC Server” Screen, as follow:



Here you select the communication interface that you want to use to communicate with the PLC:

a) **Connect via TCP/IP network**:

If your PLC is already connected to the network and you do know its IP address, then simply enter the PLC’s IP address into the textbox under the heading “Server’s IP Address and its check box will be selected automatically. The PLC FServer port by default is 9080 (unless you have changed it previously) and you can leave it as such.

b) **Connect via TLServer**

If you need to use TLServer (https://docs.triplc.com/tl74/#4711) to communicate with the PLC, you must start the TLServer software on the same PC as the i-TRiLOGI software and configure the serial interface on the TLServer. When TLServer is properly configured you can then go back to i-TRiLOGI to select the checkbox for "TLServer" whose IP address is the localhost: 127.0.0.1:9080. Since TLServer is a legacy program that is no longer needed for programming any Super PLC, we will not spend too much time describing it here but will instead refer you to download the TL6 Reference Manual from:
https://triplc.com/documents/TL6ReferenceManual.zip
(https://triplc.com/documents/TL6ReferenceManual.zip)

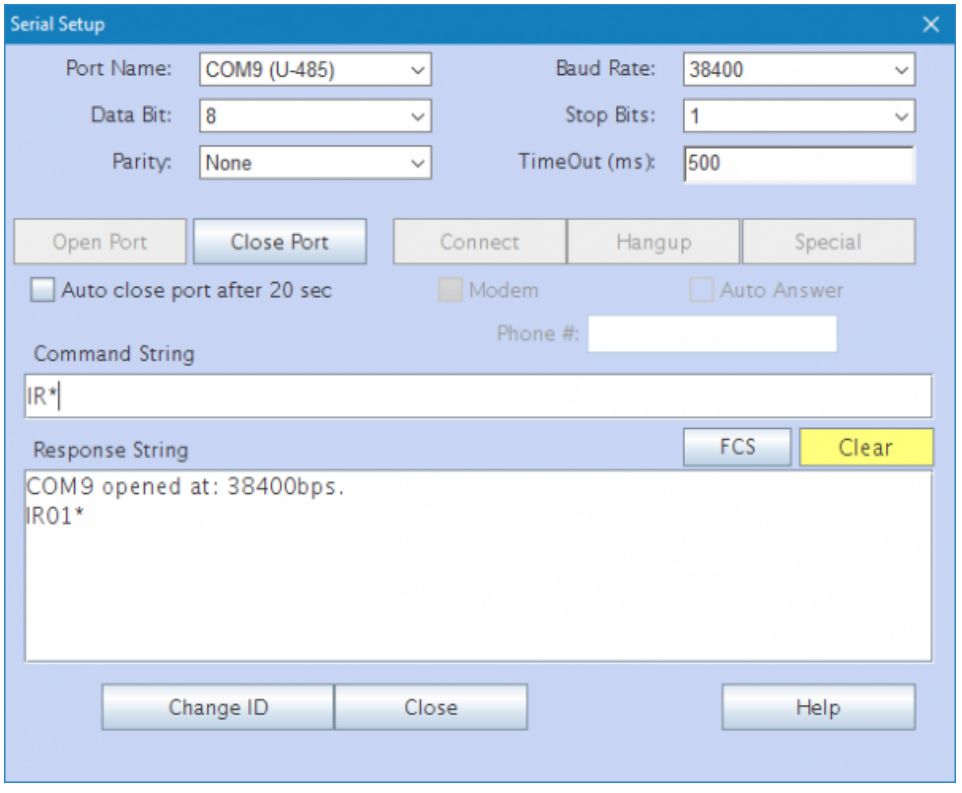
c) **Connect via Serial/USB port**

If your PLC is not connected to the WiFi or Ethernet network, or if you don't know the IP address of the PLC, then you should still be able to communicate with the PLC via its serial port (RS232 or RS485). Simple check the "Serial/USB Port" check box and the "Configure Serial port" button will appear.

Normally if there is only one serial port found on the PC, i-TRiLOGI will select and open it automatically. The COM port will assume the default settings of 38400 bps, 8 data bit, 1 stop bit and no parity.

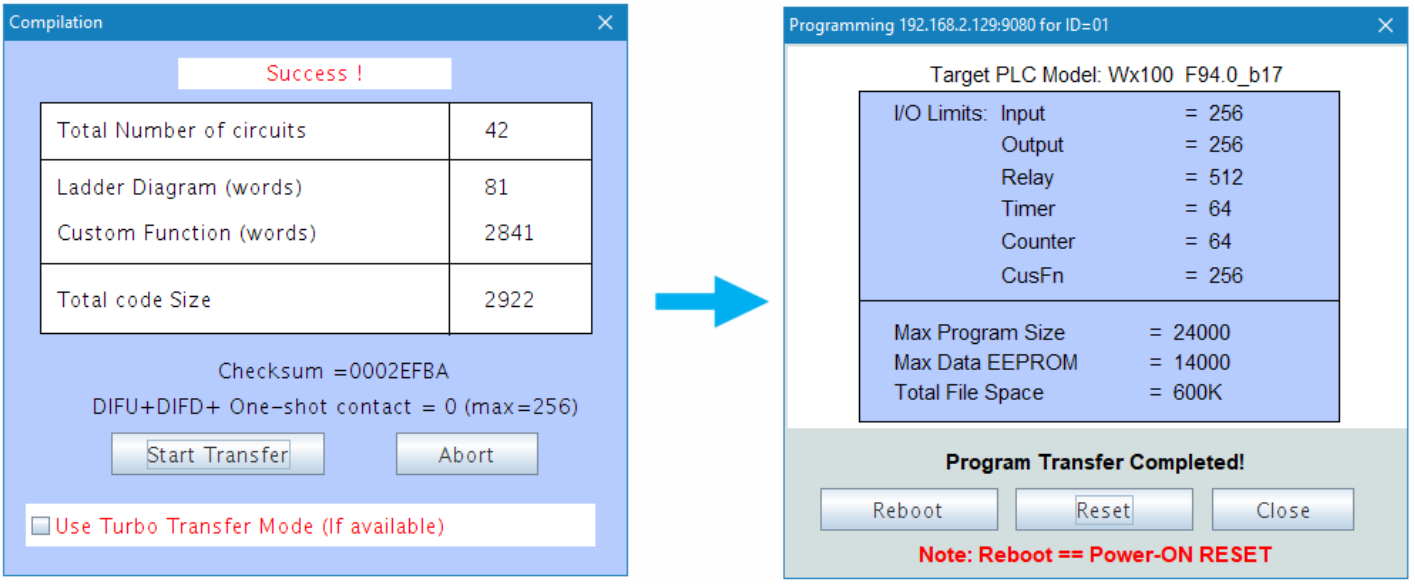
Assuming that there is only one PLC connected to the PC's serial port, you can simply click the "Detect ID" button on the login panel and i-TRiLOGI will send a point-to-point "IR*" command to the PLC to retrieve the ID. If the ID (00 to FF) appears on the ID text field it indicates that the connection is successful and you can continue by clicking the "Connect" button to connect to the PLC using the serial COM port.

However, if the communication failed and an error message appears, then you will to check further by clicking on the "Configure Serial Port" button. The following panel should appear:



You should then click on the “Port Name” choice box to select the serial port that is used to connect to the PLC. Make sure that the baud rate, number of data bit, number of stop bit and the parity match that of the PLC (Unless changed by the SETBAUD command, the default settings when the PLC is booted up is always 38400bps, 8 data bit, 1 stop bit and no parity). Click on the “Open Port” button if it isn’t already opened. If the COM port is properly opened it should report e.g. “COMxx opened at 38400bps”.

If there is only one PLC connected to the serial port, you can enter the command “IR*” on the Command String text box and press the enter key. The PLC should send back a response such as “IR01*” indicating that its ID is “01”. Once you get a response it indicates that the connection is successful and you can now close this window and go back to the login dialog screen, click “Detect ID” and it should retrieve the correct ID. Then click the Connect button and i-TRiLOGI will begin to compile and transfer the “WxDEMO.PC7” program to the PLC as illustrated by the following screen captures. At the end of the program transfer, click either the Reset or Reboot button to complete the process.

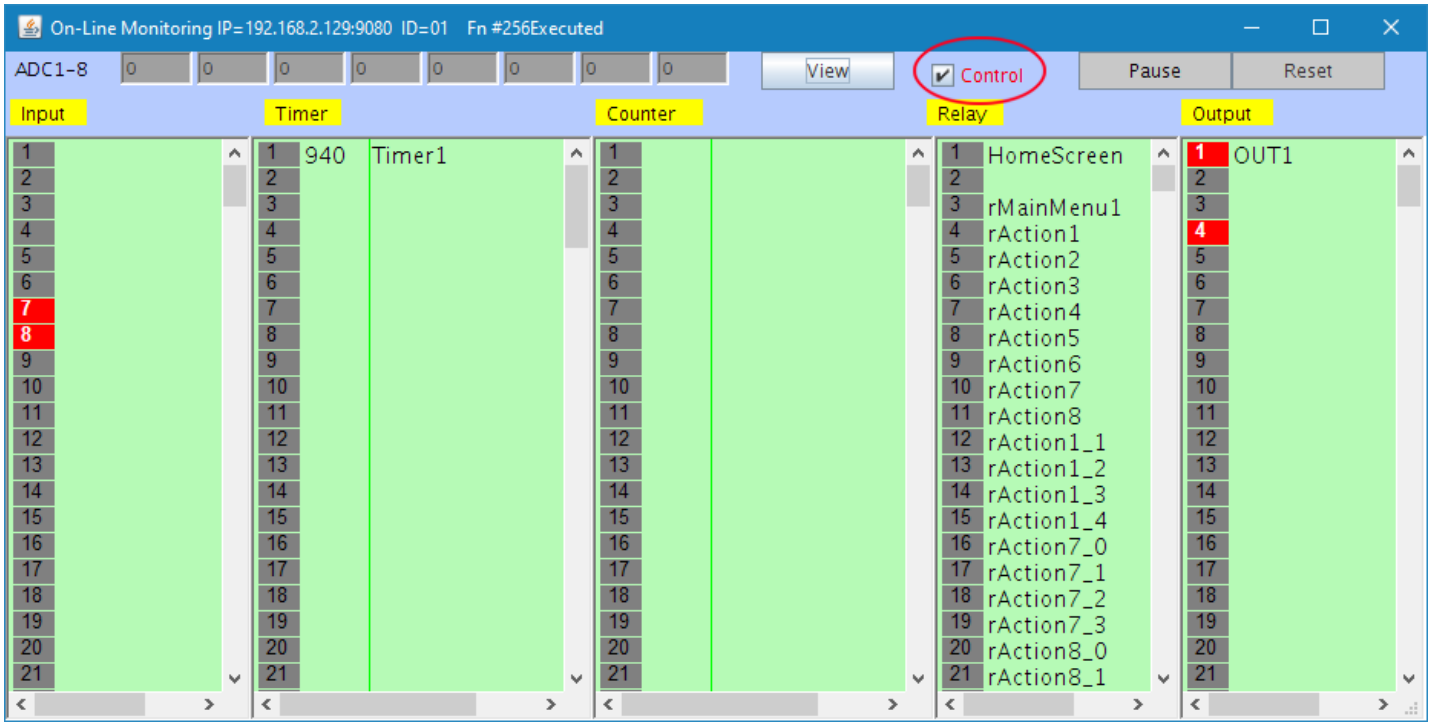


3.2.2 On-line Monitoring

- **Note:** This section intends to only give you a quick overview of the on-line monitoring capability of the i-TRiLOGI software which is an extremely important tool for programmer. We will describe what you can do with the online monitoring action in greater details later when we discuss the ladder+BASIC programming and debugging.

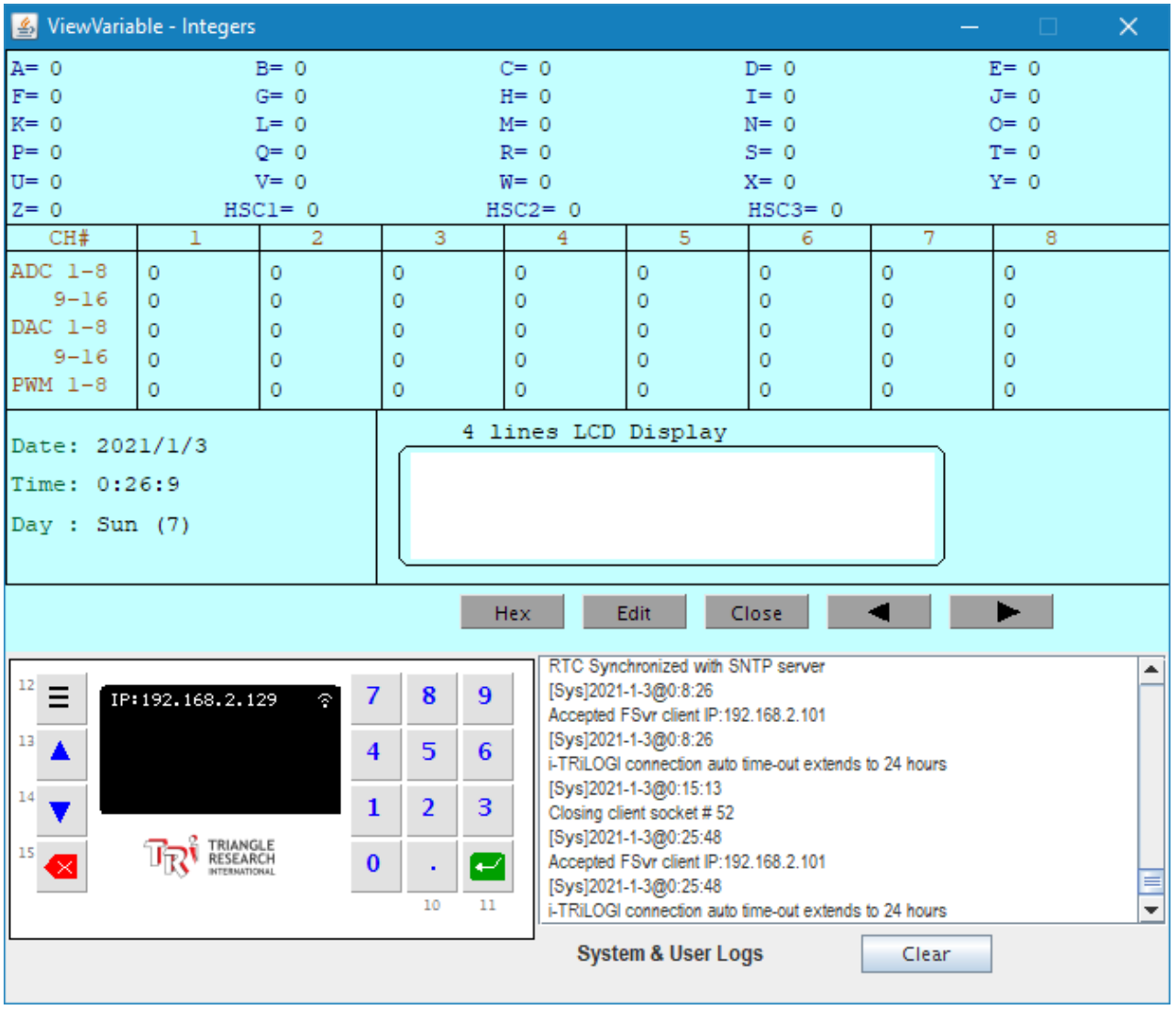
First, click “Controller” menu from i-TRiLOGI main menu and select “On-line Monitoring”. You will be presented the same login screen as described in the last section (https://docs.triplc.com/trilogi/#4945). You can select to connect to the PLC via either TCP/IP, TLServer or serial port and once connected you will see the

following screen. On this screen you can see the ON/OFF status of all the PLC's input, output, relay, timer and counter contacts. You can also observe the timer and counter present values as they count up or count down, as well as values of the first 8 ADCs in the PLC.



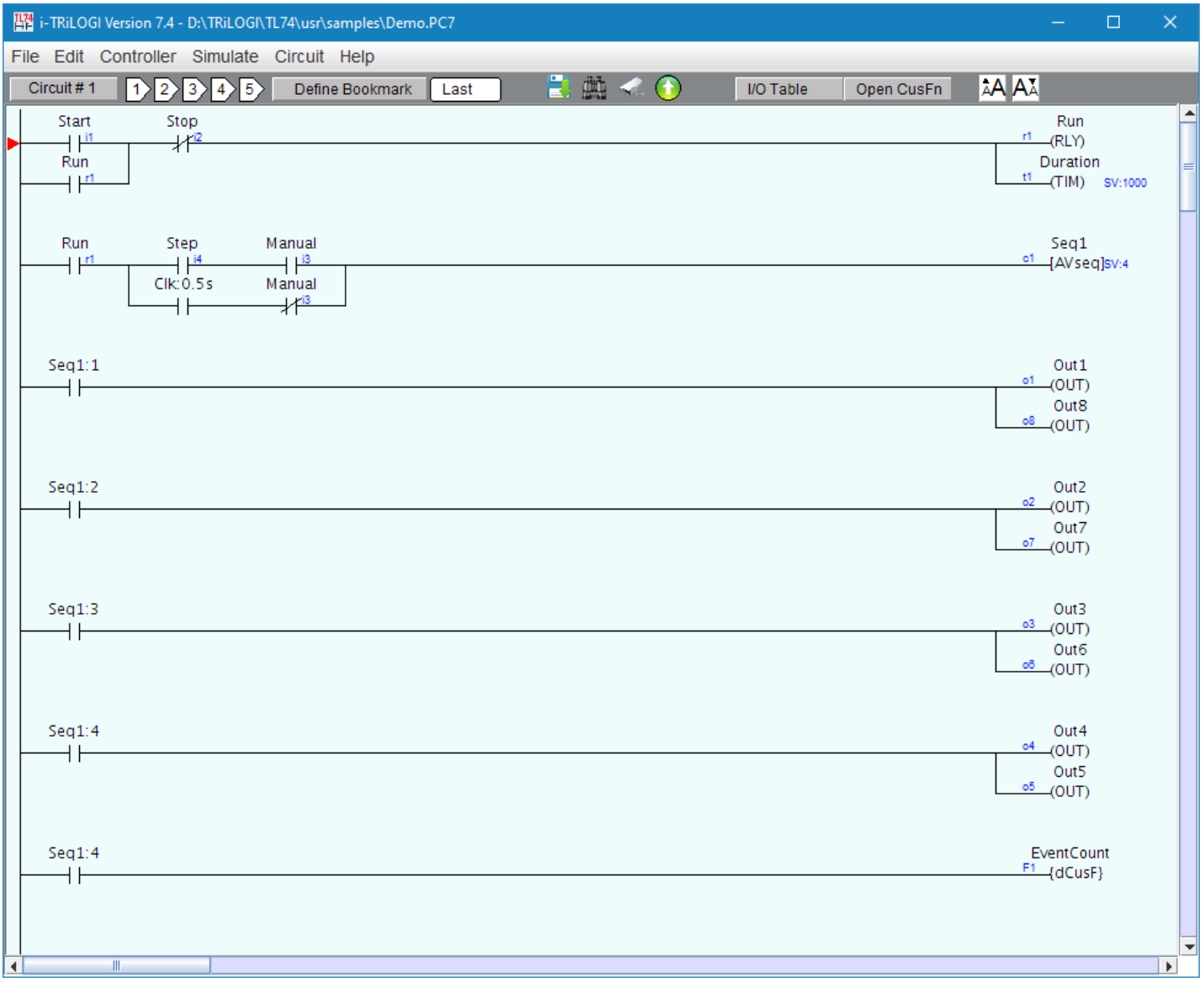
If you enable the control checkbox (see the red circle) you can also actively turn ON and OFF any of these contacts by clicking on the I/O label or its indicator lamp using the mouse click. These changes can be either momentary (with left mouse click) or persistent (with right mouse click – at least until the PLC program changes their logic states). Hence the online monitoring is also a CONTROL screen, allowing you to control the PLC's operation to a certain extent.

Besides ladder logic elements such as I/Os, timers and counters, all TBASIC internal variables, such as data memory, numeric and string variables, PWM settings etc can also be viewed by clicking on the "**View**" button on the On-line Monitoring screen. You will then be presented with the following "View Variable" window. There are multiple screens in the View Variable window that you can scroll by clicking on the right and left arrow on the screen or by using the keyboard. We will describe what you can do with the View Variable screen in greater details in later sections.



Chapter 4 - Ladder Logic Programming

4.1 Video Tutorial



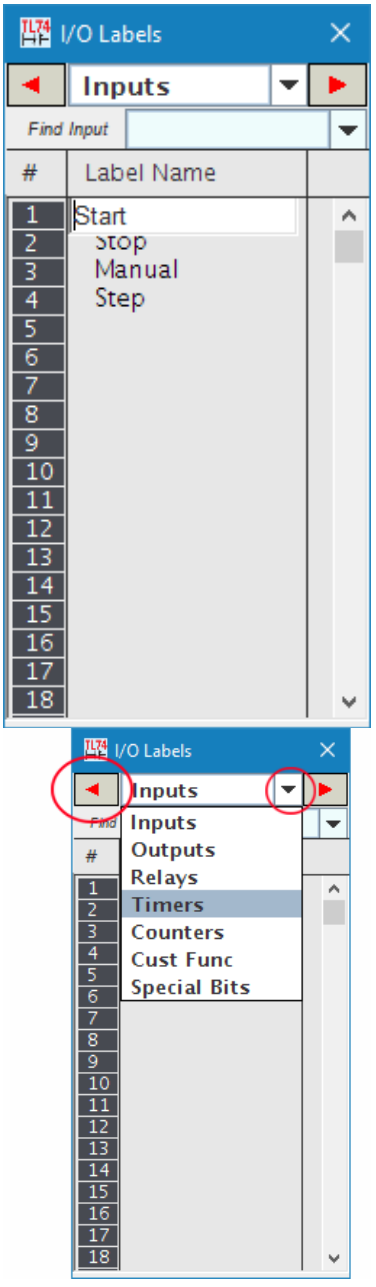
Simply follow the steps below to create your first ladder logic circuit.

Step 1 - Define I/O Label Names

- Open pull-down “File” menu and select “New”.
- 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 (counter) : Seq1



1. Click on the I/O Table button located on right hand side of the upper status bar to open up the I/O label editing Window. You can also achieve the same with the <F2> function key.
2. 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.
3. Move the 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.

4. 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.

5. Complete entry of the other two input label names “Manual” and “Step” as above. Note that i-TRiLOGI Version 6.2 and up allows I/O label names of up to 16 characters. However, if you wish to keep compatibility with Version 5.x to 6.1x, you should use no more than 10 characters to define the I/O names. Also, white spaces between names are not acceptable and will be automatically converted to the underscore character (‘_’). e.g. If you enter the name: “F-series PLC” for an I/O, it will be accepted as “F_series_P”.

6. 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

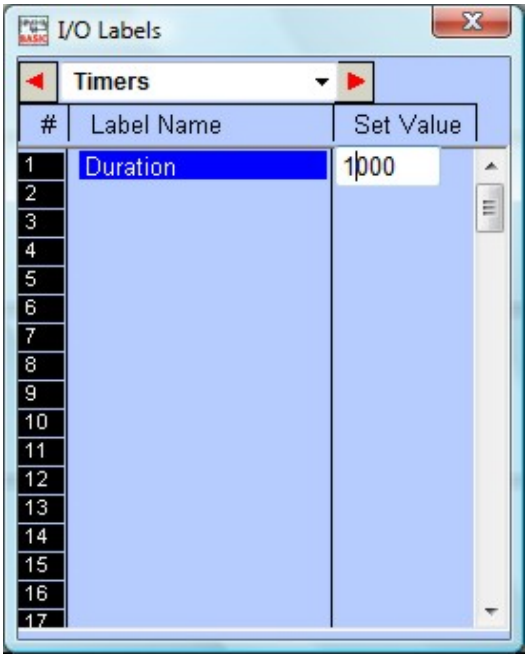
- You can **shift the Items in the I/O table up or down** or insert a new label between two adjacent, pre-defined labels. Simply press the <Ins> key or Right-Click the mouse button to pop up the “Shift I/O” menu, which allows you to shift the selected I/O. However, please note that if you shift the I/O down, the last entry in the I/O table (e.g. Input #256) will be lost.

- From i-TRiLOGI 6.2 and up, shifting of Custom Function Label names will now shift the custom function content along with the label name. (In previous versions of i-TRiLOGI, shifting of the I/O label would not shift the function content, therefore making it untenable to use I/O label shift to reorganize custom functions. Warnings are provided if such an action were to result in overwriting of an existing custom function.

- i-TRiLOGI Version 6.2 and up allows I/O label names of up to 16 characters. However, if you wish to keep compatibility with Version 5.x to 6.1x, you should use no more than 10 characters to define the I/O names.

Step 2 - Timer/Counter Label Names

1. Timer table has an extra column “Set Value” located to the right of the “Label Name” column.
2. 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.
3. 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.



4. 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. i-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 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.

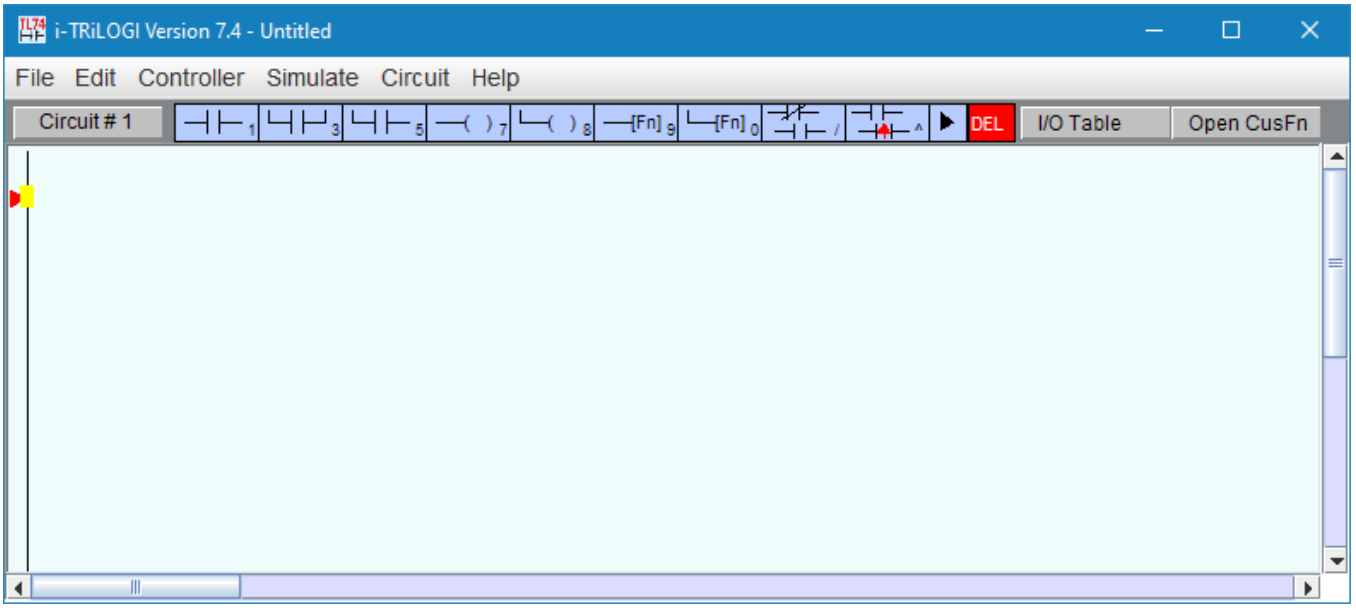
If you are at the “Timers” table, pressing the right cursor key again will bring up the “Counters” table. Enter the name: “Seq1” at the label column for Counter #1. Press <Enter> and the text entry field will be opened at the “Set Value” column. For now, let’s enter a preset value of “4” for “Seq1”.

5. 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>.

Step 3 - Create 1st Ladder Circuit

We are ready to create Circuit #1 as shown below:

1. 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 i-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.

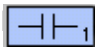
<1> – Left click to insert a normally-open series contact.
<2> – Right click to insert a normally-closed series contact.

<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.

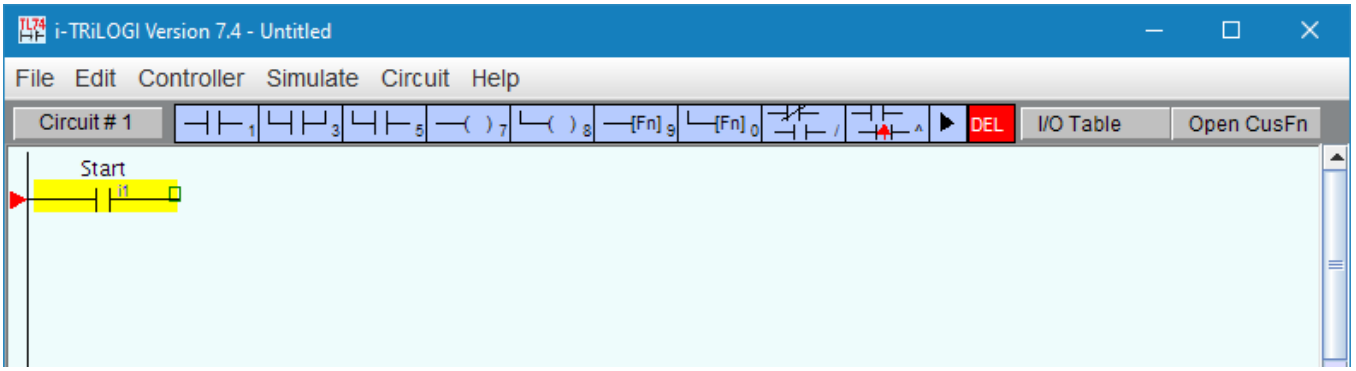
<7> – Insert a normal coil which may be an output, relay, timer or counter.

	<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.
	<^> – Convert the element to a rising-edge triggered contact (one shot)
	Click to move the highlight bar to the right (same effect as pressing the right arrow key). This can be used to move the cursor to a junction which cannot be selected by mouse click.
	Double-click to delete a highlighted element. This acts as a safety against mistake.

2. Now insert the first element by left-clicking on t  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.

   Note: In i-TRiLOGI version 6.x, if you pick any undefined I/O you will be prompted to enter the label name and what you entered will automatically be updated in the I/O table.

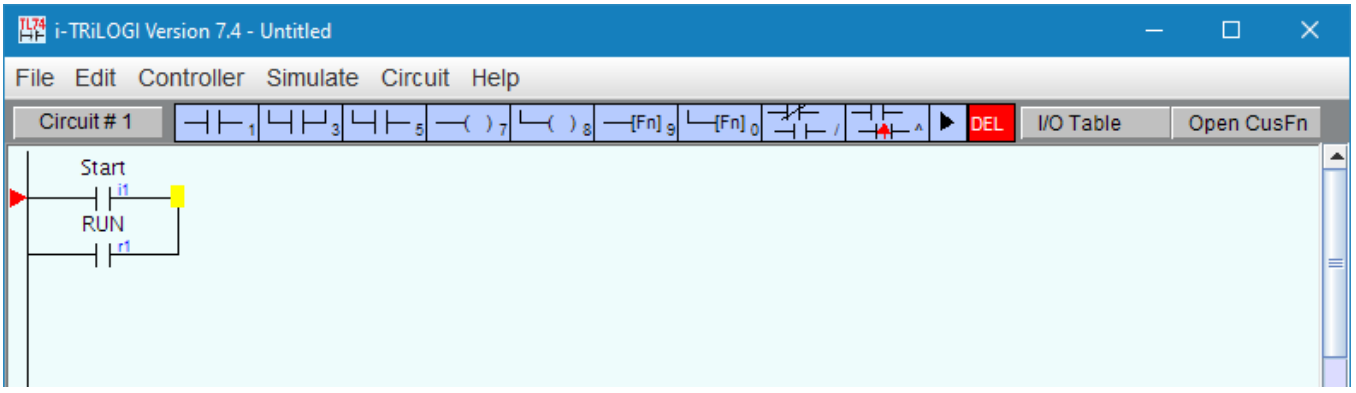
3. The contents in the table are not normally meant to be edited at this moment . Scroll to the “Input” table and click on the label name “Start” and a normally-open contact will be created at Circuit #1.



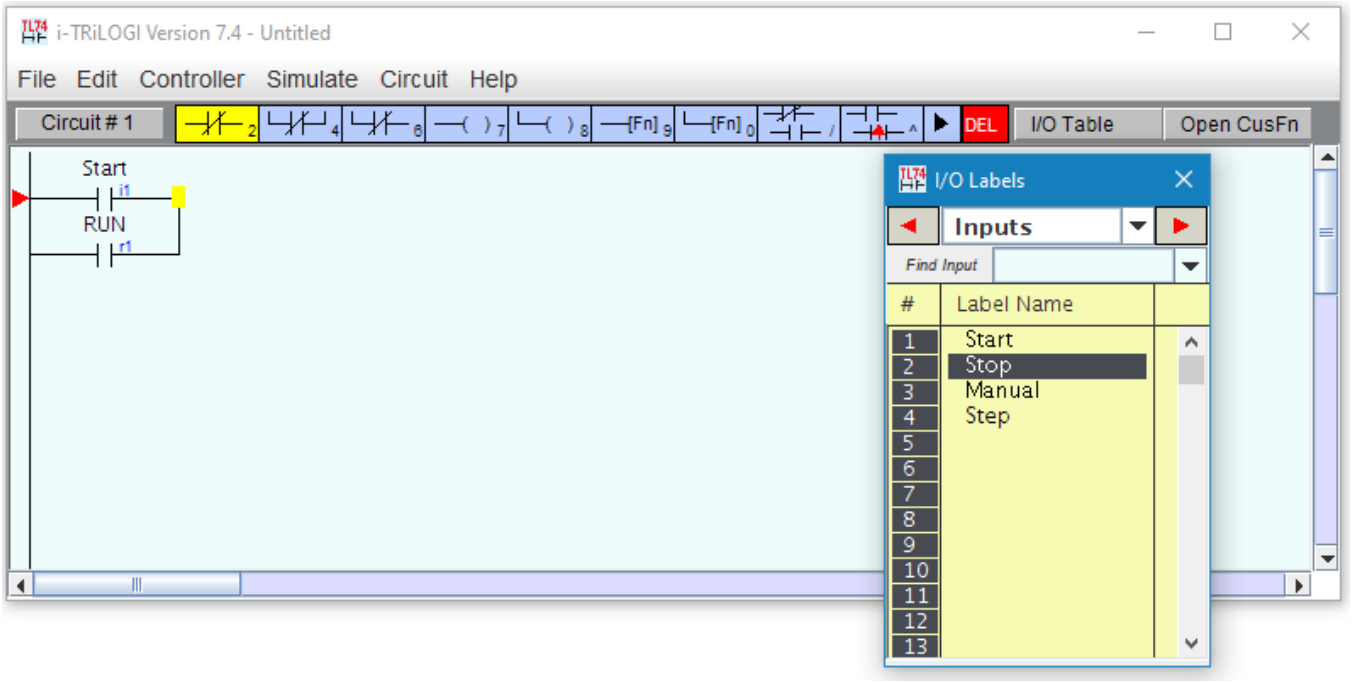
If you observe the highlight bar carefully, you will notice a dark green color square at the right end of the highlight. This indicates the insertion location where a series contact will be attached. You can change the insertion location to the left or the right of the highlight bar by pressing the 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

4. Next, create the contact “RUN” which is parallel to the “Start” contact by left-clicking on t icon. The I/O table will appear again. Scroll to the “Relay” table and select the “RUN” relay.

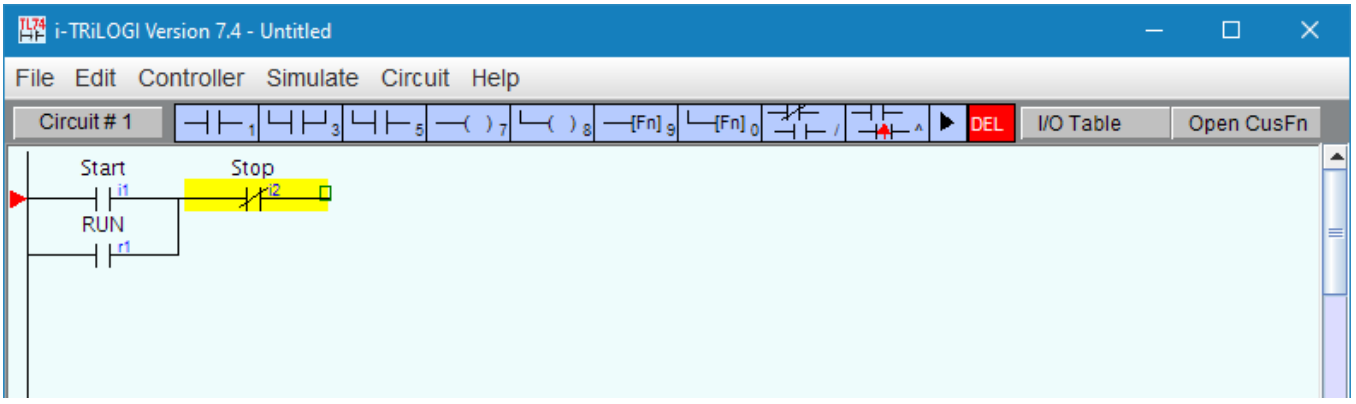
5. To insert the normally-closed “Stop” contact in series with the “Start” and “Run” contacts, you need to move the highlight bar to the junction of the “Start” and “Run” contact. First click on the “Start” contact to select it. Then click on t icon to move the highlight bar to the junction, as follow:



6. Next, **right**-click on t icon. It will change into yellow color normally-closed contact as shown in the following diagram. You are now inserting a normally-closed series contact at the location of the highlight bar. Pick the “Stop” label from the “Input” table to add the series contact.



This is the resulting circuit:

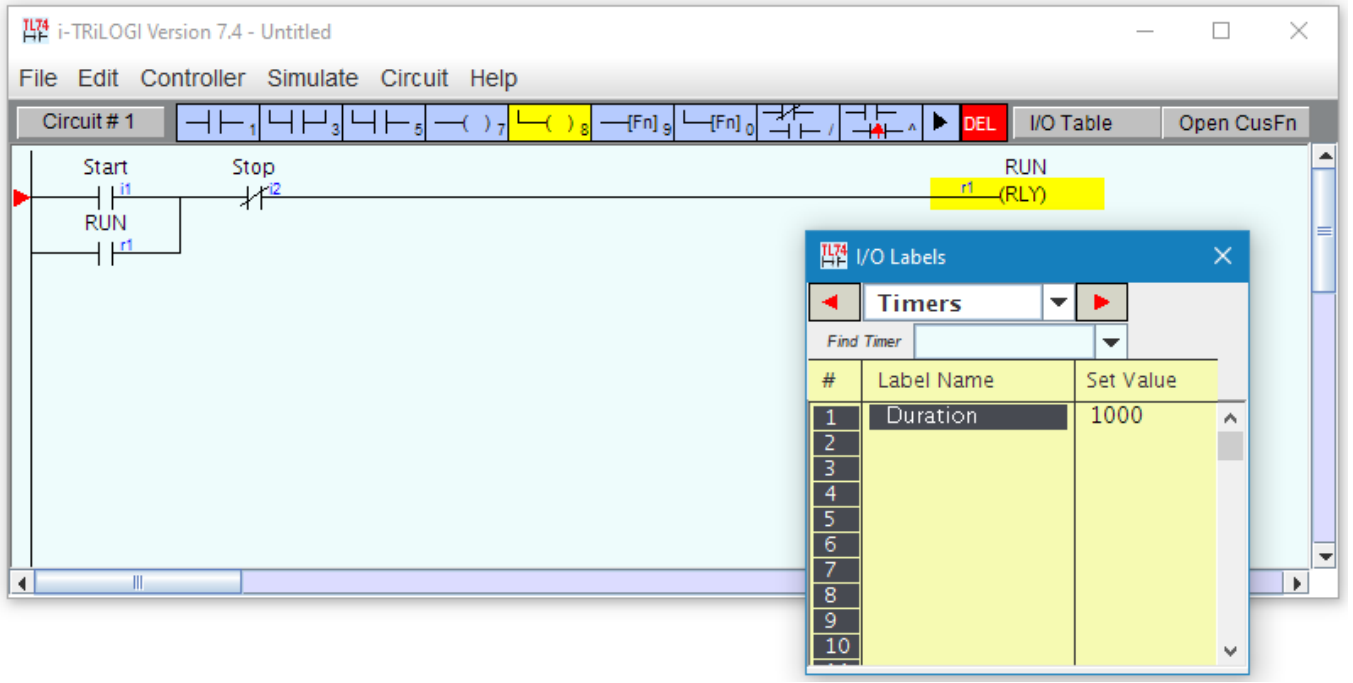


7. We will now connect a relay coil “Run” to the right of the “Stop” contact. Click on t —( )7 icon to insert the

coil. Select “RUN” label from the “Relay” table. Remember that an input can never be used as a coil. Fortunately, i-TRiLOGI is smart enough not to call up the “Inputs” table when you are connecting a coil, to avoid unintentional errors.

Notice that the coil symbol —(RLY) indicates that this is a relay coil, which is helpful in identifying the function of the coil. i-TRiLOGI automatically places the coil at the extreme right end of the screen and completes the connection with an extended wire.

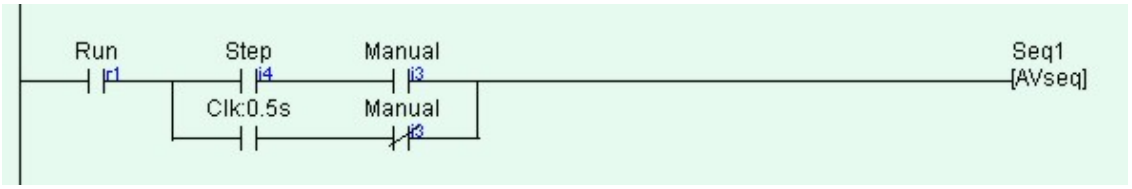
8. Right below the relay coil is a parallel timer coil with label name “Duration”. To create this coil, click on the  icon. This allows you to connect a parallel coil to the existing coil. The “I/O” table will pop up for selection again. Since we want to choose a timer, scroll to the “Timer” table and pick the first timer with the label “Duration” to complete the circuit. Press the <Enter> key once to complete Circuit #1.



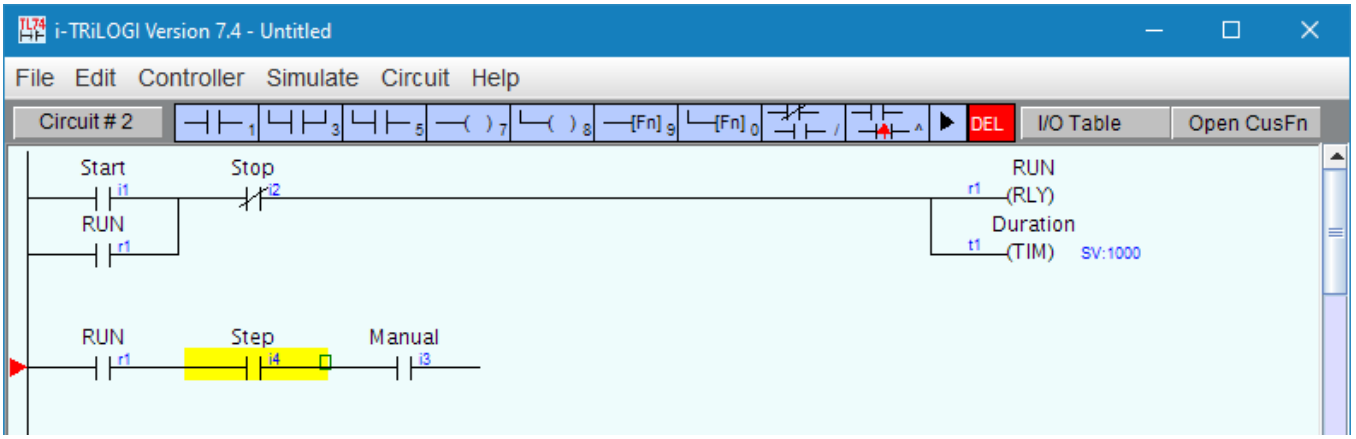
Congratulation! You have just successfully created your first ladder logic circuit. It is that simple! It may be a good time to save your program now by pressing <CTRL-S> key or select “Save” from the “File” menu and give a file name for you new program.

Step 4 - Series and Parallel Contacts

We will now create Circuit #2 as shown below.



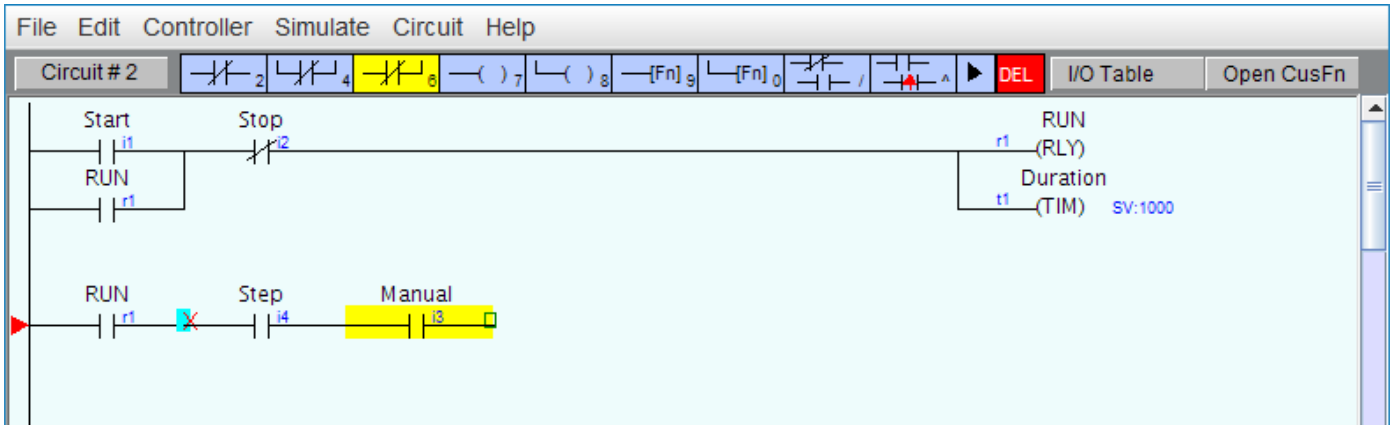
1. Follow the steps listed in STEP 3 to create the following circuit fragment:



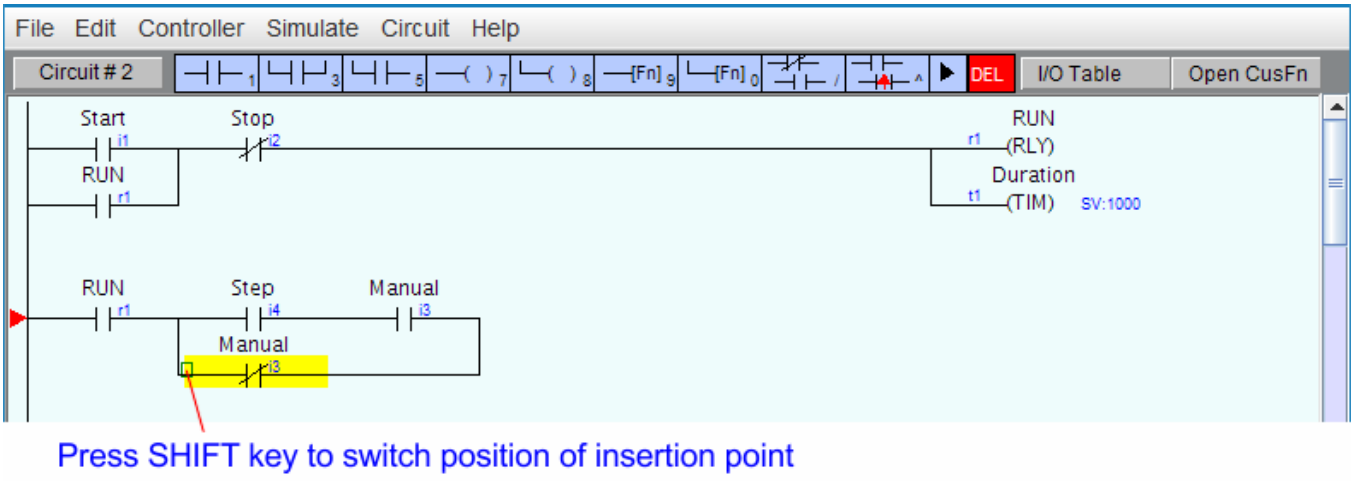
2. Next, we want to enclose the two series contacts “Step” and Manual” with a parallel branch that contains two elements. First, we will create the branch for the N.C. “Manual” contact.

3. Click on the element “Step” to highlight it. Then right-click on t icon to create a N.C. parallel circuit

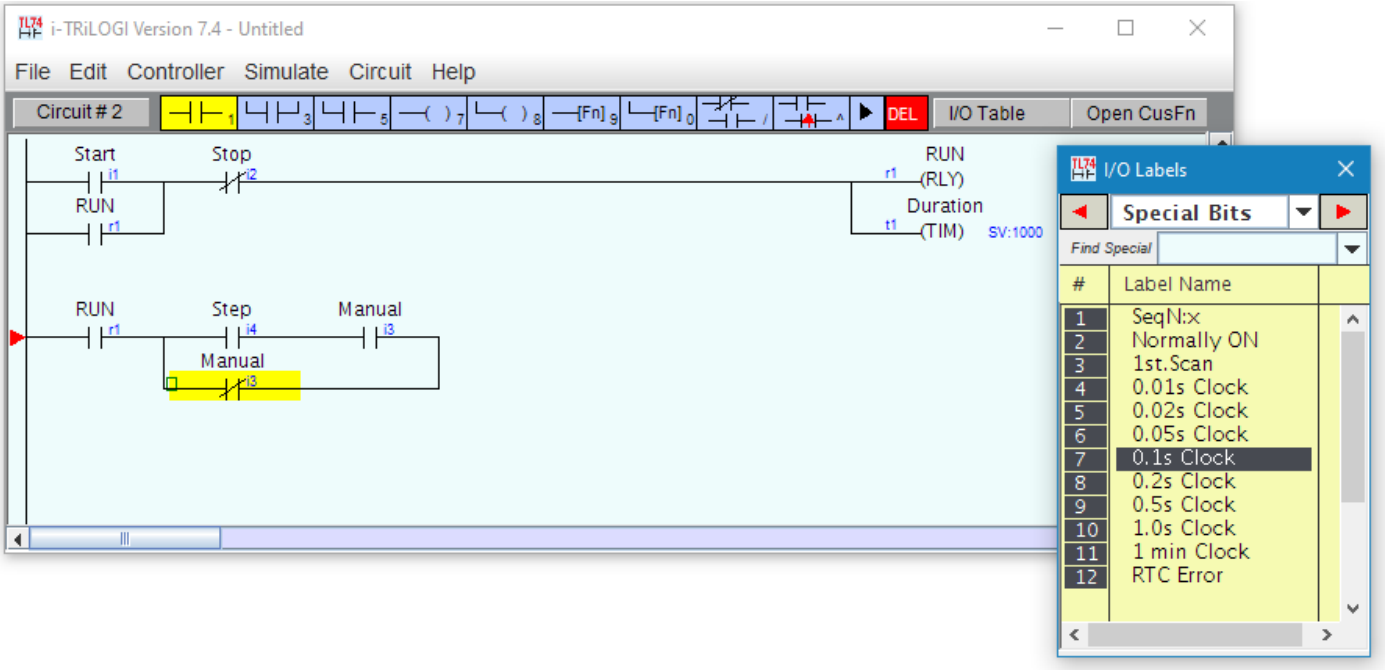
   that encloses both the “Step” and the “Manual” contacts. A cross will appear at the left hand end of the “Step” contact, indicating that this is the starting location of the parallel circuit. You should now click on the “Manual” contact to select the ending location for the parallel circuit. The yellow highlight bar will be positioned at “Manual” contact now.



4. 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 contact [“Manual”] in this case 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:



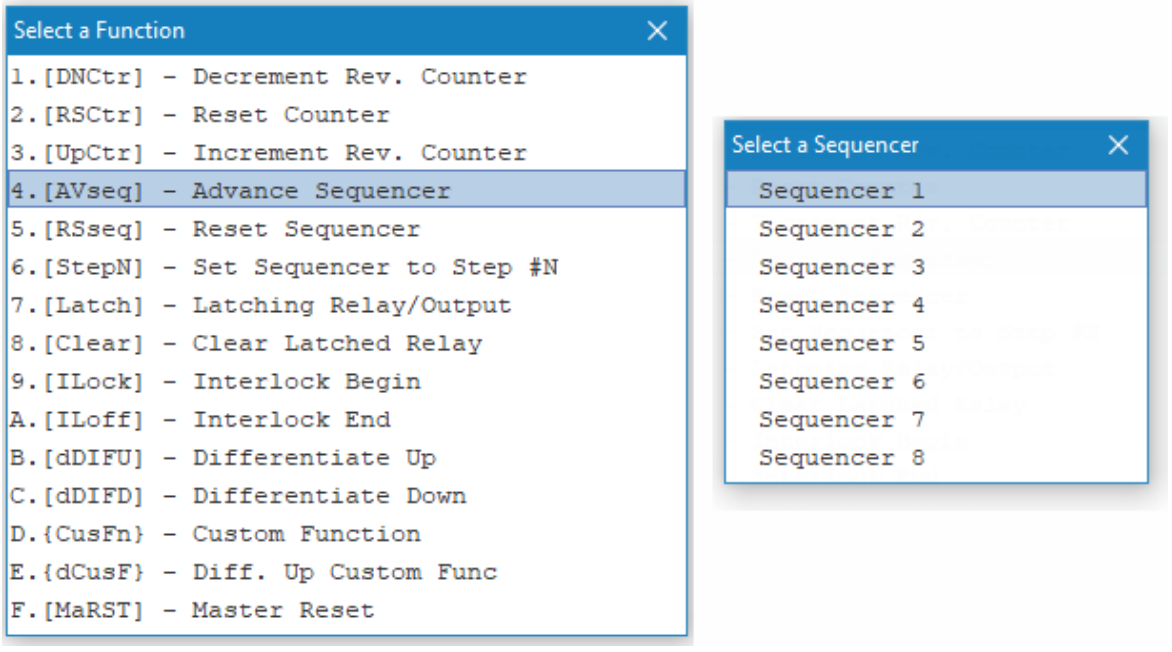
5. Next, we want to insert the special bit “Clk:0.5s” to the **LEFT** of the “Manual” contact. Press the <SHIFT> key to move the insertion point to the left end of the highlight bar as shown above. Then left-click on t icon to create a normally-open contact. Scroll the I/O table to the “Special Bits”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



> 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.

6. Next, move the highlight bar to the right end junction of the parallel circuits.

7. Now, click on t —[Fn]$_g$ 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].



8. When prompted, select “Sequencer 1”. This function 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

Step 5 - Sequencer & CusFn

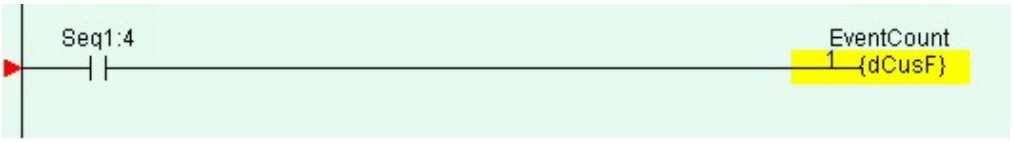
1. Circuits #3 to #6 are similar to one another. They make use of the Sequencer to turn on the Outputs 1 to 8 to create a pattern of “running lights” when executed. The label “Seq1:1” of the contact in Circuit #3 represents Step #1 of Sequencer 1. Remember that each sequencer can have up to 32 steps (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.

2. To create the normally-open contact “Seq1:1”, left-click on t  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. At this point, you should enter the number “1” since we are using Step #1.

3. 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 which correspond 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.

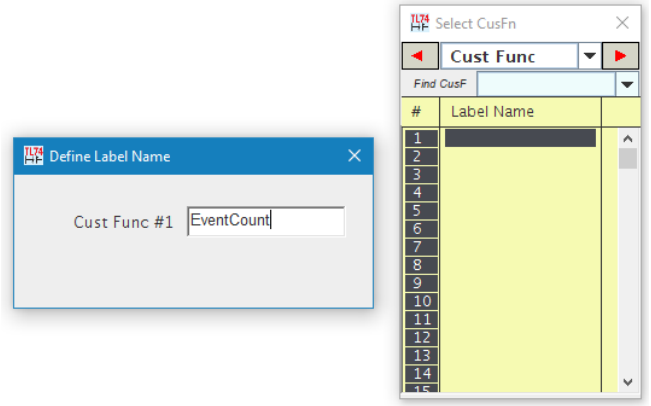
4. Circuits #4, 5 and 6 are very similar to Circuit #3 and you shouldn’t have problem creating them. Complete these circuits and we are ready for some interesting simulation exercises. When you have created all the circuits, press <Enter> key or <ESC> key at the last blank circuit to end “Ladder Edit” mode.

5. Next, we want Seq1:4 contact to execute a “Custom Function” (the abbreviation is CusF). A custom function is one that you can create to perform very sophiscated control actions using the TBASIC language. The Custom function can contains multiple TBASIC statements that gives the PLC extremely powerful capabilities unmatched by many ladder-only type of PLCs. The Circuit we wish to create is as follow:



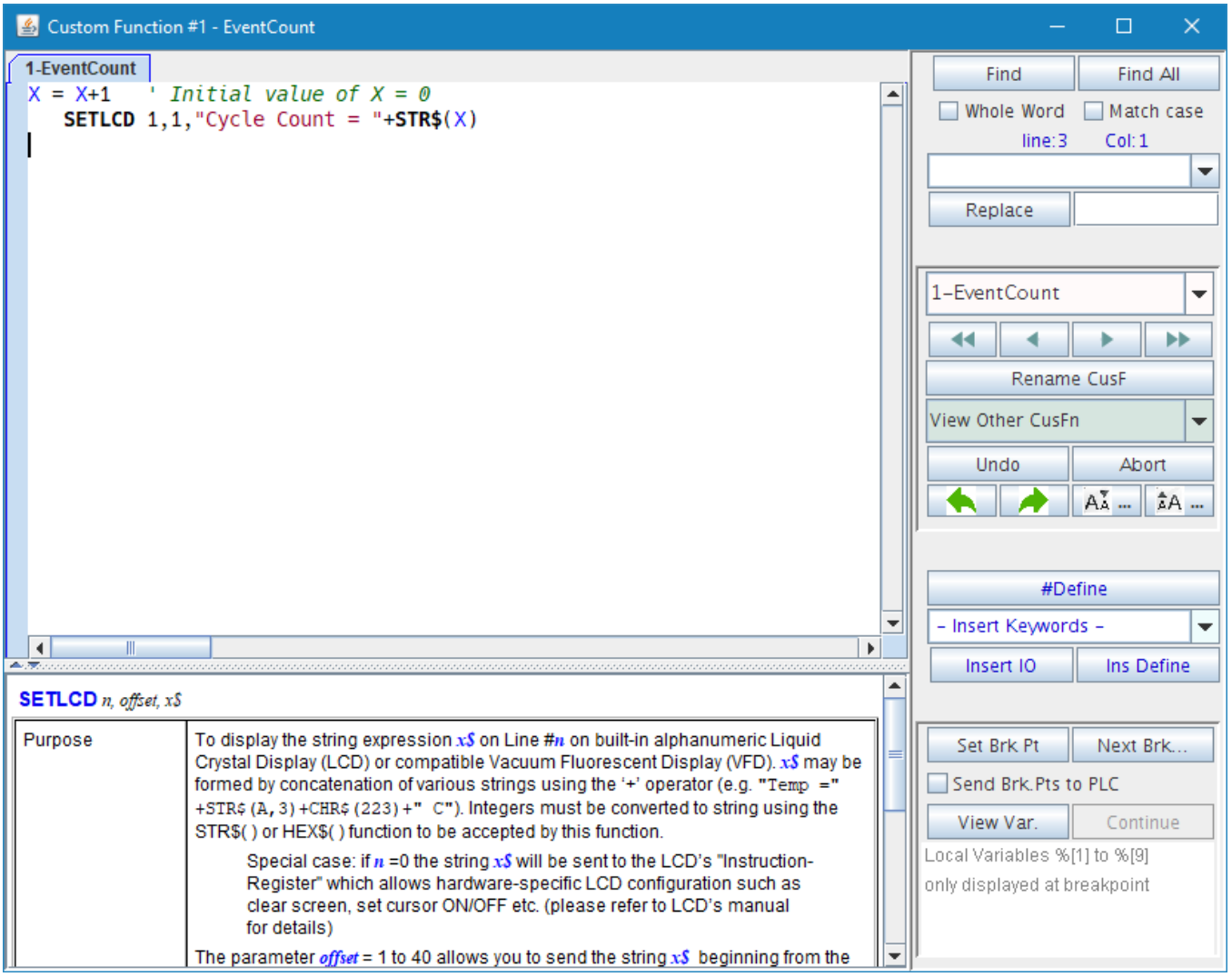
6. First, create the contact “Seq1:4” as per previous circuits. Then left-click on t [Fn]g to connect to a Special

Function coil. Select from the popup menu the item “E: Diff. Up Custom Function” to create a “Differentiated Up” version of Custom Function (For a full explanation of the meaning of Differentiated Up”, please refer to TBASIC Introduction (https://triplc.com/TRiLOGI/Help/tbasic/tbasicintro.htm) The Custom function popup menu will appear for you to select the Custom Function number you wish to insert. Up to 256 Custom functions can be defined and you can click on the first custom function. as follow:



Since the custom function has not been defined with a label name, you will be prompted to give it a label name. The label name given to the CusF must be unique and are subject to the same limitations as that of the I/O label names. Lets enter the label name “EventCount” for function #1.

7. When the Custom Function editor has been opened, enter the following text and then close the editor window by click on t  or by pressing the <ESC> key. If you would like to exit the editor window without saving the changes, click on t  button.



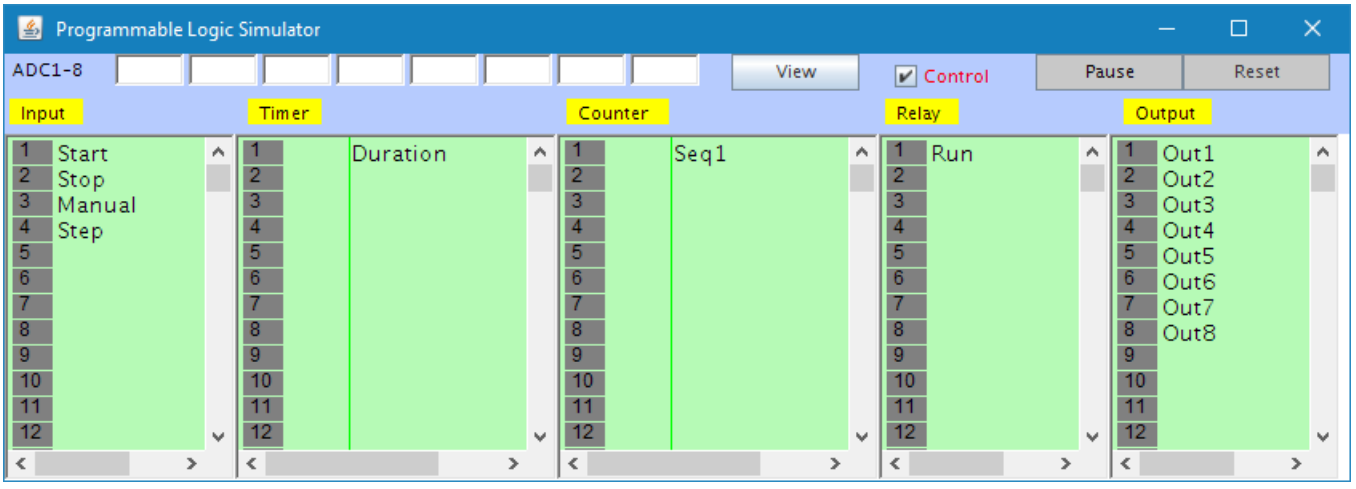
The purpose of this custom function is to keep a cycle count of the number of times the Sequencer has completed the complete sequencing and display the cycle count on the PLC's built-in LCD display (or on the Simulator's "View Variable" screen). This custom function should be run ONLY ONCE whenever the Seq1:4 contact closes, which is why the differentiated version of Custom function must be used in this case ({dCusF}, not {CusFn})

8. We can make our program more comprehensive to other users by utilizing the “Comments” feature of i-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. Starting from i-TRiLOGI version 6.7 and 7.4, text in comment circuit will automatically wrap to the next line when it exceeds the current screen width so that makes it easier to insert and modify comments from time to time.

A comment circuit may be moved around or deleted just like any other ladder circuits. If you wish to edit the comment circuit, 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.

STEP 6 - Simulate Now!

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”. i-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 (https://triplc.com/TRiLOGI/Help/laddertutorial.htm) page, make all 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 which contains up to 512 elements, and the Timer table which 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 which indicates the ON/OFF state of respective I/O. A red color lamp represents the ON state of an I/O, whereas a dark grey color lamp represents an OFF state. The I/O number is indicated in the middle of the lamp.

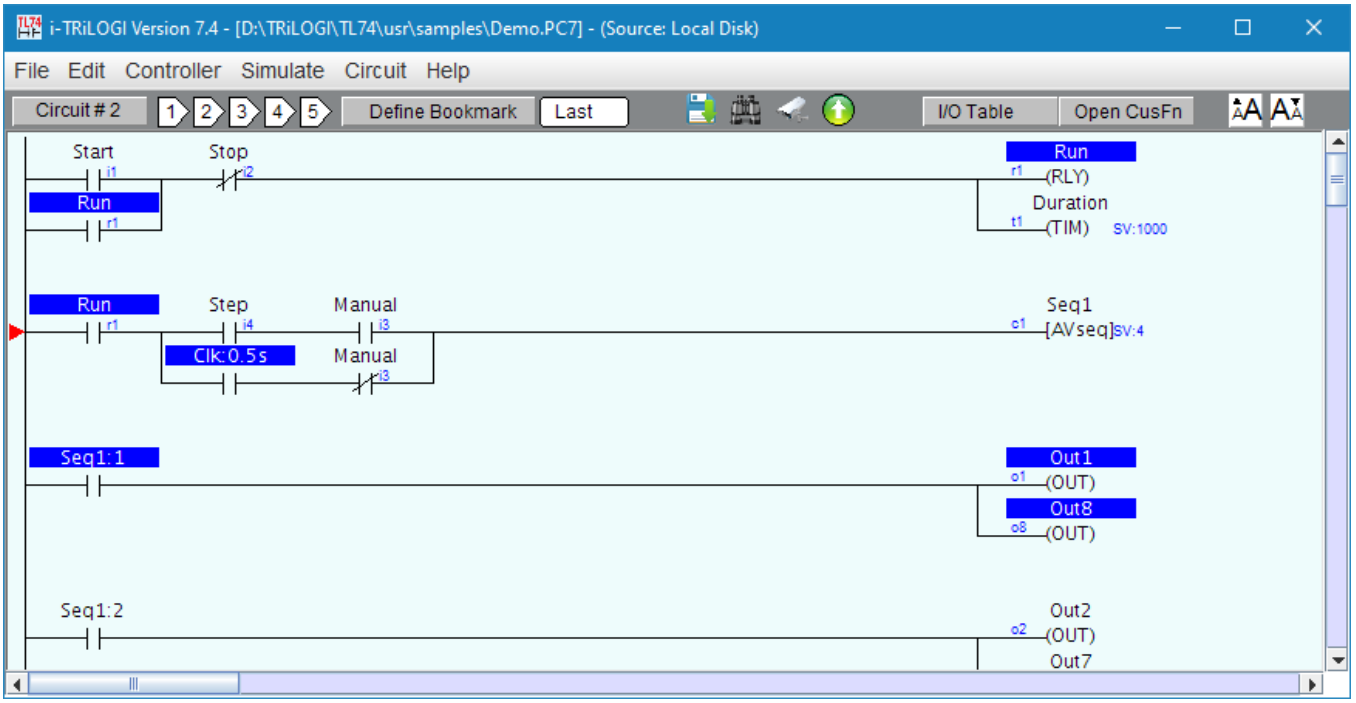
   The simulator require the use of the mouse to work properly so it is important to remember the mouse button actions as follow:

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)

4. 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!

5. 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…..

6. 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:



7. 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.

8. 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.

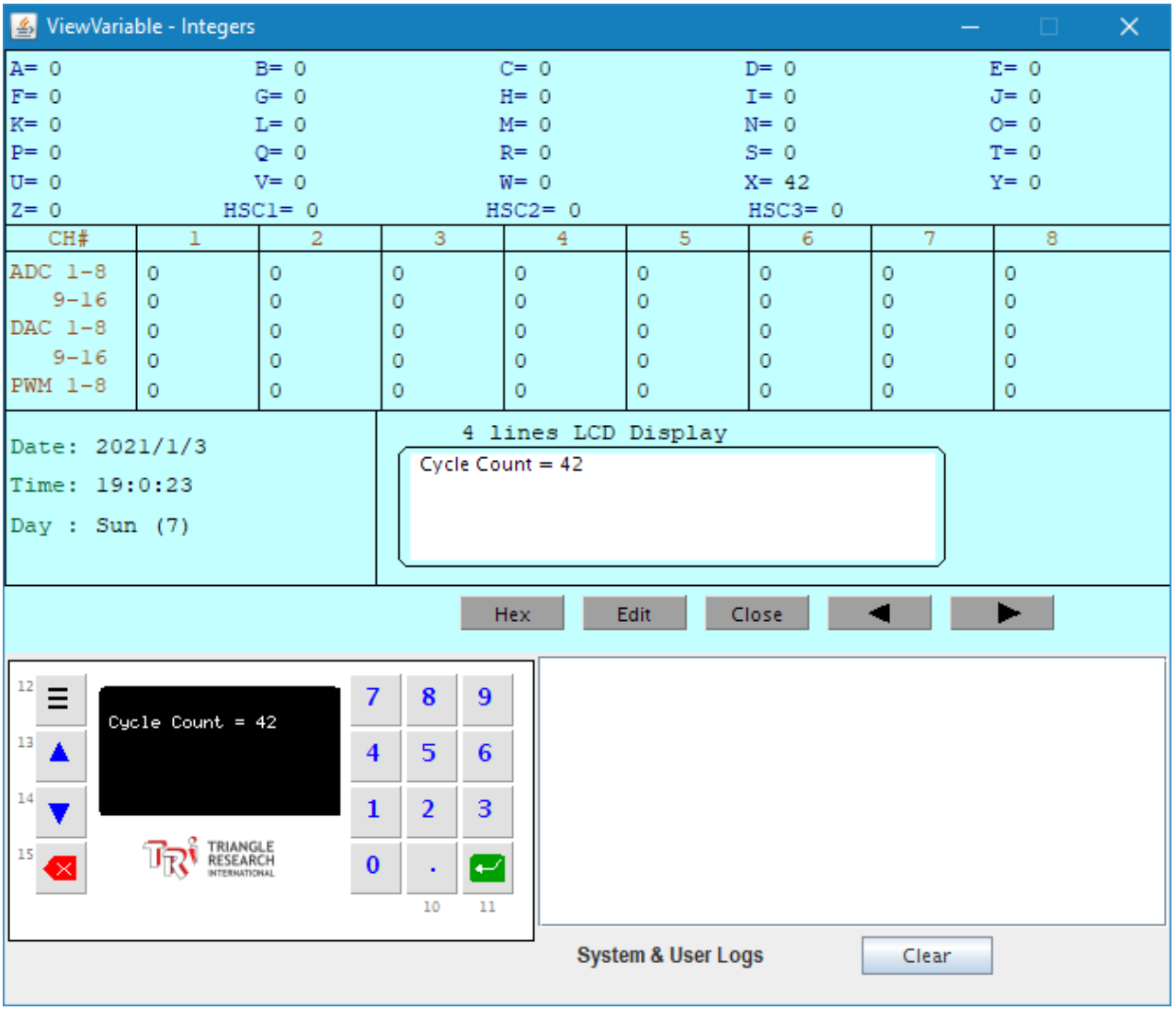
9. Let’s restart the system by turning ON the “Start” input momentarily again. Next, we want to turn ON the “Manual” input. Move the mouse pointer to the “Manual” input and then press the right mouse button. “Manual” input will be “stuck at “ON” state even after you have released the right mouse button. Click on “Manual” button using the right mouse button again and it will be turned to OFF.

10. 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.

11. 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.

12. 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!

13. You can also observe the execution of the Custom Function #1 by clicking on t[View] button on the “Programmable Logic Simulator” screen and the following window will appear:



14. This screen shows the values of TBASIC variables A to Z and many other data. It also provides a simulated 4 lines LCD display that show case the action of the SETLCD statement contained in Custom Function #1. As you can see, the value of variable X used in the Custom function is also shown in the variable section. The value of X is converted by the STR$(X) statement into a string and displayed after the text string “Cycle Count = ” at line #1, column #1 of the LCD area.

We have completed this hands-on session and have successfully created a simple ladder +BASIC program. We have also performed real time simulation to test the program's functionality. By now you would probably have a good appreciation of i-TRiLOGI's superb capability and ease of use and are ready to include i-TRiLOGI as an integral part of your programming needs.

Chapter 5 - i-TRiLOGI Main Menu

5.1 - File Menu

The File menu provides commands for the opening/saving of i-TRiLOGI files either on the local harddisk 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.pc6" ("Untitled.pc7" when running TL7.x).

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 hard drive with its current name.

3. Open (Local Drive) / Save As (Local Drive)

The default file extension for opening a i-TRiLOGI file 6.x is ".PC6" and ".PC7" for TL7.x. This signals to the TL6 application that the file is in Unicode format and TL6 will then load it accordingly. However, TL6 is also able to read ".PC5" files created by older TL5 applications. TL6.x assumes that all ".PC5" files are stored in ASCII format and will load it accordingly. To display ".PC5" files in the file dialog, simply enter the string "*.PC5" in the "File Name:" text field and press <Enter>. The window will now display only files with "*.PC5" extension. You can then navigate to the folder that contains the ".PC5" files to pick the file you want to open.

The default file extension for saving a i-TRiLOGI file in TL6 is ".PC6" and in TL7 is ".PC7". This signals to the TL6/TL7 application to save the data in Unicode format. Unicode is required to store international characters which cannot be properly saved in the ASCII format used by TL5. However, you may still save the currently opened i-TRiLOGI file in TL6 or TL7 to a file with an extension ".PC5", which will then instructs TL6/TL7 application to save the file in ASCII format. Files saved with ".PC5" extension may then be opened using older TL5 applications.

Import Notes:

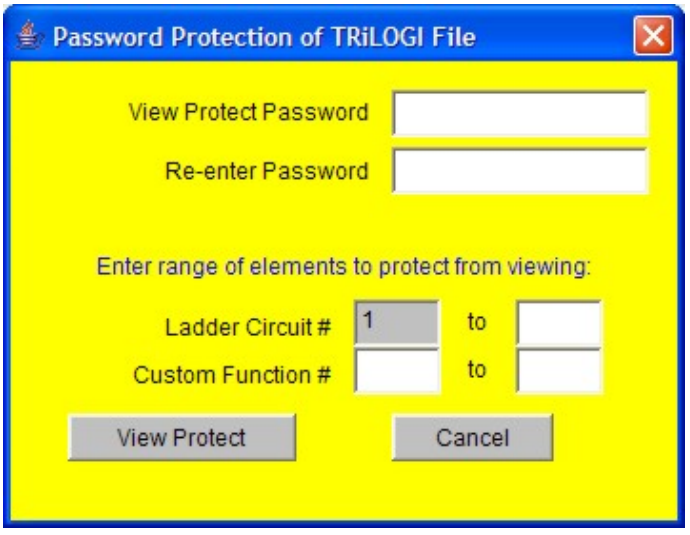
a. ".PC6" / ".PC7" files are saved in Unicode and are 100% compatible with each other. i.e. TL7.x can read .PC6 file and vice versa. However, while TL7.x should be able to compile any PC6 file the reverse is not true, since PC7 program may contain commands that are not available to TL6.x compiler.

b. ".PC5" files are saved in ASCII code format, and is not NOT interchangeable with PC6 / PC7 files. Therefore, you must not simply change a file name extension from ".PC5" to ".PC6" / ".PC7" via Windows File manager and then attempt to open it from TL6. Doing so will cause TL6 to complain that it is not a valid i-TRiLOGI file.

c. Any Unicode characters used in the program that cannot be represent by 8-bit ASCII code will be lost when the file is saved in ".PC5" format and it is not recoverable. So if you use non-English I/O labels your program will most likely fail to compile when it is saved as a ".PC5" file, since the I/O labels will be converted to ASCII and partially truncated. Thus, it is important to keep the original copy of your .PC6 program so that you can work on it to resolve the Unicode conversion issues.

4. View Protect / Un-Protect

This feature allows you to prevent others from viewing a pre-defined range of ladder logic and custom functions. When you select "View Protect" command, you will be asked to enter an unlocking password, a range of ladder logic starting from circuit #1, as well as a range of custom functions whose content you wish to prevent others from viewing, as follow:



The moment you click on the "View Protect" button, you can no longer view or print the protected range of ladder circuits and custom functions. When you save the view-protected program, it will be saved in an encrypted format and it cannot be opened using older version of i-TRiLOGI software. However, you will still be able to add new ladder circuits and custom functions to this program as well as modify those unprotected ladder circuits and custom functions. Of course, you will still be able to compile, simulate and transfer the protected program to the PLC as usual.

A view-protected file can be unprotected by selecting the “Un-protect” command from the “File” menu and supplying the unlocking password. Note that this unlocking password is strictly for un-locking the viewing restriction and it has nothing to do with other username and/or password required for interacting with the PLC.

This View Protection command is extremely useful for OEMs who wish to allow end users of their equipment to modify or append to the PLC’s program for ease of interfacing to other equipment, but without revealing the core content of the PLC program to the end user. Besides being able to protect the OEM’s intellectual property, it will also help to prevent the end users from mistakenly modifying the core program which can lead to disastrous result.

5. Write Compiled Code to Disk

This feature is available since i-TRiLOGI version 5.32. You can write the compiled program code to a disk file so that you can send the compiled code (with a “.CO5” extension) to your end customer to upload to the Super PLC using a standalone “CO5 Uploader” program. This allows you to protect your source program file content, while giving the end users of your equipment the ability to upgrade the PLC program. The end users DO NOT need to install the i-TRiLOGI or the Java JRE in order to use the CO5 Uploader program. so that makes it more flexible for you to distribute self-upgrade for the end users.

The CO5 Uploader program can be obtained by emailing to “sales@tri-plc.com” and installed by first unzipping the “Install CO5 Uploader.zip” file found in your i-TRiLOGI Version 6.x CD-ROM or and then run the “Setup.exe” program. Please note that the “CO5 Uploader” is a copyrighted program and Triangle Research International is the copyright owner of this program. However, Triangle Research International authorize the licensed users of the i-TRiLOGI version 5.xx software to freely distribute the “Install CO5 Uploader.zip” program to their end users at no charge.

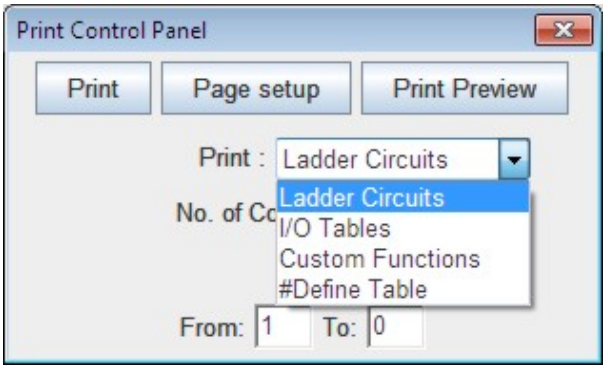
Note:

- Although i-TRiLOGI version 6.x supports Unicode for its user-interface, the compiled code produced by TL6 is identical to that of TL5 and therefore the compiled file is still saved in ASCII format. To maintain backward compatibility the “Write Compiled Code to Disk” function still produces “.CO5″ file that can be uploaded using the CO5 Uploader” program.

- Beginning from i-TRiLOGI version 6.41, you can also use the i-TRiLOGI software to transfer the .CO5 file to the PLC. This allows licensed i-TRiLOGI users to exchange compiled .CO5 file without revealing the source code.

6. Print

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.

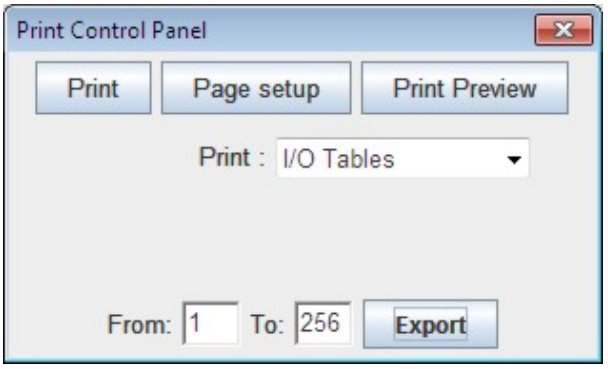
You can use the “Print preview” button to check the pagination of the printing on screen. You can select paper size and print orientation. etc. by clicking the “Page setup” button. Empty custom functions will be automatically skipped to save paper. When you select to print the “Ladder Circuits” a special “No. of Element” 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 “No. of Element” parameter, the “out-of-page” part of those ladder circuits will not be printed.

Wider I/O Table

Previously, the I/O Table label names could only be 10 characters, but i-Trilogi 6.2 and up allows label names of up to 16 characters (see the IOTable (https://docs.triplc.com/trilogi/#6004) section for more details). It is now possible to print an I/O Table with label names up to 16 characters and still fit all of the information within the page width. The only difference is that Relays #257 to 512 will be printed on a second page to provide more room.

Export the I/O Table

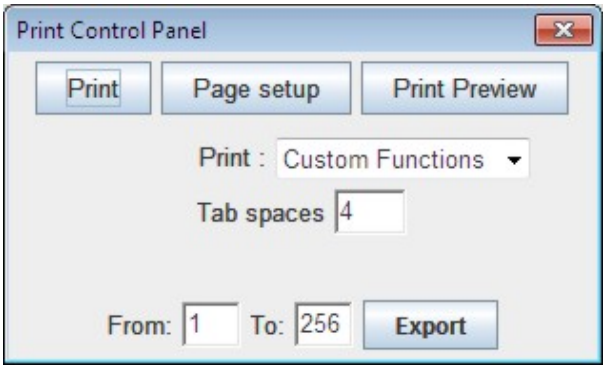
Now you can export the I/O Table to a “.csv” file that can be opened by any program that has “.csv” compatibility. If you use Microsoft Excel, you can open the “.csv” file and the I/O table contents will automatically be displayed in an Excel spreadsheet. This provides limitless possibilites for printing the contents of the I/O Table and for integrating them into other documents. To export the I/O Table, just choose the “I/O Tables” option from the Print Control Panel and click on the Export button, as shown in the Print Control Panel below. You will then be prompted to save the I/O Table as a “.csv” file.



NOTE: You can select the range of I/Os you want to export to a ".csv" file, just like you can select the range of I/Os you want to print.

Export Custom Functions

Now you can export custom functions to a ".txt" file that can be opened any text editing program. It is best to use a text editor that interprets carriage returns as new-line characters as well so that each custom function is displayed as it would be in the i-Trilogi Custom Function Editor instead of as one long line that is very difficult to read. A good editor to use is Wordpad or Microsoft Word. To export Custom Functions, just choose the "Custom Functions" option from the Print Control Panel and click on the Export button, as shown in the Print Control Panel below. You will then be prompted to save the Custom Functions as a ".txt" file.



NOTE: You can select the range of Custom Functions you want to export to a "txt" file, just like you can select the range of Functions you want to print.

Export #Define Table

The new #Define Table feature introduced since version 6.43 can be exported to a ".txt" file that can be opened any text editing program or MS Excel program. To export Custom Functions, just choose the "#Define " option from the Print Control Panel and click on the Export /> button.

7. Exit

Execute this command to exit orderly from the i-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.

5.2 - Edit Menu

5.2.1 Abort Edit Circuit

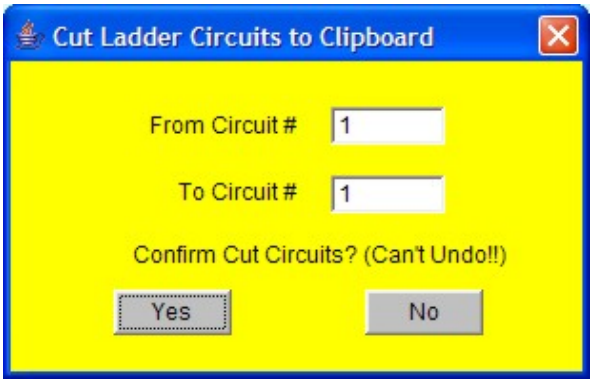
Changes made to the current ladder circuit can be aborted if you execute this command before pressing <Enter> to accept changes made to the current circuit. If changes have already been accepted by pressing the <Enter> key, then this command will have no effect. This command is useful if you wish to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

5.2.2 Undo <Ctrl+Z>

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

5.2.3 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 altogther. 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.

5.2.4 Copy Circuit (Ctrl+C)

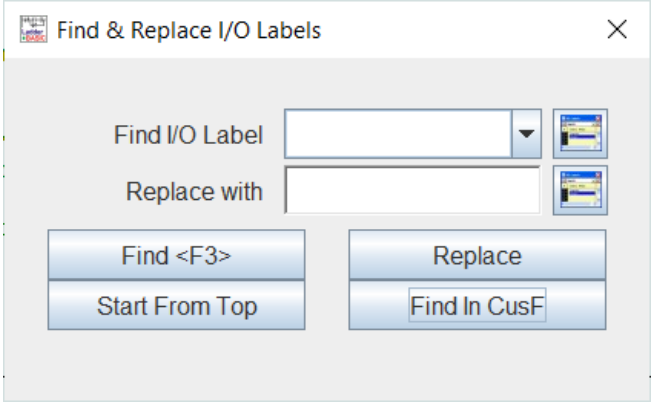
You can copy a block of circuits from the current ladder program and store them into the clipboard for pasting into another part of this ladder program or into another ladder program file altogether. The range dialog box similar to “Cut Circuit” will appear for you to enter the range of circuit to copy.

5.2.5 Paste Circuit <Ctrl+V>

When you execute this command, the block of ladder circuit which 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.

5.2.6 Find and Replace <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. 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.



For more details, refer to section 6.7 Ladder Logic Editor Search (https://docs.triplc.com/trilogi/#6023).

The Find command can also be used to search for a keyword in a TBASIC program. (https://docs.triplc.com/trilogi/#5809)

5.2.7 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.

5.2.8 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 (https://docs.triplc.com/trilogi/#6004)

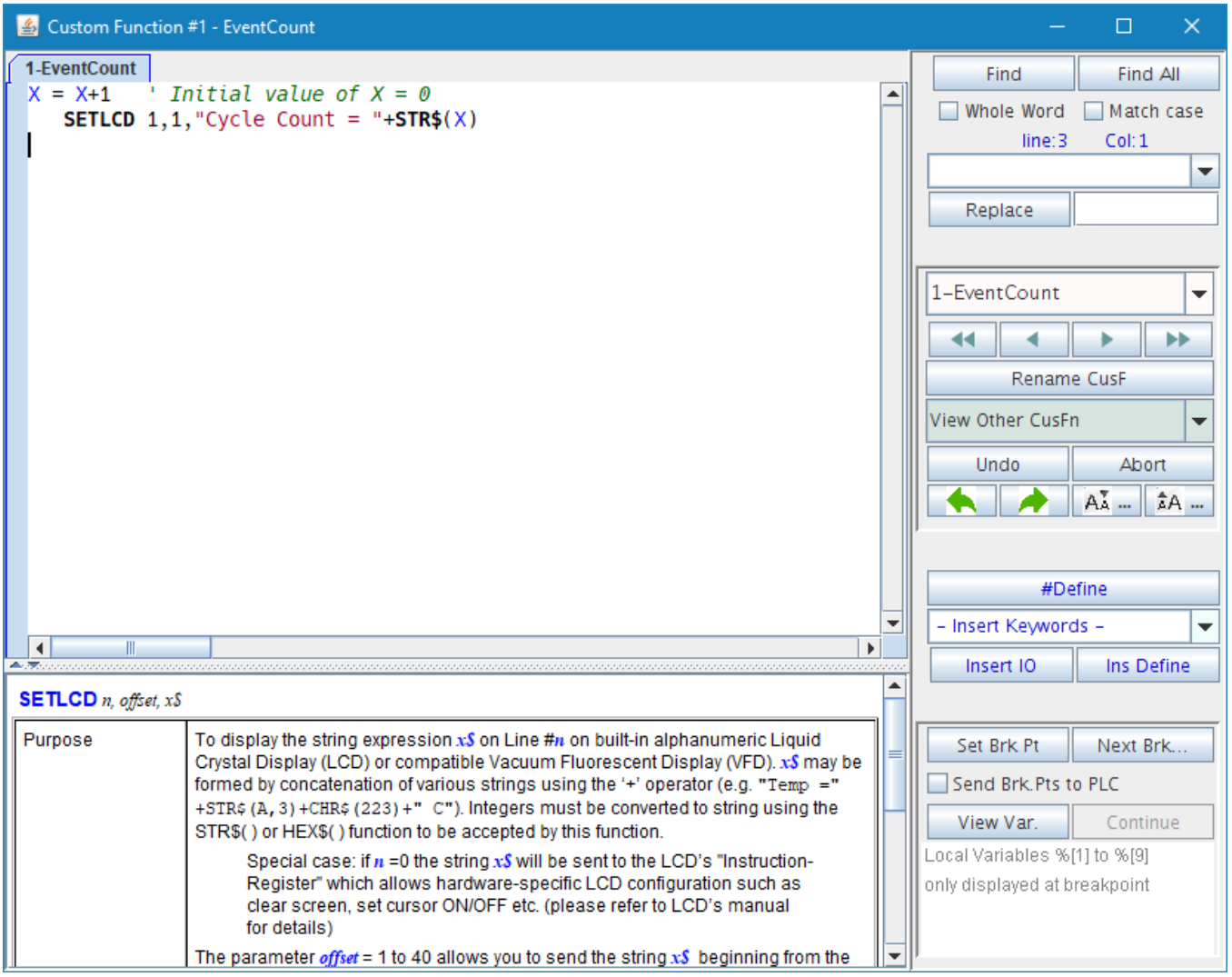
5.2.9 View I/O Type on Ladder <F3>

Toggle between display or no display of the I/O type and physical I/O number 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 and physical I/O number, which is now

linked to the I/O labelname. e.g. i1=input 1, o12=output 12, r25= relay 25, t1= timer 1 and c5=counter 5. When i-TRiLOGI first starts, the display is enabled but you have the option of turning it off if you find it distracting.

5.2.10 Edit Custom Function <F7>

Opens up the Custom Function Editor window for you to enter the TBASIC (https://docs.triplc.com/trilogi/#5430) 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 i-TRiLOGI file can contain a maximum of 256 custom functions. At any one time only one custom function will be opened for editing. The custom function number and the required label name will be displayed on the Title of the Custom Function editor window:



NOTE: This is a new and improved custom function editor. For a full description of its features, see sections 7.2 through 7.8 (https://docs.triplc.com/trilogi/#5789)

5.2.11 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.

5.2.12 Set TAB Width

You can change the number of spaces the editor used to display the <TAB> character that you insert to format the code hierarchically.

5.2.13 Zoom Out

You can zoom out multiple levels, which affects all windows in the same instance of TRiLOGI (including ladder logic, custom functions, and monitoring).

The shortcut key is F11 on the keyboard. There are also shortcut buttons on the Ladder Logic quick access toolbar available in browse mode. See the Ladder Editor Help page (https://docs.triplc.com/trilogi/#5260) for more information.

5.2.14 Zoom In

You can zoom in multiple levels, which affects all windows in the same instance of TRiLOGI (including ladder logic, custom functions, and monitoring).

The shortcut key is F12 on the keyboard. There are also shortcut buttons on the Ladder Logic quick access toolbar available in browse mode. See the Ladder Editor Help page (https://docs.triplc.com/trilogi/#5260) for more information.

5.3 - Controller Menu

All commands in this menu are for communication with the PLCs. The PC running i-TRiLOGI must be able to connect to the PLC via either Ethernet/WiFi or via the serial port.

If you are using the TLServer software, then note that TLServer can be running on the same computer that i-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 F-server or 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 server. If you have enabled the “use password” option in the F-Server or if you are connecting via TLServer, then you must be positively authenticated before you are able to log-in to the server.

1. Select Controller <Ctrl-I>

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 connect to the PLC F-Server or 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 PLC server if you have not yet done so. However, once you are connected, this command changes into “Disconnect from Server” and this is the way for you to log out of the currently connected PLC so that you can use the Username/Password dialog box to log-in to another PLC of a different IP Address/port number.

3. On-Line Monitoring <Ctrl+M>

See On-Line Monitoring (https://docs.triplc.com/trilogi/#4951) help page 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 log-in 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 i-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.

NOTE: When doing a program transfer, it is possible to abort a transfer even after the transfer has started. However, the PLC will be left in PAUSE state if a program transfer has been aborted to prevent execution of an incomplete program.

NOTE: The Program Transfer window is now wider to accommodate file names with longer strings and/or in other languages.

5. Transfer .CO5 File To PLC

This new command is added from i-TRiLOGI version 6.41 onward. It lets you transfer a file that has been saved previously to the hard disk using the “File -> Write Compiled Code To Disk” function under the “File (https://docs.triplc.com/trilogi/#5187)” Menu.

Using this function you can send compiled code to other users to be transferred into the PLC using a licensed copy of i-TRiLOGI v6.41 and above, but without revealing your source code. To ensure that the file is not corrupted when you email to others, try to compress it using a Zip or Rar software.

6. Open Matching Source File

You can use this command to query the connected PLC for the filename of the last i-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 or on the PC's hard disk. 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.

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.

Note to Unicode users: If you created your file name using Unicode instead of plain ASCII, the Unicode filename will not be saved into the F-Series / M-series PLC since the PLCs do not support Unicode in its internal memory. In such case you cannot use this command to open the matching source file. You would have use the "File -> Open (Local Drive)" command to manually open the file for monitoring purpose.

7. 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.

8. 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 which 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 i-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 i-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 PLC.

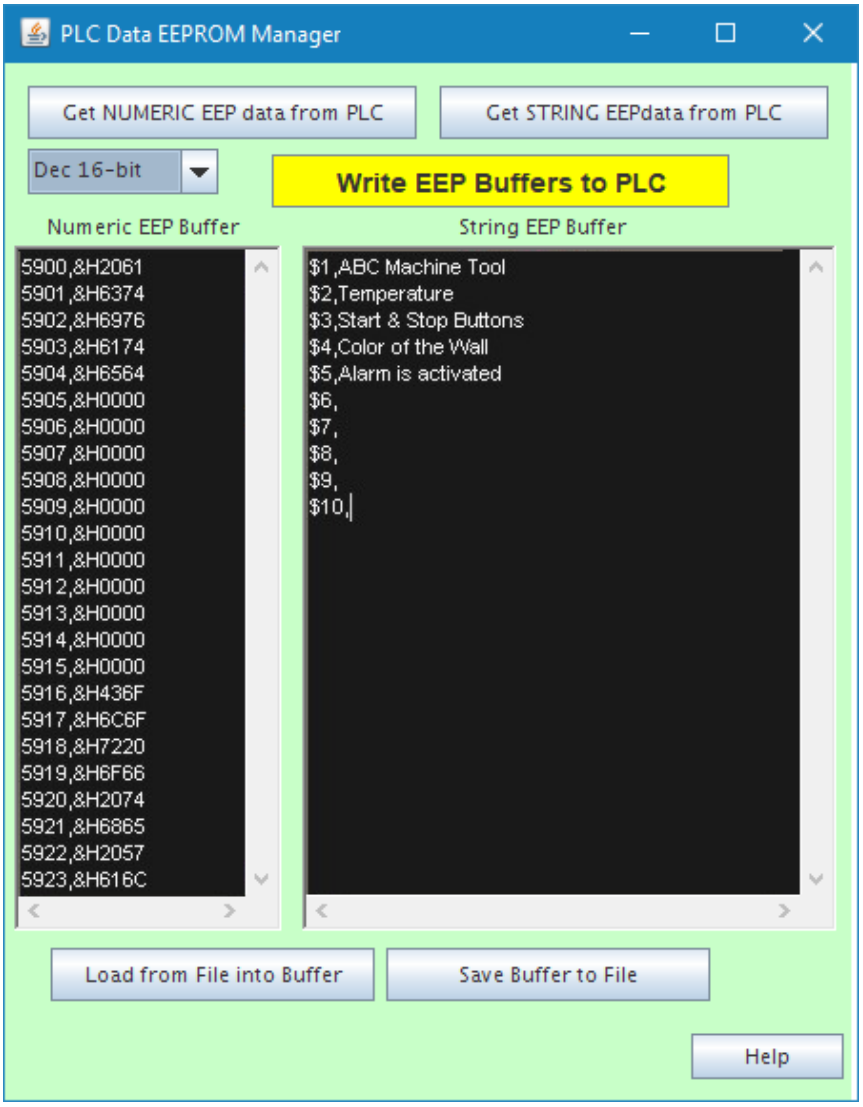
9. Reboot PLC (= Power ON Reset)

This new command is added to i-TRiLOGI version 6.47 build 01. It can force the PLC to reboot as if it has been powered cycle so that it can load new configuration or clear any system errors. This command however does not work on the older T100MD+/T100MX+ PLCs.

---

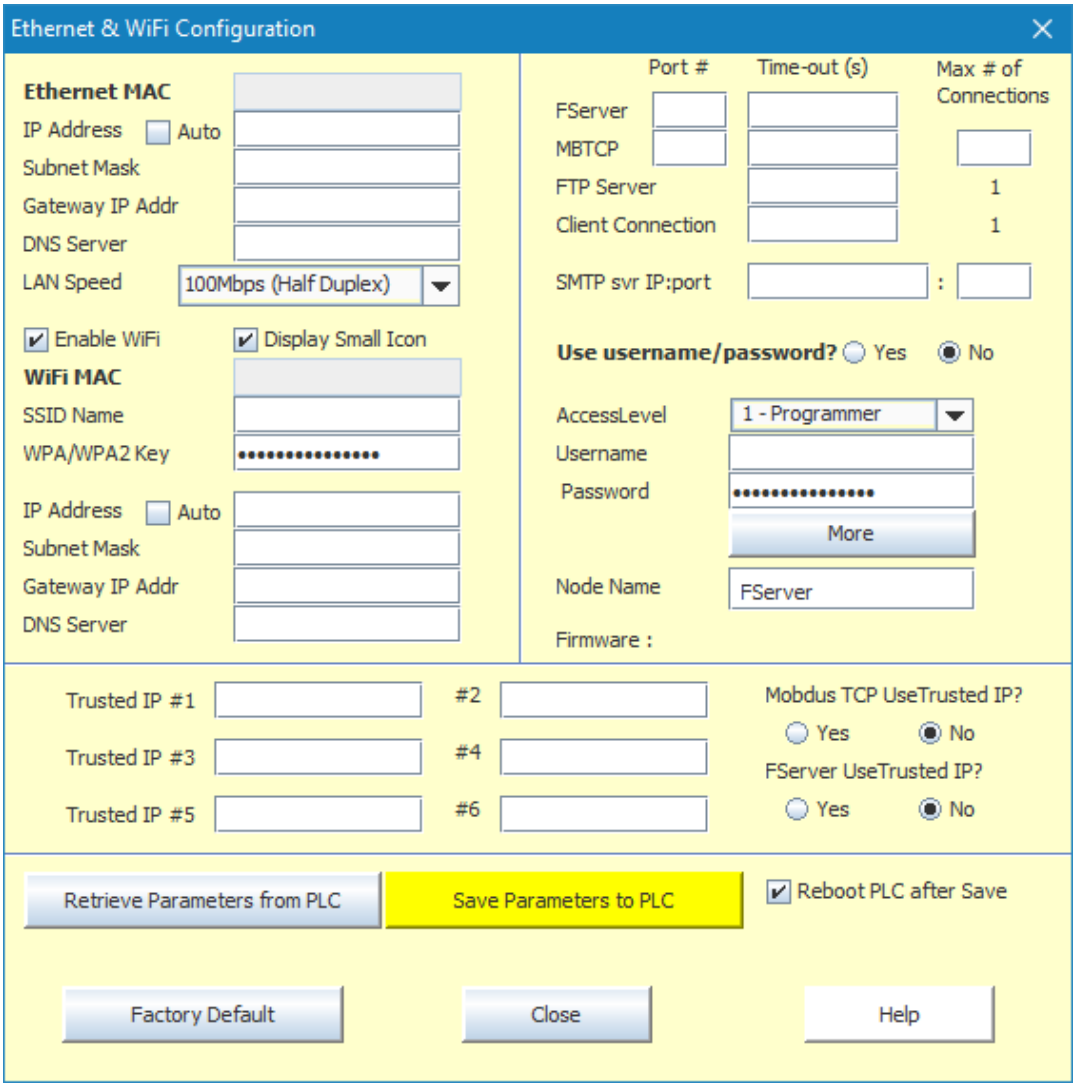
10. EEPROM/FRAM Manager

This new command is added starting from i-TRiLOGI version 6.1. Please click here for more details (https://triplc.com/TRiLOGI/Help/EEPMgr.htm)



11. Ethernet & ADC Configuration

Please click here for more details (https://triplc.com/TRiLOGI/Help/EthernetCFG.htm)



12. Auto Analog Calibration

Use this command to calibrate the first 16ADC and first 8 DAC. Please click here for more details (https://triplc.com/TRiLOGI/Help/autoanalogcalib.htm)

5.4 - Simulate Menu

i-TRiLOGI allows you to perform almost 100% simulation of your PLC's program off-line on your PC. This is a great tool for quickly testing a program 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.

i-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 i-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 (https://triplc.com/TRiLOGI/Help/simulator.htm)” window will open for you to conduct the simulation test of your i-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 (https://triplc.com/TRiLOGI/Help/simulator.htm)” screen so that at any time you can reset the simulator without affecting the logic states of the inputs.

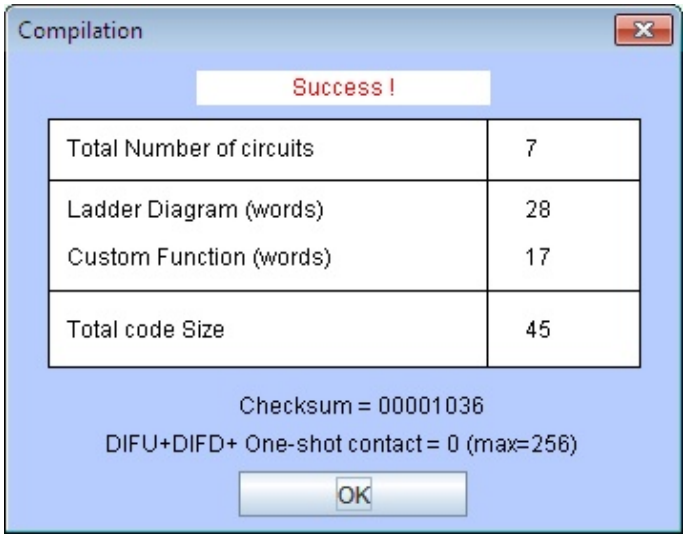
3. Continue Run (no reset) <F9>

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 i-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.

5.5 - 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 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.

You can save and close the comment the same way as before or by clicking the new OK button. It is now also possible to exit the comment and discard any changes made since it was last opened by clicking on the new Abort button.

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.

Note: Since i-TRiLOGI is Unicode-aware, you can enter comments text in any international language and as long as the file is saved with a “.PC6” the comments will be properly saved into the data file and will be properly reloaded when the file is next open.

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 (https://triplc.com/TRiLOGI/Help/editladder.htm) 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.

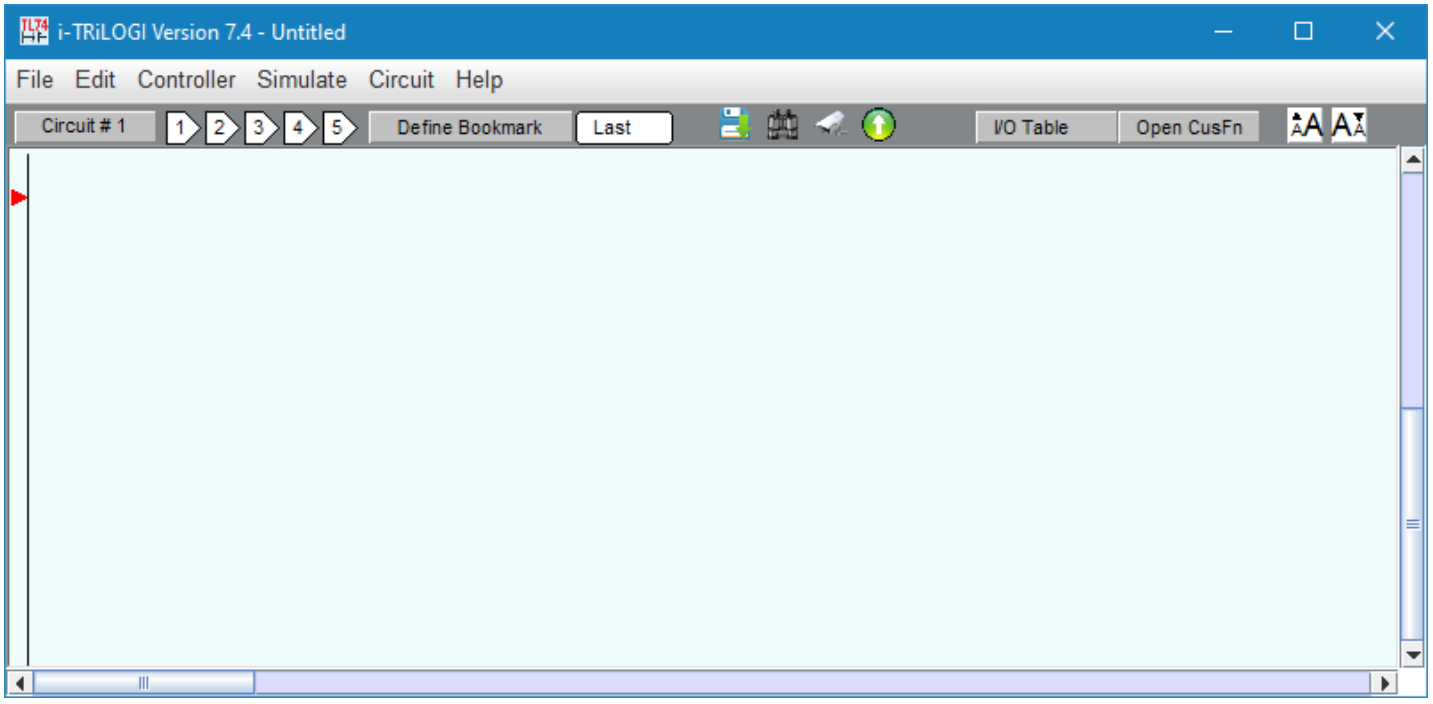
Chapter 6 - Ladder Logic Reference

6.1 Using Ladder Logic Editor

i-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 each mode below.

I. 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 “.



Mouse Actions

Since i-TRiLOGI Version 6 and up 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.

Keyboard Actions

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

1. **<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.

2. **<F1>** – Activates the context-sensitive help function to display on-line help.

3. **<F2>** – Opens the I/O Table to create the I/O Label Names

4. **<F3>** – Turns ON/OFF display of the I/O type and the physical I/O number for ladder logic contacts on the screen (The physical I/O number is now linked to the labelname). 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 and number. e.g. i1=input 1, o12=output 12, r25= relay 25, t1= timer 1 and c5=counter 5.

5. **<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.

6. **<F7>** – Opens any custom function. If the currently selected circuit contains a custom function, then it will be opened for editing. Otherwise i-TRiLOGI will ask you to select a custom function # from a menu.

7. **<F8>** – Compiles the i-TRiLOGI program to show the compilation statistics.

8. **<F9>** – Runs the simulator without resetting any I/O.

9. **<Ctrl-F9>** – Resets all I/Os and then runs the simulator.

10. **<Ctrl-F8>** – Resets all I/Os except inputs and then runs the simulator.

11. **<F11>** – Zoom Out (applies to ladder logic and custom function editors, monitoring and view variables screens)

12. **<F12>** – Zoom In (applies to ladder logic and custom function editors, monitoring and view variables screens)

13. **<Up>/<Dn><PgUp>** and **<PgDn>** keys – 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.

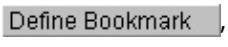
Navigating Circuits in Browse Mode

Although most of the time you will probably be navigating the ladder circuit diagram using the scroll bar or cursor keys to find the circuit that you are looking for, there exist several short cuts to enable you to quickly jump to specific circuits in the program. E.g. If you select "Goto" from the "Edit" menu, you will be prompted by a "Goto Circuit" window to enter the circuit number to jump to immediately. You can also call up the "Goto Circuit" window by pressing <Ctrl-G> key anytime.

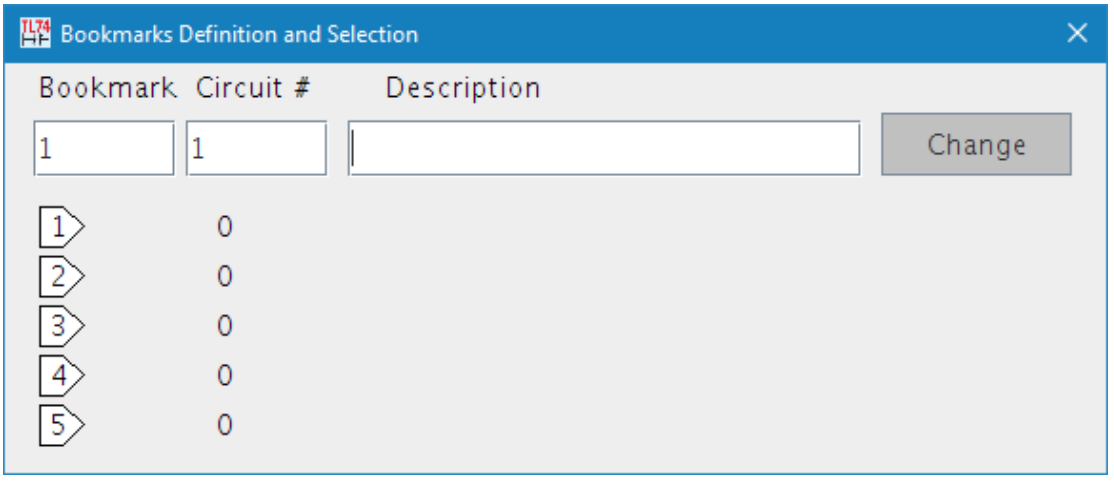
If you have a large ladder program with multiple rungs of circuits, then you will find the "Bookmark" feature very useful for quickly jumping from one circuit to another. The bookmarks appear as a row of 5 buttons along the top of the i-TRiLOGI editor windows: 

You can define up to five bookmarks by clicking on 

, each bookmark corresponds to a specific circuit #. By default all 5 bookmark points to circuit #1. Once you have defined the circuit # for a bookmark, then when you click on the bookmark button the editor instantly jumps to the pre-defined circuit number.

When you click on 

, the following window will appear



You can select the bookmark No. either from the choice box or by clicking one of the bookmark icons, e.g. 1 Then enter or edit the circuit # to associate with the selected bookmark. You can optionally enter/edit a description for the bookmark to help you recall the purpose of that section of ladder circuit which the bookmark is associated with. Then click the Change button to update the circuit # and description at the corresponding bookmark location.

In i-Trilogi version 6.2 and later the bookmark you have created will be displayed in the corresponding ladder circuits. New programs created will set all the bookmarks to a default value of 0 so that no bookmarks are displayed until they have been manually created.

In i-Trilogi version 6.2 and later there is a feature that allows you to immediately go the last circuit that was selected. This is implemented by clicking on the Last button that is located beside the bookmark buttons, which are described above. Whether you move to a new circuit by using the “Goto Circuit” command or a bookmark or just by clicking on a circuit with the mouse, the “Last” button will remember your last circuit position. This also allows you to easily switch back and forth between two circuits an unlimited number of times because if you don’t select a new circuit after pressing the “Last” key, you will go back to the circuit you came from.

Note: The bookmark definitions are saved along with the program body into the “.PC6” file, which means that the bookmark that you defined for a particular program can be recalled later when you open its “.PC6” file.

Quick Access Toolbar

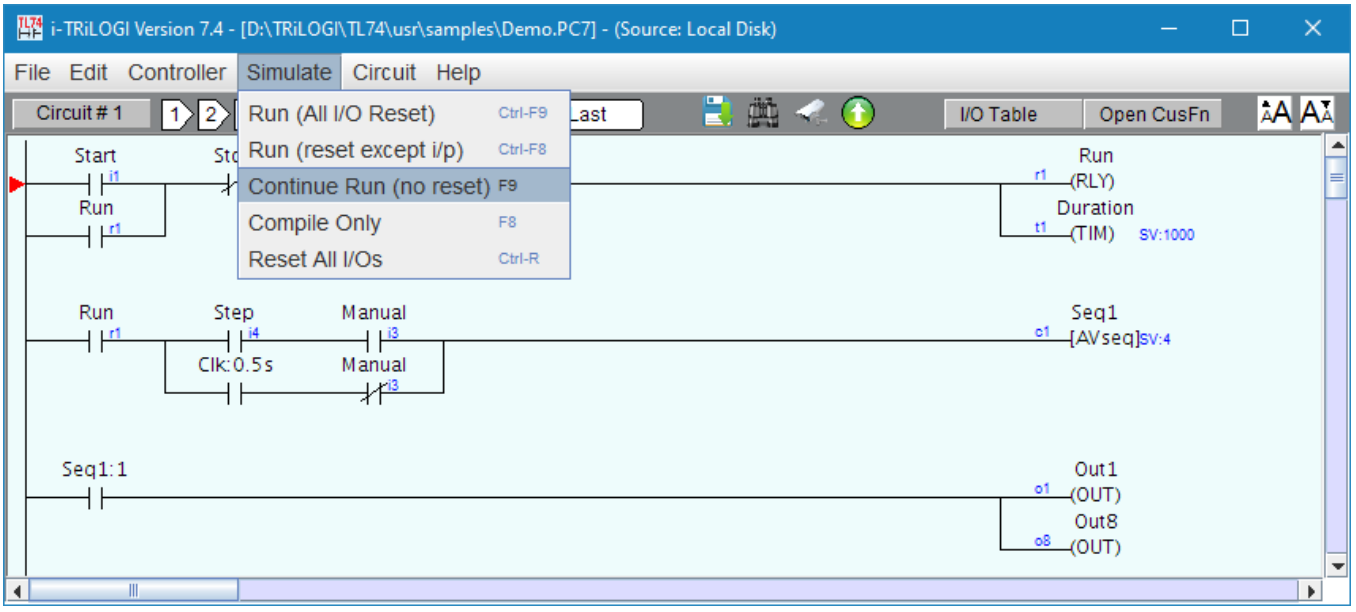
Trilogi 6.5/7.1 and up now includes a quick access toolbar that allows for the following single click operations:

1) File Save
2) Find Ladder Element/TBASIC text
3) Online monitoring
4) Program Transfer

5) I/O Table I/O Table Access
6) Open CusFn Custom Function access options
7) Shrink
8) Enlarge

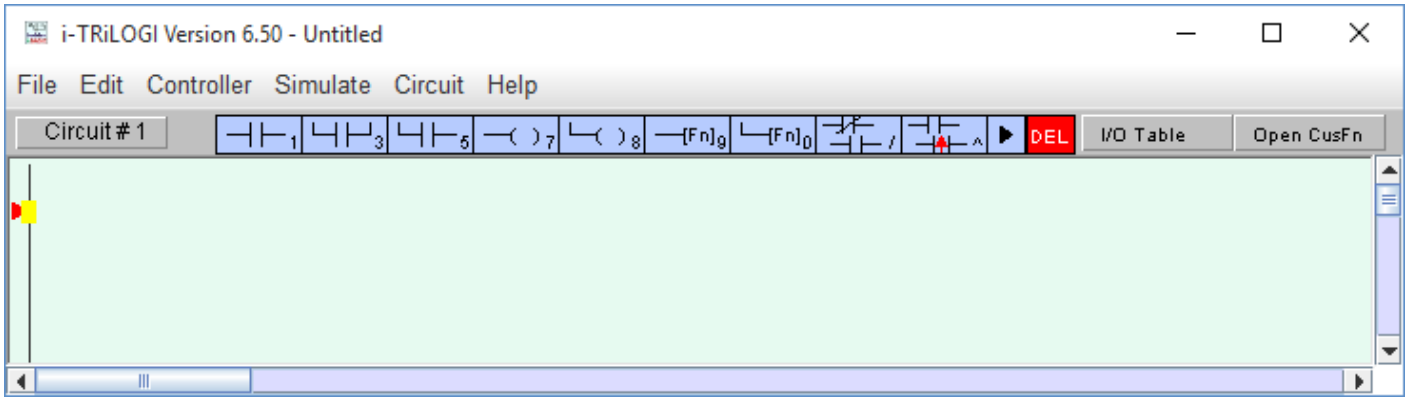
II. The Menu Bar

Clicking on any item on the main menu bar will open a drop down menu with multiple options. All menu options have been described in details in Chapter 5 (https://docs.triplc.com/trilogi/#5183).



III. The Circuit Editing Mode

i-TRiLOGI comes with a smart editor which 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:



Mouse Actions

One-Click Ladder Logic Navigation – Starting from version 6.50/7.10 it is possible to select an element with one click once the circuit is selected. This allows for easier navigation between elements within a circuit.

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

Right Click – When you click on a highlighted element using the right mouse button, you are provided with a list of options including:

1) Rename – Edit the label name of the element.
2) Delete – Remove the element
3) Invert – Toggle NO/NC contact (applicable to contacts only)
4) One-Shot – Toggle between standard and 1-shot contact types (applicable to contacts only)
5) Differential – Toggle Function Type. Switch between {dCusF} and {CusFn} (applicable to custom functions only).
6) Open CusF – 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.
7) End Edit Mode – Revert back to Browse Mode

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 highted 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.
	<^> – Convert the element to a rising-edge triggered contact (one shot)
	Click to move the highlight bar to the right (same effect as pressing the right arrow key). This can be used to move the cursor to a junction which cannot be selected by mouse click.
	Double-click to delete a highlighted element. This acts as a safety against mistake.

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 name 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 “Ladder Logic Programming Tutoria (https://docs.triplc.com/trilogi/#4966)l” 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.

Create Ladder Circuit Using The Keyboard

It is also possible to create 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 which 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.

<SHIFT>	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 <SHIFT> key. (Note: prior to Version 5.32 only the <TAB> key was used for this purpose, but <TAB> key does not work on JRE 1.4 and up, so we added the <SHIFT> key to achieve this action). The position of the insertion point 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.

6.2 Contacts and Coils

Ladder Logic Fundamentals: Contacts, Coils, Timers & Counters

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 circumtances 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

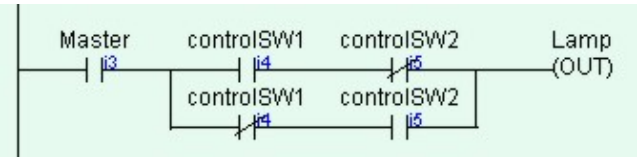
activate it. Limit switches are those small switches that are placed at certain location so that when a mechanical device moves towards it, the contact will be closed and when the device moves away from it, the contact will be open.

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 three-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 each have to have multiple set of contacts. But if you use a PLC, then these two switches only need to have a single set of contact 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 and hence this is a logical OR connection.

Rising Edge One-Shot Contact

NEW! A new type of contact, available for the F-Series PLCs only, is the Rising-Edge contact, which looks like this:  in the ladder logic toolbar and like this:  when placed in a ladder logic circuit. This type of contact will detect a change of status from off to on and then send a single pulse out (one shot). This contact can be used for any physical input or output or any internal bits (relay, counter, timer). In the case of physical input and output rising-edge contacts, a rising edge will be detected if the I/O has changed from off to on from one I/O scan to another (any I/O status changes that happen during a ladder logic scan wont matter for physical I/O). In the case of the internal bit rising-edge contacts, a rising edge will be detected if the internal bit has changed from off to on from one ladder logic scan to another (any internal bit status changes that happen during an I/O scan wont matter for internal bits).

Here are some examples of this for the physical I/O:

Ex1. In the circuit shown below, if Input1 is off for an I/O scan and then on for the next I/O scan, a single pulse (one shot) will be sent to Output1 and Output1 will be turned on for one program scan time (turned off on the next I/O scan). On the same I/O scan that Output1 is turned off and all following I/O scans, whether Input1 is still on or has been turned off, a rising edge will not have been detected and Output1 will remain off. For a new rising edge to be detected, Input1 must be scanned as off first and then scanned as on in a following I/O scan.

Ex2. In the circuit below, when Input1 goes from the off status to on status (as described previously), a single pulse will be sent to a [Latch] function that will latch Output1 to the on status. The actual Latch function will only be activated for a single scan time (just as Output1 was in the previous example), but it will permanently latch Output1 to on until Output1 is unlatched using the [Clear] function. This way Output1 will remain on even though Input1 has only sent a single pulse and Output1 will not be affected by any further rising-edge detections from Input1 (can only be latched once until it is unlatched).

The same principles can be applied to internal bits and coils as were previously described for physical input contacts and output coils.

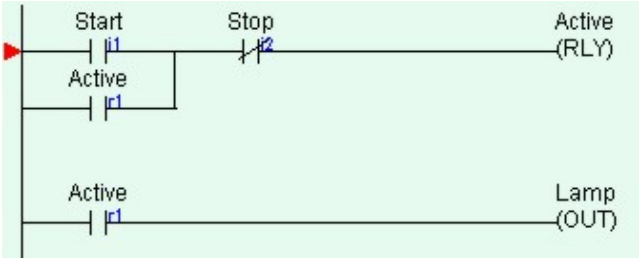
Once you understand these fundamental principles of interpretting a ladder diagram, everything should become clearer and simpler. Ladder diagram programming can be used to create a rather sophisticated control system. However, In i-TRiLOGI we augment its power further by allowing a ladder program to activate customized functions created in TBASIC (https://docs.triplc.com/trilogi/#5434).

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 which 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 follows:



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 or Normally-Closed and there is no limit to the number of contacts a relay can have.

Output 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.

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.

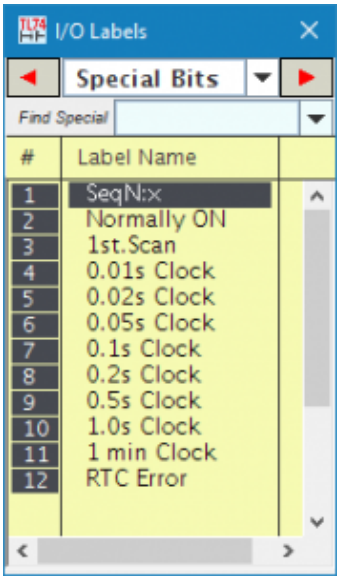
Counter Coils

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 decrement 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 remain 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].

6.3 Special Bits

i-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 (https://docs.triplc.com/trilogi/#5260) 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:

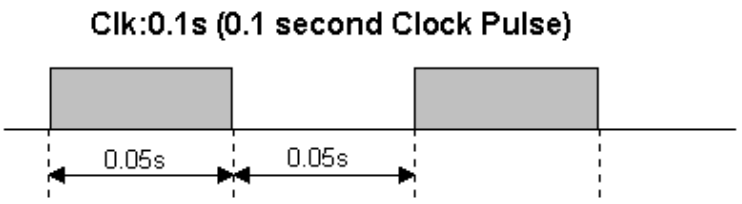


1. Clock pulse bits

The 8 clock pulses supported by i-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:



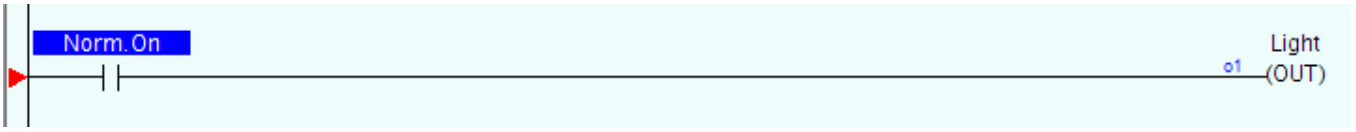
The clock pulse bits are often used with counter instructions to create timers. Additionally, they can be used as timing source for “Flasher” circuit. A reversible counter can also work with a clock pulse bit to create secondary clock pulses of periods that are multiples of the basic clock pulse rate.

2. SeqN:X

These are special “Sequencer (https://docs.triplc.com/trilogi/#5382)” 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. Click this link for detailed explanation and working examples on how to use a Sequencer (https://docs.triplc.com/trilogi/#5382).

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

Since the Wx100 PLC does not have a built-in battery-backed real-time clock, this bit will always be turned ON when a Wx100 PLC is first powered ON and the RTC will be reset to factory default date and time of 2016/1/1 00:00:00. This bit can thus be used to trigger a custom function to set the PLC’s RTC from an NTP time server

on the Internet. The RTC.Err bit will be cleared automatically when the RTC is synchronized with the Internet time server or is set by a host computer.

Note that the RTC.Err will not be set again when a Wx100 PLC undergoes a soft reboot as long as the RTC.Err has been previously cleared by a real-time clock update (either by a NTP server or by a host computer via hostlink or Mobdus commands).

FMD and Nano-10 PLC

This bit is turned ON if a Nano-10 or FMD PLC does not have one of the following installed: FRAM-RTC-0, FRAM-RTC-256 or I2C-FRAM-RTC 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.

Fx-Series PLC and SmartTiLE-Fx based Custom PLCs

Since the SmartTILE-Fx brain board used on all Fx-Series PLCs has a built-in battery-backed RTC, the RTC.Err bit will usually be cleared if the RTC has been previously set from a host computer or set by the program.

The RTC.Err will still be set if the RTC data is corrupted, or the battery is not installed, or the battery-backed RTC is damaged. The RTC.Err contact can be used to activate an alarm of some kind if real-time clock data is important.

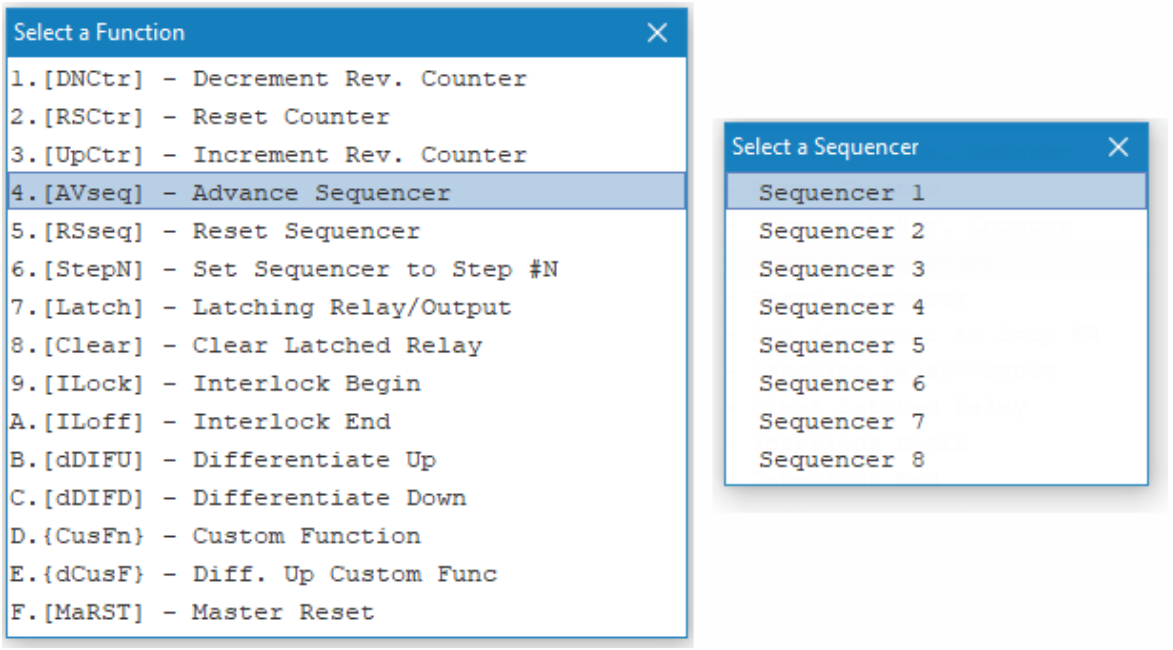
> **Note:** Unlike the Wx100 PLC, a FMD, Nano-10 or Fx PLC do not inherently support the <SNTP> command to automatically synchronize with an NTP time server. However, the PLC can still use the <TCPCONNECT> TBASIC command to make a TCP connection to a time server and extract the return data to set its own time. See “TestEthernet.PC6” and “NIST Timeserver by Gary Dickinson.docx” in “C:\TRiLOGI\TL6x\usr\samples\Ethernet”.

M-Series PLC (Legacy)

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.

6.4 Special Functions

During ladder circuit editing (https://docs.triplc.com/trilogi/#5260), when you click on the  or  icon to create a special function coil, a special function menu will pop up as shown below:



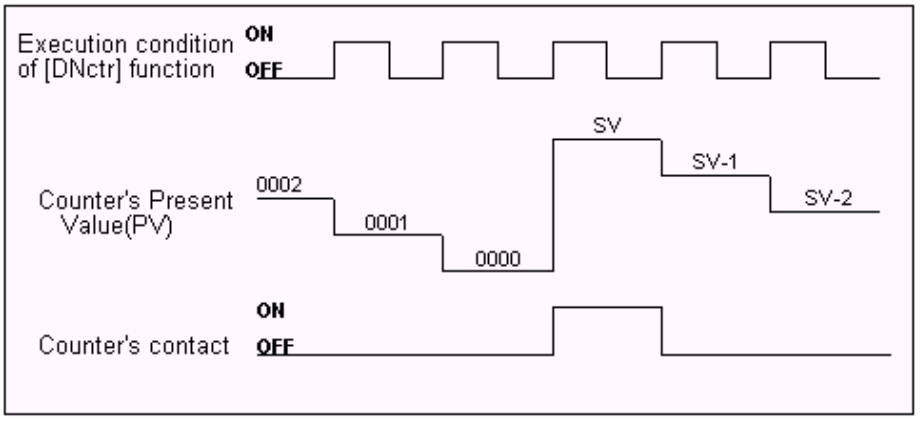
1. Reversible Counter Functions: [DNctr], [Upctr] and [RSctr]

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

The ordinary down-counter (created by clicking on the —( )$_7$ 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 which 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]



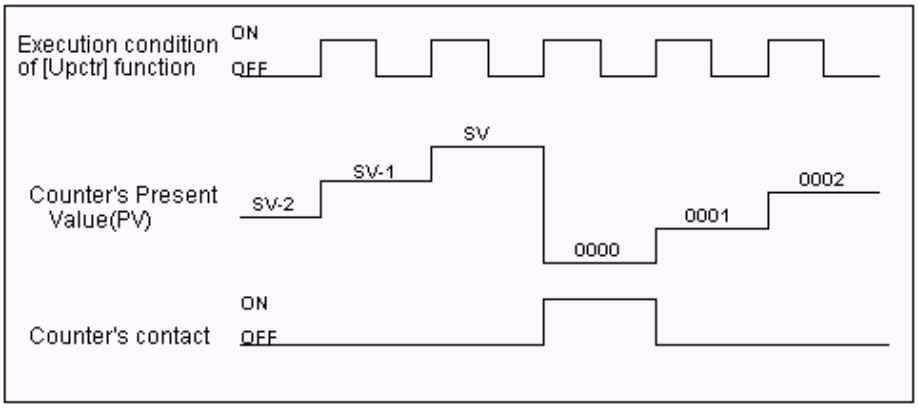
Each time when the execution condition of a [DNctr] function changes from OFF to ON, the present value of the designated counter is changed as follow:

a. 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.

b. 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).

c. 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:

a. 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 documentation on Using i-TRiLOGI Sequencers (https://docs.triplc.com/trilogi/#5382)

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” [ILoff] functions work together to control an entire section of ladder circuits. If the execution condition of an [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 [ILoff] will behave as follow:

a. all output coils are turned OFF.

b. all timers are reset to inactive.

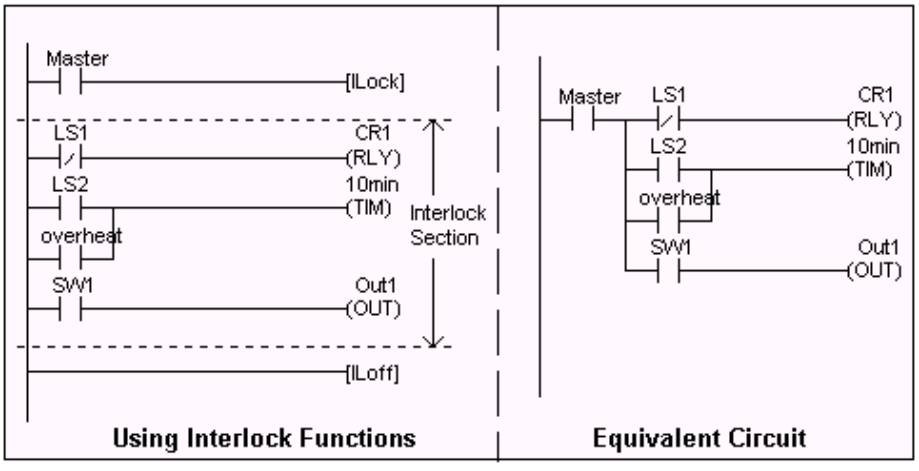
c. all counters retain their present values.

d. Latched relays by [Latch] function are not affected.

e. [dDIFU] and [dDIFD] functions are not executed.

f. all other functions are not executed.

> i-TRiLOGI does not let you create a branch circuit from the middle of a ladder circuit. However, you can achieve the same result using the Interlock functions. An Interlock section is equivalent to a master control relay controlling a number of sub-branches as follow:



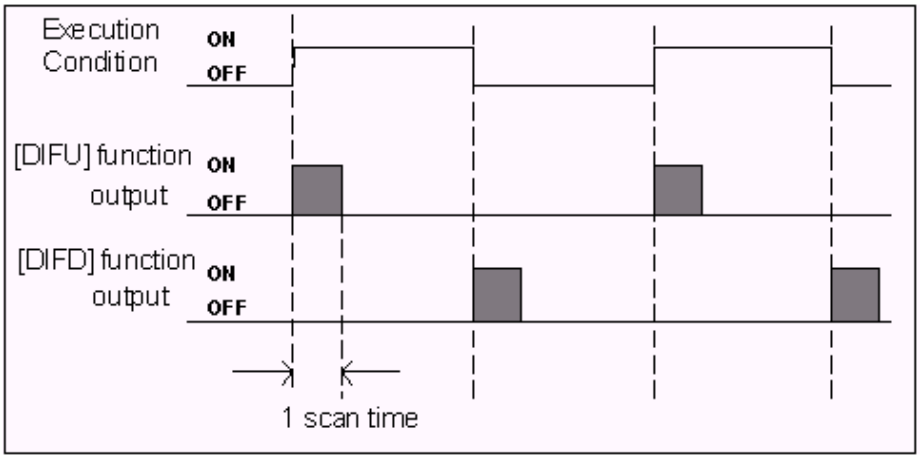
Note that [ILoff] is the only function that does not need to be energized by other program elements. When you use one or more [ILock] functions, there must be at least one [ILoff] function before the end of the program. Otherwise the compiler will warn you for the missing [ILoff]. The logic simulator always clears the Interlock at the end of the scan if you omit the [ILoff] function.

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

6. Differentiate Up and Down [d DIFU] and [d DIFD]

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.



NOTE: The [dDIFU] function can now be implemented as a single contact with the Rising-Edge contact that is triggered once every time a rising edge is sensed. More information on this new contact can be found in the Contacts section of the “Contacts and Coils (https://docs.triplc.com/trilogi/#5262)” help page.

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 editor provided by i-TRiLOGI Version6.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.

6.5 Using Sequencers

A sequencer is a highly convenient feature for programming machines or processes which 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

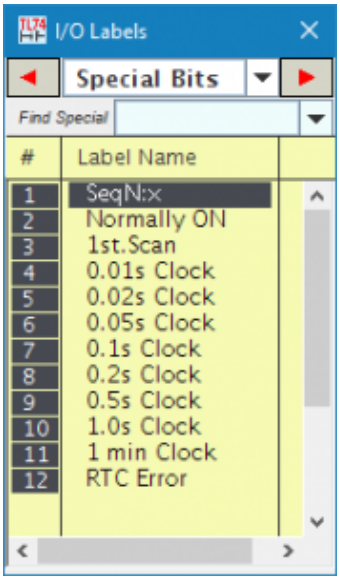
i-TRiLOGI Version 5 supports eight sequencers of 32 steps each. Each sequencer uses one of the first eight counters (Counter #1 to Counter #8) as its step counter. Any one or all of the first eight counters can be used as sequencers “Seq1” to “Seq8”.

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 i-TRiLOGI. Simply create a “contact” (NC or NO) in ladder edit mode (https://docs.triplc.com/trilogi/#5260). 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.

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] (https://triplc.com/TRiLOGI/Help/specialfn.htm#1) 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.

Other Sequencer Applications

a. Controlling a State Machine

You can use a sequencer to construct a state machine to control the execution of a process that is determined by which "state" the program is currently operating at. The sequencer counter is used to keep track of the state and the SeqX:YY contact is used in the ladder program to trigger different circuit or to call

up different custom functions.

b. Driving Stepper Motor

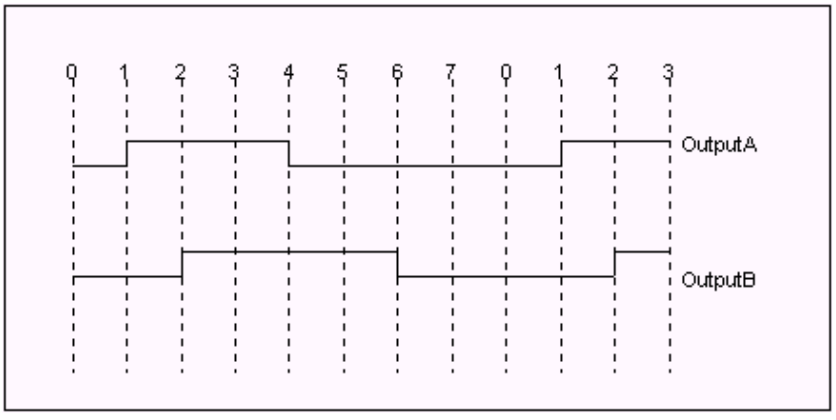
Some TRi Super PLCs have built-in high speed stepper motor drivers which should be used whenever possible if you want to drive a stepper motor. But it is also possible to use a sequencer to drive a low speed stepper motor directly using low speed expansion outputs on the PLC. E.g. This can be used to illustrate how stepper motor works in an educational demo setup.

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.

c. Replacing a Drum Controller

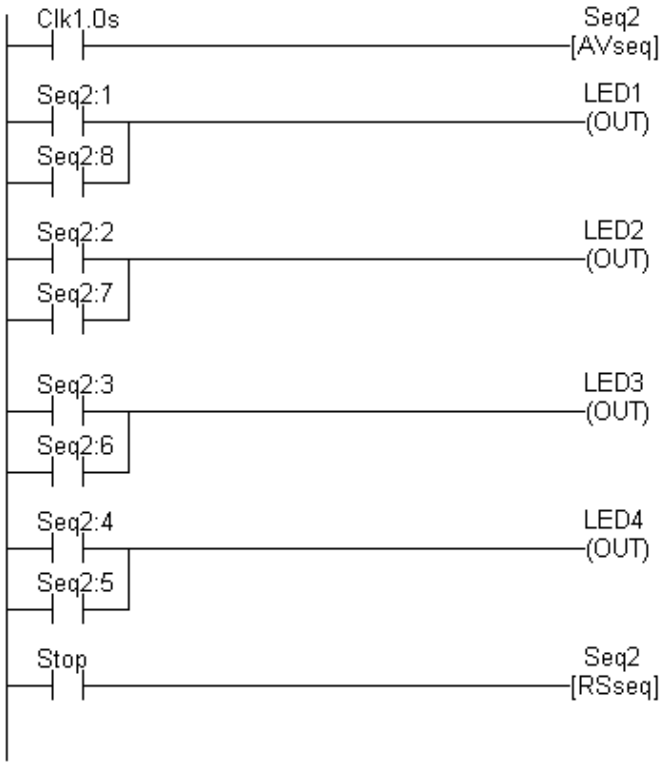
A drum controller can be replaced easily by a sequencer if the timing of the drum’s outputs can be divided into discrete steps. Assuming a drum controls two outputs with the timing diagram shown in the following figure:



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.

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.



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.

6.6 I/O Table

Unlike many ladder logic editors which are “numeric-centric” – meaning all ladder components are referred to by their I/O numbers, i-TRiLOGI programs are “labelname-centric” because i-TRiLOGI constructs the ladder logic strictly based on the use of label names and the compiler generates the correct I/O representation based on the labelnames defined in the I/O table. The advantages are that ladder programs constructed from label names are far easier to understand and remember, and you can rearrange the I/O location for a certain label without changing the program at all (e.g. move a load to another output driver).

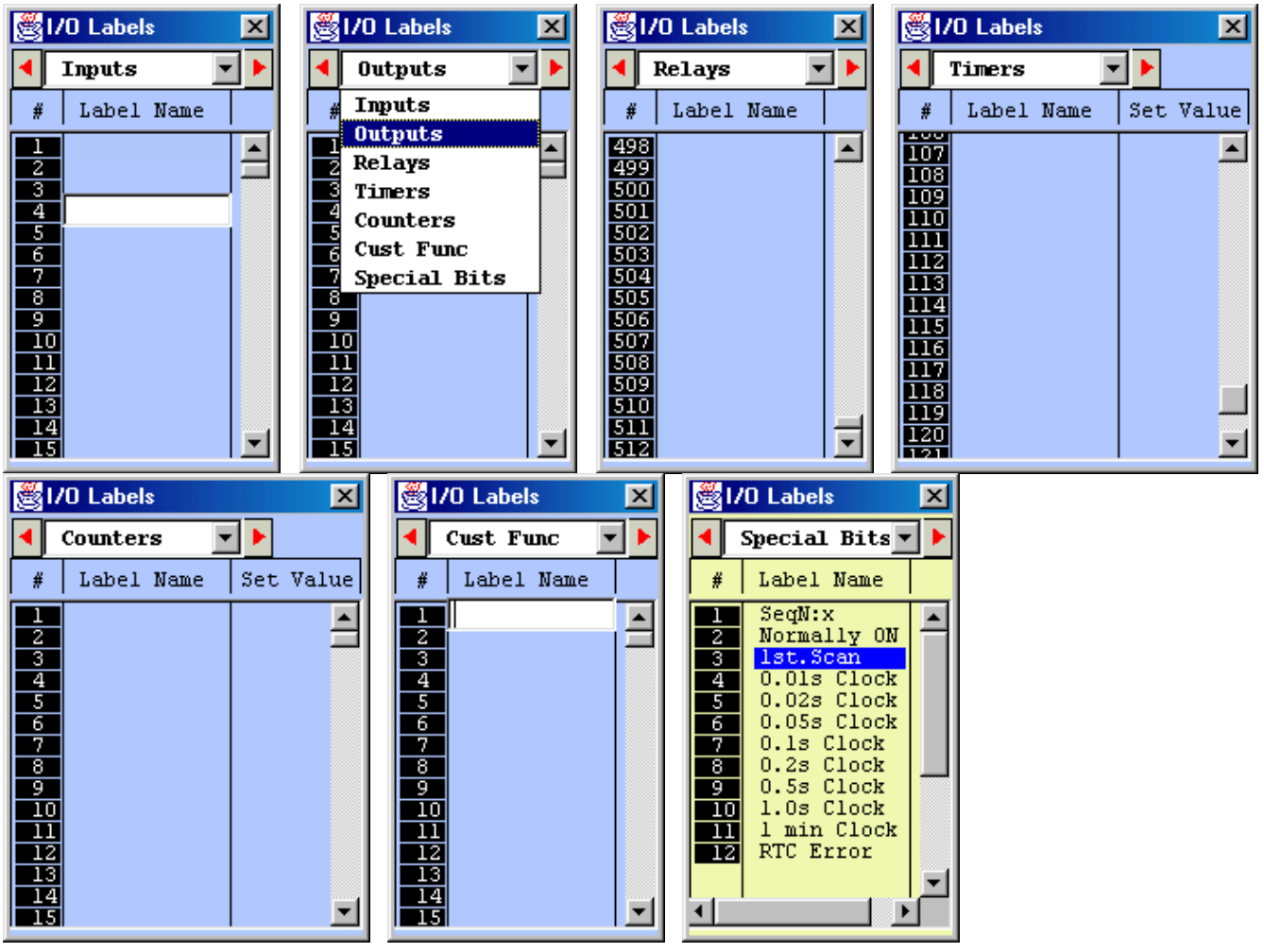
6.6.1 Open I/O Table

You can open the I/O table by either pressing the <F2> key or by clicking the “I/O Table” button, which has been added to the top status line of the i-Trilogi Window. It is good to remember this short cut key since you will be using it very often. The first time you press <F2> or click the “I/O Table” button, the input table will be opened.

6.6.2 Navigating the I/O Table

You can then scroll to other tables using one of the three methods:

- Using the left/right cursor keys.
- Click on the word “Inputs” – a choice box will open for you to select the I/O table you wish to scroll to.
- Click on the two buttons beside the “Inputs” choice box to scroll left or right from table to table.



NOTE: With i-Trilogi version 6.2 and up, the I/O Table edit or select pop up is no longer modal. I.e. it is now possible to have I/O table opened for reference while you browse or edit a ladder circuit.

6.6.3 Editing Label Names

To create or edit labelname for an I/O, simply click on the I/O number in the opened table and a text entry box will appear for you to enter/edit a label name. Once you have finished editing, press the <Enter> key to close the box and the name will appear on the I/O table. You can also use the keyboard to move the blue color highlight bar to an I/O location and press the <Space> bar to edit the label.

6.6.4 Editing Set Values (Timers and Counters only)

Timer and Counter Tables each has an extra “Set Value” column. If you define a new timer/counter label the Set Value field for this newly defined timer/counter will be opened for entry. Otherwise you can manually open this field for entry by clicking on it or by pressing the <End> key. Only numeric values between 0 and 9999 should be entered. If you enter an illegal character, the text field will not be closed when you press <Enter>.

6.6.5 Label Name Restrictions

You can enter up to 16 Unicode characters per label name. Special symbol such as “+”,”-“,”%” etc cannot be used.. No space is allowed between characters. Compressed font is used to display the label when the label name exceeds a certain width. Otherwise normal font is used. If you enter more than 16 characters, then an error message will inform you that the maximum label name width has been exceeded when you press the <Enter> key to close the text entry field.

For some languages such as Chinese Korean or Japanese, using all 16 characters for a label name may cause the label name to be wider than the I/O table itself and therefore part of the label name will be covered by the border. If this happens, you can drag on the lower right hand corner of the I/O table to increase its width by up to 50%.

6.6.6 Cust Func Table

This table is meant for you to define a label name for a custom function, which is now required. Previously, it was possible to leave a CusFn unnamed in order to keep compatibility with older i-TRiLOGI software versions 4.x. However, with i-Trilogi version 6.2 and up, it is Mandatory to use a label name for a custom function and simply leaving it blank with a default CuFn # wont work anymore.

Important Notes

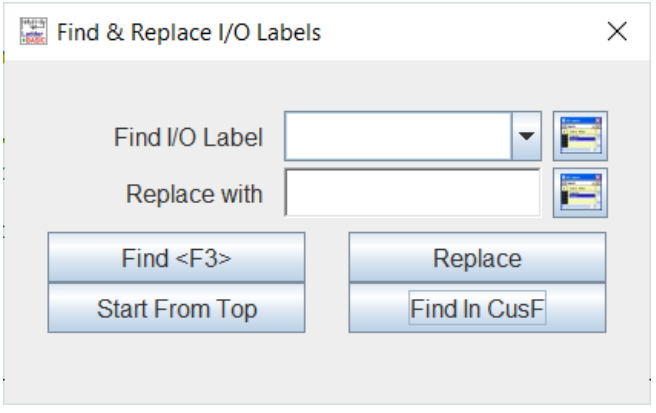
a. You can shift the Items in the I/O table up or down or insert a new label between two adjacent, pre-defined labels. Simply press the <Ins> key or **Right-Click** the mouse button to pop up the "Shift I/O" menu which allows you to shift the selected I/O. However, please note that if you shift the I/O down, the last entry in the I/O table (e.g. Input #256) will be lost.

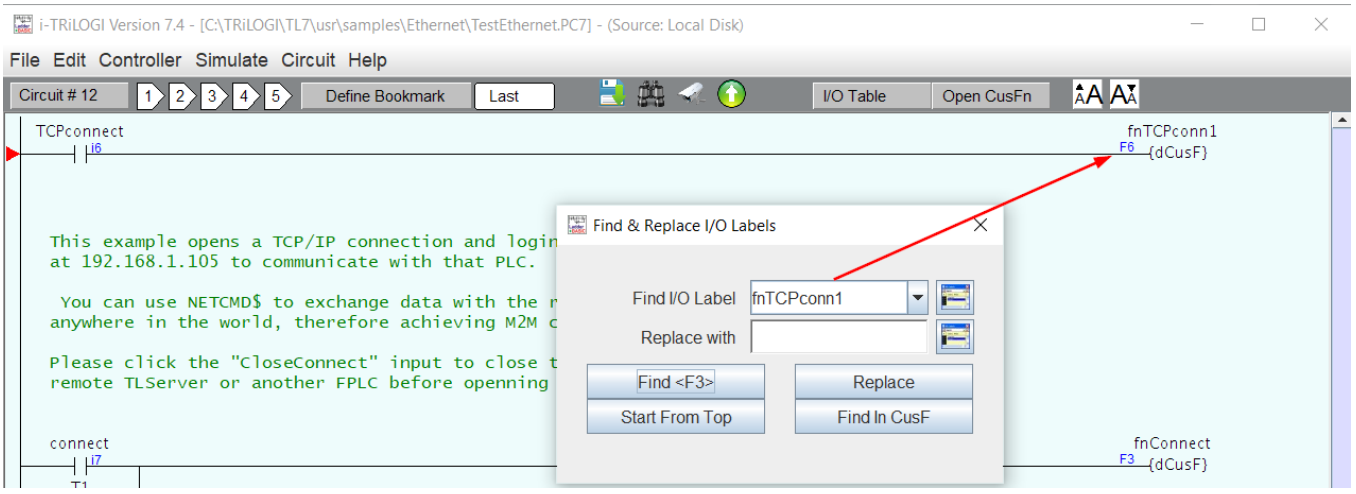
b. Shifting of Custom Function Label names will now shift the function content along with the label name. (In previous versions of i-Trilogi, shifting of the I/O label would not shift the function content, therefore making it untenable to use I/O label shift to reorganize custom functions. Warnings are provided if such an action were to result in overwriting of an existing custom function.

c. i-TRiLOGI Version 6.2 and up allows I/O label names of up to 16 characters. However, if you wish to keep compatibility with Version 5.x to 6.1x, you should use no more than 10 characters to define the I/O names.

6.7 Ladder Logic Editor Search

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. 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. The Find command can also be used to search for a keyword in a TBASIC program. (https://docs.triplc.com/trilogi/#5809)

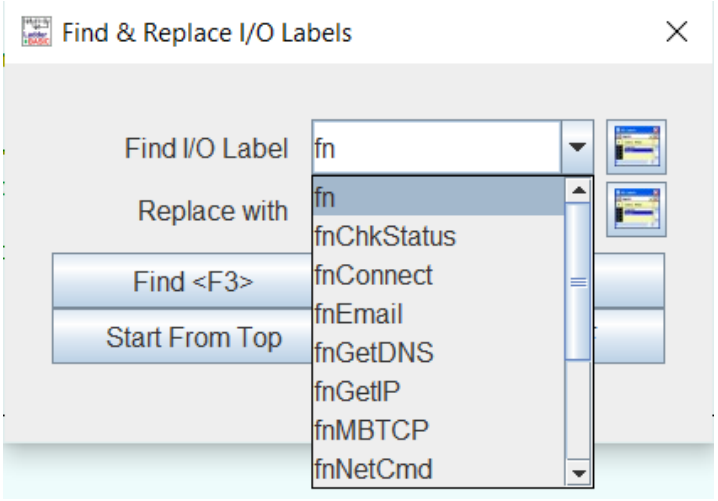




6.7.1 Find Ladder Element

There are three ways to choose a ladder element to search for:

1. Enter into the text field  a string that partially or fully matches the label name you wish to locate.
   Note that as text is entered, an auto-complete process will start such that a list of potentially matching label names will appear in a drop down menu. Any of these can be selected via mouse click or keyboard 'Enter' (use the arrow keys or scroll bar to traverse the menu).



2. Press the <F2> key or click on the table icon to open up the I/O table and pick the label name from the I/O table.

3. Click the  icon to call up the I/O Table and pick the label name from the I/O table.

6.7.2 Replace Ladder Element

There are two ways to enter text in the replace field and both will overwrite text found that is equivalent to the Find I/O Label field.

1. You can manually enter new text in the Replace with field.

2. Click the  icon to call up the I/O Table and pick the label name from the I/O table. This will be automatically entered in the 'Replace with' field.

Clicking Replace will both find and replace the text (no need to click Find first). The search and replace function will iterate through the ladder logic circuits and advance to the next matched result each time Replace is clicked. You can select a Label Name from the I/O Table to find and another to replace it with, allowing you to easily swap or update label names.

6.7.3 Find and Replace Text in CusFn

Refer to chapter 7.6 Custom Function Editor Search (https://docs.triplc.com/trilogi/#5809) for more details on searching for text in the custom function editor.

Chapter 7 - TBASIC Programming

7.1 TBASIC Introduction

i-TRiLOGI Version 6 & 7 support 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 (https://docs.triplc.com/trilogi/#5569).

TBASIC is a structured-text language 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.

Like any common structured text programming language, standard loops such as FOR and While, and conditional statements such as IF, ElseIF are supported by TBASIC.

There are also many built-in keywords to accomplish complex tasks such as interfacing to digital and analog I/O, performing serial and network communication, managing a PID control system, and handling interrupts. These built-in commands simplify programming tasks and allow for more reliable control.

7.2 Create Custom Function

There are three simple ways to create a new CusFn:

1. Edit Menu

From the “Edit” pull-down menu, select the item “Edit Custom Function” and select the function number from a pop-up CusFn selection table which may range from 1 to 256. 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. Keyboard Shortcut

You may also use the hotkey <F7> to open up the selection table.

3. Ladder Circuit

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 double clicking the left mouse button while the highlight bar is at the [CusFn] or [dCusFn]. Alternatively, you can open the CusFn by clicking the right mouse button while the highlight bar is at the [CusFn] or [dCusFn].

Open a Function

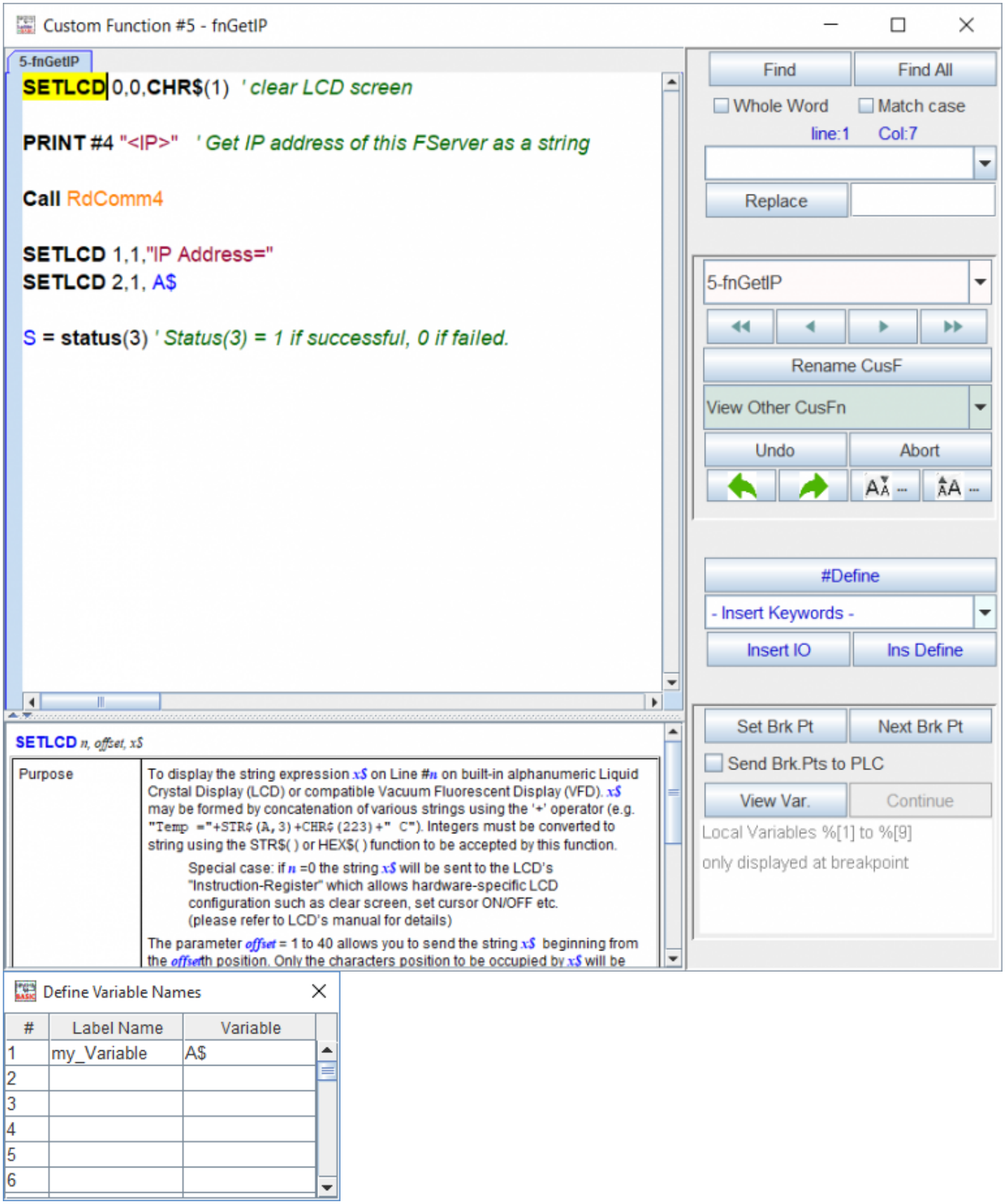
You can open a custom function by clicking on the “Open CusFn” button on the status bar below the pulldown menu.

7.3 Custom Function Editor

7.3.1 The i-TRiLOGI Custom Function Editor

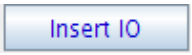
The custom function editor window allows creation of any number of lines of TBASIC program statements. Since this is a standard text editor, you should have no problem using the key and mouse controls to edit the text. E.g. To copy a paragraph of text, select it using the mouse and the press <Ctrl-C>. Alternatively, you can use <Ctrl-X>, which will cut the text (copy and delete). Once you have copied or cut the desired text, move the text editing cursor to the destination and press <Ctrl-V> to paste it to the new location.

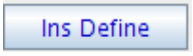
i-TRiLOGI 6.7 / 7.4 and up introduces an updated custom function editor with multiple view tabs to allow quick viewing of other custom functions. The custom function editor now has a whole new interface with split panes that provide text editing for writing code (just like before) on the upper pane and a web browser/TBASIC help information on the lower pane. The first time the custom function editor is opened after starting i-TRiLOGI, the text editor pane will be empty and the lower pane will display some information about Triangle Research, our products, or software update information (see below). The lower pane can also be used to display help or error messages when writing or compiling your program. A #Define button has been added as of i-TRiLOGI version 6.42 which allows you to define meaningful label names for any variables or expression.



7.3.2 New Function Editor Features.

As of i-TRiLOGI version 6.7 / 7.4, the following improvements have been added:

1. Tabbed Editor for switching between editing a single function and simultaneously viewing multiple functions.
    - Allows multiple view windows to appear as tabs next to the main editor window for easy reference.
    - Allows you to quickly switch a function from View tab into the Editor tab by right clicking inside the View tab window or left-clicking on the View tab itself.
    - View tabs are now syntax-formatted and allow mouse over viewing of variable's value.
2. Added two new options "Whole Word" and "Match case" checkboxes to the "Find" function.
3. Pick a keyword from the "-Insert Keywords-" choice box by typing the first few characters into the choice box and it will present a menu of keywords beginning with the characters you just typed.
4. A new 

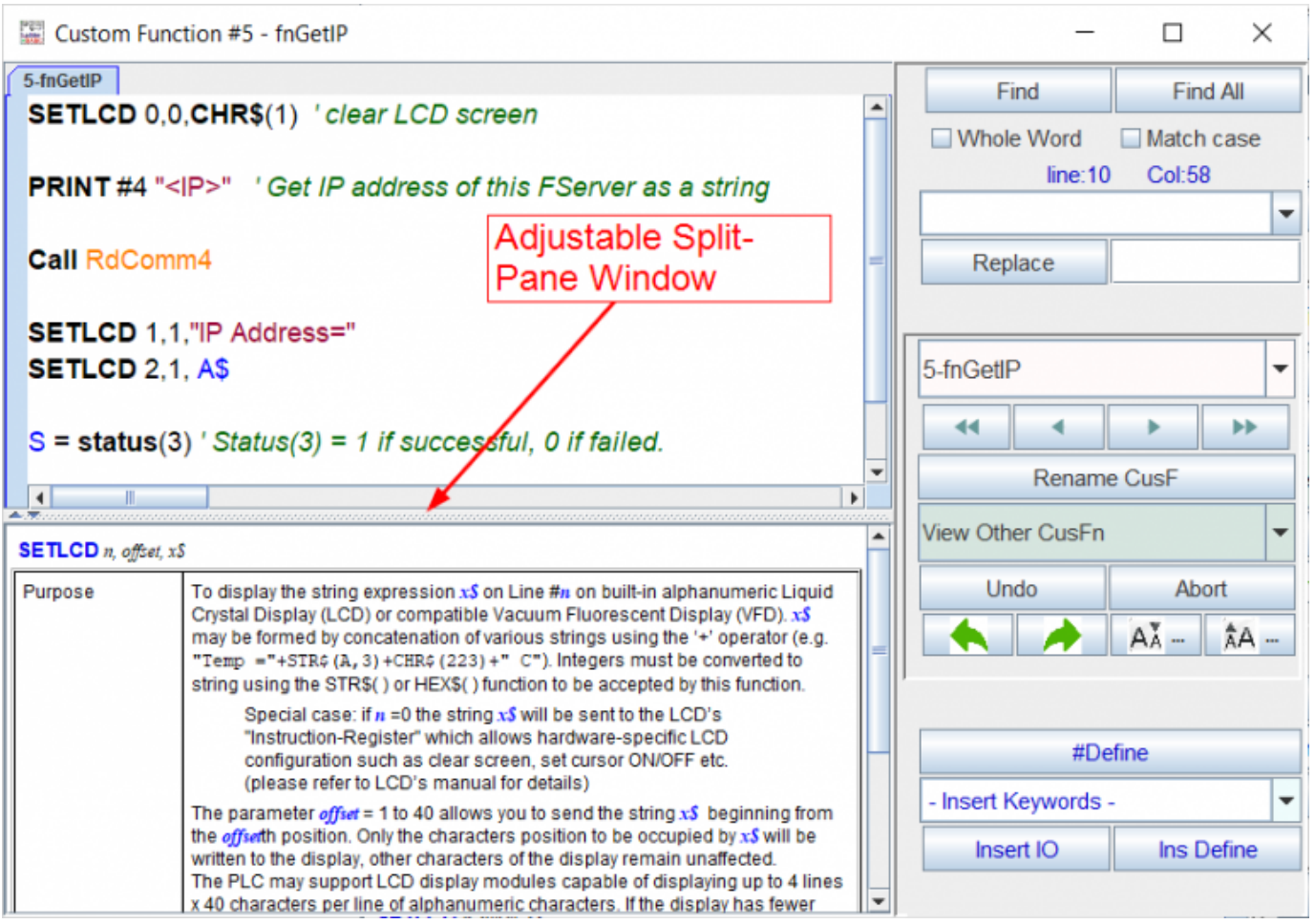
button lets you pick an I/O label from the I/O table and insert into the custom function editor (can also be done by pressing F2 key)
5. A new 

'Ins Define" button lets you pick a #Define name from I/O table and insert into the custom function editor.

7.4 Custom Function Editor Layout & Navigation

7.4.1 Split Pane Window

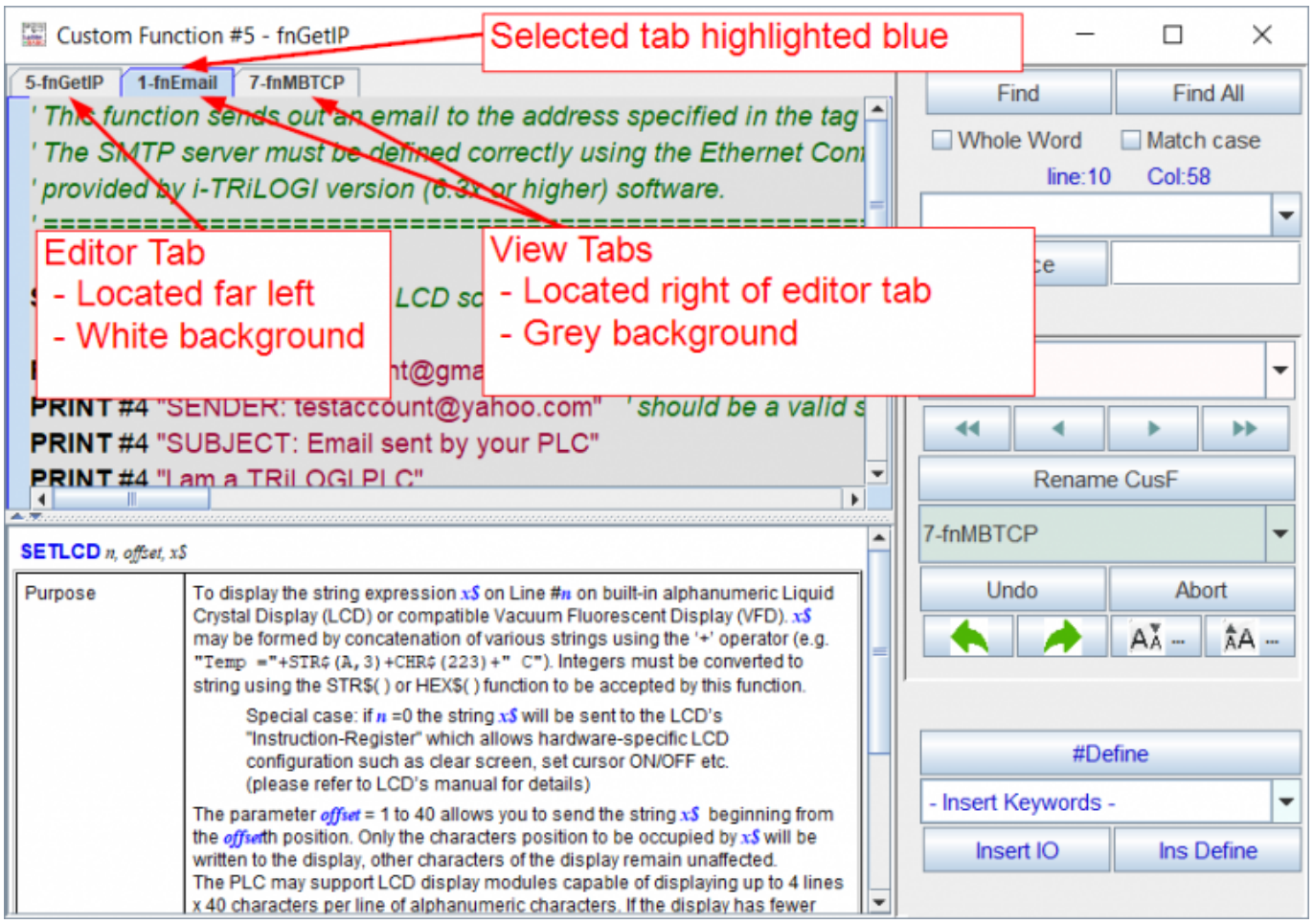
The upper pane is used as a text editor for writing TBASIC code and the lower pane can either display information about Triangle Research (as shown above), or provide immediate help for any TBASIC keyword (as shown below). To use the lower pane for help on TBASIC keywords, simply select the keyword from the "Select Keyword" drop box or highlight the text in the editor. For Example: in the following screenshot, the code "SETLCD" is highlighted and the syntax for "SETLCD" would be displayed as it is in the screenshot.

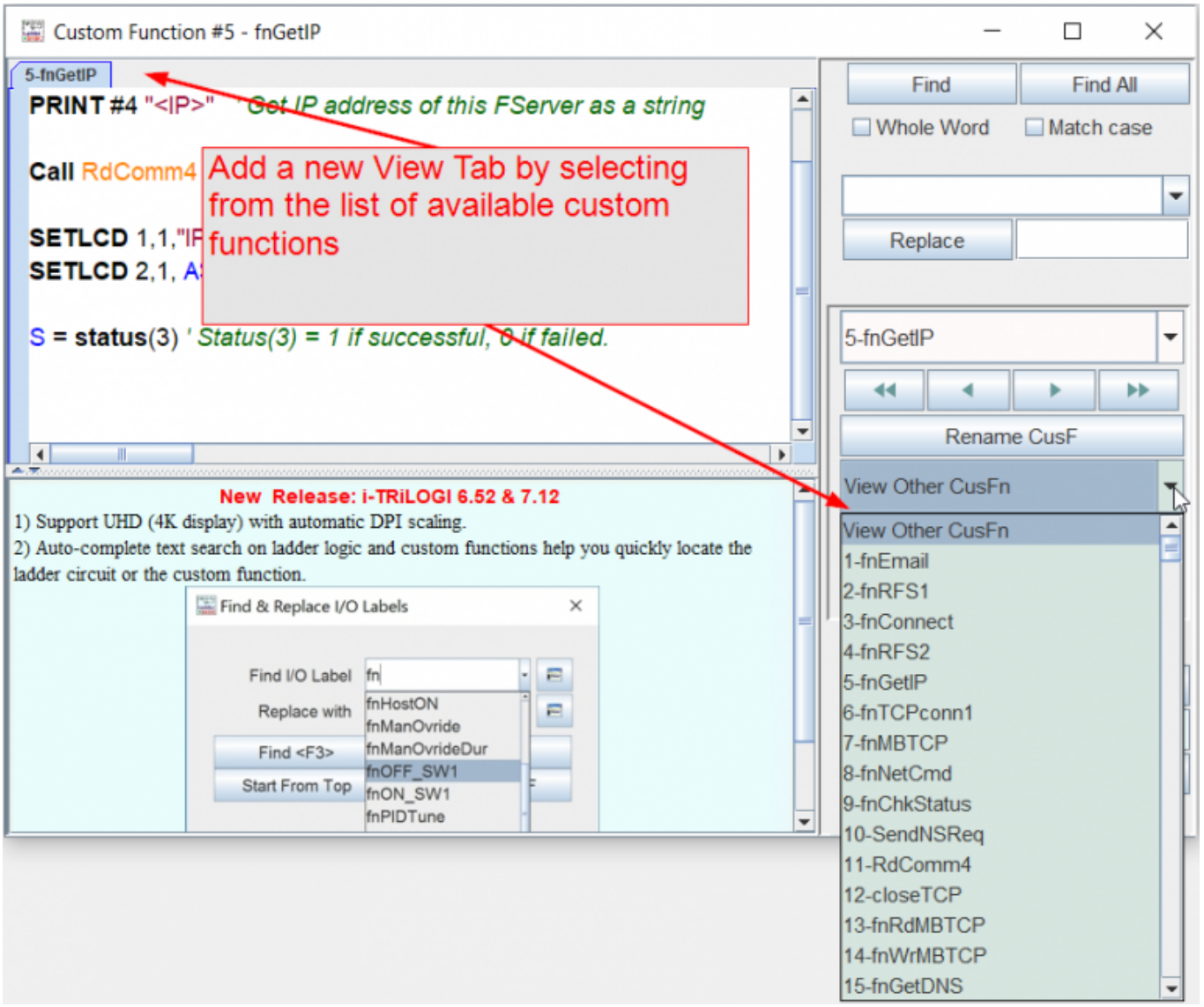


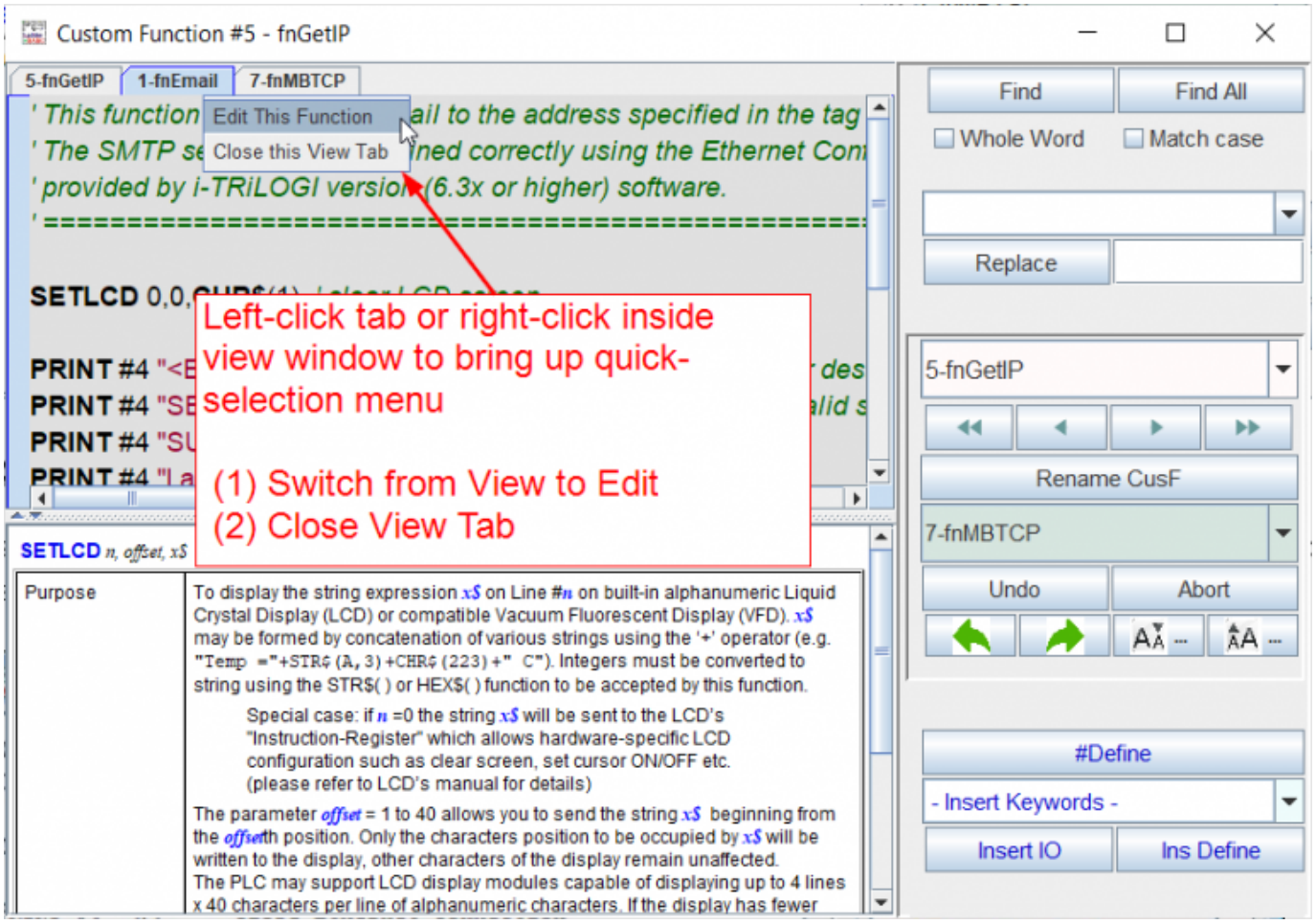
7.4.2 Tabbed Editor

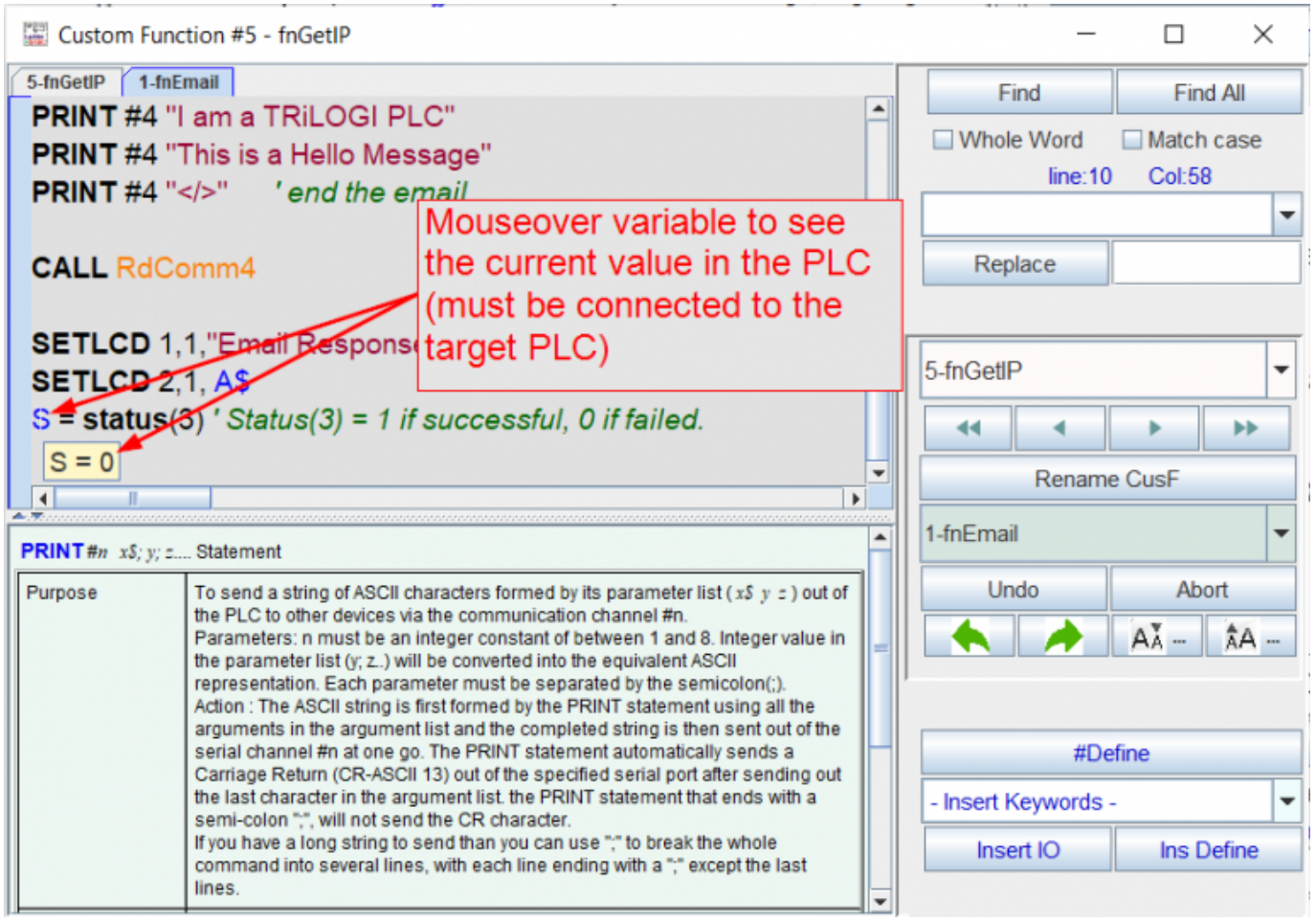
I-TRiLOGI version 6.7 / 7.4 introduces a new tabbed editor format that enables switching between editing a single function and simultaneously viewing multiple functions.

- Allows multiple view windows to appear as tabs next to the main editor window for easy reference.
- Add new tabs by selecting from the View Function drop-down menu.
- Allows you to quickly switch a function from View tab into the Editor tab by right clicking inside the View tab window or left-clicking on the View tab itself.
- View tabs are now syntax-formatted and allow mouse over viewing of variable's value.


Custom Function #5 - fnGetIP
Selected tab highlighted blue
5-fnGetIP
1-fnEmail
7-fnMBTCP
Editor Tab
- Located far left
- White background
View Tabs
- Located right of editor tab
- Grey background
PRINT #4 "SENDER: testaccount@yahoo.com"
PRINT #4 "SUBJECT: Email sent by your PLC"
Find
Find All
Whole Word
Match case
line:10
Col:58
Rename CusF
7-fnMBTCP
Undo
Abort
#Define
- Insert Keywords -
Insert IO
Ins Define
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 offsetth 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


Custom Function #5 - fnGetIP
5-fnGetIP
PRINT #4 "<IP>" ' Get IP address of this FServer as a string
Call RdComm4
SETLCD 1,1,"IP
SETLCD 2,1, A
S = status(3) ' Status(3) = 1 if successful, 0 if failed.
Add a new View Tab by selecting from the list of available custom functions
Find
Find All
Whole Word
Match case
Replace
5-fnGetIP
Rename CusF
View Other CusFn
New Release: i-TRiLOGI 6.52 & 7.12
1) Support UHD (4K display) with automatic DPI scaling.
2) Auto-complete text search on ladder logic and custom functions help you quickly locate the ladder circuit or the custom function.
Find & Replace I/O Labels
Find I/O Label
fn
Replace with
Find <F3>
Start From Top
fnHostON
fnManOvride
fnManOvrideDur
fnOFF_SW1
fnON_SW1
fnPIDTune
View Other CusFn
1-fnEmail
2-fnRFS1
3-fnConnect
4-fnRFS2
5-fnGetIP
6-fnTCPconn1
7-fnMBTCP
8-fnNetCmd
9-fnChkStatus
10-SendNSReq
11-RdComm4
12-closeTCP
13-fnRdMBTCP
14-fnWrMBTCP
15-fnGetDNS


Custom Function #5 - fnGetIP
5-fnGetIP
1-fnEmail
7-fnMBTCP
Edit This Function
Close this View Tab
Left-click tab or right-click inside view window to bring up quick-selection menu
(1) Switch from View to Edit
(2) Close View Tab
Find
Find All
Whole Word
Match case
Replace
5-fnGetIP
Rename CusF
7-fnMBTCP
Undo
Abort
#Define
- Insert Keywords -
Insert IO
Ins Define
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 offsetth 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

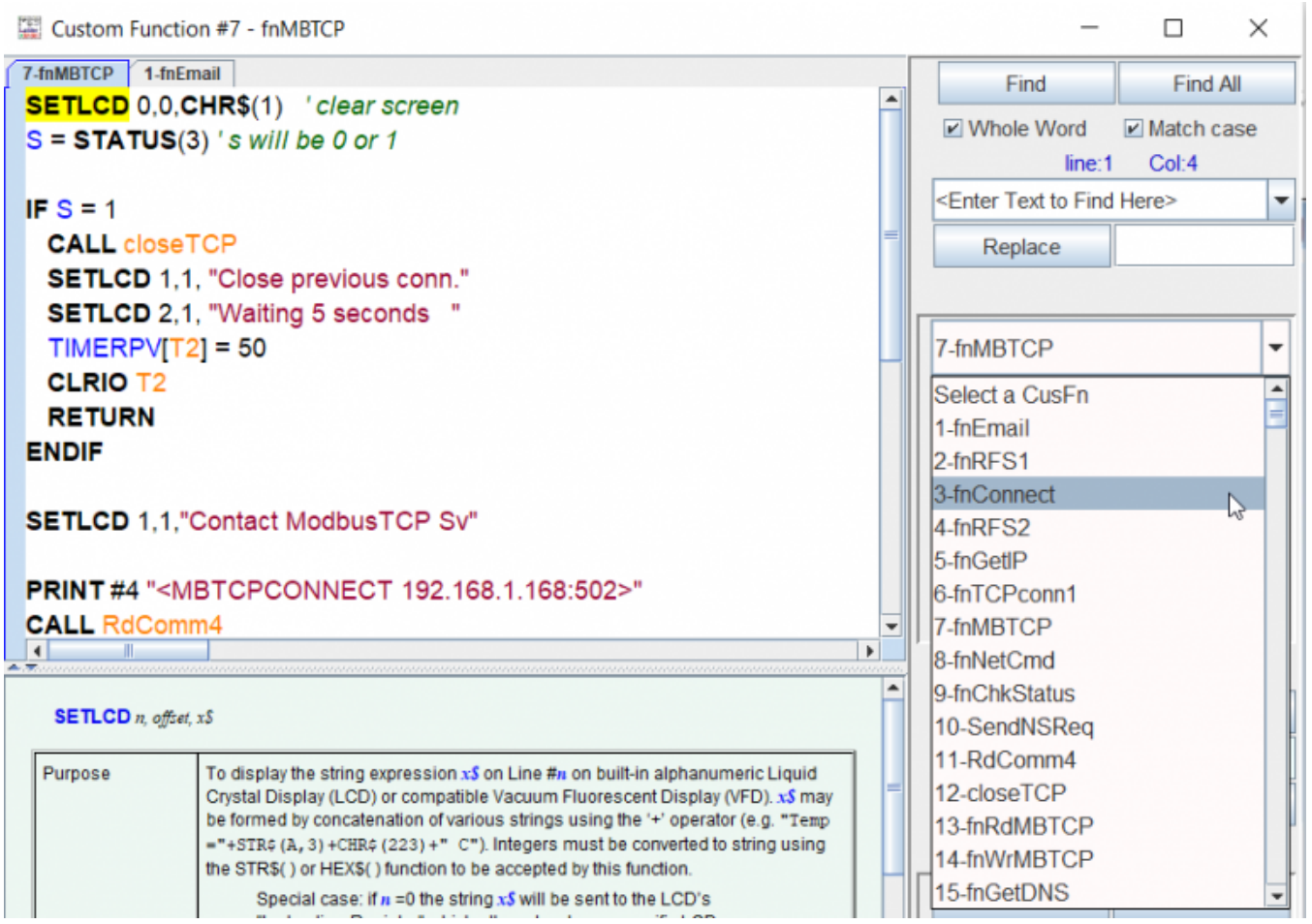

Custom Function #5 - fnGetIP
5-fnGetIP
1-fnEmail
PRINT #4 "I am a TRiLOGI PLC"
PRINT #4 "This is a Hello Message"
PRINT #4 "</>" ' end the email
CALL RdComm4
SETLCD 1,1,"Email Response
SETLCD 2,1, A$
S = status(3) ' Status(3) = 1 if successful, 0 if failed.
S = 0
Mouseover variable to see the current value in the PLC (must be connected to the target PLC)
Find
Find All
Whole Word
Match case
line:10 Col:58
Replace
5-fnGetIP
Rename CusF
1-fnEmail
Undo
Abort
#Define
- Insert Keywords -
Insert IO
Ins Define
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 channel #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. the 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.

7.4.3 Navigating Custom Functions

There are multiple ways to navigate custom functions in i-TRiLOGI 6.7 / 7.4 and up.

Drop-down Menu

It is now possible navigate to any other function directly by choosing the function name from the “Select Function To Edit” drop box, as shown below.



Arrow Keys

Alternatively, you can scroll from one custom function to the next one using the < and > 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.

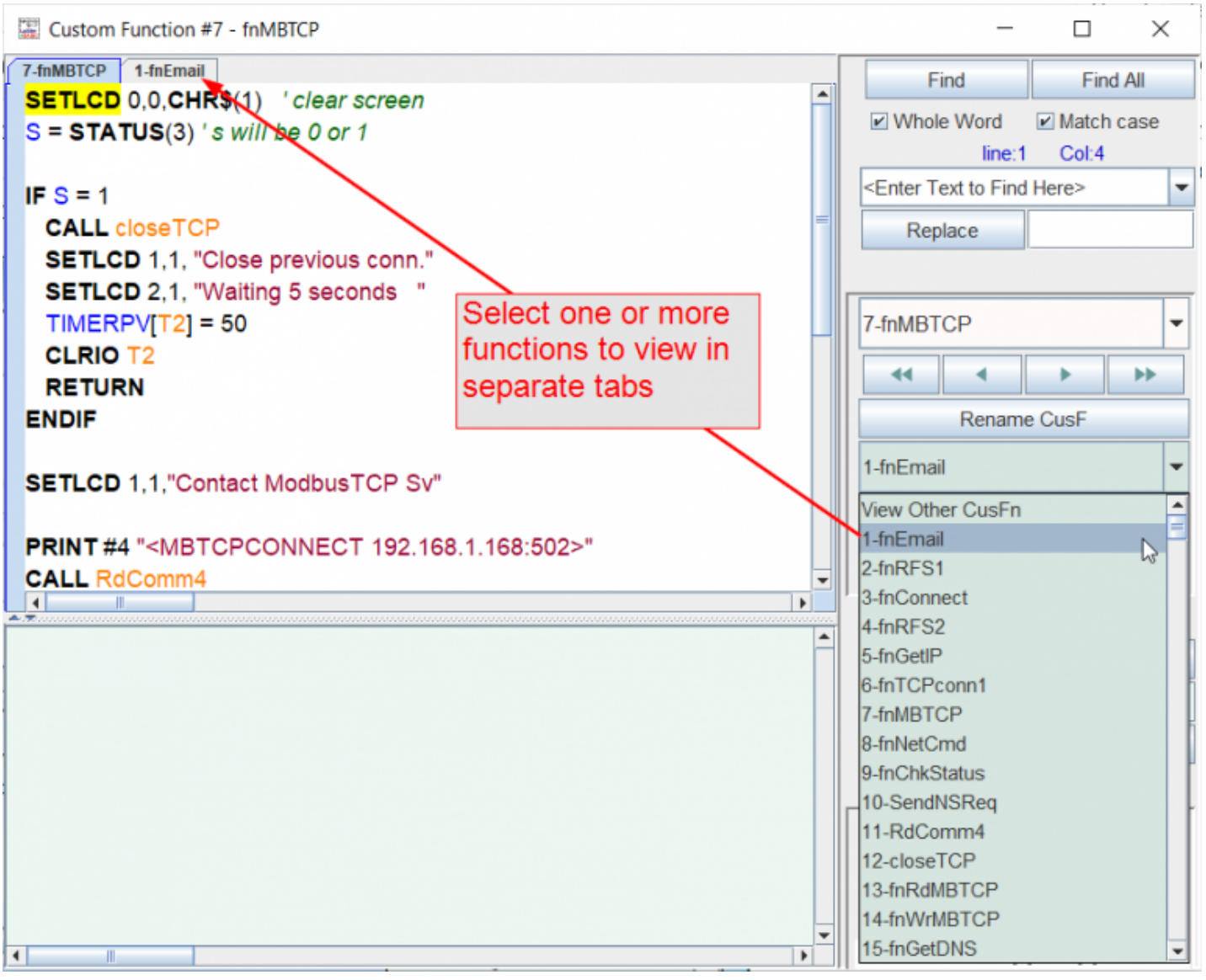
Forward and Back Keys

You can traverse backwards and forwards through only functions that have been edited using the  and  keys. This means that an existing function that was not edited since the program was last opened will be ignored.

This is useful if only a couple of functions are being worked on, but they are separated by many other functions, which would be more time consuming to switch between using the other methods above.

7.4.4 View Other Custom Functions

The “View Other Functions” choice box allows you to view another custom function via a read-only tab window without closing the currently edited custom function, as described in section 7.4.2 Tabbed Editor. This is very convenient when you need to view the content of another custom function while editing one custom function.



7.4.5 Save Changes

All changes are updated automatically. The editor can be closed after making changes and those changes will be recalled when the function is opened again. However, the .PC6/.PC7 program must be saved to retain the custom function (and ladder logic) changes before the file is closed. This is done via the *File –> Save* or *ctrl + s* keyboard shortcut operations.

7.4.6 Undo Changes 

Click this button to undo any changes you just made to the custom function. You may also use the <Ctrl-z> shortcut key to achieve the same.

The “Undo” button (shown below) can be used to undo the last change to the text in the currently opened Custom Function.

7.4.7 Aborting Changes 

The “Abort” button can be used to abort any changes made to the currently opened Custom Function since it was opened. The custom function editor will be closed when you abort an edit.

Note: If you navigate away from the currently opened custom function, or if you exit a custom function by using the <ESC> key or by clicking the  button in the top-right corner of the editor, then the content of the custom function currently in the editor will be updated into the internal memory and you will not be able to undo or abort changes. Hence you can only abort changes to the currently opened custom function before you exit the editor or navigate away to another Custom Function.

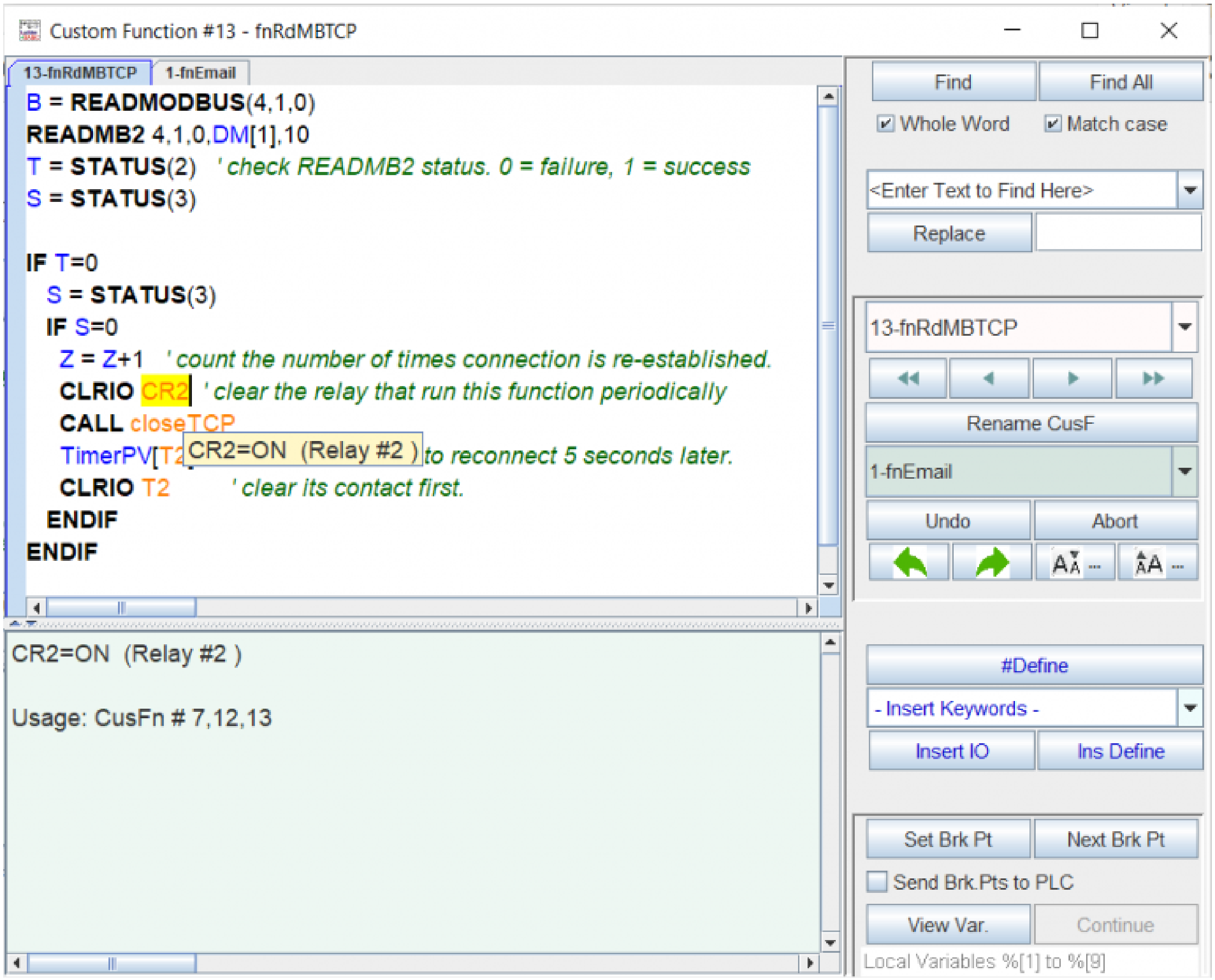
7.5 Custom Function Editor Interface

7.5.1 Select an I/O Label

If the selected text is an I/O label name, the lower pane will display its I/O type and the I/O number as well as the custom function # where the same label name have been used. This can be very useful to find out where a certain variable may be changed in other custom functions.

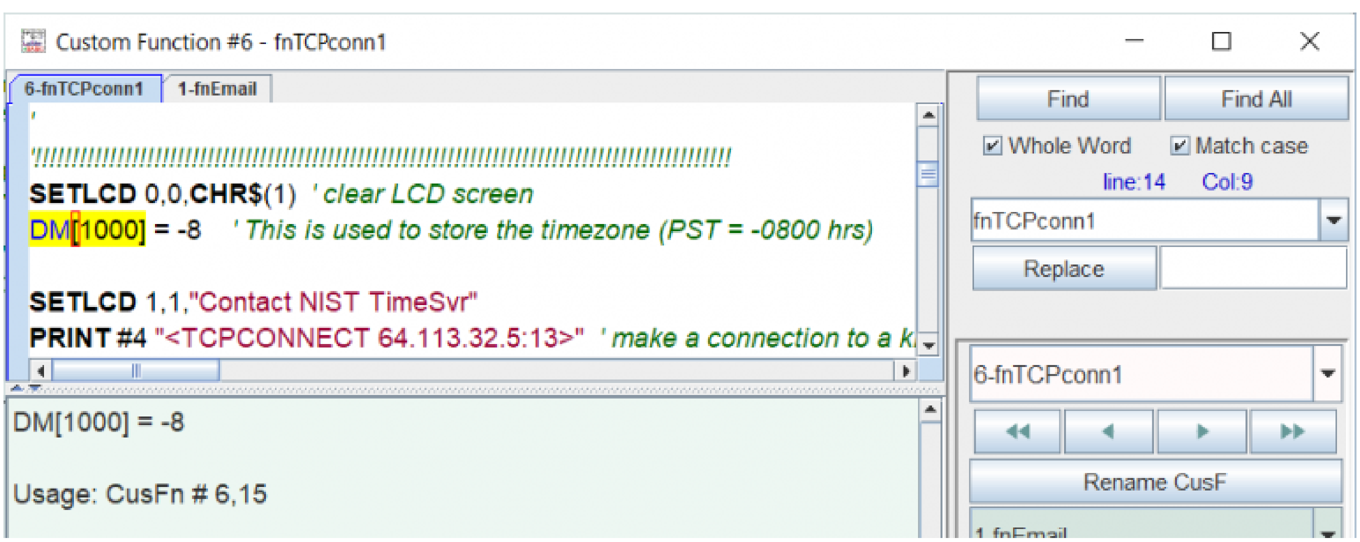
7.5.2 Mouseover an I/O Label

Without clicking or selecting a label, it is possible to mouseover and display a callout window with details about the label.

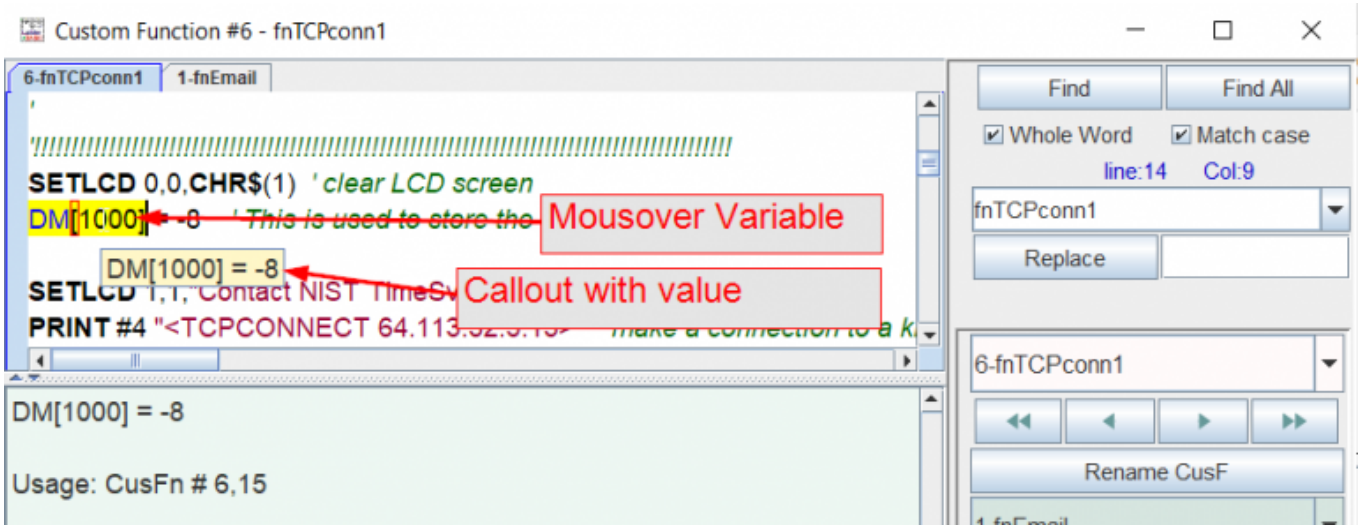


7.5.3 Select a Variable

The value is obtained from the simulator if i-TRiLOGI is currently disconnected from any TLServer or F-Server. If i-TRiLOGI is currently connected to a TLServer or F-server and online monitoring is active, then the program will retrieve the current value in real-time from the connected PLC. In the following example, variable DM[1000] has a current value = -8 and it is being used in Custom function #6 and 15.

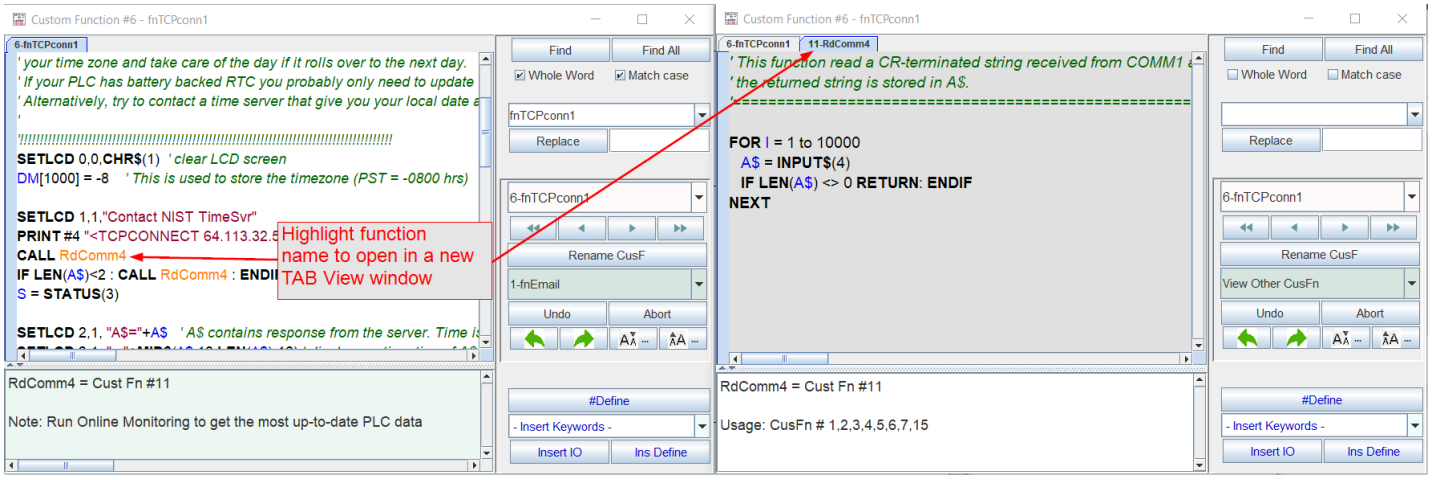


Now in version 6.5x/7.1x and higher it is possible to mouseover a variable to bring up a small callout window with details. Again if the software is in active online monitoring the current value of the variable will be retrieved from the PLC in real time.



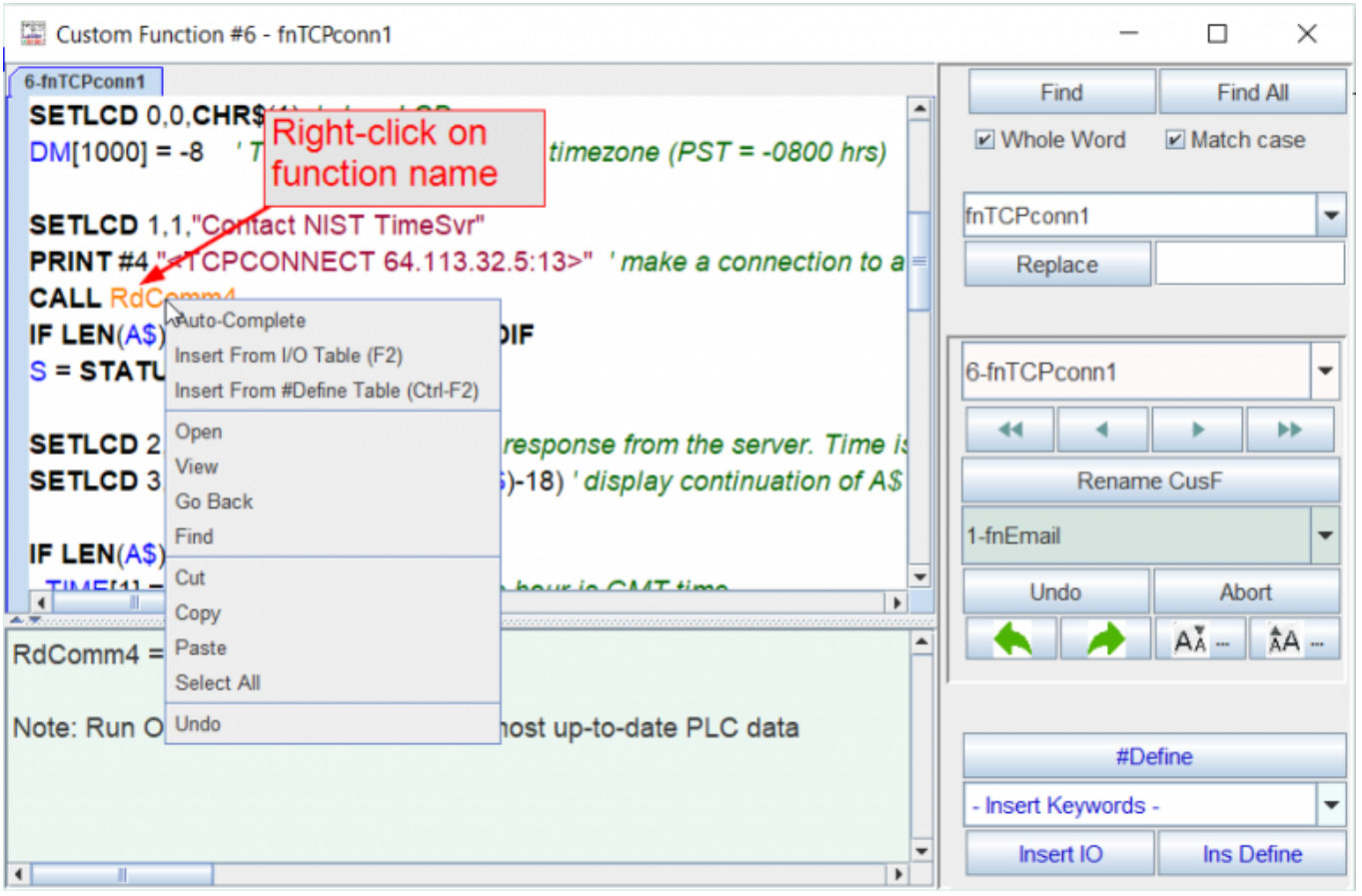
7.5.4 Select a Custom Function Name

If the selected text is the name of a custom function, the function will open in a new view tab window just like selecting a function to view from the drop-down menu. This enables the programmer to view the code of another custom function CALLed by this function without leaving the current function or having to find it from the list.



7.5.5 Right-Click Custom Function Name

When the cursor is over a custom function name in the editor, it is possible to right-click on the function and perform the tasks described below from the list of options:



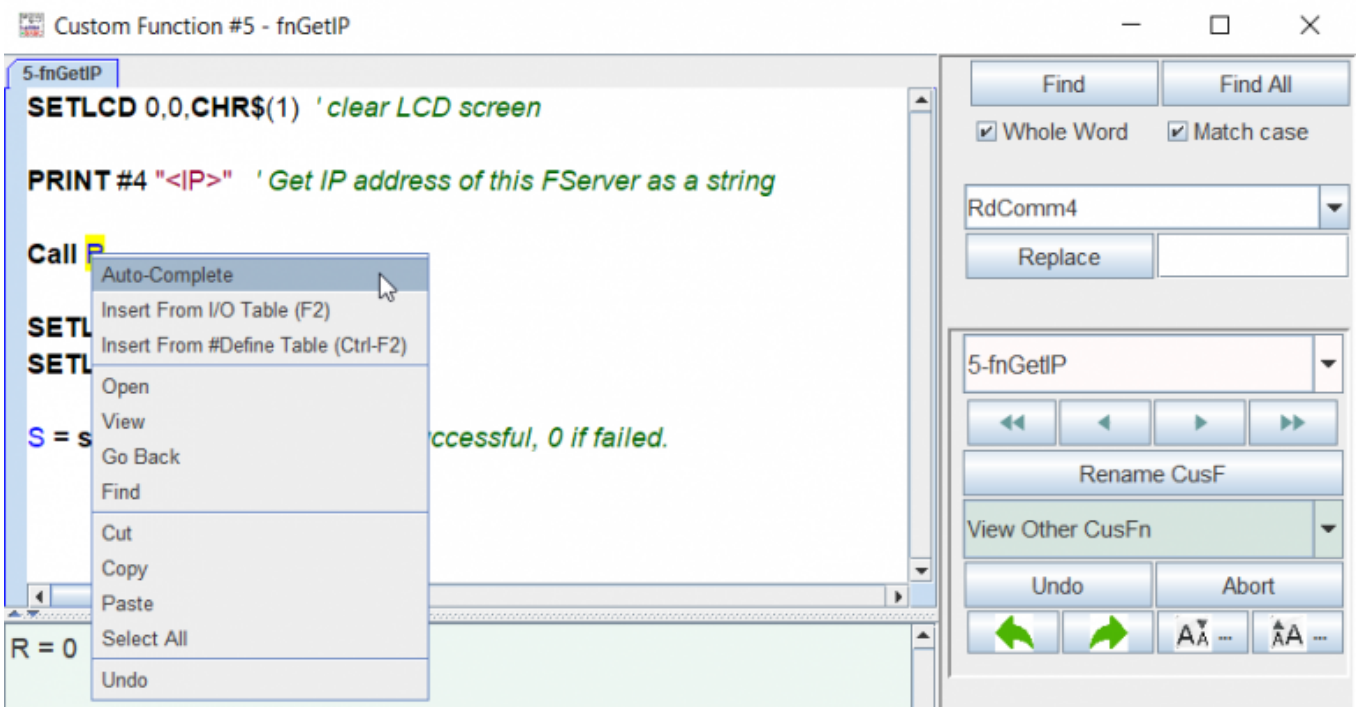
Open: You can edit the selected custom function by right-clicking on the selected name and select “Open” from the popup menu. This will switch from the current function to the right-clicked function in the editor tab (not a pop-up window).

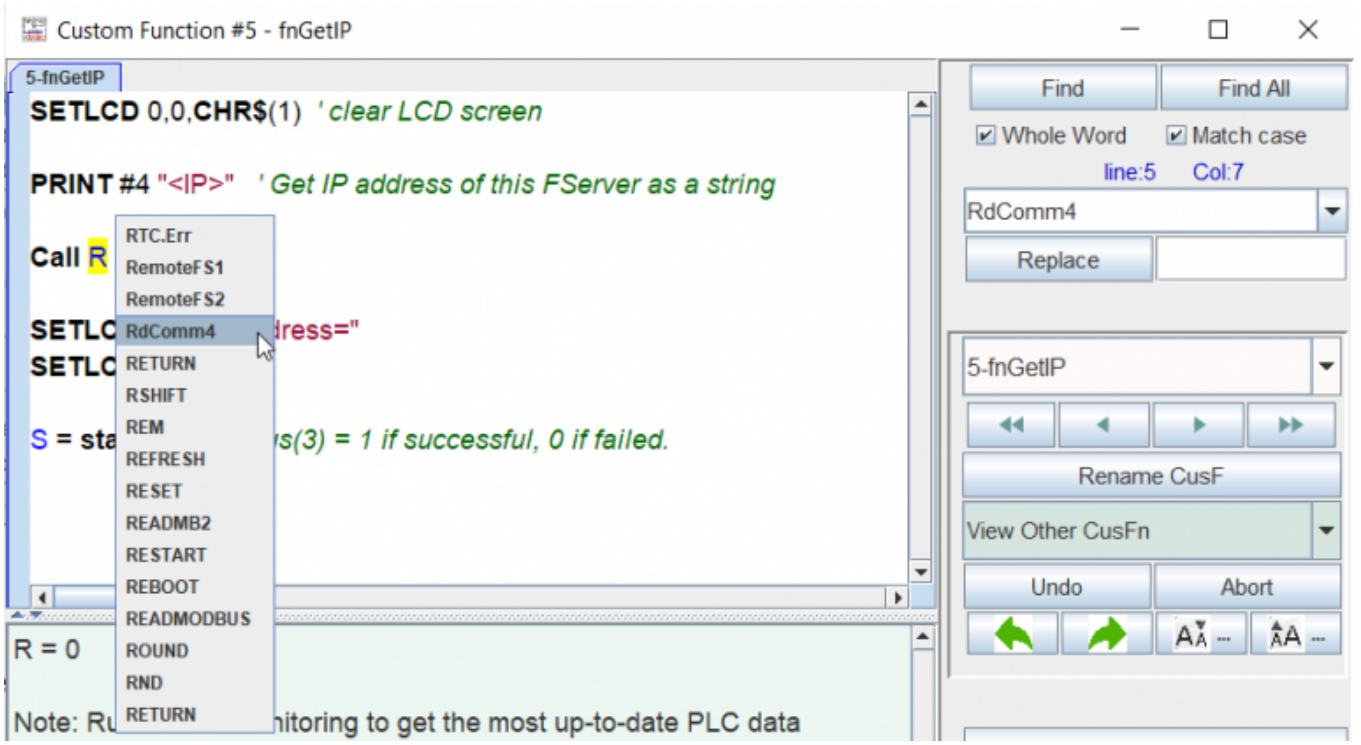
View: You can view the selected custom function by right-clicking on the selected name and select “View” from the popup menu. This will not change the function being edited and the right-clicked function will open in a new view tab window just like selecting a function to view from the drop-down menu.

Go Back: If you have changed functions since opening the editor, selecting “Go Back” will take you to the previously opened function. You can continue to go back until the first function opened is reached.

7.5.6 Auto-Complete

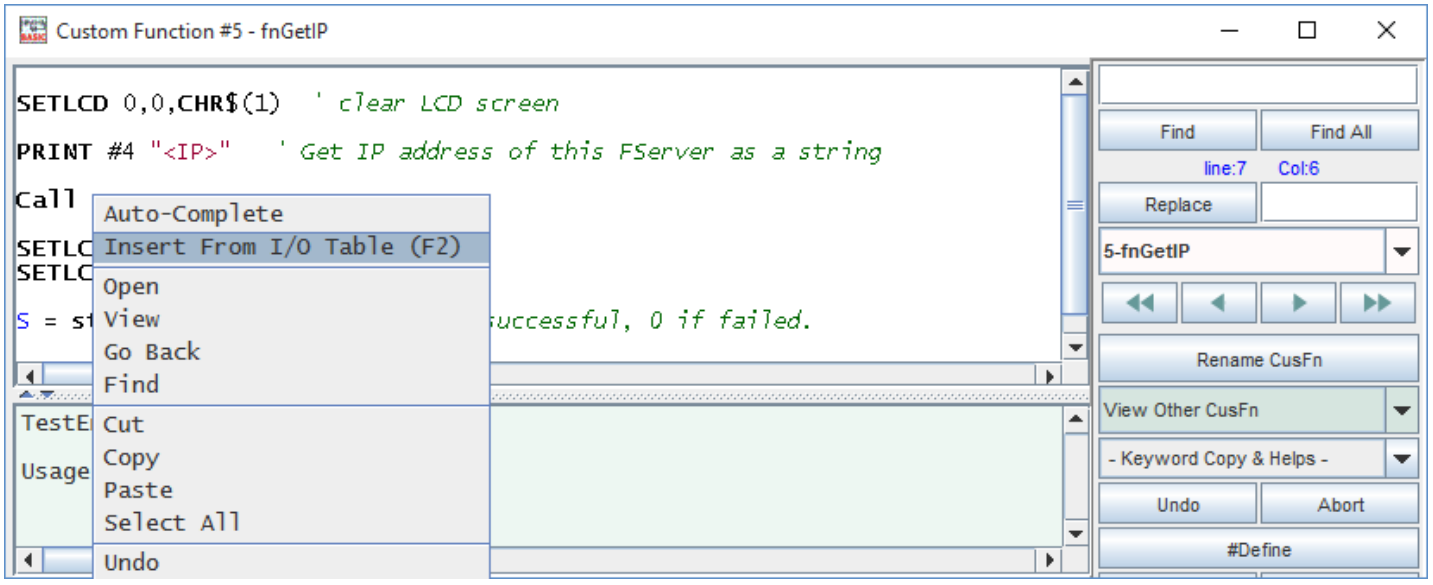
If the selected text contains part of the name of a custom function, TBASIC function, label name (I/O Table or define), then it is possible to right click and select Auto-Complete to bring a list of applicable function/label names to select from.





7.5.7 Insert from I/O Table (F2)

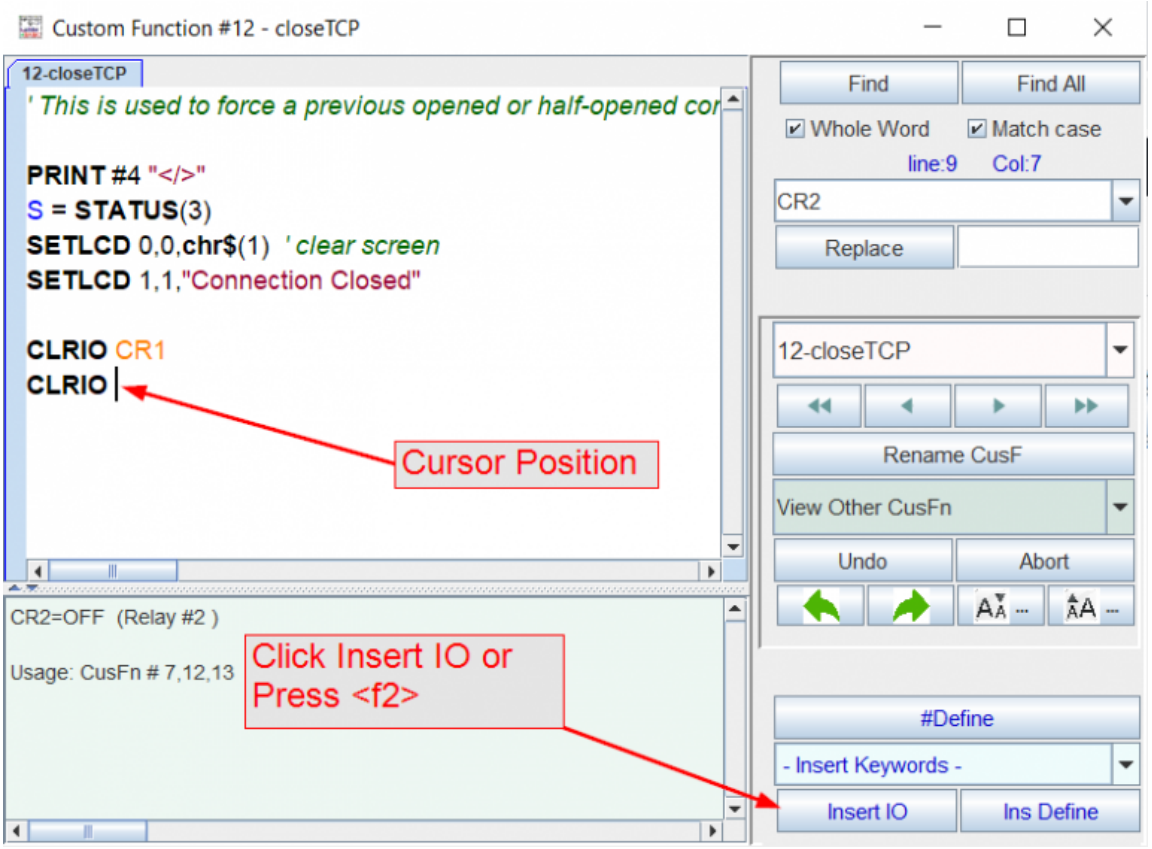
It is possible to insert elements from the I/O table by right clicking anywhere in the editor or by pressing the F2 shortcut. This could be used to select a custom function to call, an input label to test, an output label to control, etc.

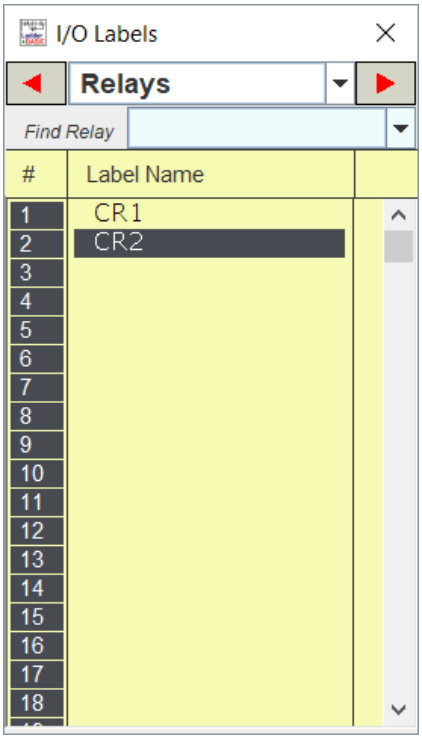


7.5.8 Insert I/O Label

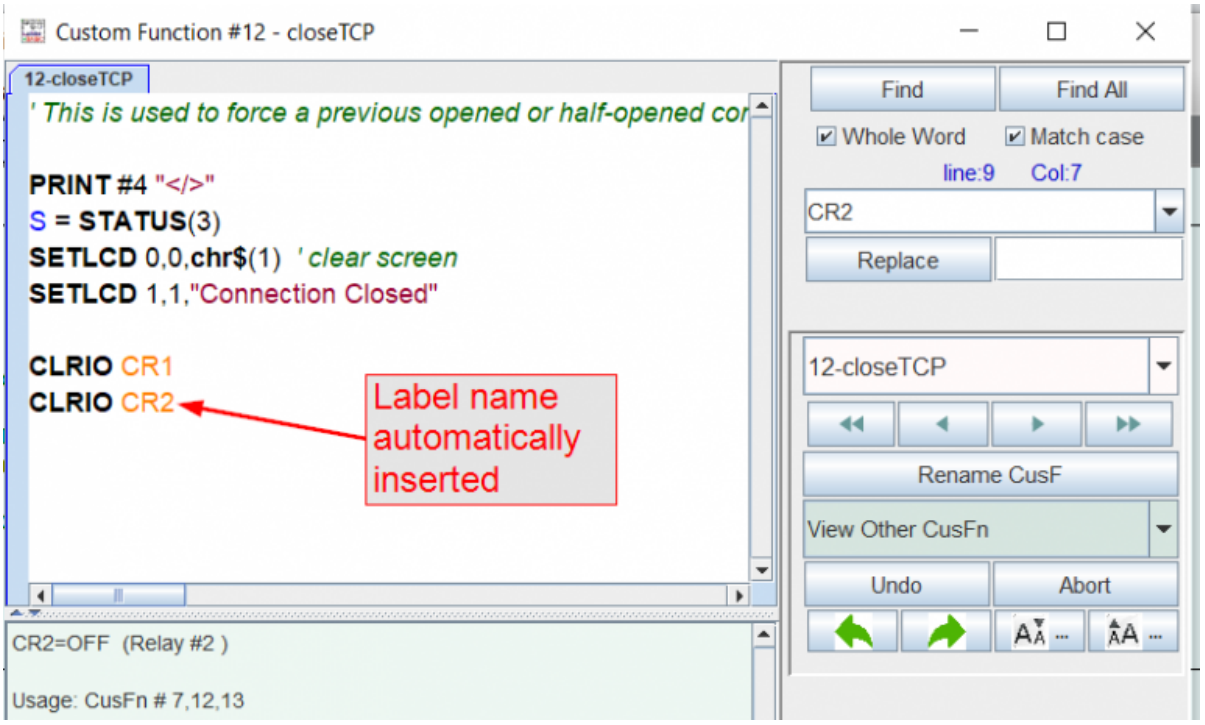
A new Insert IO button lets you pick an I/O label from the 'I/O Table (https://docs.triplc.com/trilogi/#6004)' and insert into the custom function editor (can also be done by pressing F2 key). Insert an I/O label into a custom function as follows:

1. **Set the cursor** to the position that the I/O label should be inserted.

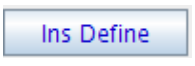




2. **Select the desired label name** from the I/O Table via mouse click or keyboard Enter on the highlighted field.
3. The corresponding **label name will now appear** in the custom function.



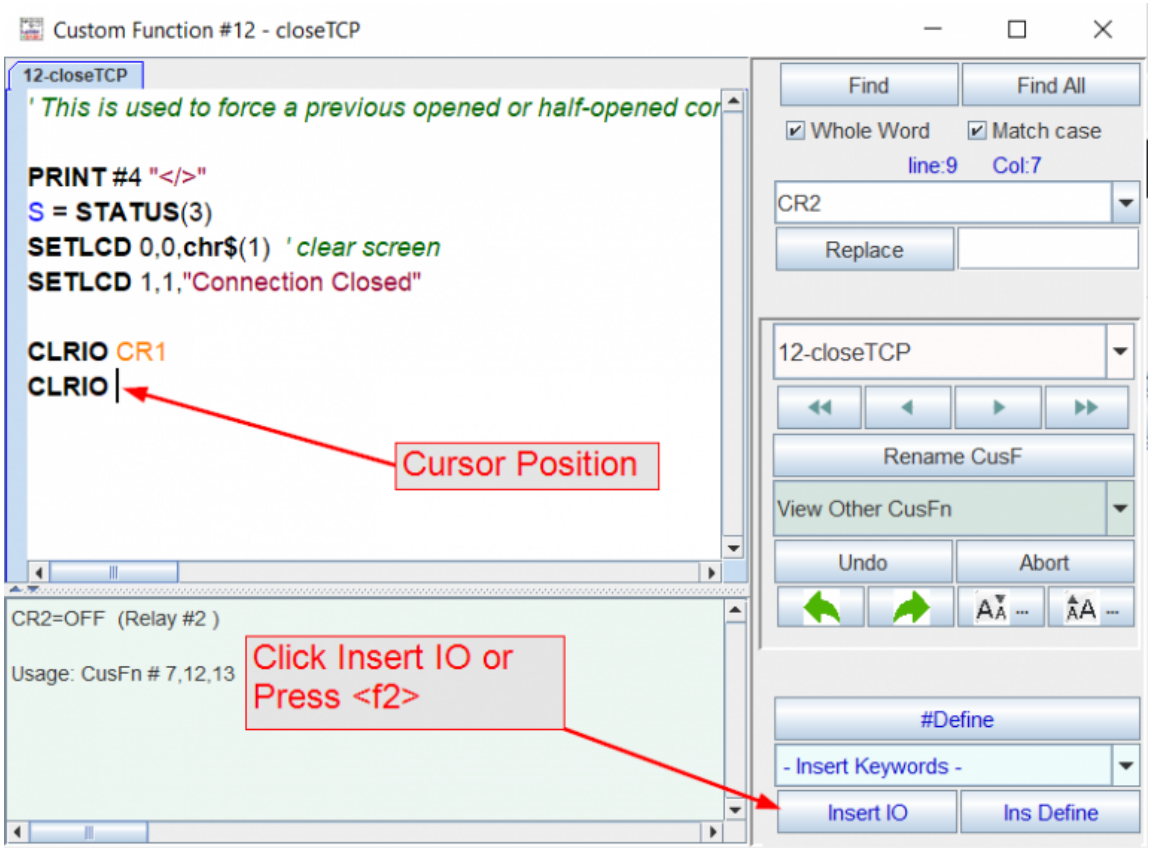
7.5.9 Insert Define Label

A new 

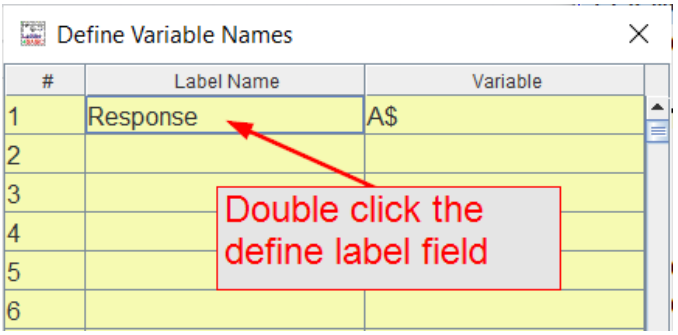
'Ins Define" button lets you pick a #Define name from the 'Define Table (https://docs.triplc.com/trilogi/#5459)' and insert into the custom function editor.

Insert a Define label into a custom function as follows:

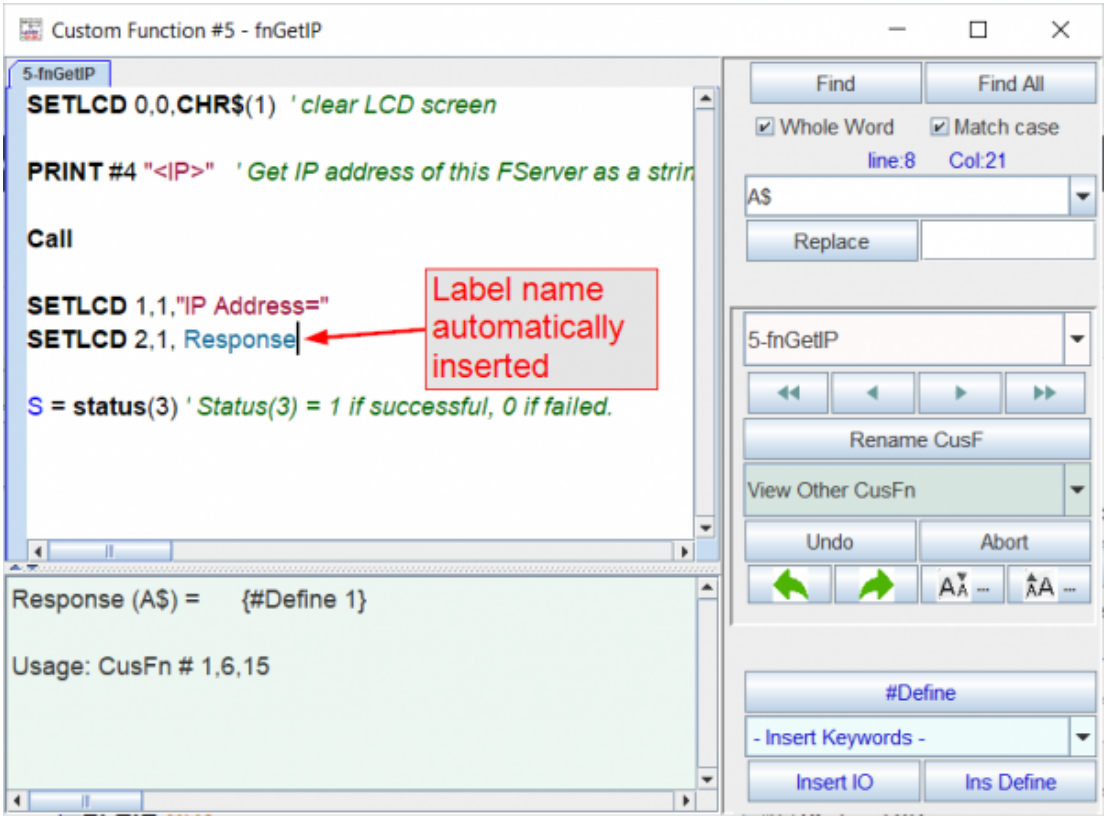
1. **Set the cursor** to the position that the Define label should be inserted.



2. **Select the desired label name** from the Define Table via mouse double-click on the highlighted label field.



3. The corresponding **label name will now appear** in the custom function.

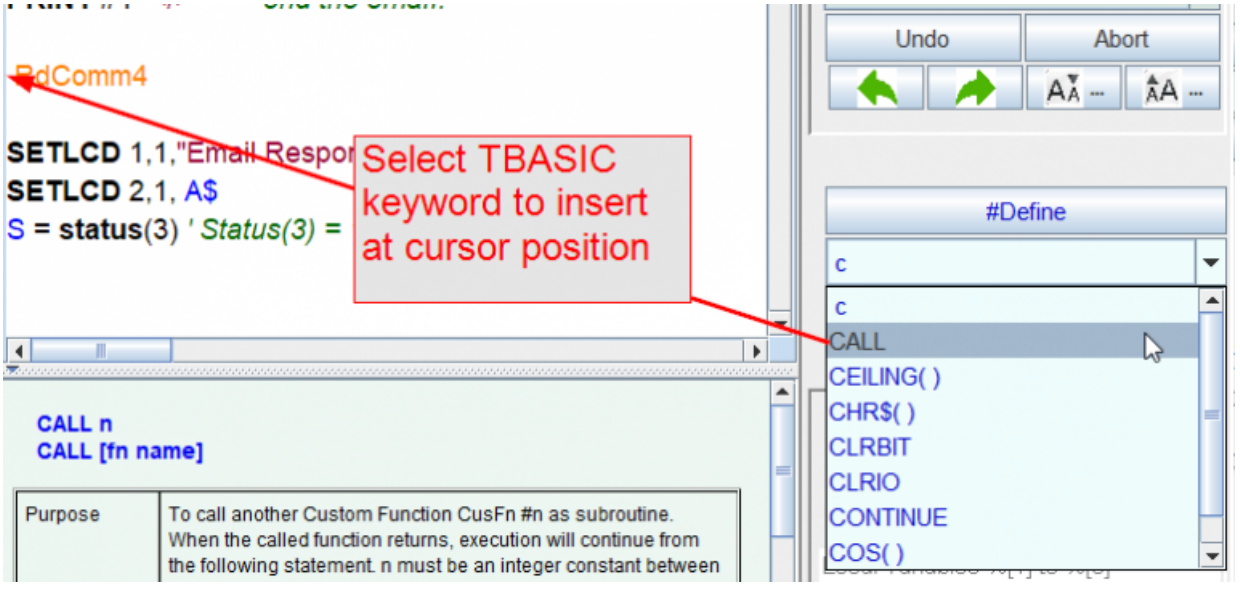


7.5.10 Insert Keywords

A new - Insert Keywords - 'Insert Keywords" field lets you pick a TBASIC command name from the list of 'TBASIC Keywords (https://docs.triplc.com/trilogi/#5725)' and insert into the custom function editor.

Insert a Keyword into a custom function as follows:

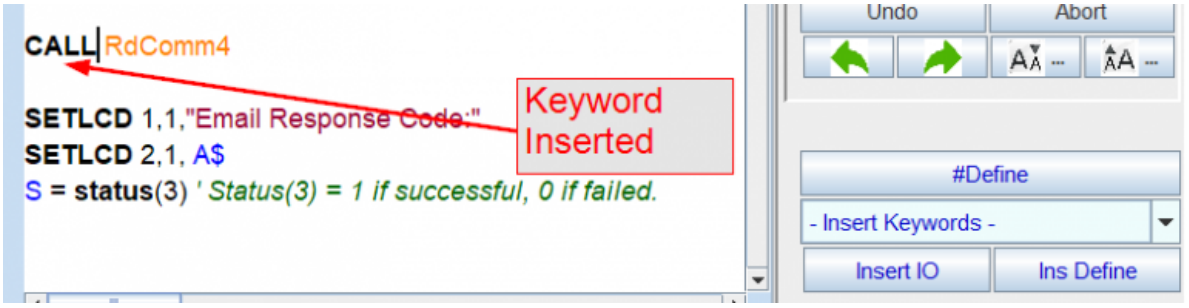
1. **Set the cursor** to the position that the Define label should be inserted.



2. Type part or all of the Keyword name in the - Insert Keywords - field. The list of Keyword names will automatically show up as long as the typed text matches part or all of an existing TBASIC

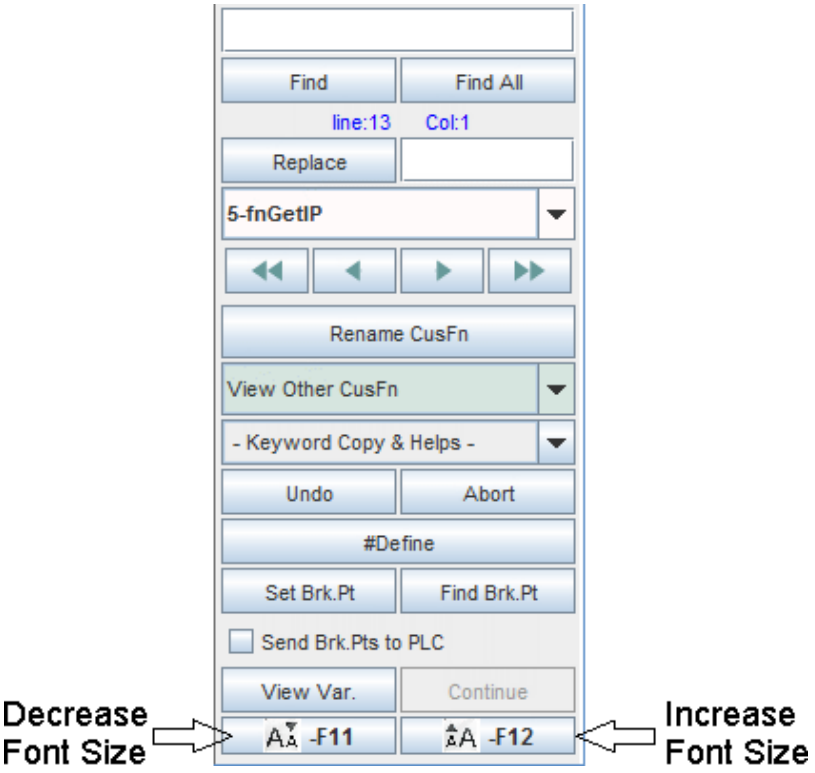
Keyword.

3. **Select the desired Keyword name** from the keywords drop-down menu via mouse click or keyboard Enter on the highlighted field.
4. The corresponding **label name will now appear** in the custom function.



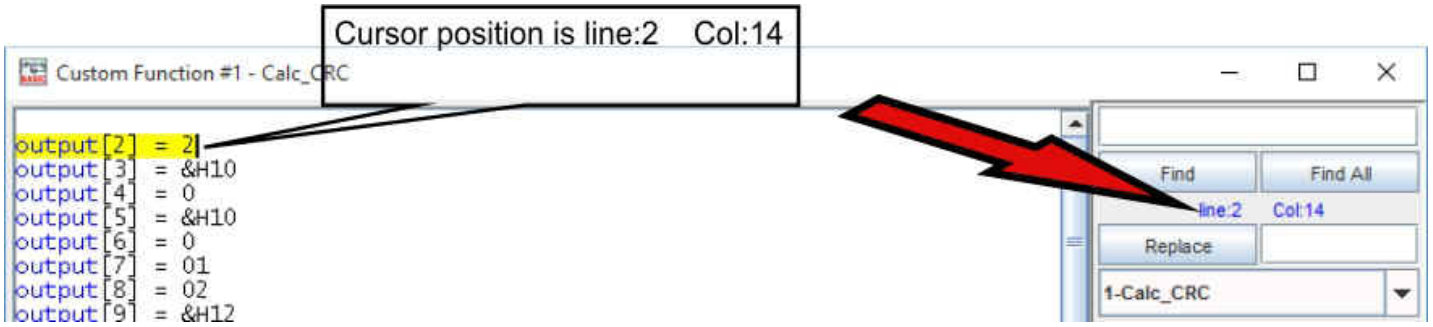
7.5.11 Increase/Decrease Font Size (F12 / F11)

Buttons have been added to the editor control panel on the right side that are used to increase or decrease the font size. The same keyboard shortcut keys that control the zoom level in the ladder logic editor will control the font size while the custom function editor window is active (F11 will decrease font size and F12 will increase font size). The zoom level and font size are not synchronized, so they can be controlled independently.



7.5.12 Cursor Position

In TRiLOGI 6.49 and higher the row and col of the cursor position within the function editor is display on the right editor panel.



7.5.13 Search / Find and Replace

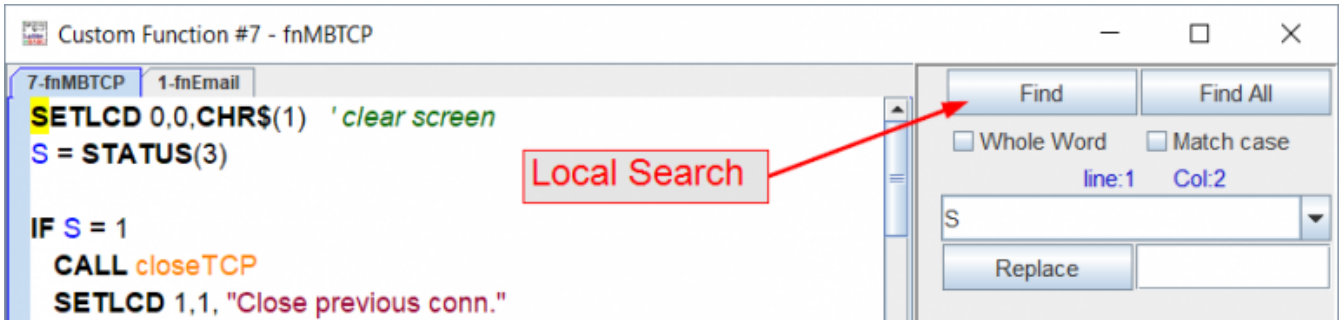
See section 7.6 Custom Function Editor Search (https://docs.triplc.com/trilogi/#5809)

7.6 Custom Function Editor Search

It is possible to search for a word or phrase in the current custom function (local search) or in all of the custom functions in your program (global search). Since version 6.7 / 7.4 it is possible to match a 'Whole Word' and/or 'Match Case' to improve the search function.

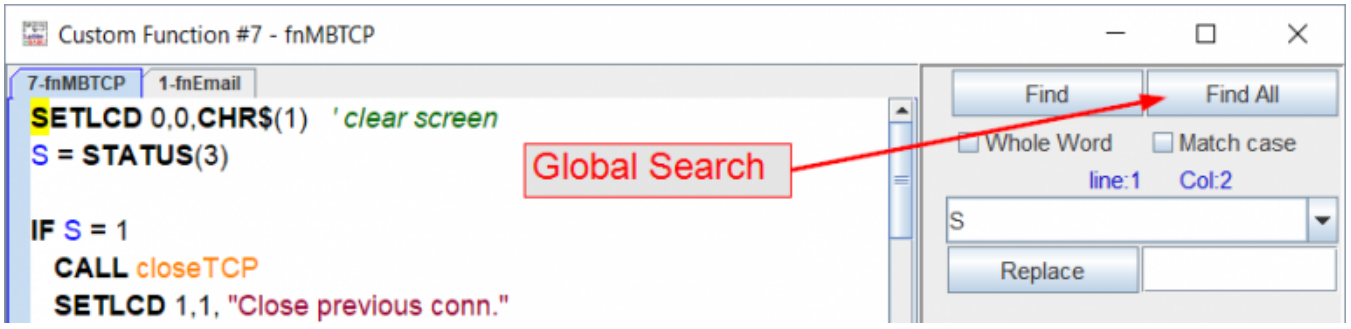
Local Search

To do a local search, simply type the text in the command line below the "Find" and "Find All" buttons and then click the "Find" button. If the text is found in the current custom function, it will be highlighted in the text editor as shown below. Also, the text "Find only in this CusF" will be displayed below the command line in the search area, indicating a local search. Each time the "Find" button is clicked, the next instance of the search text will be highlighted until the text cant be found anymore. At this point the message in the search area will change to "Text Not Found" and the next time "Find" is clicked, the first result will be highlighted again.



Global Search

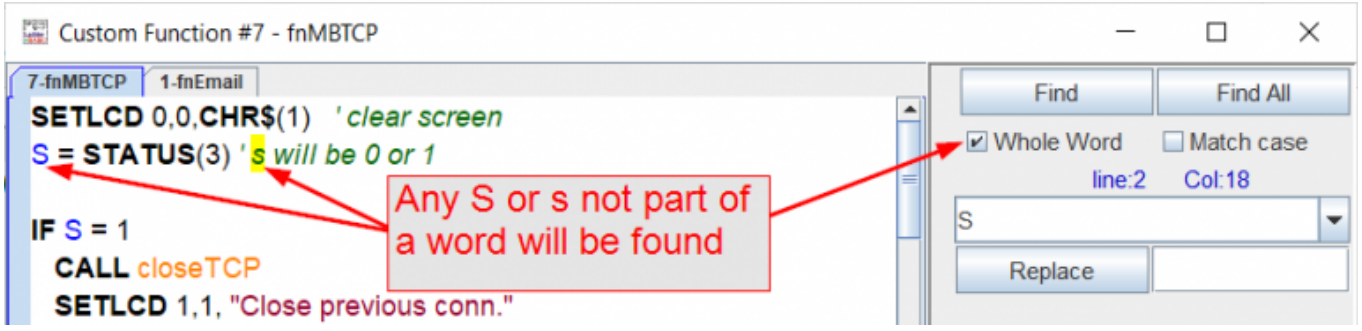
To do a global search, simply type the text in the same command line and click the "Find All" button. If the text is found in any custom function within the program, it will be highlighted in the text editor as shown below. Also, the text "Find in all CusF" will be displayed below the command line in the search area, indicating a global search. Each time the "Find All" button is clicked, the next instance of the search text will be highlighted until the text cant be found anymore (in any custom function). At this point the first result will be highlighted again. If no text matches the search text, the message in the search area will change to "Text Not Found".



NOTE: Highlighting of text found in the editor during a user initiated search or during compilation error tracking now works with JRE (Java Runtime Environment) 1.5 and 1.6

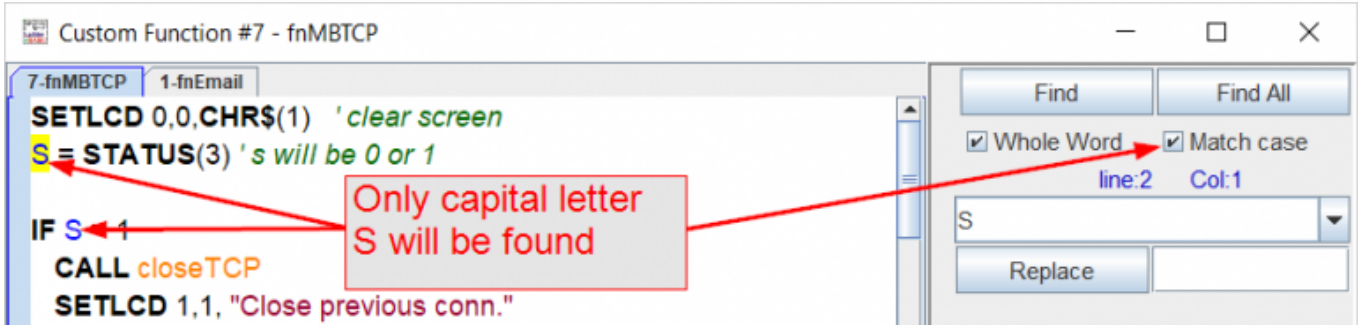
Match Whole Word

Check the 'Whole Word' box to search for an exact word or phrase. This is helpful to search for more common character sequences part of a larger word that needs to be found as an independent word.



Match Case

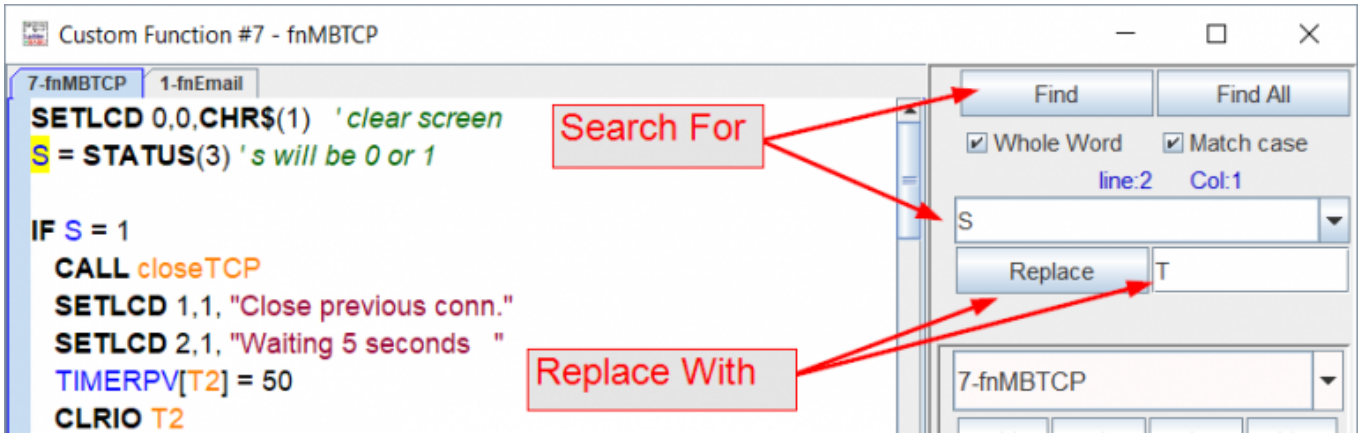
Check the 'Match Case' box to search for an exact combination of lowercase and uppercase letters, words, or phrases. This is helpful to find code that may frequently repeated in comments, but with a different case. It is especially useful for finding the A-Z variables when combined with 'Whole Word'.



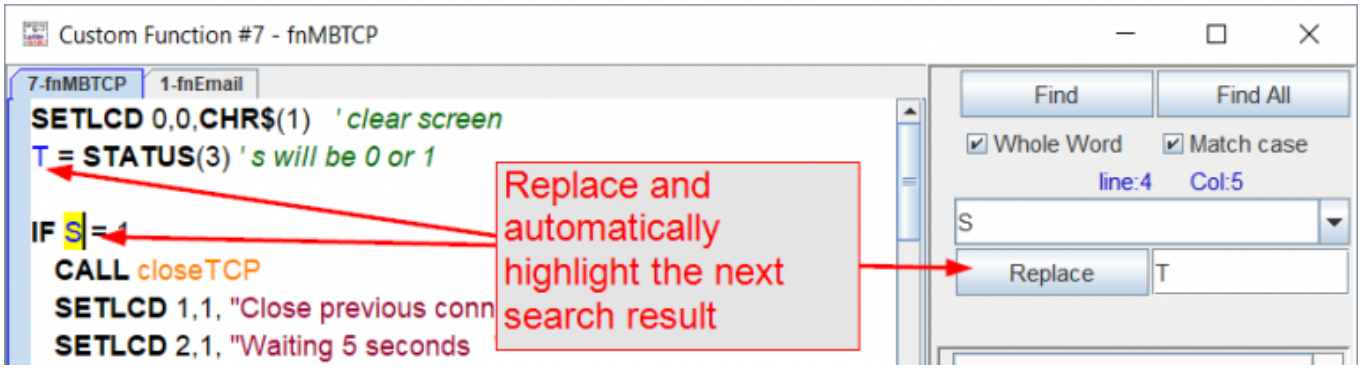
Replace Text

A replace feature is now available in the custom function editor below the Find/Find All text field in TRiLOGI version 6.49 and higher.

To replace text, enter the new text in the replace field and do a local or global search to find the text that is to be replaced.



When the searched text is found and highlighted as in the above figure, click on Replace and the text in the Replace field will overwrite the highlighted text. The next instance of the text being searched (if any) will be highlighted automatically.



Continue to click replace until all instances of the searched text have been highlighted and replaced. Replace will only replace and find text in the same function. Find All is still needed to search text outside the function that needs to be replaced.

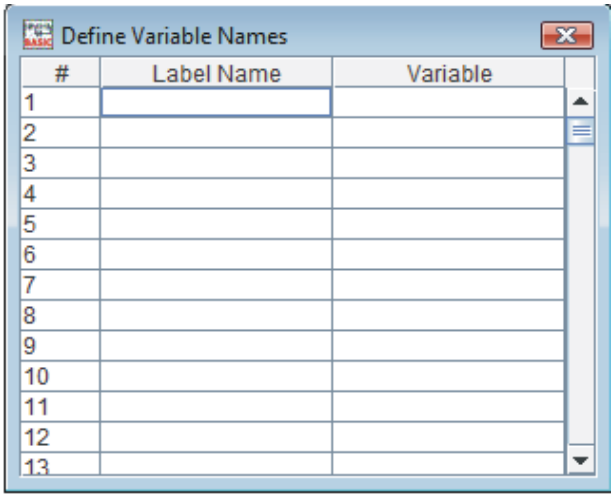
7.7 Custom Function Editor #Define Table

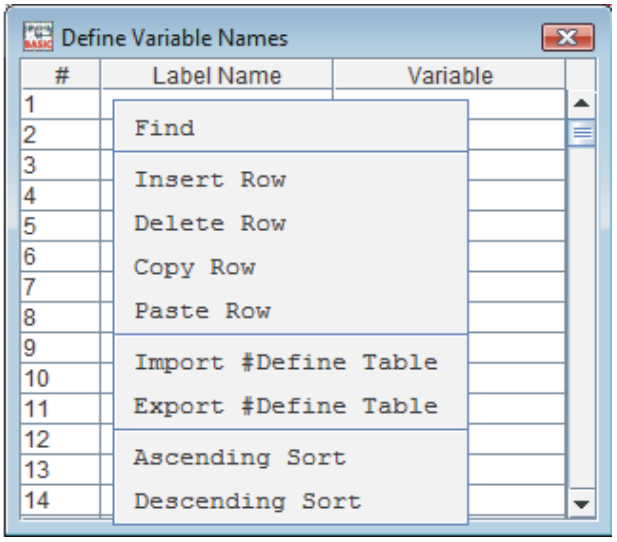
#Define Table 

In previous versions of i-TRiLOGI it was not possible to define variable names for the A-Z, DM[], A$-Z$, etc variables. However, in i-TRiLOGI versions 6.42 and up, a define button has been added to the custom function editor that allows users to create a table of variable and expression definitions for all integer/string variables available in TRiLOGI as well as constant values. The following sub-sections will describe in details how to use the #Define Table.

When the #Define button is pressed a table of definitions will open as a new window, as shown by the picture on the left below. Only the Label Name and Variable columns can be edited such that the name you would like to use should go in the Label Name column and the name of the variable (eg. DM[1], A, A$, etc.) should go in the Variable column. (See section 7.7.8 below for more examples of using the #Define table)

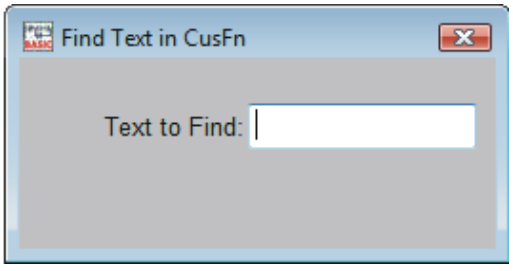
It is then possible to right-click on any of the cells to bring up the list of options shown by the picture on the right below.





7.7.1 Find

If you select the “Find” menu, you will see the following window popup and you will be able to type text in the field that will be searched for within the Define table.



7.7.2 Insert Row

A new blank row will be inserted above the row where the cell is highlighted when the mouse was right clicked. The highlighted row and all other rows below will be shifted down by 1 row and their row number will increase by 1.

7.7.3 Delete Row

The row that was highlighted when the mouse was right clicked will be deleted. All other rows below the highlighted row will be shifted up by 1 row and their row number will decrease by 1.

7.7.4 Copy Row

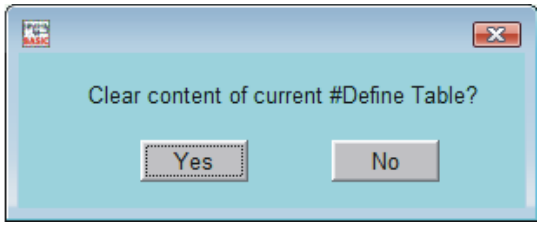
The row that was highlighted when the mouse was right clicked will be copied.

7.7.5 Paste Row

The row that was previously copied will be pasted to the row highlighted when the mouse was right clicked.

7.7.6 Import #Define Table

You will first be asked if you want to clear the content of the #Define Table and you will want to say 'Yes' to proceed with the import (say 'No' and do an export first if you want to save the current table).



An example define table exported to an Excel file (automatically exported in "Tab Delimited" format) is shown above.

Note: row #1, which must be included exactly as shown above in order for the file to properly be imported into the define table.

	A	B	C
1	#	#Define Name	Value
2	1	Name1	DM[1]
3	2	Name2	DM[2]
4	3	Name3	DM[3]
5	4	Name4	A
6	5	Name5	B
7	6	Name6	C
8	7	Number1	1
9	8	StringName1	A$
10	9	StringName2	Constant String
11	10	Expression1	D = A + B + C
12	11	Expression2	B$ = A$ + "Extension"
13	12	Expression3	If DM[1] = DM[2]:DM[3] = 1111:ENDIF

You will then need to browse your PC for the .txt (Tab delimited text file) that you have previously exported or created and are now importing. Once you select it, the define table will be populated with all the definitions.

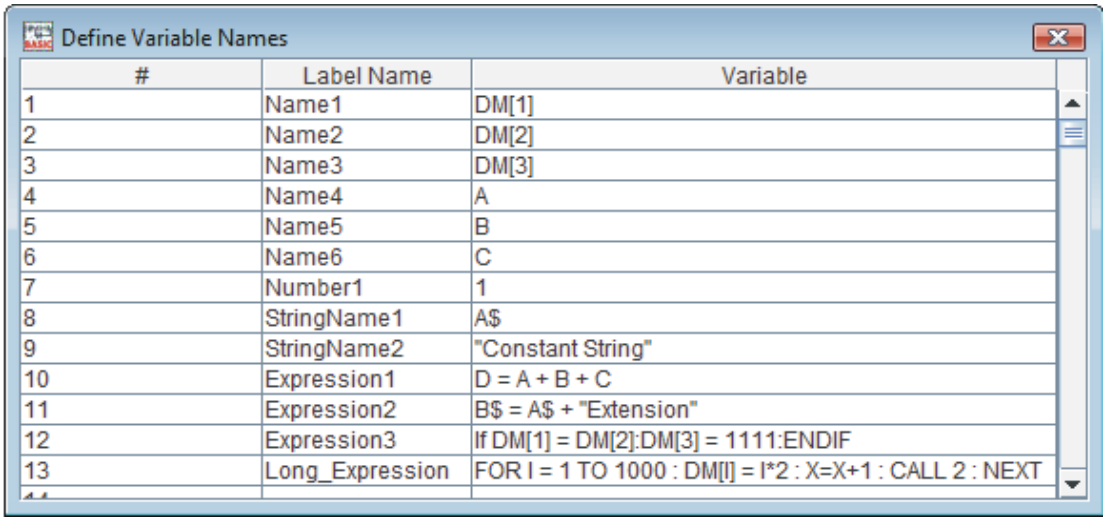
Note: If you are have manually created a "#Define" table using Excel spreadsheet program, you will need to use the correct format in your spreadsheet (shown above) and ensure that the file is saved in the "Tab Delimited format" (not the standard spreadsheet format) with a ".txt" file extension.

7.7.7 Export #Define Table

You can export all the items in the #Define table of a currently opened .PC6 file to the PC harddisk so that the these definitions can be imported into another .PC6 program. When you select "Export #Define Table" you will just need to name and save the file with ".txt" extension to your desired location. The data are stored in TAB-delimited format, which can be opened directly by a Microsoft Excel program for viewing or editing.

7.7.8 #Define Example

Here is an example of a possible Define Table :



#	Label Name	Variable
1	Name1	DM[1]
2	Name2	DM[2]
3	Name3	DM[3]
4	Name4	A
5	Name5	B
6	Name6	C
7	Number1	1
8	StringName1	A$
9	StringName2	"Constant String"
10	Expression1	D = A + B + C
11	Expression2	B$ = A$ + "Extension"
12	Expression3	If DM[1] = DM[2]:DM[3] = 1111:ENDIF
13	Long_Expression	FOR I = 1 TO 1000 : DM[I] = I*2 : X=X+1 : CALL 2 : NEXT

The following explanations will reference the above example define table.

Creating Variable Definitions – It is possible to define label names for any variable available in TRiLOGI. The following results will happen based associated program code :

TRiLOGI Program Code	**Result**
Name1 = 10	DM[1] would contain the value 10

Name4 = Name5 * 5	A would contain B x 5
DM[Name2] = Number1	DM[DM[1]] would equal 1

Creating Constant Definitions – It is possible to define label names for constant values. The following results will happen based associated program code :

TRiLOGI Program Code	**Result**
A = A + Number1	A would contain its current value plus 1
DM[Number1] = 2	DM[1] would equal 2

Creating Expression Definitions – It is possible to define label names for entire expressions or code snippets. The following results will happen based associated program code :

TRiLOGI Program Code	**Result**
Expression1	D would equal A + B + C
IF A = 1 Expression3 ELSE B = 1 ENDIF	If A equals 1, Expression3 will execute and the following code will run :If DM[1] = DM[2]:DM[3] = 1111:ENDIF If A does not equal 1, B will be equal to 1
Long_Expression	The following code would execute :FOR I = 1 TO 1000 : DM[I] = I*2 : X=X+1 : CALL 2 : NEXT

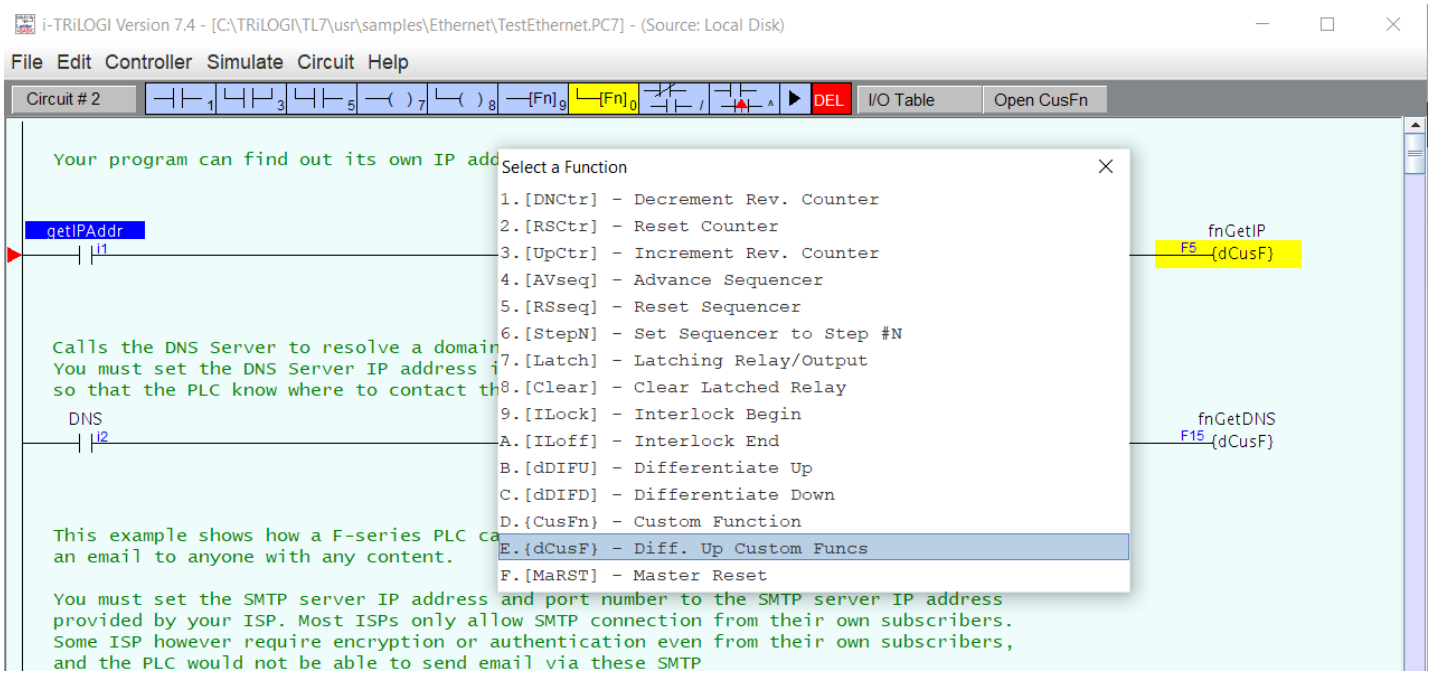
7.8 Using Custom Function

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:

7.8.1 Triggered by Ladder Logic Special function coil [CusFn]

A custom function may work the same way as any other special functions in the i-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 [Fn]9 or [Fn]0 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 : [dCusF] Diff. Up Custom Funcs”

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.

NOTE: It is now possible to right-click on a highlighted custom function and toggle between a [CusFn] and [dCusF].

7.8.2 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

1. 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.

2. 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.

3. 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.

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.

4. Like all ladder circuits, the relative position of the circuit which 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.

5. 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.

   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.

7.8.3 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 which 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 which can trigger a CusFn:

- Special Interrupt inputs

A Super PLC (such as FMD/Fx-Series) 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).

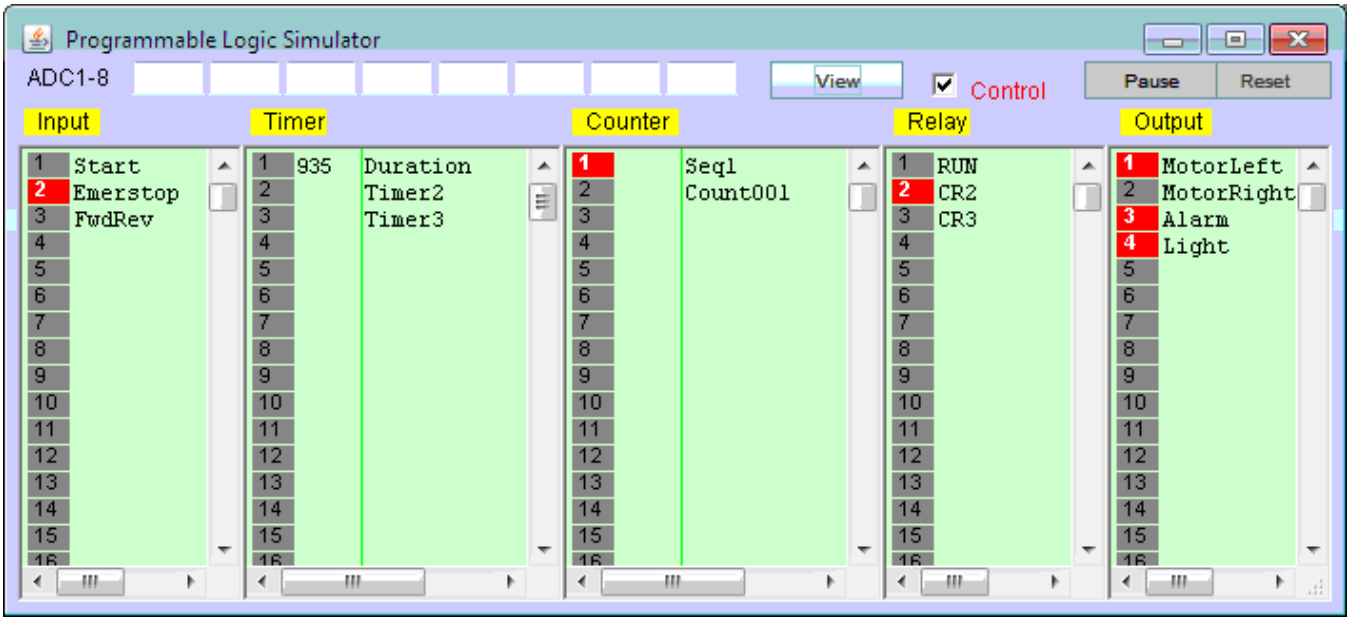
- High Speed Counters (HSC) Reach Target Count

A Super PLC contains some “High Speed Counter (https://triplc.com/TRiLOGI/Help/tbasic/hscdef.htm)” 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.

7.9 Simulation

7.9.1 Simulation Run of CusFn.

i-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 that simulate a LED being turned ON. You can pause the logic simulator at any time by pressing the <P> key or clicking on the [Pause] button. Likewise the simulator engine can be reset by clicking on the [Reset] button.

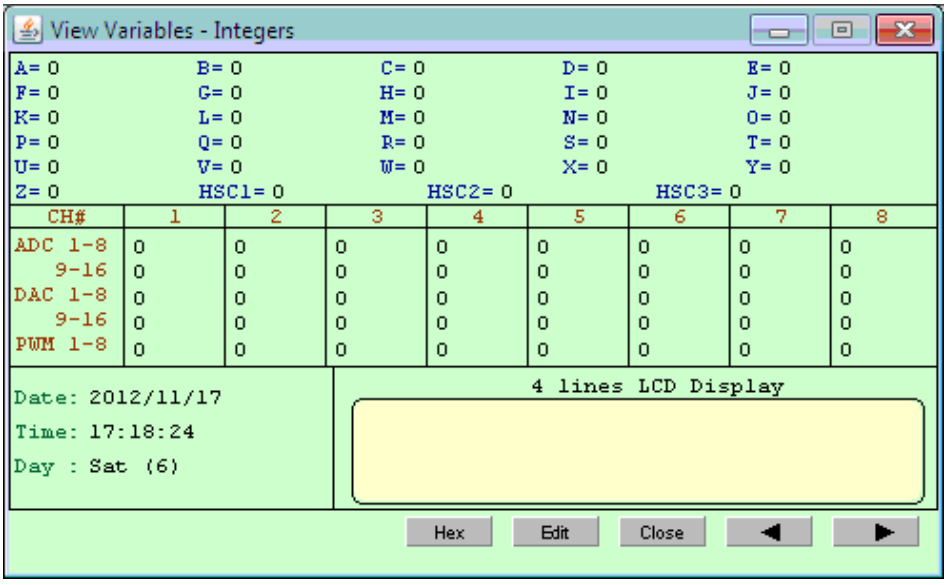
Simulation of ADC Inputs

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).

7.9.2 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:



a) 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 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].

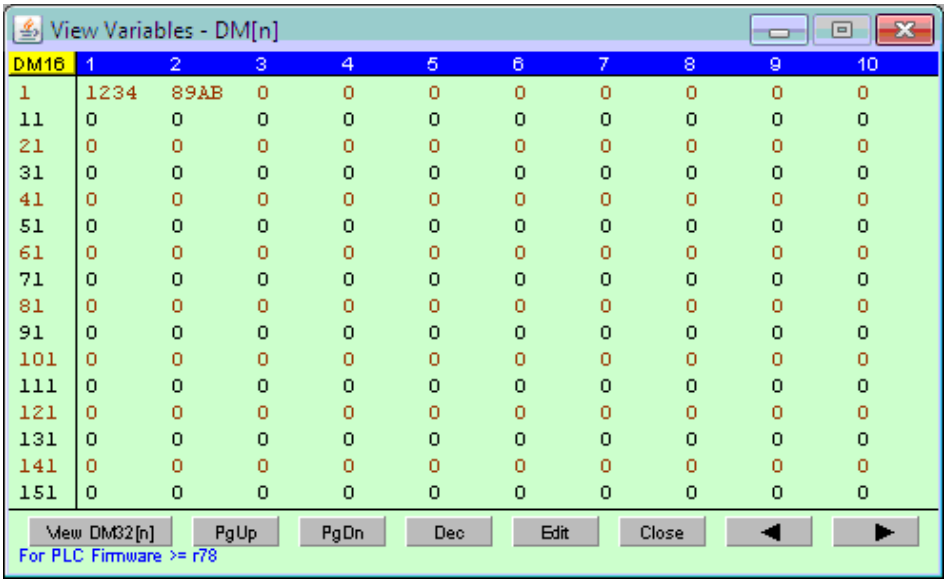
The present values of the first 3 high speed counters: HSC1 to HSC3 are also shown on this page. Note that ADC data for any particular A/D channel #n will only be shown if an ADC(n) function has been executed. Otherwise the ADC value shown on screen will not reflect the true current value of the ADC port.

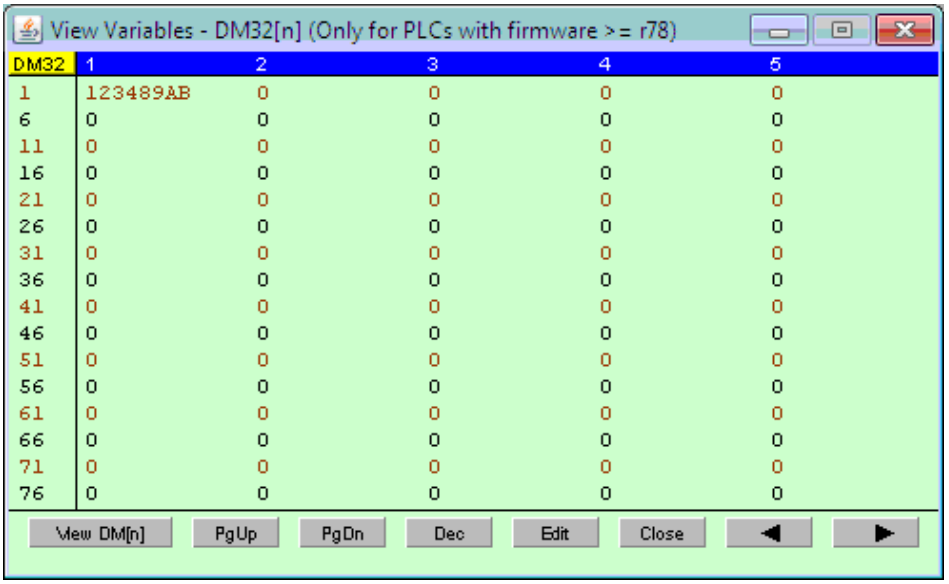
b) Data Memory Screen

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.

Starting from version 6.45 you can use two consecutive 16-bit DM variables as a single 32-bit DM32 variable in the following manner: DM[1] & DM[2] = DM32[1]; DM[3] & DM[4] = DM32[2].... DM[2n-1] & DM[2n] = DM32[n]. A new button "View DM32[n]" is added and when clicked, toggles the view of the DM variables between DM and DM32, as shown below. The range of DM[1] to DM[4000] are mapped to DM32[1] to DM32[2000].

Note: Although this version of TRiLOGI allows you to view any super PLC's DM[] as DM32[] during online monitoring, only new Super PLCs with firmware version r78 or later is able to actually use DM32[n] in its program. i-TRiLOGI will check the PLC firmware version number during program transfer and will not transfer TBASIC program that contains DM32[] variables to those PLC with < r78 firmware.





c) 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.

d) 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 they are more commonly used by programmers to identify the bit patterns in these variables.

7.9.3 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], DM32[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;
DM32[100]=&H12789ABC
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.

7.9.4 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.

7.10 Monitoring of TBASIC Variables

If you execute the “On-Line Monitoring/Control” command from the “Controller” pull-down menu, i-TRiLOGI 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 as described in Section 7.9.2. 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 i-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 i-TRiLOGI programming environment is really what really sets the M-series/F-Series PLCs apart from many other PLCs.

7.10.1 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. The PLC can also be halted by pressing the <P> key. A halted PLC can subsequently be released from the halted mode by pressing the <P> key again. You can also change any internal variable data using the “Edit” button on the View Variable screen.

7.10.2 Using LCD Display for Debugging

You should take advantage of the built-in LCD display port of the PLC 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.

7.11 Debugging With PAUSE/Breakpoints

7.11.1 Using the TBASIC PAUSE Statement

When debugging your program it is often easier to identify program bugs by pausing the PLC at a certain execution point so that you can examine the data in the variables to understand why the program does not behave the way you expect it to. TBASIC offers the "**PAUSE**" statement which is always available in all versions of i-TRiLOGI software and supported on all firmware versions of the Super PLCs.

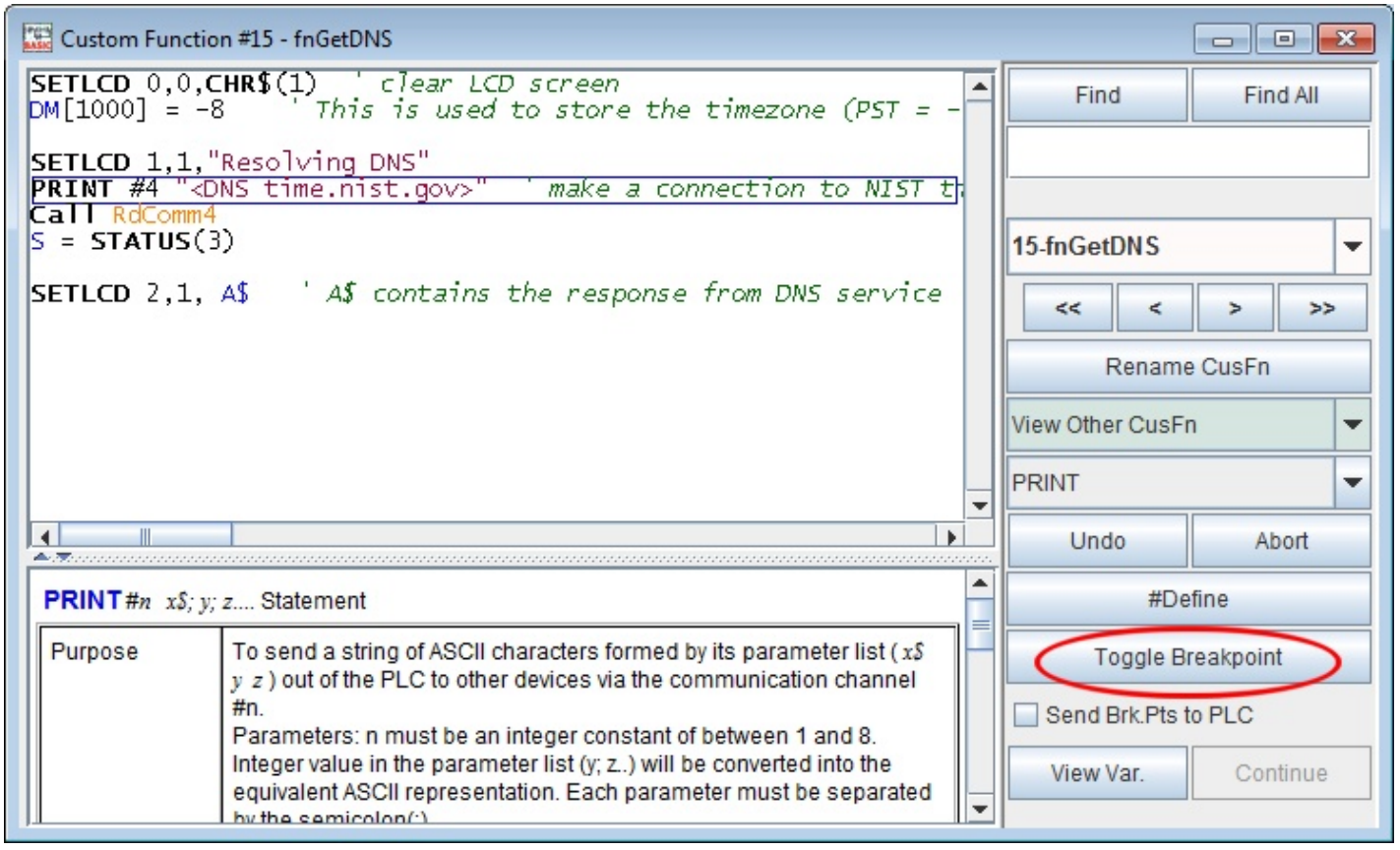
When the program reaches a **PAUSE** statement it will enter a PAUSE Mode and you can then examine the variable in the PLC. You can add as many **PAUSE** statement as you need in different part of the program to halt the program execution. However, since the software does not show you exactly where the program is halted except to tell you that a **PAUSE** statement is known to have been executed in a particular function number, and if you have more than 1 **PAUSE** statement in a particular custom function then you will need to put some markers so that you know where the program has stopped. The marker could be displaying of some text on the LCD (e.g. "Pause @21") that a certain PAUSE point has been reached or you can assign a certain value to a reserved variable and the value corresponds to the location of your PAUSE statement. This unfortunately means that you need to modify your program in order to break the PLC program at a certain location.

Although it may be only a small effort to add and delete **PAUSE** statements in the TBASIC program and then re-run the test immediately during simulation, if you don't use the simulator but instead debug your program directly on the PLC, then you must re-transfer the program to the PLC every time you add or delete a **PAUSE** statement. This can be time consuming if your program is large or you have a very long custom function. So the new **BREAKPOINT** features described in the next section can be good a time saver.

7.11.2 Using Breakpoints

Starting from i-TRiLOGI version 6.45 a new "Breakpoint" feature has been added to the program so that debugging can be more easily performed without the need to modify and transfer the program to the PLC. The Breakpoint feature is supported on all Super PLCs with firmware version r78 and above. The break point feature is always available to the simulator in this software version but the program does not let you send the program break points to a PLC with firmware older than r78.

To set a break point in any part of the TBASIC program, open up the custom function and place the cursor on the line where you want the program to pause when it reach the line. Then click on the "Toggle Breakpoint" button to set the break point. A breakpoint that is set will be shown up as a blue rectangular box around the line, as shown below:



If you click on the “Toggle Breakpoint” button again the blue box will disappear, which means the breakpoint has been cleared.

In TRiLOGI 6.49 and higher it is possible to find breakpoints that have been set by clicking on the  Find Breakpointbutton. This will iterate through all of the break points that have been set in the program.

Note:

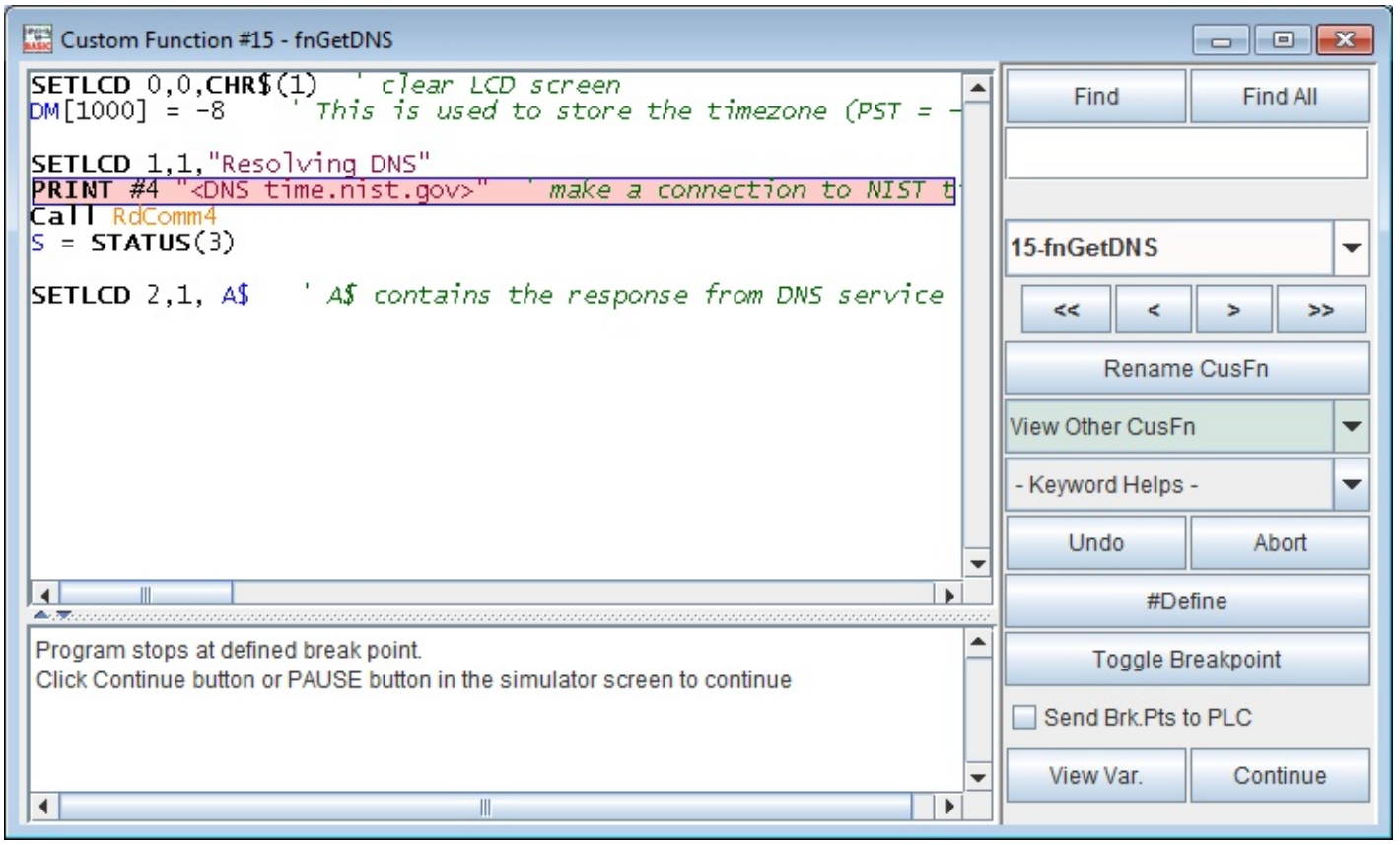
1. You can define up to a maximum of 8 breakpoints in a program.

2. Comment/Remark lines and blank lines are not compiled and breakpoint should not be set on any of these lines.

3. The breakpoints that you’ve defined are saved along with the program. Each breakpoint can be cleared individually by placing the cursor on the breakpoint line and the click the “Toggle Breakpoint” button. If you wish to clear all breakpoints in the program you can click on the “Edit” pull down menu and select “Clear All Breakpoints”.

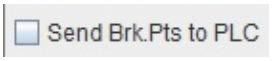
7.11.3 Debugging Using Breakpoint on Simulator

With a break point set in the custom function and if you run the program using the simulator, when the simulator runs to the line where a breakpoint is set the program will pause and the custom function where the breakpoint is defined will be opened with the breakpoint line highlighted as shown in the following picture:



At this point you can examine the variable data by clicking on the “View Var.” button, which will open up the “View Variable” screen. You can release the program from its PAUSE state by either clicking on the “Continue” button or by clicking on the “Pause” button on the on-line monitoring screen. The program will continue execution until it hits the next breakpoint. Note that when the program is halted at a breakpoint you can define more breakpoints so as to track the program execution. You can remove old breakpoint at any time if you run out of the maximum 8 breakpoints limit.

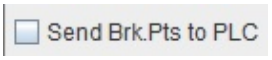
7.11.4 Debugging Using Breakpoints on PLC

Notice the 

checkbox below the “Toggle Breakpoint” button? If you wish to halt the actual PLC (not the simulator) when the program runs to the breakpoint, you can send all your defined breakpoints to the PLC by clicking on this checkbox. i-TRiLOGI will connect to the PLC and check if the program in the PLC is the same as the currently open program by verifying the checksums. If the program is current then it will transfer all the defined break point to the PLC. With the checkbox shown as “checked” any subsequent addition or deletion of breakpoints will be immediately transferred to the PLC

When the PLC program reaches the defined breakpoint it will stop execution and the PAUSE LED on the PLC will light up. If you then go online monitoring the program will open up the custom function and highlight the breakpoint line where the breakpoint was encountered.

Note that the breakpoint sent to the PLC is volatile. If you reboot or power-on reset the PLC all the breakpoints will be disabled. This ensure that production PLC will not halt at any breakpoint that you forget to clear.

To manually clear all the breakpoints, you can un-check the 

checkbox and the i-TRiLOGI program will disable the breakpoints in the PLC. If the checkbox was already unchecked but for some reasons the PLC breakpoints have not been cleared yet, then you will need to first check the checkbox to transfer the breakpoints to the PLC and then followed with an uncheck to disable them.

If a PLC stops at a breakpoint that is not defined in your current file then the i-TRiLOGI program will give you an alert and will not be able to display the line where the program break.

7.12 Error Handling

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

7.12.1 Syntax Error

i-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@triplc.com (mailto:support@triplc.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.

7.12.2 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, i-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!
STEPMOVE 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 8 - TBASIC Language Reference

8.1 Statement, Function & Delimiter

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.

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 seek we use a mix of upper and lower case characters in this manual.

> i-TRiLOGI 6.x only supports a subset of built-in functions available on i-TRiLOGI 7.x. In addition, i-TRiLOGI 7.x provides a mechanism for user-created custom function to be CALLed by other custom functions just like a built-in function. The caller can pass a list of parameters to the function enclosed between ‘(‘ and ‘)’. The called function can also return a variable to the caller. We will describe this in greater details in TL7 Addendum.

DELIMITER

A TBASIC program consists of many statements. Each statements are usually separated by a different line. The new line therefore acts as a "delimiter" which separate 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
```

8.2 Integer Data & Operators

The TBASIC compiler in i-TRiLOGI 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 (two adjacent 16-bit DMs can also form a 32-bit DM32[] variable). 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.

a) 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 “&HFFFFFB2E” 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 &HFFFFFB2E. 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 : ENDIF
b) IF DM[1] <> &HFFFFFB2E” CALL 5: ENDIF
```

b) 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.

Specific examples with each variable type are shown below.

TBASIC supports the following integer variables:

- 26 Integer variables A, B, C….Z which are 32-bit variables. Note that the variable name is a single character by default; however, as mentioned above, you can append a comment to the variable. EG: A_temperature, A_distance – both refer to the same variable A
- 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. A comment can also be appended to DM[] variables as follows: DM[1]_Current_status_of _product1 – refers to variable DM[1].

- DM[1] to DM[4000] can also be used as 32-bit variables **DM32**[1] to **DM32**[2000] in the following manner:

```
DM[1] and DM[2] == DM32[1],
DM[3] and DM[4] == DM32[2]
   ...
DM[2N-1] and DM[2N] = DM32[N].
```

  If you require lots of 32 bit variables it may be simpler to use only the DM32 and not the 16-bit DM. If you mix the use of both DM and DM32 then you need to manage the memory properly to ensure that they don't overwrite each other memory space.The simulator fully support the use of DM32 variables in any expression. Wx100 and all SmartTILE-Fx based PLCs support DM32[n] variable. For FMD or Nano-10 PLCs only those with firmware > r78 supports DM32 variables. PLCs with older firmware cannot directly use DM32 in the program.

```
E.g.   DM32[100] = DM[1] * A
```

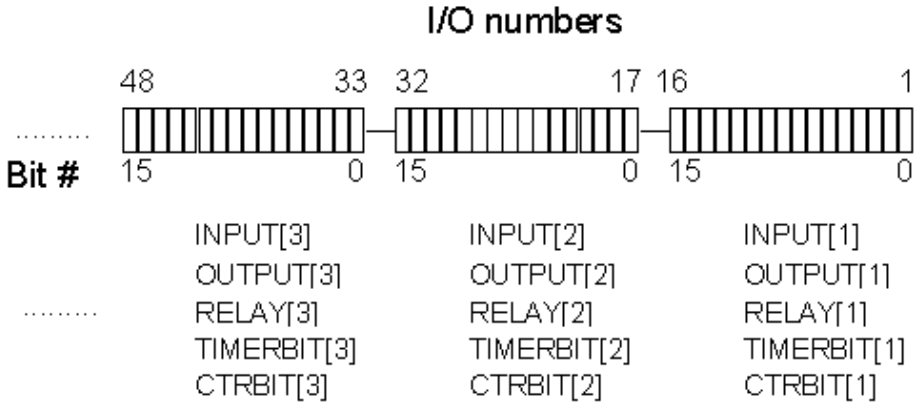
- System variables. These are special integer variables that relate to the PLC hardware, which will be described in the next section.

c) System variables:

NOTE: All of the following System Variables can have comments appended to them with the same format as described in section 5 (Integer Variable Comments).

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 and F-Series PLCs support High Speed Counters (HSC), which 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.

Special Variables – Legacy variables used 4 x special 16 bit integer variables

EMEVENT[1] to EMEVENT[4] – emEvent[1] is also used for email purpose. 16 x 16-bit integer variables:
EMINT[1] to EMINT[16] 16 x 32-bit integer variables: EMLINT[1] to EMLINT[16]

d) 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)

For detailed explanation of these functions please refer to the next chapter: “TBASIC Language Reference”

e) Hierarchy of Operators

The hierarchy of operators represent the priority of computation. Eg. X = 3 + 40*(5 – 2). The compiler will generate codes to compute 5 – 2 first because the parentheses has the higher hierarchy, the result is then multiplied by 40 because multiplication has a higher priority then 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

f) Integer Variable Comments

With i-TRiLOGI version 6.2 and above, you can now attach comments to any variable/register name of any length to make program easier to read. The compiler will ignore any alphanumeric characters (A to z, 0 to 9 and '_') that are attached behind the variable name following an underscore character "_". It is important to understand that by appending these comments to variables, no new variables are being created.

For Example: "X_Some_Integer" corresponds to variable "X". An integer variable (such as DM[1]) can have different comments each time it is referenced in the same program because the compiler ignores the comments anyways. Although, in most cases it may be best to limit a variable to one comment to avoid potentially overwriting data since no new variables are actually created, as mentioned above. For Example: If an integer variable, DM[1], was named "DM[1]_one_integer" in one part of a program and named "DM[1]_two_integer" in another part of the program, then they will still both refer to DM[1]. If each variation has different integer data, then the data that was stored in the last variation that was updated will be the data in DM[1]. Then if the previous variation is accessed, it won't contain the data that was originally stored in it.

> We recommend you consider using the #Define Table (see next section) or #DEFINELOCAL command to assign easy-to-read names to variables which work much better than using this method of identifying a variable. We continue to support this legacy feature to maintain compatibility with some exiting user's program that may still be using this method of naming variables.

g) Variable Define Table (NEW!)

With TRiLOGI version 6.42 and higher, it is possible to create a table of variable/constant/expression definitions. Please refer to Section 7.3 (https://docs.triplc.com/tl74/#5459) for details and examples.

h) Using #DEFINELOCAL keyword within a Custom Function

Starting from i-TRiLOGI version 6.47, besides using the #Define table you can also define label names that only has local scope by using the **#DEFINELOCAL**labelname = expression. E.g

```
#defineLocal RoomTemperature = DM[100]
```

You can then freely use the label name “RoomTemperature” within this custom function in place of DM[100] so that the program is more readable.

The **#DEFINELOCAL** keyword is useful if you only want the labels to be valid in the current function. It could make it easier to create a library function that you can re-use in other applications without exporting and importing the #Define table. Note that the compiler will first process any label names defined using the **#DEFINELOCAL** before it processes the #Define table.

However, unlike the #Define table the program does not conduct a check on whether you have duplicate **#DEFINELOCAL** on the same label, or whether a label so defined is a reserved keyword. So the programmer must exercise care when using **#DEFINELOCAL** to avoid ending up with hard-to-debug compilation errors.

8.3 String Data & Operators

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 r44 and above and all F-Series PLCs, 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 i-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 Operators: 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$),

4. String Variable Comments

With i-TRiLOGI version 6.2 and above, you can now attach comments to any variable name of any length to make program easier to read. The compiler will ignore any alphanumeric characters (A to z, 0 to 9 and '_') that are attached behind the variable name following an underscore character "_". It is important to understand that by appending these comments to variables, no new variables are being created. For Example:

“A$_Some_String” corresponds to “A$” (or $$[1]).

A string variable (such as A$) can have different comments each time it is referenced in the same program because the compiler ignores the comments anyways. Although, in most cases it may be best to limit a variable to one comment to avoid potentially overwriting data since no new variables are actually create, as mentioned above.

For Example: If a string variable, A$, was named “A$_one_string” in one part of a program and named “A$_two_string” in another part of the program, then they will still both refer to A$. If each variation has different string data, then the data that was stored in the last variation that was updated will be the data in A$. Then if the previous variation is accessed, it wont contain the data that was originally stored in it.

> We recommend you consider using the #Define Table (see next section) or #DEFINELOCAL command to assign easy-to-read names to variables which work much better than using this method of identifying a variable. We continue to support this legacy feature to maintain compatibility with some exiting user’s program that may still be using this method of naming variables.

5. Variable Define Table

This new feature allows you to define an acceptable name for any variable, constant or even an entire expression. See #Define Table (https://docs.triplc.com/tl74/#5459) for details.

8.4 Floating-point Data & Operators

8.4.1 Floating-point Number Representation

i-TRiLOGI version 7.x (TL7) uses IEEE, 32-bit Single Precision format to represent all float numbers. For more information on how single precision float are encoded into 32-bit of data, please refer to the following link:

http://en.wikipedia.org/wiki/Single-precision_floating-point_format (http://en.wikipedia.org/wiki/Single-precision_floating-point_format)

However, in-depth knowledge of how float are actually encoded are not required for TL7 users since TL7 automatically handles the floating-point data conversion to and from its binary representation. All you need to know are the range of numbers that an IEEE single precision floating-point number can represent, as follow:

Minimum positive value	1.18×10^{-38}
Maximum positive value	3.4×10^{38}
Minimum negative value	-3.4×10^{38}
Maximum negative value	-1.18×10^{-38}
±Zero	Yes
± Infinity	Yes
NaN (not a number)	Yes

Special formats are used to represent special numbers such as zeros there are positive zero and negative zeros but they compare as equal), infinity and NaN. These are supported by TL7.

> **Note**: Having the basic knowledge of how the float data are actually encoded in 32-bit binary number is useful when you need to transport the numbers to and from external devices via serial or Ethernet communication. TL7 supports special function "**Float2Bits(float)**" to convert a float into its 32-bit IEEE format and the function "**Bits2Float(integer)**" to decode a 32-bit IEEE representation (of a single precision float) back into a floating-point number. This makes it very simple to send and retrieve float data via Modbus and Hostlink commands that traditionally support only 16-bit and 32-bit integer data.

8.4.2 Floating-point Variables

TL7 added the following new Floating-point variables:

Floating-point variables	Type
A# to Z#	Global variable
FP[1] to FP[1000]	Global variable
%[1] to %[9]	Local variables*

- Support of “Local variables” is a completely new addition to TL7 that allows up to 9 floating-point parameters to be passed to and used within a custom function. Please refer to Section 1.3 in this manual for more detailed explanation of local variables.
- Support of “Local variables” is a completely new addition to TL7 that allows up to 9 floating-point parameters to be passed to and used within a custom function. Please refer to Section 1.3 in this manual for more detailed explanation of local variables.

8.4.3 Floating-point Constant

Float constants can be entered into TBASIC using the popular floating-point format as shown in the following examples:

Number	Format 1	Format 2	Format 3	Format 4
123.4567	123.4567	1.234567E2	1.234567e+02	+1.234567e+02
-0.0004567	-0.004567	-4.567E-4	-4.567e-4	-4.56700e-04
12345	12345.0	1.2345E4	+1.2345e04	1.2345e+04
0	+0.0	-0.00		

It is important to note that if you want to use a whole number as a floating-point number in an expression, you should append it with a decimal point and a 0. i.e. 123 should be entered as 123.0. The reason being that certain operators such as “/” (divide by) operate differently on floating-point numbers than that on integers.

E.g.

```
A# = 11.0/2.0 = 5.5
A# = 11/2 = 5
```

In the second expression, 11/2 is treated as an integer divide between two integers 11 and 2 and the result is an integer 5 without any fractional part. When the result is finally assigned to A# it has already lost its fractional part. But as long as one of the two operands is a float (such as 11.0 or 2.0) the divide operator will perform floating-point division and return a float.

Likewise,

```
A# = 345/0 => Run time error: Divide by Zero
A# = 345/0.0 = +Infinity
```

Special Floating-point Constant

Due to their relatively rare use in control system, TL7 does not create special keywords to represent special float numbers such as +infinity. If you really need to use such special numbers in your program you can use the Bits2Float( ) function to convert their 32-bit integer representation into these special numbers, as follow:

+ Infinity E.g. A# = Bits2Float(&H7F800000)

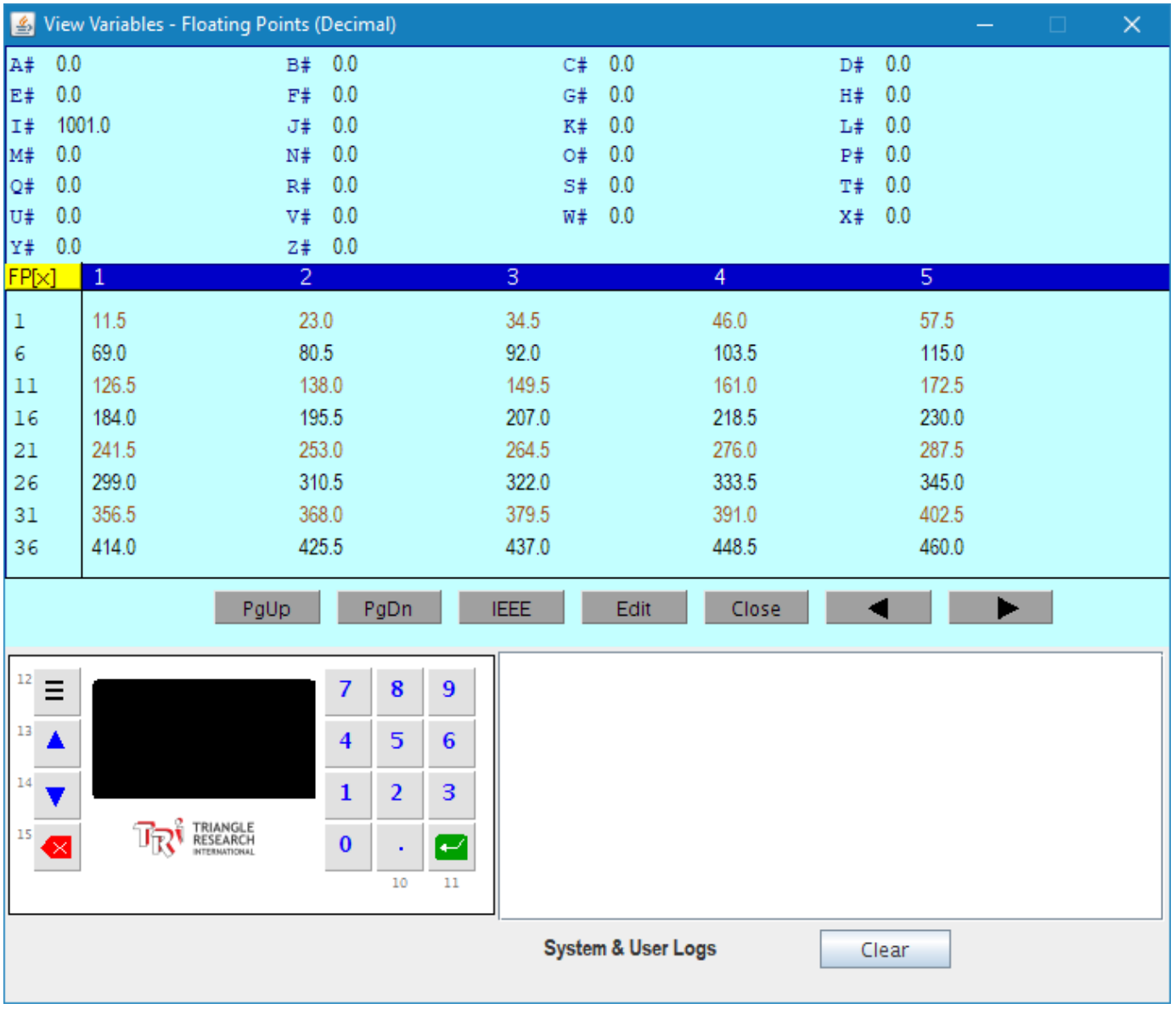
– Infinity E.g. B# = Bits2Float(&HFF800000)

NaN E.g. C# = Bist2Float(&H7FC00000)

TL7 however thus support the keyword **NaN** (not a number) since this may be used to return to a caller to alert the caller that it has supplied invalid parameters being supplied by the caller.

8.4.4 Viewing of Floating-point Variables

TL7 added an additional screen to the “View Variable” screen of the simulator/online monitoring screen to display all the new floating-point variables A# to Z# as well as the floating-point array FP[1] to FP[1000] as shown below (this screen is not available in TL6 since TL6 does not support floating-point operation at all):



You can click on the “PgUp” or “PgDn” button on the screen or use the “PgUp” and “PgDn” keys on your keyboard to scroll through all the pages to view all 1000 FP[ ] variables.

The third button: “IEEE” – let you switch the view of the values of these floating-point variables between human readable decimal format and their actual IEEE 32-bit single precision format. In IEEE mode these values are displayed as 8 characters of hexadecimal digits.

If you are interested in seeing how the floating-point values are represented in IEEE format you can use the following online calculator to verify the data you observe on the screen:

http://www.h-schmidt.net/FloatConverter/ (http://www.h-schmidt.net/FloatConverter/)

8.4.5 Floating-point Operators

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

i) Assignment Operator (=):

A floating-point variable (FP[], A# to Z#, %[1] to %[9]) may be assigned a numeric value or the result of a numeric expression using the assignment operator ” = “. E.g.

```
A# = 1000 X = H*I+J + len(A$)/FP[5]
```

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)

A numeric expression using the above arithmetic operator can include both integers and floating-point numbers. As long as one of the two numbers is a float the integer number will be converted to float and then the floating-point operator will be used and the result is a float. However if both operands of an arithmetic operator are integers then the integer operator will be used and an integer number will be returned.

iii) 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#<>0.0 **OR** B# >=1000

> The programmer should bear in mind that a decimal floating-point number in a computer is at best an approximation of the real data and not an exact number. Any arithmetic operation carried out on a number could result in rounding or truncation errors of the resulting. So comparison of two numbers using the “Equal” operator may not always work as expected. For example:

```
A# = 0.0
FOR I = 1 to 1000
   A# = A# + 0.1
NEXT
IF A# = 100.0 THEN
    SETLCD 1,1, "True"
ELSE
    SETLCD 1,1, "False"
ENDIF
```

At first glance you may expect that the result should be “True” since adding 0.1 to A# 100 times should means A# = 100.000 after the program is run? However, when you execute this program you may be surprised that the result is “False”. Further examination of the execution result you can see that after adding 0.1 to A# 100

times, A# actually contain the value “99.999”, which is quite close to 100.0 but not exactly. The error is due to cumulative truncation errors resulting from floating-point addition. Hence you should be careful and try to avoid using the “Equal” operator to compare two floating-point numbers directly.

8.4.6 Built-in Floating-point Functions

There are a number of built-in floating-point functions such as trigonometric functions (SIN, COS, TAN) , Logarithmic functions (Ln and Log10), power and square root function as well as exponential (ex) functions. The following are a quick reference list of all the built-in floating point functions available only in TL7.x:

ARCCOS (https://triplc.com/TRiLOGI/Help/tbasic/arccos.htm)	ARCSIN (https://triplc.com/TRiLOGI/Help/tbasic/arcsin.h
EXP (https://triplc.com/TRiLOGI/Help/tbasic/exp.htm)	FLOAT2BITS (https://triplc.com/TRiLOGI/Help/tbasic/float2bits
LOG10 (https://triplc.com/TRiLOGI/Help/tbasic/log10.htm)	POWER (https://triplc.com/TRiLOGI/Help/tbasic/power.h
STR$(x#) (https://triplc.com/TRiLOGI/Help/tbasic/str$.htm)	STR$(x#, w) (https://triplc.com/TRiLOGI/Help/tbasic/str$.ht

For details on each instruction please click on the keyword to bring you to the specific TBASIC help file (the same file that you will see on the TL7 custom function editor Help-pane.

8.5 TBASIC Keywords

For details on each instruction please click on the keyword to bring you to the specific TBASIC help file

Note: You can call up exactly the same help file on the i-TRiLOGI Custom Function Editor’s bottom help pane when you select any keywords inside the editor pane. You can subsequently double-click the title of the keyword inside the help pane to view the whole help file on your default browser.

TBASIC Keywords for all TRi Super PLC (including the legacy M-series)

ABS (https://triplc.com/TRiLOGI/Help/tbasic/abs.htm)	ADC (https://triplc.com/TRiLOGI/Help/tbasic/adc.h
CLRIO (https://triplc.com/TRiLOGI/Help/tbasic/clrio.htm)	CRC16 (https://triplc.com/TRiLOGI/Help/tbasic/crc16.ht
GetHigh16 (https://triplc.com/TRiLOGI/Help/tbasic/gethigh16.htm)	GOTO (https://triplc.com/TRiLOGI/Help/tbasic/goto.htı
HSTIMER (https://triplc.com/TRiLOGI/Help/tbasic/hstimer.htm)	If..Then..Else (https://triplc.com/TRiLOGI/Help/tbasic/if.htm
LEN (https://triplc.com/TRiLOGI/Help/tbasic/len.htm)	LET (https://triplc.com/TRiLOGI/Help/tbasic/let.h
NETCMD$ (https://triplc.com/TRiLOGI/Help/tbasic/netcmd.htm)	OUTCOMM (https://triplc.com/TRiLOGI/Help/tbasic/outcomm.
PMOFF (https://triplc.com/TRiLOGI/Help/tbasic/pmon.htm)	PRINT # (https://triplc.com/TRiLOGI/Help/tbasic/print.ht
ReadMB2 (https://triplc.com/TRiLOGI/Help/tbasic/readmb2.htm)	REFRESH (https://triplc.com/TRiLOGI/Help/tbasic/refresh.h
SAVE_EEP (https://triplc.com/TRiLOGI/Help/tbasic/save_eep.htm)	SAVE_EEP$ (https://triplc.com/TRiLOGI/Help/tbasic/saveeep2.
SetDAC (https://triplc.com/TRiLOGI/Help/tbasic/setdac.htm)	SetHIGH16 (https://triplc.com/TRiLOGI/Help/tbasic/sethigh16.
SetProtocol (https://triplc.com/TRiLOGI/Help/tbasic/setprotocol.htm)	SetPWM (https://triplc.com/TRiLOGI/Help/tbasic/setpwm.h
StepHome (https://triplc.com/TRiLOGI/Help/tbasic/stephome.htm)	StepMove (https://triplc.com/TRiLOGI/Help/tbasic/stepmove.
STRCMP (https://triplc.com/TRiLOGI/Help/tbasic/strcmp.htm)	STRLWR$ (https://triplc.com/TRiLOGI/Help/tbasic/strlwr.ht

VAL (https://triplc.com/TRiLOGI/Help/tbasic/val.htm)	WHILE (https://triplc.com/TRiLOGI/Help/tbasic/while.ht

TBASIC Keywords for Fx-series, FMD or Nano PLCs (TL6 Version >= 6.49 and all TL7)

Continue (https://triplc.com/TRiLOGI/Help/tbasic/continue.htm)	Exit (https://triplc.com/TRiLOGI/Help/tbasic/exit.
GET_SUBNET (https://triplc.com/TRiLOGI/Help/tbasic/get_subnet.htm)	I2C_READ (https://triplc.com/TRiLOGI/Help/tbasic/i2c_read
SET_GATEWAY (https://triplc.com/TRiLOGI/Help/tbasic/set_gateway.htm)	SET_IPADDR (https://triplc.com/TRiLOGI/Help/tbasic/set_ipadd

TBASIC Keywords for Version 7.x only

ARCCOS (https://triplc.com/TRiLOGI/Help/tbasic/arccos.htm)	ARCSIN (https://triplc.com/TRiLOGI/Help/tbasic/arcsin.htm)
EXP (https://triplc.com/TRiLOGI/Help/tbasic/exp.htm)	FLOAT2BITS (https://triplc.com/TRiLOGI/Help/tbasic/float2bits.htm)
LOG10 (https://triplc.com/TRiLOGI/Help/tbasic/log10.htm)	POWER (https://triplc.com/TRiLOGI/Help/tbasic/power.htm)
STR$(x#) (https://triplc.com/TRiLOGI/Help/tbasic/str$.htm)	STR$(x#, w) (https://triplc.com/TRiLOGI/Help/tbasic/str$.htm)

TBASIC Keywords for Version >= 7.4 (Include Wx100-specific commands)

ALPHAPAD_SHOW (https://triplc.com/TRiLOGI/Help/tbasic/alphapad_show.htm)	ALERT_SHOW (https://triplc.com/TRiLOGI/Help/tbasic/ale
DRAW_ROUNDRECT (https://triplc.com/TRiLOGI/Help/tbasic/draw_roundrect.htm)	DRAW_TEXT (https://triplc.com/TRiLOGI/Help/tbasic/dr
FILL_RECT (https://triplc.com/TRiLOGI/Help/tbasic/fill_rect.htm)	FILL_ROUNDRECT (https://triplc.com/TRiLOGI/Help/tbasic/fill_r
NUMPAD_SHOW (https://triplc.com/TRiLOGI/Help/tbasic/numpad_show.htm)	SCREENCLEAR_AREA (https://triplc.com/TRiLOGI/Help/tbasic/scree

TBASIC Keywords for Version >= 7.5 (Include new Wx100 MQTT & JSON commands)

Download MQTT Quick Start Guide (https://triplc.com/documents/MQTT_Quickstart1-Connect_Wx100_to_HiveMQ.pdf)

MQTT_CONFIG (https://triplc.com/TRiLOGI/Help/tbasic/mqtt_config.htm)	MQTT_CONNECT (https://triplc.com/TRiLOGI/Help/tbasic/mqt
MQTT_UNSUBSCRIBE (https://triplc.com/TRiLOGI/Help/tbasic/mqtt_unsubscribe.htm)	GET_JSONVAL$ (https://triplc.com/TRiLOGI/Help/tbasic/get

Support Links

FAQ (Https://Triplc.Com/Faq.Htm)

Forums (Https://Triplc.Com/Smf/)

Documentation

Information

Products Page (Https://Triplc.Com/)

Testimonials (Https://Triplc.Com/Testimonials.Htm)

Resources (Https://Triplc.Com/Resources.Htm)