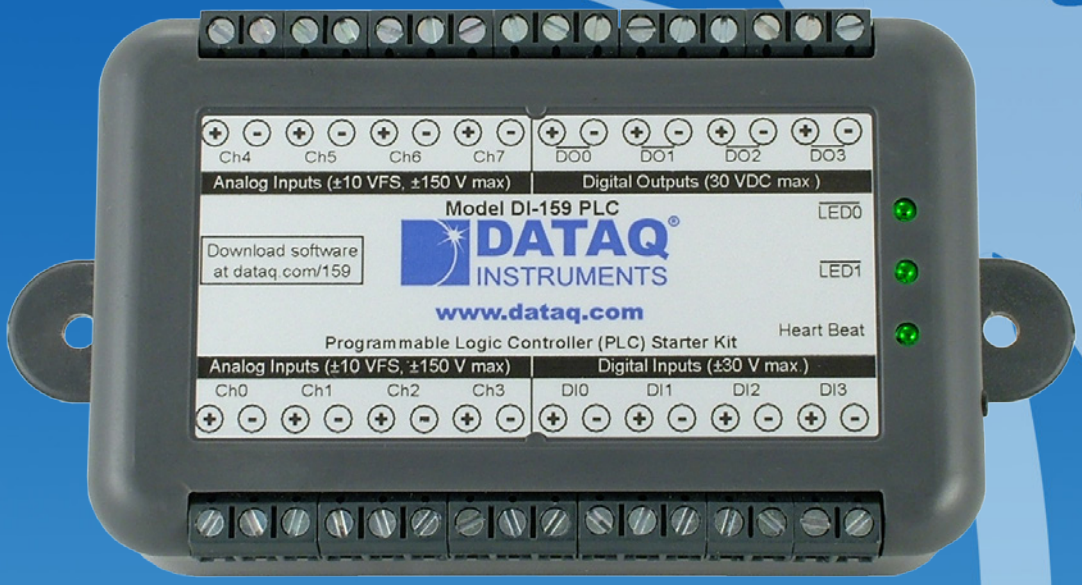
DI-159 High Speed PLC with Embedded BASIC



- ✓ Low-cost, Data Acquisition and Control
- ✓ Built-in BASIC Programming Language
- ✓ Stand-alone or PC-connected Operation
- ✓ Supports High-and Low-speed Applications
- ✓ Eight Built-in Analog Input Channels
- ✓ Four Built-in Digital Output Channels
- ✓ Four Built-in Digital Input Channels
- ✓ Replaces Controllers many times its Price and Complexity

DATAQ® INSTRUMENTS

DI-159 PLC Description

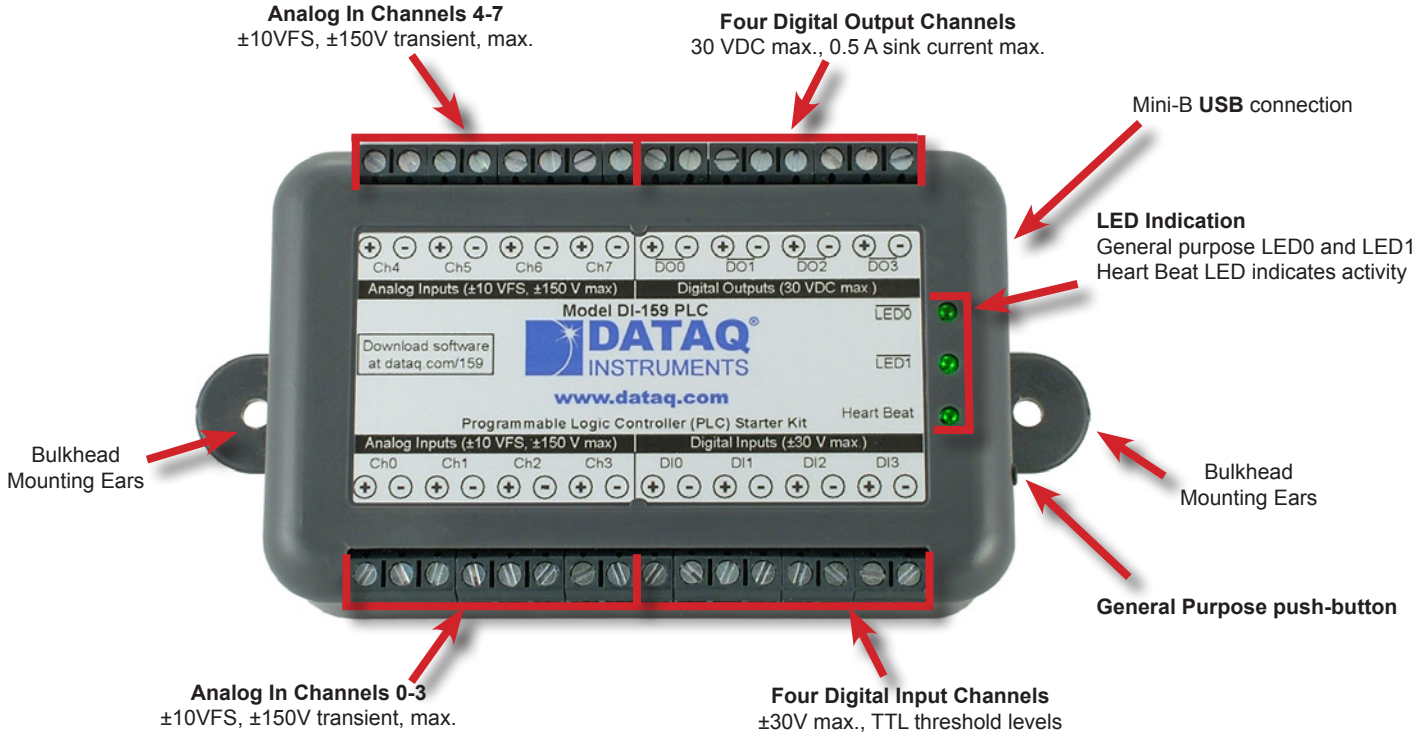
Model DI-159 PLC (programmable logic controller) is a first-of-its-kind product to leverage low cost, but extremely powerful 32-bit microcontrollers to provide a fully integrated solution that is easily programmed for a vast array of data acquisition and control applications. Alternative products (both PACs and PLCs) use complex programming language environments like C, .NET, LabVIEW, Domore!, Productivity3000, or other proprietary software that requires either a major time investment to understand and apply, or the need to hire expensive system integrators and consultants. In contrast, the integrated BASIC programming approach of the DI-159 PLC allows anyone with rudimentary programming skills to construct effective control systems in mere minutes. The product so tightly integrates the programming language with data acquisition and I/O hardware that many complex control scenarios may be implemented with only a few BASIC instructions.

The DI-159 PLC offers most of the data acquisition features of the popular DATAQ Instruments DI-149 Data Acquisition Starter Kit. It provides eight protected analog input channels, each with a ±10 VFS range and 10 bits of ADC resolution. Also provided are four each general purpose digital inputs and outputs. The outputs can switch up to 30 V/500 mA loads, and the inputs detect TTL thresholds over a maximum applied range of ±30 V peak. Also provided for general-purpose use are two LEDs and one push button switch.

Wrapped around this data acquisition core is an embedded BASIC language compiler called StickOS from www.CPUStick.com that provides single-instruction access to each analog input and digital I/O port, as well as the LEDs and switch. The language supports real time performance to allow precisely timed operations that are so critical for many process control applications. It features four tick timers, each independently programmable to 250 µs resolution. This real time core is then leveraged by a powerful and familiar BASIC command set that supports block statements (IF, THEN, ELSE, etc.), strings, arrays, mathematical expressions, and much more.

The DI-159 PLC provides a USB port interface and can be used under any operating system that can run a terminal emulator and hook a COM port. A connected terminal emulator provides direct access to the DI-159 PLC's embedded BASIC programming environment and, depending upon the emulator, the ability to save and load an unlimited number of programs beyond the DI-159 PLC's built-in flash memory limit of three. Emulators that are available online for free are Terminator and Konsole (Linux), iTerm2 (OS X), and PuTTY and TeraTerm (Windows). Also for the Windows environment the DI-159 PLC is supplied with a Visual Basic. NET application that allows terminal emulation, and BASIC program storage and retrieval. The DI-159 PLC's BASIC compiler is pre-programmed with a demo application that works with the Windows application to record DI-159 PLC analog and digital data to a connected PC in a Microsoft Excel-compatible format. Also included with the DI-159 PLC is a USB cable. Purchase the optional AC adaptor (connects to the USB cable) for stand-alone, computer-independent control applications.

DI-159 PLC Close-up



DI-159 PLC Features

Eight Analog Input Channels

Connect the DI-159 PLC to any pre-amplified signal in the typical range of ±5 to ±10 VFS.

Four Digital Inputs

Four discrete inputs allow the DI-159 PLC to access and process external, discrete (on/off) events.

Four High Voltage Digital Outputs

Four digital outputs allow the DI-159 PLC to initiate external discrete (on/off) control. Loads up to 30 V peak at 500 mA are supported.

10-bit ADC resolution

Provides 19.5 mV resolution across the entire DI-159 PLC measurement range.

Ruggedized Inputs

All eight DI-159 PLC analog inputs are protected to ±150 V peak. Discrete inputs are protected to ±30 V peak. Accidents that happen are quickly forgiven.

Noise-cancelling Analog Input Configuration

The noise-immunity of differential inputs minimizes the effects of common mode noise so often encountered in industrial measurements.

USB Interface

The DI-159 PLC is provided with a built-in USB interface, allowing it to be connected, powered, and operated from any laptop or desktop PC. For stand-alone control applications, the provided USB interface cable provides power to the DI-159 PLC using an optional AC adaptor.

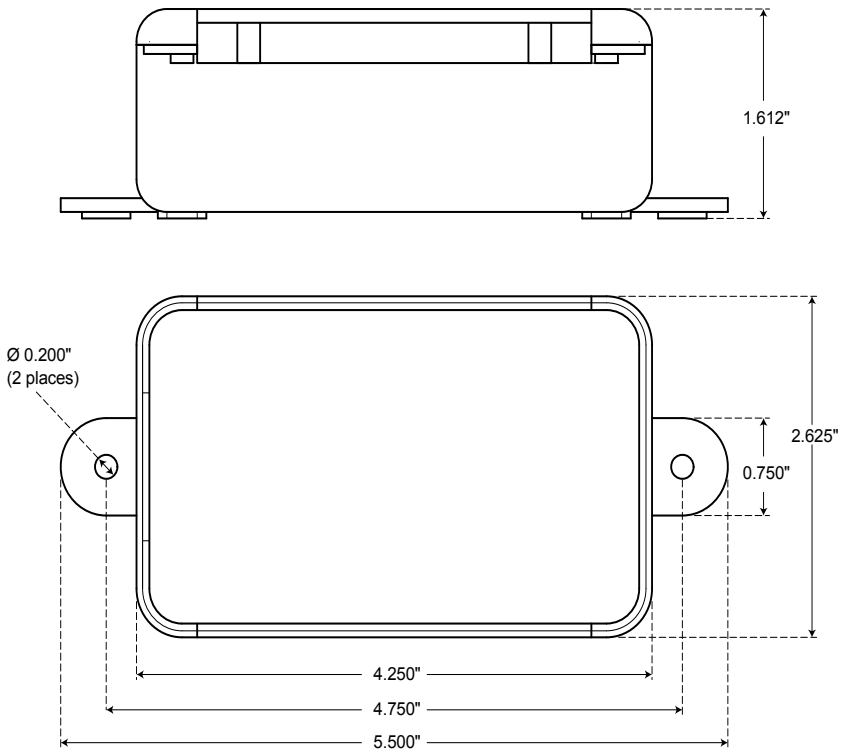
Wide OS support

A signed driver is provided with the DI-159 PLC that allows trouble-free installation for Windows XP and both 32- and 64-bit versions of Windows Vista, Windows 7, and Windows 8. Since the DI-159 PLC hooks a COM port, ubiquitous support is provided for any other OS using standard programming tools and emulators.

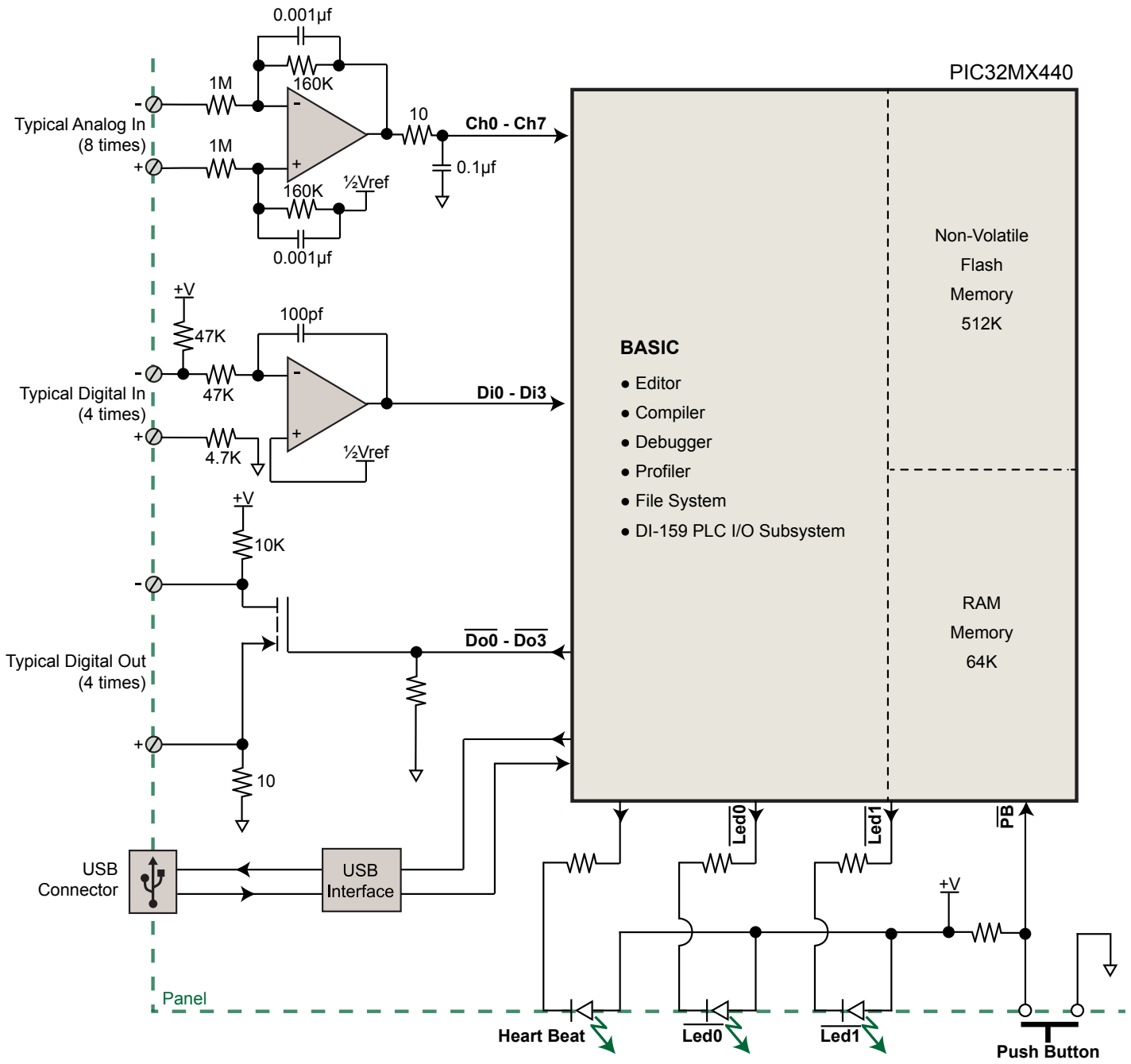
Embedded BASIC Programming Environment

The DI-159 PLC embeds a microcontroller-resident, interactive BASIC programming environment that includes a program editor, interactive debugger, performance profiler, and flash-based file system, all running entirely within DI-159 PLC hardware and controlled thru an interactive command-line user interface. Access the programming environment through any terminal emulator operating under any operating system.

DI-159 PLC Dimensional Drawing

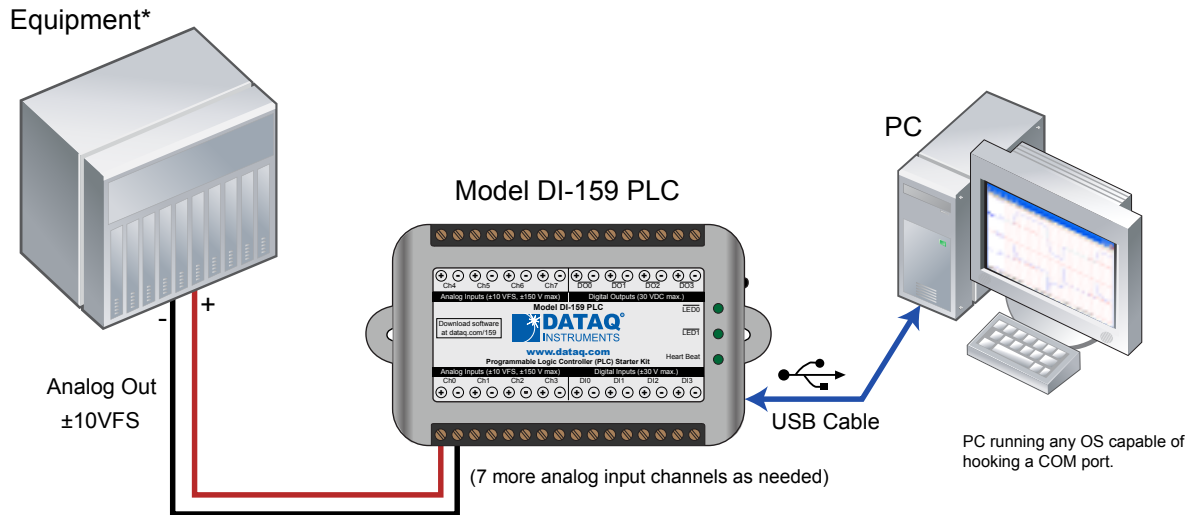


DI-159 PLC Block Diagram



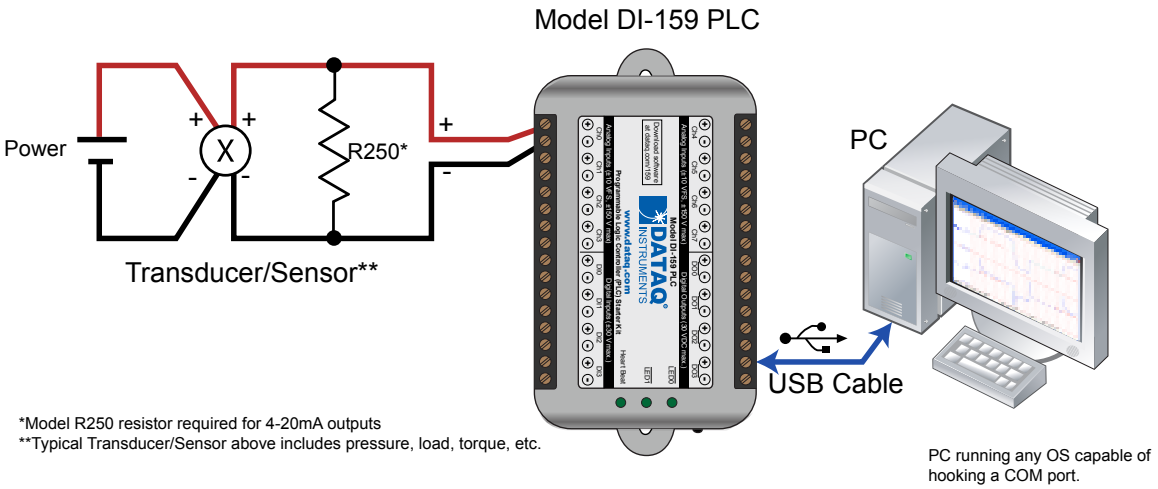
Typical DI-159 PLC Analog Connections

Typical voltage mode connection



* DATAQ Instruments' hardware and software products are NOT designed to be used in the diagnosis and treatment of humans, nor are they to be used as critical components in any life-support systems whose failure to perform can reasonably be expected to cause significant injury to humans.

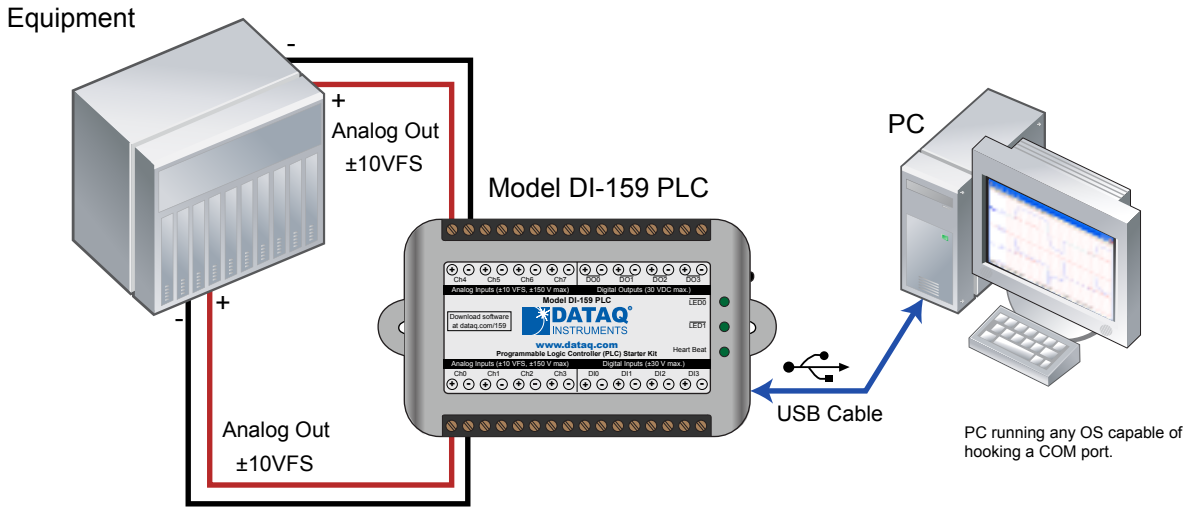
Typical current mode connection



*Model R250 resistor required for 4-20mA outputs
**Typical Transducer/Sensor above includes pressure, load, torque, etc.

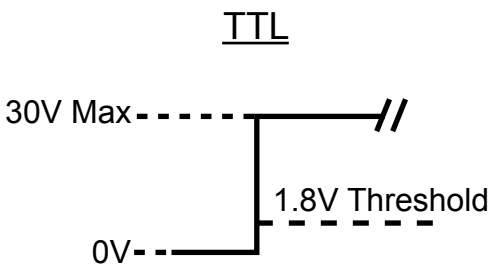
Typical DI-159 PLC Analog Connections

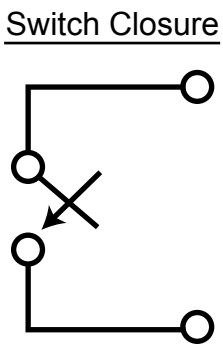
Typical multi-channel analog connection

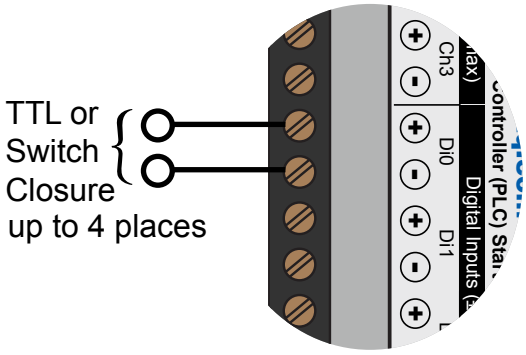


DI-159 PLC Digital Input Connections

The DI-159 PLC's provides four discrete input channels that can adapt to a wide range of signal types. Internal pull-ups are provided, so they can be used with any type of dry-contact switches. When connecting to process discretes, the inputs will tolerate ±30 V without damage. Software access to the digital input ports is tightly integrated with the embedded BASIC programming language, and easily manipulated. See the programming examples in this data sheet for more information.

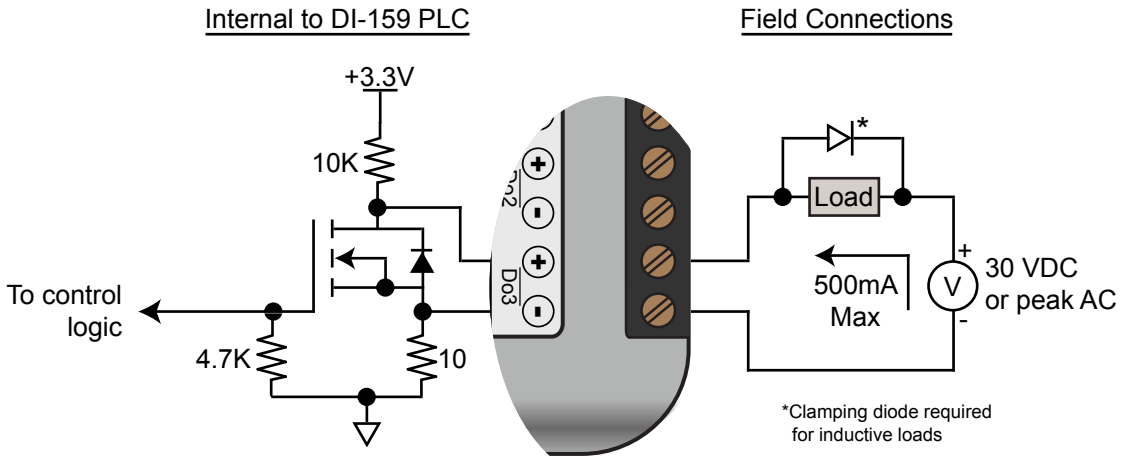




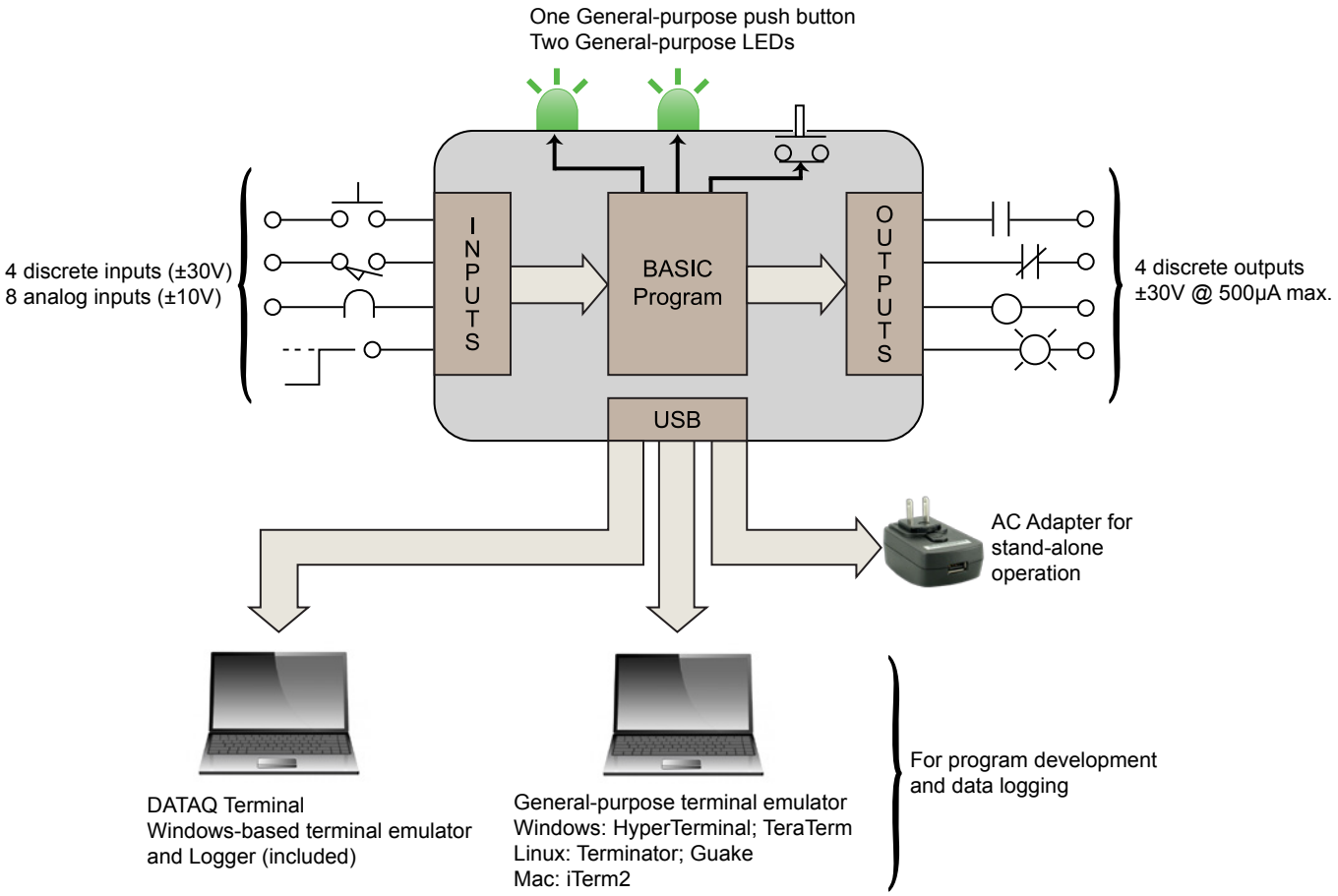


Typical DI-159 PLC Digital Output Connections

The DI-159 PLC's digital outputs are not your ordinary transistor switches. Each of four channels features a dedicated MOSFET that can handle high voltage and current loads with built-in electrostatic discharge protection. It can switch loads of up to 30 Volts (DC or peak AC) and 500 mA. Use it to control relays when the load to be switched exceeds the 30 V/500 mA spec of the port, or to control the load directly where it does not. Software access to the digital output ports is tightly integrated with the embedded BASIC programming language, and easily manipulated. See the programming examples in this data sheet for more information.

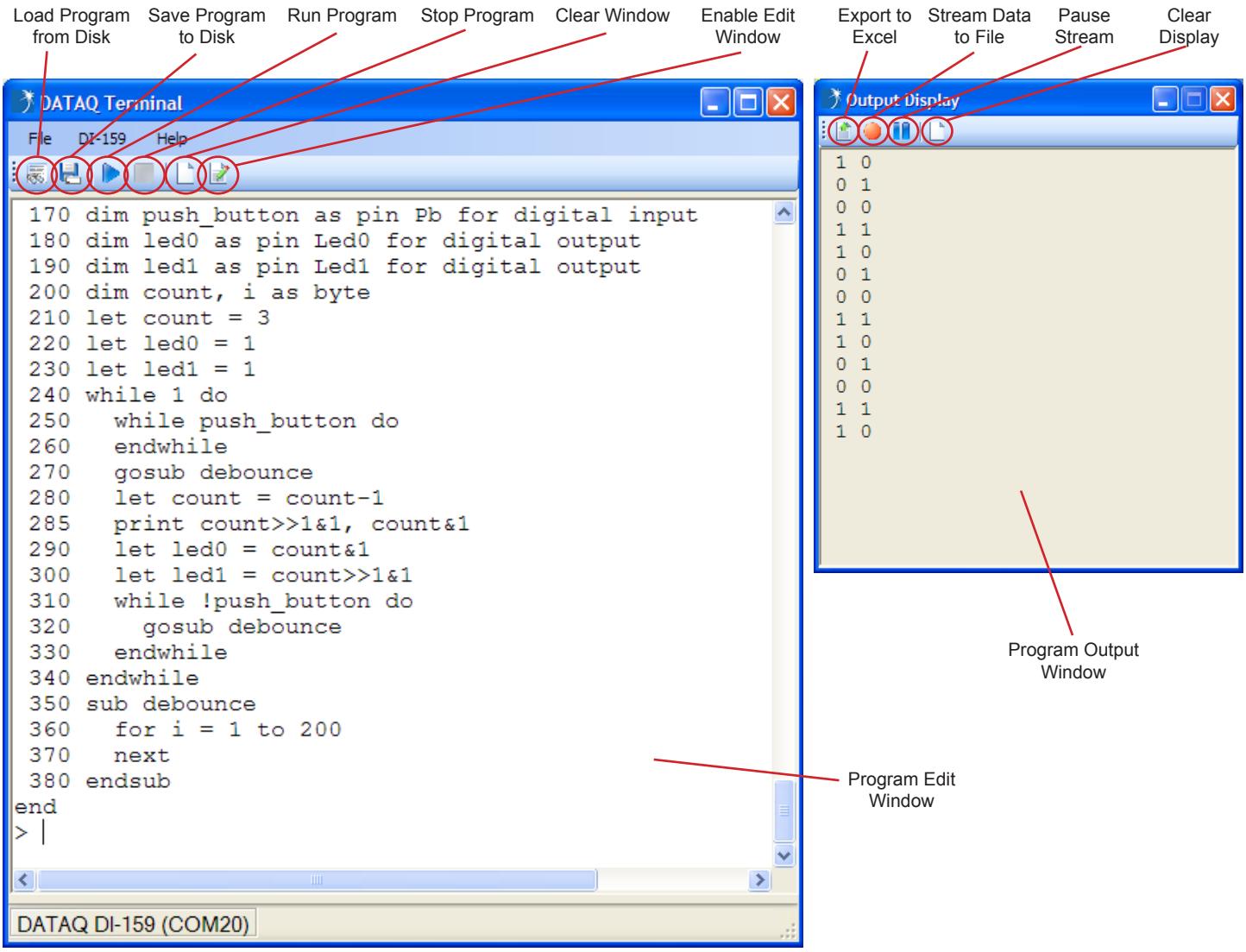


Typical DI-159 PLC Deployment Block Diagram



Included Terminal Emulator for Windows

The DI-159 PLC can connect to any terminal emulator program running under any operating system that can hook a COM port. DATAQ Terminal is a VB.NET-based, open source VT-100 terminal emulator and data recorder that runs under Windows, and is included with the DI-159 PLC. It provides the ability to store and retrieve BASIC program files to and from the hard drive, and the contents in its Output window can be directed to a text file for a permanent record. Then use the Terminal program to open the file in Microsoft Excel. Finally, the terminal emulator supports VT100-like editing using your keyboard's cursor-control keys.



The DI-159 PLC BASIC Programming and Control Environment

The BASIC programming engine that is embedded within the DI-159 PLC offers a powerful development platform that doesn't sacrifice ease-of-use or utility. The environment offers transparent line-by-line compilation, as well as integer variable, string variable, and array support. Block-structured programming is also supported using easily-recognized IF, FOR, WHILE, DO, and GOSUB constructs. This environment is so familiar and comfortable to anyone with rudimentary programming skills, that you'll know how to use it after a simple glance and review. See the *BASIC Quick Reference Guide* following the example programs for an overview of all supported commands and statements . The following is a brief list of the BASIC programming environment's major features:

Integrated Program Debugger

Easily track down problems with your code using breakpoints, assertions, and watch points. Debugging tools also include program trace mode, single stepping, and edit-and-continue. You can even use the built-in profiler to determine where your program spends its time so you can optimize it as required.

Built-in File System

The DI-159 PLC's BASIC program supports flash memory that allows you to load and store multiple programs (three maximum), each with its own file name. An unlimited number of programs may be loaded using any terminal emulator that supports file transfers, like the DATAQ Terminal utility program provided with the instrument and the freely available TeraTerm application.

Real Time PLC Performance

The embedded BASIC engine even offers real time performance to rival its most costly competitive alternatives. Four timers are available for general-purpose use, each providing jitter-free and consistent timing operations with 250 microsecond resolution. The timers provide maskable software interrupts that you can use for very precise scheduling of process events without the need for complex, external hardware timers.

Immediate-mode allows Keyboard-control over Inputs and Outputs

A simple, single line instruction reads any analog input. Another sets (or reads) the state of a digital I/O port. For example, led0=0 lights general-purpose LED0, and led0=1 turns it off.

Autorun Mode and Built-in Flash Memory Enable Stand-alone Control

After you've written and tested your BASIC program using a connected PC with a terminal emulator, save it to the DI-159 PLC's non-volatile memory, and type *autorun* at the command prompt. Then deploy the DI-159 PLC powered from the optional AC adaptor as a stand-alone controller. Upon applying power to the DI-159 PLC, the instrument automatically initializes itself and begins running your program. It's that simple.

Programming Examples

Aside from actually applying a DI-159 PLC in your control application, the best way to understand how easy the instrument is to use is by example. The following just scratches the surface, but should give you a solid understanding of the range of DI-159 PLC control possibilities. Note that explanatory comments appear in these examples on the same line as the code to conserve space. Since the BASIC engine supports comments using the familiar REM statement, comments would actually appear as program lines.

Example #1

Object

Boilerplate code that must be included at the beginning of every program that uses the specified I/O points. *

Code

```
 10 dim c0 as pin Ch0 for analog input            'map analog input 0 to BASIC variable "c0"
 20 dim c1 as pin Ch1 for analog input            'map analog input 1 to BASIC variable "c1"
 30 dim c2 as pin Ch2 for analog input            'map analog input 2 to BASIC variable "c2"
 40 dim c3 as pin Ch3 for analog input            'map analog input 3 to BASIC variable "c3"
 50 dim c4 as pin Ch4 for analog input            'map analog input 4 to BASIC variable "c4"
 60 dim c5 as pin Ch5 for analog input            'map analog input 5 to BASIC variable "c5"
 70 dim c6 as pin Ch6 for analog input            'map analog input 6 to BASIC variable "c6"
 80 dim c7 as pin Ch7 for analog input            'map analog input 7 to BASIC variable "c7"
 90 dim i0 as pin Di0 for digital input           'map digital input 0 to BASIC variable "i0"
100 dim i1 as pin Di1 for digital input           'map digital input 1 to BASIC variable "i1"
110 dim i2 as pin Di2 for digital input           'map digital input 2 to BASIC variable "i2"
120 dim i3 as pin Di3 for digital input           'map digital input 3 to BASIC variable "i3"
130 dim o0 as pin Do0 for digital output          'map digital output 0 to BASIC variable "o0"
140 dim o1 as pin Do1 for digital output          'map digital output 1 to BASIC variable "o1"
150 dim o2 as pin Do2 for digital output          'map digital output 2 to BASIC variable "o2"
160 dim o3 as pin Do3 for digital output          'map digital output 3 to BASIC variable "o3"
170 dim push_button as pin Pb for digital input   'map pushbutton to BASIC variable "push_button"
180 dim led0 as pin Led0 for digital output       'map LED0 to BASIC variable "led0"
190 dim led1 as pin Led1 for digital output       'map LED1 to BASIC variable "led1"
200 rem your program starts here
```

Comment

These instructions map the various DI-159 PLC analog input and digital I/0 points so the BASIC program can use them. You can rename them as necessary (e.g. change "c0" to "MotorVoltage"), and you can even omit those that will not be used by your program. In this example all I/O points have been mapped to the variable names that immediately follow the "dim" statement.

* Note that for clarity this code will not be shown in all other examples, so all subsequent programming examples begin with line 200.

Example #2

Object

Flash general-purpose LED0 at a precise one second on/off interval.

Code

```
200 let led0 = 0                        'start with LED on
210 configure timer 0 for 1 s           'configure one of four timers for 1 sec interval
220 on timer 0 do gosub flasher         'execute subroutine "flasher" when timer fires
230 while 1 do                          'do nothing while waiting for the timer to fire
240 endwhile                            '
250 sub flasher                         'end up here when timer fires
260   let led0 = !led0                  'invert the state of LED0 (turn off if on, and on if off)
270 endsub                              'return to waiting for the timer to fire again
```

Comment

This example demonstrates the real time power of the BASIC program, the ease with which it can manipulate peripherals, and one of many block statements (gosub, in this case). A single statement configures a timer for a precise interval, and another single statement defines the state of the peripheral (LED0 in this case.) Timer intervals can be configured in seconds (s), milliseconds (ms), or microseconds (us) and can range from milliseconds to hours.

Example #3

Object

Flash both LEDs at precisely different rates, LED1 at four times the rate of LED0.

Code

```
200 let led0 = 0                    'begin with both LEDs on
210 let led1 = 0                    '
220 configure timer 0 for 1000 ms   'configure first of four timers for 1 sec interval
230 configure timer 1 for 250 ms    'configure second of four timers for 1/4 sec interval
240 on timer 0 do gosub flash_led0  'execute subroutine "flash_led0" when timer0 fires
250 on timer 1 do gosub flash_led1  'execute subroutine "flash_led1" when timer1 fires
260 while 1 do                      'do nothing while waiting for the timers to fire
270 endwhile                        '
280 sub flash_led0                  'end up here when timer0 fires
290   let led0 = !led0              'invert the state of LED0 (turn off if on, and on if off)
300 endsub                          'return to waiting for timers to fire again
310 sub flash_led1                  'end up here when timer1 fires
320   let led1 = !led1              'invert the state of LED1 (turn off if on, and on if off)
330 endsub                          'return to waiting for timers to fire again
```

Comment

Extends the example above to include two timers, each running at a precise, independent, and different rate. Each timer indirectly controls the state of a peripheral, in this case the two general-purpose LEDs.

Example #4

Object

Flash both LEDs at precisely different rates, LED1 at four times the rate of LED0, and control digital outputs DO0 and DO1 in the same way.

Code

```
200 let led0 = 0                    'begin with both LEDs on
210 let led1 = 0                    '
220 let o0 = 1                      'begin with both digital outputs on
230 let o1 = 1                      '
240 configure timer 0 for 1000 ms   'configure first of four timers for 1 sec interval
250 configure timer 1 for 250 ms    'configure second of four timers for 1/4 sec interval
260 on timer 0 do gosub flash_led0  'execute subroutine "flash_led0" when timer0 fires
270 on timer 1 do gosub flash_led1  'execute subroutine "flash_led1" when timer1 fires
280 while 1 do                      'do nothing while waiting for the timers to fire
290 endwhile                        '
300 sub flash_led0                  'end up here when timer0 fires
310   let led0 = !led0              'invert the state of LED0 (turn off if on, and on if off)
320   let o0 = !o0                  'invert the state of digital out 0 (turn off if on, and on if off)
330 endsub                          'return to waiting for timers to fire again
340 sub flash_led1                  'end up here when timer1 fires
350   let led1 = !led1              'invert the state of LED1 (turn off if on, and on if off)
360   let o1 = !o1                  'invert the state of digital out 1 (turn off if on, and on if off)
370 endsub                          'return to waiting for timers to fire again
```

Comment

A slight modification to Example #3 includes digital outputs DO0 and DO1 in the state changes of LEDs 0 and 1.

Example #5

Object

Use BASIC's bitwise expressions to change the state of the LEDs in binary-count order from 00 (off, off) to 11 (on, on) each time the general-purpose pushbutton is pressed. Also introduces BASIC's FOR/NEXT block statement.

Code

```
200 dim count, i as byte      'define variables COUNT and I as byte (0-255)
210 let count = 3             'set two LSBs of COUNT to 1
220 let led0 = 1              'set initial LED states to match initial COUNT value (both LEDs off)
230 let led1 = 1              '
240 while 1 do                'loop continuously
250   while push_button do    'wait for pushbutton to be pressed (low true)
260   endwhile
270   gosub debounce          'de-bounce the pushbutton to get one clean transition
280   let count = count-1     'the LEDs are low true, so we'll decrement COUNT
290   let led0 = count&1      'LED0 is LSB, so mask COUNT LSB state by ANDing with 1
300   let led1 = count>>1&1   'LED1 is second LSB, so right-shift COUNT one bit, then AND with 1
310   while !push_button do   'wait for the pushbutton to be released
320     gosub debounce        'de-bounce pushbutton again to get a clean transition
330   endwhile                'end one pushbutton cycle
340 endwhile                  'do it again
350 sub debounce              'pushbutton de-bounce subroutine.
360   for i = 1 to 200        'do nothing for 200 cycles while the pushbutton settles down
370   next
380 endsub                    'return from the subroutine
```

Comment

Note that the pushbutton and LEDs are low true. A 0 written to either LED lights it, and the pushbutton transitions from 1 to 0 when pressed.

Example #6

Object

Create a square wave output with a frequency that's proportional to the magnitude of the voltage applied to an analog input channel.

Code

```
200 dim o0 as pin Do0 for frequency output   're-dimension digital out 0 to be a frequency output
210 dim New_o0                               'define variable New_o0
220 let o0 = 100                             'set the square wave frequency output at o0 to 100 Hz
230 configure timer 0 for 100 ms             'set timer to update 10 times per second
240 on timer 0 do gosub Update               'go to subroutine Update whenever timer 0 fires
250 while 1 do                               'loop while waiting for the timer to fire
260 endwhile                                 '
270 sub Update                               'get here when the timer fires
280   let New_o0 = c0/100+100                'calculate a new frequency where c0 = analog input 0
290   if o0!=New_o0 then                     'don't write the value if it hasn't changed
300     let o0 = New_o0                      'New_o0 is a different value so change the frequency
310   endif
330 endsub
```

Comment

Note that the pushbutton and LEDs are low true. A 0 written to either LED lights it, and the pushbutton transitions from 1 to 0 when pressed. Any DI-159 PLC digital output port may be programmed to output a precise frequency using the method shown here. Frequency can range from 0 to several kHz in 1 Hz steps. Analog input values are in millivolts and range from ±10,000. The calculation in line 280 above yields a frequency scaled between DC and 200 Hz for a –full scale to +full scale range respectively.

Example #7

Object

Print variables in this order: sample counter, all eight analog channels, and the state of the digital inputs and outputs. Pressing the general-purpose pushbutton resets the sample counter, and the digital output states reflect the value of the sample counter. Finally, LED0 and LED1 display the state of the two LSBs of the sample rate counter.

Code

```
200 dim cr
210 configure timer 0 for 1 s
220 on timer 0 do gosub readin
230 on push_button==0 do cr = 0
240 while 1 do
250 endwhile
260 sub readin
270  cr = cr+1
280  let o0 = cr&1
290  let o1 = cr&2
300  let o2 = cr&4
310  let o3 = cr&8
320  print cr, c0, c1, c2, c3, c4, c5,c6,
     c7, i0, i1, i2, i3, o0, o1, o2, o3
330  led0 = o0
340  led1 = o1
350 endsub
```

```
'dimension cr, which will be used as a counter
'set up a timer for a sample rate of 1 Hz
'go to subroutine 'readin' when the timer fires
'reset the sample counter if the pushbutton is pressed
'do nothing while waiting for the 1 Hz timer to fire
'arrive here when the timer fires
'increment the sample rate counter
'assign dig out 0 to the LSB of the sample counter
'assign dig out 1 to bit 1 of the sample counter
'assign dig out 2 to bit 2 of the sample counter
'assign dig out 3 to the MSB of the sample counter
'stream sample counter, all analog input values, and
'state of the digital inputs and outputs
'assign the state of dig out 0 to LED0
'assign the state of dig out 1 to LED1
'return
```

Comment

This example, when used with the provided DATAQ Terminal program for Windows, allows recorded values to be streamed to a CSV file via the PRINT statement, which is easily imported to Microsoft Excel.

DI-159 PLC Embedded BASIC Quick Reference Guide

Commands

<Ctrl-C>	stop running program
auto [*line*]	automatically number program lines
clear [**flash**]	clear ram [and flash] variables
cls	clear terminal screen
cont [*line*]	continue program from stop
delete ([*line*][-[*line*]]|*subname*)	delete program lines
dir	list saved programs
edit *line*	edit program line
help [*topic*]	online help
list ([*line*][-[*line*]]|*subname*)	list program lines
load *name*	load saved program
memory	print memory usage
new	erase code ram and flash memories
purge *name*	purge saved program
renumber [*line*]	renumber program lines (and save)
run [*line*]	run program
save [*name*|**library**]	save code ram to flash memory
undo	undo code changes since last save
upgrade	upgrade StickOS firmware!
Uptime	print time since last reset

Device Statements

timers:

configure timer *n* **for** *n* (**s**|**ms**|**us**)
on timer *n* **do** *statement*

off timer *n*	disable timer interrupt
mask timer *n*	mask/hold timer
interrupt unmask timer *n*	unmask timer interrupt

watchpoints:

on *expression* **do** *statement*

off *expression*	disable expr watchpoint
mask *expression*	mask/hold expr watchpoint
unmask *expression*	unmask expr watchpoint

Expressions

the following operators are supported as in C, in order of decreasing precedence:

n	decimal constant
0xn	hexadecimal constant
'c'	character constant
variable	simple variable
variable[expression]	array variable element
variable#	length of array or string
( )	grouping
! ~	logical not, bitwise not
*** / %**	multiply, divide, mod
+ -	add, subtract
>> <<	shift right, left
<= < >= >	inequalities
== !=	equal, not equal
| ^ &	bitwise or, xor, and
|| ^^ &&	logical or, xor, and

Strings

V$ is a null-terminated view into a byte array v[]

string statements:

dim, input, let, print, vprint
if *expression relation expression* **then**
while *expression relation expression*
do until *expression relation expression*

string expressions:

"literal"	literal string
variable$	variable string variable$
[*start:length*]	variable substring
+	concatenates strings

string relations:

<= < >= >	inequalities
== !=	equal, not equal
~ !~	contains, does not contain

General Statements

Line	delete program line
line statement // comment	enter program line
variable[$] **=** *expression, ...*	assign variable
? [**dec**|**hex**|**raw**] *expression, ...*[;]	print strings/expressions
assert *expression*	break if expression is false
data *n* [, ...]	read-only data
dim *variable*[$][[*n*]] [as ...], ...	dimension variables
end	end program
halt	loop forever
input [**dec**|**hex**|**raw**] *variable*[$], ...	input data
label *label*	read/data label
let *variable*[$] **=** *expression, ...*	assign variable
print [**dec**|**hex**|**raw**] *expression, ...*[;]	print strings/expressions
read *variable* [, ...]	read data into variables
rem *remark*	remark
restore [*label*]	restore data pointer
sleep *expression* (**s**|**ms**|**us**)	delay program execution
stop	insert breakpoint in code
vprint *var*[$]=[**dec**|**hex**|**raw**] *expr, ...*	print to variable

Modes

analog [*millivolts*]	set analog voltage scale
autorun [**on**|**off**]	autorun mode (on reset)
echo [**on**|**off**]	terminal echo mode
indent [**on**|**off**]	listing indent mode
numbers [**on**|**off**]	listing line numbers mode
pins [*assign* [*pinname*|**none**]]	set/display pin assignments
prompt [**on**|**off**]	terminal prompt mode
step [**on**|**off**]	debugger single-step mode
trace [**on**|**off**]	debugger trace mode
watchsmart [**on**|**off**]	low-overhead watchpoint mode

Block Statements

if *expression* **then**
  [**elseif** *expression* **then**] [**else**]
endif

for *variable* **=** *expression* **to** *expression* [**step** *expression*]
[(**break**|**continue**) [*n*]]
next

while *expression* **do**
  [(**break**|**continue**) [*n*]]
endwhile

do
  [(**break**|**continue**) [*n*]]
until *expression*

gosub *subname* [*expression, ...*]

sub *subname* [param, ...] [**return**]
endsub

Variables

all variables must be dimensioned

variables dimensioned in a sub are local to that sub
simple variables are passed to sub params by reference
array variable indices start at 0
v is the same as v[0], except for input/print statements

ram variables:

dim *var*[$][[*n*]]
dim *var*[[*n*]] **as** (**byte**|**short**)

flash parameter variables:

dim *varflash*[[*n*]] **as flash**

pin alias variables:

dim *varpin*[[*n*]] **as pin** *pinname* **for** \
(**digital**|**analog**|**frequency**) \
(**input**|**output**) \

absolute variables:

dim *varabs*[[*n*]] **at address** *addr*
dim *varabs*[[*n*]] **as** (**byte**|**short**) **at address** *addr*

system variables (read-only):

analog getchar keychar msecs nodeid random seconds ticks ticks_per_msec

DI-159 PLC Specifications

Signal Inputs

Analog Inputs

Number of Channels:	8
Configuration:	Differential
Full Scale Range:	±10VFS
Input impedance:	2 MΩ, differential
Isolation:	none
Overall inaccuracy:	±64mV (at 25°C)
Minumum common mode rejection:	40db @ 50-60 Hz and @ 25°C
Max input without damage:	±75 V peak continuous ±150 V peak, one minute or less
Max common mode voltage:	±10V
Analog frequency response:	-3db @ 1,000 Hz

Digital Inputs

Number of Channels:	4
Pull-up value:	47 KΩ
Isolation:	none
Input high voltage threshold:	1.8 V minimum
Input low voltage threshold:	1.4 V maximum
Absolute maximum values:	±30 VDC

Digital Outputs

Number of Channels:	4
Isolation:	none
Absolute max ratings:	Voltage: 30 VDC or peak AC Sink current: 0.5 A Source current: 3 mA On resistance < 2 Ω

Power

Power Consumption:	<1.0 Watt, via USB interface

* 11,000 Hz for 11 enabled channels (8 analog, 3 digital)

ADC Characteristics

Resolution:	Overall: approx. 1 part in 1,024 (10-bit) Above zero: approx. 1 part in 511 Below zero: approx. 1 part in 512
Max sample throughput rate:	10,000 Hz*
Min sample throughput rate:	11.44 Hz
Sample rate timing accuracy:	50 ppm

Indicators, Controls, and Connections

Interface:	USB 2.0 (mini-B style connector)
Indicators (LED):	Three. Two for general-purpose use , one reserved for activity indication.
Push button:	General-purpose use
Input Connections:	Two 16-position terminal strips

Environmental

Operating Temperature:	0°C to 35°C (32°F to 95°F)
Operating Humidity:	0 to 90% non-condensing
Storage Temperature:	-20°C to 45°C (-4°F to 113°F)
Storage Humidity:	0 to 90% non-condensing

Physical Characteristics

Enclosure:	Hardened Plastic
Mounting:	Desktop; bulkhead
Dimensions:	2.625D × 5.5W × 1.53H in. (6.67D × 13.97W × 3.89H cm.)
Weight:	< 4 oz. (< 140 grams)

Software Support

Embedded:	StickOS(TM) BASIC
Downloadable:	DI-159 PLC Windows-based Utiltiy software for terminal emulation, program archive, data logging. Provided as a VB.NET application that supports Windows XP and both 32- and 64-bit versions of Windows Vista, Windows 7, and Windows 8.

Ordering Guide

Description	Order No.
DI-159 Programmable Logic Controller (PLC) Data acquisition and control package consisting of embedded StickOS(TM) BASIC, DI-159 PLC hardware, and USB cable. Windows-based utility software is available via free download from www.dataq.com/159	DI-159

Optional Accessories

101085 Power supply adapter (USB to AC).	**101085**	**101017-RPS** Australian Adapter for power supply 101085.	**101017-RPS**
101017-RPE European Adapter for power supply 101085.	**101017-RPE**	**101017-RPA** SPARE US Adapter for power supply 101085 (one already ships with 101085).	**101017-RPA**
101017-RPK UK Adapter for power supply 101085.	**101017-RPK**		

Included

DI-159-PLC

USB Cable (1 meter)

Optional Accessories

101085

101017-RPS

101017-RPK

101017-RPE

101017-RPA*

***USA adapter is included with purchase of 101085**

241 Springside Drive
Akron, Ohio 44333
Phone: 330-668-1444
Fax: 330-666-5434

Data Acquisition Product Links

(click on text to jump to page)

Data Acquisition | Data Logger | Chart Recorder