

General Purpose

Application Specific Function Block Manual

Version 13.0.1

NOTE

Progress is an on-going commitment at Giddings & Lewis. We continually strive to offer the most advanced products in the industry; therefore, information in this document is subject to change without notice. The illustrations and specifications are not binding in detail. Giddings & Lewis shall not be liable for any technical or editorial omissions occurring in this document, nor for any consequential or incidental damages resulting from the use of this document.

DO NOT ATTEMPT to use any Giddings & Lewis product until the use of such product is completely understood. It is the responsibility of the user to make certain proper operation practices are understood. Giddings & Lewis products should be used only by qualified personnel and for the express purpose for which said products were designed.

Should information not covered in this document be required, contact the Customer Service Department, Giddings & Lewis, 660 South Military Road, P.O. Box 1658, Fond du Lac, WI 54936-1658. Giddings & Lewis can be reached by telephone at (920) 921-7100.

DISCLAIMER: All programs in this release (application demos, application specific function blocks (ASFB's), etc.), are provided "AS IS, WHERE IS", WITHOUT ANY WARRANTIES, EXPRESS OR IMPLIED. There may be technical or editorial omissions in the programs and their specifications. These programs are provided solely for user application development and user assumes all responsibility for their use. Programs and their content are subject to change without notice.

Release 2002

© 1993-2002 Giddings & Lewis, Controls, Measurement, and Sensing, A Company of Thyssen Krupp Technologies

IBM is a registered trademark of International Business Machines Corporation.

Windows 95, 98, NT, Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation.

Pentium and PentiumPro are trademarks of Intel Corporation.

ARCNET is a registered trademark of Datapoint.

PiC900, PiCPro, MMC, PiCServoPro, PiCTune, PiCProfile, LDO Merge, PiCMicroTerm and PiC Programming Pendant are trademarks of Giddings & Lewis.

Table of Contents:

General Purpose ASFB Manual

CHAPTER 1-Application Specific Function Block Guidelines	1-1
Installation.....	1-1
Revisions	1-1
Network 1	1-1
Network 2	1-2
Network 3	1-2
ASFB Input/Output Descriptions.....	1-2
Network 4	1-2
Using ASFBs.....	1-3
CHAPTER 2-General Purpose ASFBs	2-1
G_CONFIG.....	2-5
G_RCVSTR.....	2-7
G_SNDSTR.....	2-8
G_BN2STR.....	2-9
G_BOO2DW	2-10
G_BOO2WD	2-12
G_BY2BIT.....	2-13
G_DW2BIT	2-14
G_DW2BOO	2-15
G_HX2STR	2-17
G_NM2STR.....	2-18
G_WD2BIT	2-19
G_WD2BOO	2-20
G_BETWN	2-21
G_GE_ALL	2-22
G_GE_ANY.....	2-23
G_LE_ALL.....	2-24
G_LE_ANY	2-25
G_FILMNG	2-26
G_READFL.....	2-28
G_WRITFL.....	2-29
G_C2F.....	2-30
G_D2ARMP	2-31
G_DEG2RD.....	2-32
G_DRMSEQ.....	2-33
G_F2C.....	2-36
G_HSTGRM.....	2-37
G_RD2DEG.....	2-39

G_CHK_32	2-40
G_CHKBIT.....	2-41
G_SHIFT	2-42
G_SHL_32	2-43
G_SHR_32.....	2-44
G_ELTMR	2-45
G_FLTMR	2-46
Index.....	Index-1

CHAPTER 1 **Application Specific Function Block Guidelines**

Installation

The following guidelines are recommended ways of working with Application Specific Function Blocks (i.e. ASFBs) from Giddings & Lewis.

The Applications CD includes the ASFB package as follows:

- .LIB file(s) containing the ASFB(s)
- source .LDO(s) from which the ASFB(s) was made
- example LDO(s) with the ASFB(s) incorporated into the ladder which you can then use to begin programming from or merge with an existing application ladder

When you install the Applications CD, the ASFB paths default to:

C:\Program Files\Giddings & Lewis\Applications *vxx.x.r*\ASFB

and

C:\Program Files\Giddings & Lewis\Applications *vxx.x.r*\Examples

where vxx.x is the PiCPro for Windows version number that these ASFBs and examples were built under. The .r is the revision number of the Application software itself.

The .LIB files and source .LDO files are put in the ASFB subdirectory. The example .LDO files are put in the Examples subdirectory.

Revisions

The first four networks of each ASFB source ladder provide the following information:

Network 1

The first network just informs you that the ASFB is provided to assist your application development.

Network 2

The second network is used to keep a revision history of the ASFB. Revisions can be made by Giddings & Lewis personnel or by you.

The network identifies the ASFB, lists the requirements for using this ASFB, the name of the library the ASFB is stored in, and the revision history.

The revision history includes the date, ASFB version (see below), the version of PiCPro used while making the ASFB, and comments about what the revision involved.

When an ASFB is revised, the number of the first input (EN_ _ or RQ_ _) to the function block is changed in the software declarations table. The range of numbers available for Giddings & Lewis personnel is 00 to 49. The range of numbers available for you is 50 to 99. See chart below.

Revision	Giddings & Lewis revisions	User revisions
1st	EN00	EN50
2nd	EN01	EN51
.	.	.
.	.	.
.	.	.
50th	EN49	EN99

Network 3

The third network describes what you should do if you want to make a revision to the ASFB.

ASFB Input/Output Descriptions

Network 4

The fourth network describes the ASFB and defines all the inputs and outputs to the function block.

Using ASFBs

When you are ready to use the ASFB in your application, there are several approaches you can take as shown below.

- Create a new application LDO starting with the example LDO for the ASFB package. The advantage is that the software declarations table for the ASFB has been entered for you.
- If you already have an application LDO, copy and paste the example LDO into yours. The software declaration tables for both LDOs will also merge.

NOTES

CHAPTER 2 **General Purpose ASFBS**

These are the general purpose application specific function blocks. Included in this package are the following files.

NOTE: Every .LDO file on the CD has a corresponding .REM file. The REM files contain all the comments found in the LDO files. If you move an .LDO file to a different location, be sure to move its REM file to the same directory.

Communications ASFBS

G_COMM.LIB	Library which contains communications application specific function blocks
G_COMMEX.LDO	Example .LDO that uses the application specific function blocks in G_COMM library
G_CONFIG.LDO	Source file for ASFB that opens and configures a serial port
G_RCVSTR.LDO	Source file for ASFB that receives strings from a serial port
G_SNDSTR.LDO	Source file for ASFB that sends strings out a serial port

Data Type Conversion ASFBS

G_DATTYP.LIB	Library which contains data type conversion application specific function blocks
G_DTYPEX.LDO	Example .LDO that uses the application specific function blocks in G_DATTYP library
G_BN2STR.LDO	Source file for ASFB that converts a DWORD to a string, displaying the data in a binary format
G_BOO2DW.LDO	Source file for ASFB that converts 32 BOOLS to a DWORD
G_BOO2WD.LDO	Source file for ASFB that converts 16 BOOLS to a WORD
G_BY2BIT.LDO	Source file for ASFB that returns one bit of a BYTE
G_DW2BIT.LDO	Source file for ASFB that returns one bit of a DWORD
G_DW2BOO.LDO	Source file for ASFB that converts a DWORD to 32 BOOLS

G_HX2STR.LDO	Source file for ASFB that converts a DWORD to a string, displaying the data in a hex format
G_NM2STR.LDO	Source file for ASFB that converts a DINT to a string including decimal point and sign
G_WD2BIT.LDO	Source file for ASFB that returns one bit of a WORD
G_WD2BOO.LDO	Source file for ASFB that converts a WORD to 16 BOOLS

Evaluation ASFBs

G_EVAL.LIB	Library which contains evaluate application specific function blocks
G_EVALEX.LDO	Example .LDO that uses the application specific function blocks in G_EVAL library
G_BETWN.LDO	Source file for ASFB that sets an output if an input is in between two other inputs
G_GE_ALL.LDO	Source file for ASFB that sets an output if an input is greater than or equal to all of the other inputs
G_GE_ANY.LDO	Source file for ASFB that sets an output if an input is greater than or equal to any of the other inputs
G_LE_ALL.LDO	Source file for ASFB that sets an output if an input is less than or equal to all of the other inputs
G_LE_ANY.LDO	Source file for ASFB that sets an output if an input is less than or equal to any of the other inputs

RAMDISK File Manipulation ASFBs

G_FILE.LIB	Library which contains RAMDISK file manipulation application specific function blocks
G_FILEEX.LDO	Example .LDO that uses the application specific function blocks in G_FILE library
G_FILMNG.LDO	Source file for file manager ASFB for creating and editing recipes or part programs on the RAMDISK

G_READFL.LDO	Source file for ASFB that reads a file from the RAMDISK into a structure or array
G_WRITFL.LDO	Source file for ASFB that writes a file from a structure or array to the RAM-DISK

Miscellaneous ASFBS

G_MISC.LIB	Library which contains miscellaneous application specific function blocks
G_MISCEX.LDO	Example .LDO that uses the application specific function blocks in G_MISC library
G_C2F.LDO	Source file for ASFB that converts temperature from Celsius to Fahrenheit
G_D2ARMP.LDO	Source file for ASFB that produces an output command based on an input command and the allowable rate of change of the input
G_DEG2RD.LDO	Source file for ASFB that converts an angle in degrees to radians
G_DRMSEQ.LDO	Source file for ASFB that implements a drum or step sequencer
G_F2C.LDO	Source file for ASFB that converts temperature from Fahrenheit to Celsius
G_HSTGRM.LDO	Source file for ASFB that collects a contact histogram for up to 8 inputs
G_RD2DEG.LDO	Source file for ASFB that converts an angle in radians to degrees

Shift Register ASFBS

G_SHFTRG.LIB	Library which contains shift register application specific function blocks
G_SHFTEX.LDO	Example .LDO that uses the application specific function blocks in G_SHFTRG library
G_CHK_32.LDO	Source file for ASFB that returns number of highest or lowest bit set in array of 32 BOOLS
G_CHKBIT.LDO	Source file for ASFB that returns number of highest or lowest bit set in array of BOOLS
G_SHIFT.LDO	Source file for ASFB that shifts an array of BOOLS right or left
G_SHL_32.LDO	Source file for ASFB that performs a shift left on an array of 32 BOOLS

G_SHR_32.LDO Source file for ASFB that performs a shift right on an array of 32 BOOLS

Timer ASFBS

G_TIMER.LIB Library which contains timer application specific function blocks

G_TMREX.LDO Example .LDO that uses the application specific function blocks in G_TIMER library

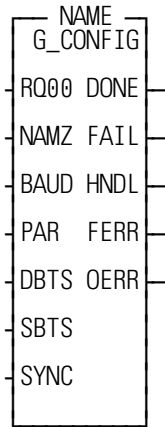
G_ELTMR.LDO Source file for ASFB that keeps track of the total elapsed time an input has been energized

G_FLTMR.LDO Source file for ASFB that flashes an output on and off when an input is energized

G_CONFIG

Configures serial port

USER/G_COMM



Inputs: RQ00 (BOOL) - set for one scan to enable execution
NAMZ (STRING) - the name of the port to open
BAUD (INT) - defines the baud rate of the port
PAR (USINT) - defines the parity of the port
DBTS (USINT) - defines the number of data bits
SBTS (USINT) - defines the number of stop bits
SYNC (USINT) - defines the synch mode of the port

Outputs: DONE (BOOL) - set when the port has been configured successfully
FAIL (BOOL) - set if an error occurred during configure
HNDL (INT) - the handle assigned to the port
FERR (INT) - error code from OPEN or CONFIG function
OERR (INT) - error code not from OPEN or CONFIG function

```
<<INSTANCE NAME>>:G_CONFIG(RQ00 := <<BOOL>>, NAMZ := <<STRING>>, BAUD := <<INT>>, PAR := <<USINT>>, DBTS := <<USINT>>, SBTS := <<USINT>>, SYNC := <<USINT>>, DONE => <<BOOL>>, FAIL => <<BOOL>>, HNDL => <<INT>>, FERR => <<INT>>, OERR => <<INT>>);
```

This function block opens and configures a serial port. This function block only needs to be called once, on the first scan of your application ladder. When it has finished executing, the send string and receive string application specific function blocks (G_SNDSTR and G_RCVSTR) can be used to read from and write to the serial port.

The string at the NAMZ input must either be 'USER:\$00' or the name used in the ASSIGN function for that device. The name must be followed by \$00.

The number entered at the BAUD input must be one of the following values: 110, 300, 600, 1200, 2400, 4800, 9600, or 19200.

The number entered at the PAR input must be one of the following values: 0 for no parity, 1 for even parity, or 2 for odd parity.

The number entered at the DBTS input must be 7 or 8.

The number entered at the SBTS input must be 1 or 2.

The number entered at the SYNC input must be one of the following values: 0 for no synch mode, 1 for send only, 2 for receive only, 3 for both send and receive, or 4 for hardware synch mode.

The HANDL output variable is used as an input to the send and receive application specific function blocks (G_SNDSTR and G_RCVSTR).

If an error occurred from the OPEN or CONFIG function, the error code will be stored in the FERR output. See appendix B of the PiC900 software manual for a description of these errors.

If an error occurred not from the OPEN or CONFIG function that prevented this function block from executing, an error code will be stored in the OERR output. A listing of these errors is shown below:

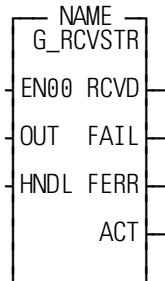
OERR Description

- 0** No error
- 1** BAUD input not an allowable value.
- 2** PAR input not 0, 1, or 2.
- 3** DBTS input not 7 or 8.
- 4** SBTS input not 1 or 2.
- 5** SYNC input greater than 4.

G_RCVSTR

Receives strings from serial port

USER/G_COMM



Inputs: EN00 (BOOL) - set continuously to enable execution
OUT (STRING) - the string read from the serial port
HNDL (INT) - the handle of the port

Outputs: RCVD (BOOL) - set for one scan when string was successfully read from the port
FAIL (BOOL) - set if an error occurred during read
FERR (INT) - error code from STATUS or READ function
ACT (INT) - the number of bytes read

```
<<INSTANCE NAME>>:G_RCVSTR(EN00 := <<BOOL>>, OUT :=  
    <<STRING>>, HNDL := <<INT>>, RCVD => <<BOOL>>, FAIL =>  
    <<BOOL>>, FERR => <<INT>>, ACT => <<INT>>);
```

This function block receives strings from a serial port.

The port must be opened and configured using the configure port application specific function block (G_CONFIG) before calling this function.

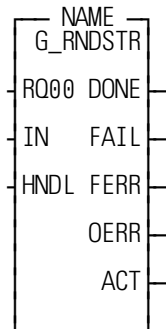
The HNDL input is the same as the output HNDL from the open and configure port function block (G_CONFIG).

If an error occurred from the STATUS or READ function, that error code will be stored in the FERR output. A complete listing of these error codes can be found in the Appendix B of the PiC900 software manual.

G_SNDSTR

Sends strings out serial port

USER/G_COMM



Inputs: RQ00 (BOOL) - set for one scan to initiate string send

IN (STRING) - the string to send

HNDL (INT) - the handle of the port

Outputs: DONE (BOOL) - reset when request is made, set when string was successfully sent out the port

FAIL (BOOL) - reset when request is made, set if an error occurred during send

FERR (INT) - error code from WRITE function block

OERR (INT) - error code for error not from WRITE function block.

ACT (INT) - the number of bytes sent

```
<<INSTANCE NAME>>:G_SNDSTR(RQ00 := <<BOOL>>, IN :=  
<<STRING>>, HNDL := <<INT>>, DONE => <<BOOL>>, FAIL =>  
<<BOOL>>, FERR => <<INT>>, OERR => <<INT>>, ACT => <<INT>>);
```

This function block sends a string out a serial port.

The serial port must be opened and configured using the configure port application specific function block (G_CONFIG) before calling this function.

The HNDL input is the same as the HNDL output from the open and configure port function block (G_CONFIG).

If an error occurred from the WRITE function that error code will be stored in the FERR output. A complete listing of these errors can be found in the Appendix B of the PiC900 software manual.

If an error occurred not from the WRITE function that prevented this function block from executing, that error code will be stored in the OERR output. A listing of these errors is shown below:

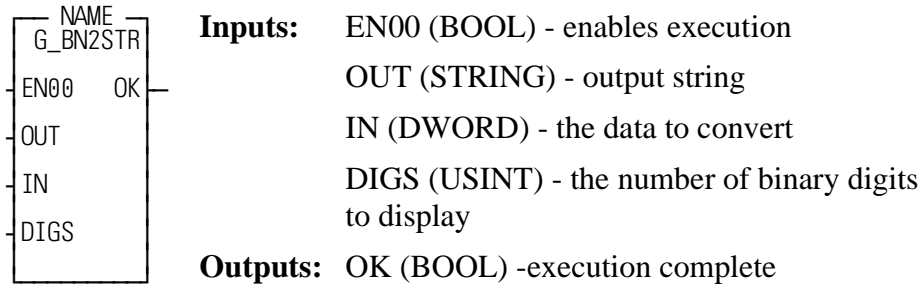
OERR Description

- | | |
|---|---|
| 0 | No error |
| 1 | A request was made and the previous request was not complete. |
| 2 | The length of the string at IN is zero. |

G_BN2STR

Converts DWORD to binary formatted

USER/G_DATTYP



```
<<INSTANCE NAME>>:G_BN2STR(EN00 := <<BOOL>>, OUT :=  
  <<STRING>>, IN := <<DWORD>>, DIGS := <<USINT>>, OK =>  
  <<BOOL>>);
```

This function block converts a double word to a string, displaying the double word in binary format.

The number entered at the DIGS input must be from 1 and 32. If it is not, then the OK will not be set.

EXAMPLES:

If IN = 16#F0F0 and DIGS = 16

then OUT = '1111000011110000'

If IN = 16#F0F0 and DIGS = 32 then

OUT = '00000000000000001111000011110000'

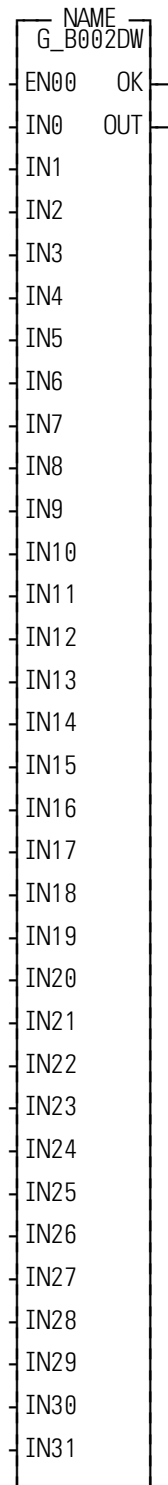
If IN = 16#3 and DIGS = 8

then OUT = '00000011'

G_BOO2DW

Converts 32 BOOLs to DWORD

USER/G_DATTYP



Inputs: EN00 (BOOL) - enables execution
IN0 (BOOL) - bit 0 of OUT (least significant bit)
IN1 (BOOL) - bit 1 of OUT
IN2 (BOOL) - bit 2 of OUT
IN3 (BOOL) - bit 3 of OUT
IN4 (BOOL) - bit 4 of OUT
IN5 (BOOL) - bit 5 of OUT
IN6 (BOOL) - bit 6 of OUT
IN7 (BOOL) - bit 7 of OUT
IN8 (BOOL) - bit 8 of OUT
IN9 (BOOL) - bit 9 of OUT
IN10 (BOOL) - bit 10 of OUT
IN11 (BOOL) - bit 11 of OUT
IN12 (BOOL) - bit 12 of OUT
IN13 (BOOL) - bit 13 of OUT
IN14 (BOOL) - bit 14 of OUT
IN15 (BOOL) - bit 15 of OUT
IN16 (BOOL) - bit 16 of OUT
IN17 (BOOL) - bit 17 of OUT
IN18 (BOOL) - bit 18 of OUT
IN19 (BOOL) - bit 19 of OUT
IN20 (BOOL) - bit 20 of OUT
IN21 (BOOL) - bit 21 of OUT
IN22 (BOOL) - bit 22 of OUT
IN23 (BOOL) - bit 23 of OUT
IN24 (BOOL) - bit 24 of OUT
IN25 (BOOL) - bit 25 of OUT
IN26 (BOOL) - bit 26 of OUT
IN27 (BOOL) - bit 27 of OUT
IN28 (BOOL) - bit 28 of OUT
IN29 (BOOL) - bit 29 of OUT
IN30 (BOOL) - bit 30 of OUT
IN31 (BOOL) - bit 31 of OUT (most significant bit)

Outputs: OK (BOOL) - execution complete
OUT (DWORD) - packed double word from inputs

```
<<INSTANCE NAME>>:G_BOO2DW(EN00 := <<BOOL>>, IN0 :=  
  <<BOOL>>, IN1 := <<BOOL>>, IN2 := <<BOOL>>, IN3 := <<BOOL>>, IN4  
  := <<BOOL>>, IN5 := <<BOOL>>, IN6 := <<BOOL>>, IN7 := <<BOOL>>,  
  IN8 := <<BOOL>>, IN9 := <<BOOL>>, IN10 := <<BOOL>>, IN11 :=  
  <<BOOL>>, IN12 := => <<BOOL>>, IN13 := <<BOOL>>, IN14 :=  
  <<BOOL>>, IN15 := <<BOOL>>, IN16 := <<BOOL>>, IN17 := <<BOOL>>,  
  IN18 := <<BOOL>>, IN19 := <<BOOL>>, IN20 := <<BOOL>>, IN2 :=  
  <<BOOL>>, IN22 := <<BOOL>>, IN23 := <<BOOL>>, IN24 := <<BOOL>>,  
  IN25 := <<BOOL>>, IN26 := <<BOOL>>, IN27 := <<BOOL>>, IN28 :=  
  <<BOOL>>, IN29 := <<BOOL>>, IN30 := <<BOOL>>, IN31 := <<BOOL>>,  
  OK => <<BOOL>>,OUT => <<DWORD>>);
```

EXAMPLES:

If IN0 is OFF and IN1 through IN31 are ON, then OUT will be 16#FFFFFFFE.

If IN15 is OFF and IN0 through IN30 are ON, then OUT will be 16#7FFFFFFF.

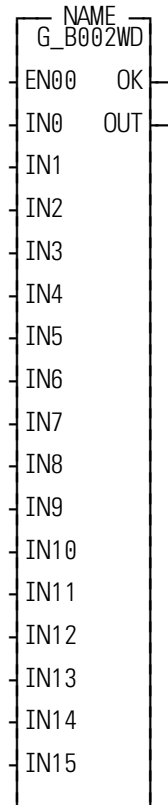
If IN0 is ON and IN1 through IN31 are OFF, then OUT will be 16#1.

If IN31 is ON and IN0 through IN30 are OFF, then OUT will be 16#80000000.

G_BOO2WD

Pack 16 BOOLs into WORD

USER/G_DATTYP



Inputs: EN00 (BOOL) - enables execution
IN0 (BOOL) - bit 0 of OUT (least significant bit)
IN1 (BOOL) - bit 1 of OUT
IN2 (BOOL) - bit 2 of OUT
IN3 (BOOL) - bit 3 of OUT
IN4 (BOOL) - bit 4 of OUT
IN5 (BOOL) - bit 5 of OUT
IN6 (BOOL) - bit 6 of OUT
IN7 (BOOL) - bit 7 of OUT
IN8 (BOOL) - bit 8 of OUT
IN9 (BOOL) - bit 9 of OUT
IN10 (BOOL) - bit 10 of OUT
IN11 (BOOL) - bit 11 of OUT
IN12 (BOOL) - bit 12 of OUT
IN13 (BOOL) - bit 13 of OUT
IN14 (BOOL) - bit 14 of OUT
IN15 (BOOL) - bit 15 of OUT (most significant bit)
Outputs: OK (BOOL) - execution complete
OUT (DWORD) - packed word with inputs

```
<<INSTANCE NAME>>:G_BOO2WD(EN00 := <<BOOL>>, IN0 :=  
<<BOOL>>, IN1 := <<BOOL>>, IN2 := <<BOOL>>, IN3 := <<BOOL>>, IN4  
:= <<BOOL>>, IN5 := <<BOOL>>, IN6 := <<BOOL>>, IN7 := <<BOOL>>,  
IN8 := <<BOOL>>, IN9 := <<BOOL>>, IN10 := <<BOOL>>, IN11 :=  
<<BOOL>>, IN12 := => <<BOOL>>, IN13 := <<BOOL>>, IN14 :=  
<<BOOL>>, IN15 := <<BOOL>>, OK => <<BOOL>>, OUT =>  
<<DWORD>>);
```

This function block packs 16 BOOLs into a WORD.

EXAMPLES:

If IN0 is OFF and IN1 through IN15 are ON, then OUT will be 16#FFFE.

If IN15 is OFF and IN0 through IN14 are ON, then OUT will be 16#7FFF.

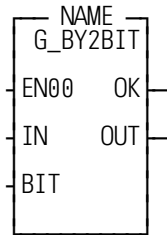
If IN0 is ON and IN1 through IN15 are OFF, then OUT will be 16#1.

If IN15 is ON and IN0 through IN14 are OFF, then OUT will be 16#8000.

G_BY2BIT

Return one bit of a BYTE

USER/G_DATTYP



Inputs: EN00 (BOOL) - enables execution

IN (BYTE) - input data

BIT (USINT) - the bit number to return

Outputs: OK (BOOL) - execution complete

OUT (BOOL) - the state of bit BIT in IN

```
<<INSTANCE NAME>>:G_BY2BIT(EN00 := <<BOOL>>, IN := <<BYTE>>,
  IN := <<DWORD>>, BIT := <<USINT>>, OK => <<BOOL>>, OUT =>
  <<BOOL>>);
```

This function block returns one bit of a BYTE variable.

The number entered at the BIT input must be between 0 and 7 or the OK output will not be set.

EXAMPLES

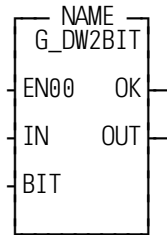
If IN=16#80 and BIT=7 then OUT will be 'ON'.

If IN=16#FE and BIT=0 then OUT will be 'OFF'.

G_DW2BIT

Return bit of a DWORD

USER/G_DATTYP



Inputs: EN00 (BOOL) - enables execution

IN (DWORD) - the input data

BIT (USINT) - the bit number to return

Outputs: OK (BOOL) - execution complete

OUT (BOOL) - the state of bit BIT in IN

```
<<INSTANCE NAME>>:G_DW2BIT(EN00 := <<BOOL>>, IN :=  
<<DWORD>>, BIT := <<USINT>>, OK => <<BOOL>>, OUT =>  
<<BOOL>>);
```

This function block returns one bit of a DWORD variable.

The number entered at the BIT input must be between 0 and 31 or the function will not execute and the OK output will not be set.

EXAMPLES:

If IN = 16#80000000 and BIT = 31 then OUT will be 'ON'.

If IN = 16#FFFFFFFE and BIT = 0 then OUT will be 'OFF'.

G_DW2BOO

Converts DWORD to 32 BOOLS

USER/G_DATTYP

NAME		
G_DW2B00		
EN00	OK	
IN	00	
	01	
	02	
	03	
	04	
	05	
	06	
	07	
	08	
	09	
	010	
	011	
	012	
	013	
	014	
	015	
	016	
	017	
	018	
	019	
	020	
	021	
	022	
	023	
	024	
	025	
	026	
	027	
	028	
	029	
	030	
	031	

Inputs: EN00 (BOOL) - enables execution
IN (DWORD) - the data to convert

Outputs: OK (BOOL) - execution complete

O0 (BOOL) - bit 0 of IN (least significant bit)
O1 (BOOL) - bit 1 of IN
O2 (BOOL) - bit 2 of IN
O3 (BOOL) - bit 3 of IN
O4 (BOOL) - bit 4 of IN
O5 (BOOL) - bit 5 of IN
O6 (BOOL) - bit 6 of IN
O7 (BOOL) - bit 7 of IN
O8 (BOOL) - bit 8 of IN
O9 (BOOL) - bit 9 of IN
O10 (BOOL) - bit 10 of IN
O11 (BOOL) - bit 11 of IN
O12 (BOOL) - bit 12 of IN
O13 (BOOL) - bit 13 of IN
O14 (BOOL) - bit 14 of IN
O15 (BOOL) - bit 15 of IN
O16 (BOOL) - bit 16 of IN
O17 (BOOL) - bit 17 of IN
O18 (BOOL) - bit 18 of IN
O19 (BOOL) - bit 19 of IN
O20 (BOOL) - bit 20 of IN
O21 (BOOL) - bit 21 of IN
O22 (BOOL) - bit 22 of IN
O23 (BOOL) - bit 23 of IN
O24 (BOOL) - bit 24 of IN
O25 (BOOL) - bit 25 of IN
O26 (BOOL) - bit 26 of IN
O27 (BOOL) - bit 27 of IN
O28 (BOOL) - bit 28 of IN
O29 (BOOL) - bit 29 of IN
O30 (BOOL) - bit 30 of IN
O31 (BOOL) - bit 31 of IN (most significant bit of IN)

```
<<INSTANCE NAME>>:G_DW2BOO(EN00 := <<BOOL>>, IN :=  
  <<DWORD>>, OK => <<BOOL>>, O0 => <<BOOL>>, O1 => <<BOOL>>,  
  O2 => <<BOOL>>, O3 => <<BOOL>>, O4 => <<BOOL>>, O5 =>  
  <<BOOL>>, O6 => <<BOOL>>, O7 => <<BOOL>>, O8 => <<BOOL>>, O9  
  => <<BOOL>>, O10 => <<BOOL>>, O11 => <<BOOL>>, O12 =>  
  <<BOOL>>, O13 => <<BOOL>>, O14 => <<BOOL>>, O15 => <<BOOL>>,  
  O16 => <<BOOL>>, O17 => <<BOOL>>, O18 => <<BOOL>>, O19 =>  
  <<BOOL>>, O20 => <<BOOL>>, O21 => <<BOOL>>, O22 => <<BOOL>>,  
  O23 => <<BOOL>>, O24 => <<BOOL>>, O25 => <<BOOL>>, O26 =>  
  <<BOOL>>, O27 => <<BOOL>>, O28 => <<BOOL>>, O29 => <<BOOL>>,  
  O30 => <<BOOL>>, O31 => <<BOOL>>);
```

This function block converts a DWORD to 32 BOOLs

EXAMPLES:

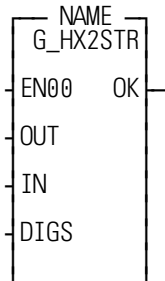
If IN = 16#FFFFFFFFE then O0 will be OFF and O1 through O31 will be ON.

If IN = 16#80000000 then O31 will be ON and O0 through O30 will be OFF.

G_HX2STR

Converts DWORD to hex string

USER/G_DATTYP



Inputs: EN00 (BOOL) - enables execution
OUT (STRING) - output string
IN (DWORD) - the data to convert
DIGS (USINT) - the number of hex digits to display

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:G_HX2STR(EN00 := <<BOOL>>, OUT :=  
  <<STRING>>, IN := <<DWORD>>, DIGS := <<USINT>>, OK =>  
  <<BOOL>>);
```

This function block converts a DWORD to a hex formatted string.

The value at DIGS must be between 1 and 8 or this function will not execute and the OK will not be set.

EXAMPLES:

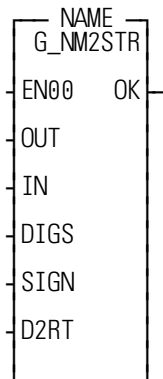
If IN = 16#ABCDE and DIGS = 8 then OUT will be '000ABCDE'.

If IN = 16#FEDCBA and DIGS = 2 then OUT will be 'BA'.

G_NM2STR

Converts DINT to formatted string

USER/G_DATTYP



Inputs: EN00 (BOOL) - enables execution
OUT (STRING) - output string
IN (DINT) - defines the data to be converted
DIGS (USINT) - defines the maximum number of digits for the number
SIGN (USINT) - defines if a + or - sign should be placed in front of the number
D2RT (USINT) = defines the number of digits to the right of the decimal point

Outputs: OK (BOOL) - execution complete

```
<<INSTANCE NAME>>:G_NM2STR(EN00 := <<BOOL>>, OUT :=  
<<STRING>>, IN := <<DINT>>, DIGS := <<USINT>>, SIGN := <<USINT>>,  
D2RT => <<USINT>>, OK => <<BOOL>>);
```

This function block converts a DINT to a formatted string.

The number entered at the DIGS input is the maximum number of digits the number can have, not including decimal point or sign. If the value of IN is too large to fit into the number of digits at DIGS, the output string will be all @ signs.

The number entered at the SIGN input should be a 0 if no sign should be placed in front of the number, or a 1 if a + or - sign should be placed in front of the number in the output string. If SIGN is 0 and the number at IN is negative, then the OUT string will be all @ signs.

The number entered at the D2RT input should be a 0 if no decimal point should be displayed, or the number of digits desired to the right of the decimal point.

EXAMPLES:

If IN = 1234567, DIGS = 7, SIGN = 1, and D2RT = 4 then OUT will be '+123.4567'.

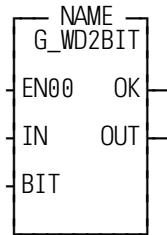
If IN = -123, DIGS = 7, SIGN = 1, and D2RT = 4, then OUT will be '-.0123'.

If IN = 567, DIGS = 5, SIGN = 0, and D2RT = 0 then OUT will be ' 567'.

G_WD2BIT

Returns one bit of a WORD

USER/G_DATTYP



Inputs: EN00 (BOOL) - enables execution
IN (WORD) - input data
BIT (USINT) - the bit number to return

Outputs: OK (BOOL) - execution complete
OUT (BOOL) - the state of BIT in IN

```
<<INSTANCE NAME>>:G_WD2BIT(EN00 := <<BOOL>>, IN := <<WORD>>,
    BIT := <<USINT>>, OK => <<BOOL>>, OUT => <<BOOL>>);
```

This function block returns one bit of a WORD variable.

The value entered at BIT must be from 0 to 15 for this function to execute.

EXAMPLES:

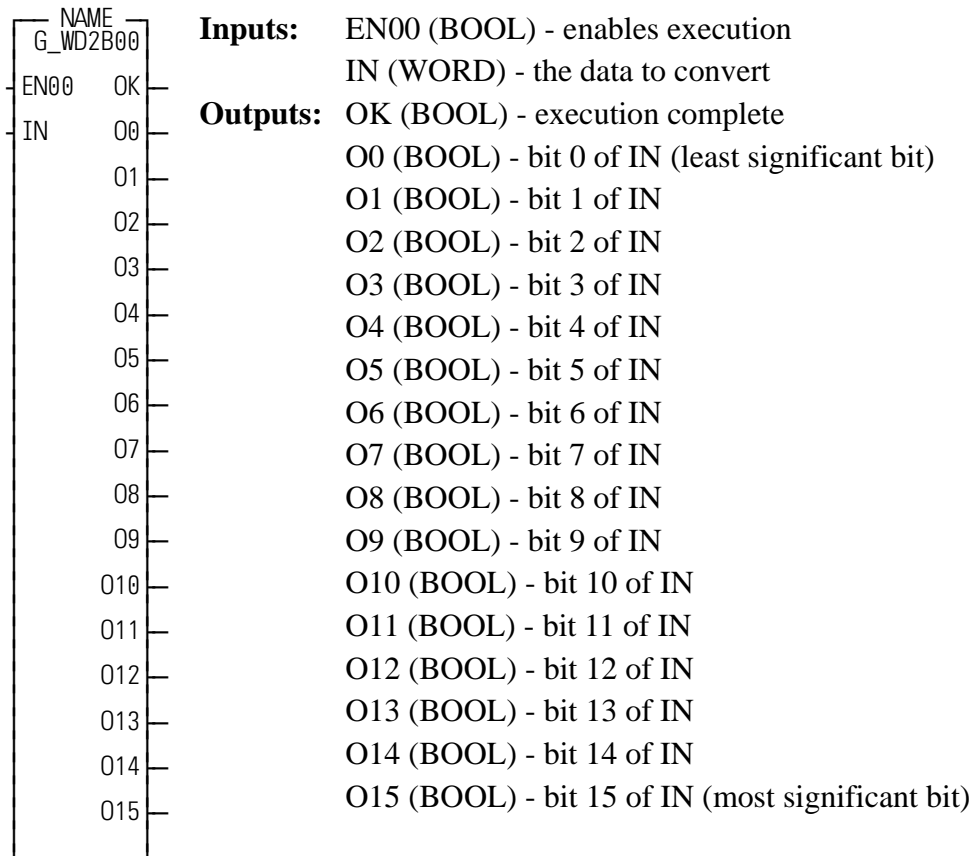
If IN = 16#8000 and BIT = 15, then OUT will be ON.

If IN = 16#FFFE and BIT = 0, then OUT will be OFF.

G_WD2BOO

Converts WORD to 16 BOOLs

USER/G_DATTYP



```
<<INSTANCE NAME>>:G_WD2BOO(EN00 := <<BOOL>>, IN :=  
<<WORD>>, OK => <<BOOL>>, O0 => <<BOOL>>, O1 => <<BOOL>>, O2  
=> <<BOOL>>, O3 => <<BOOL>>, O4 => <<BOOL>>, O5 => <<BOOL>>, O6  
=> <<BOOL>>, O7 => <<BOOL>>, O8 => <<BOOL>>, O9 => <<BOOL>>,  
O10 => <<BOOL>>, O11 => <<BOOL>>, O12 => <<BOOL>>, O13 =>  
<<BOOL>>, O14 => <<BOOL>>, O15 => <<BOOL>>);
```

EXAMPLES:

This function block converts a WORD to 16 BOOLs.

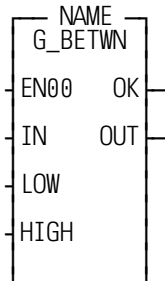
If IN = 16#8000, then O15 will be ON and O0 through O14 will be OFF.

If IN = 16#FFFE, then O0 will be OFF and O1 through O15 will be ON.

G_BETWN

Check for in between

USER/G_EVAL



Inputs: EN00 (BOOL) - enables execution
IN (DINT) - defines the input to be compared to the other inputs
LOW (DINT) - defines the lower limit for comparison
HIGH (DINT) - defines the upper limit for comparison

Outputs: OK (BOOL) - execution complete
OUT (BOOL) - set if IN is in between LOW and HIGH

```
<<INSTANCE NAME>>:G_BETWN(EN00 := <<BOOL>>, IN := <<DINT>>,
  LOW := <<DINT>>, HIGH := <<DINT>>, OK => <<BOOL>>, OUT =>
  <<BOOL>>);
```

This function block compares an input with two other inputs and sets an output if the first input is in between the other two.

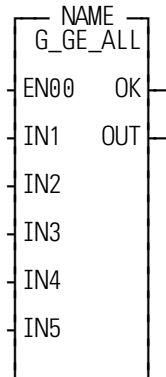
If $LOW \leq IN \leq HIGH$, then OUT will be set.

If $IN < LOW$ or if $IN > HIGH$, then OUT will be reset.

G_GE_ALL

Greater than or equal to all

USER/G_EVAL



Inputs: EN00 (BOOL) -enables execution
IN1 (DINT) - this input will be compared to all the other inputs
IN2 (DINT) - first input to compare to IN1
IN3 (DINT) - second input to compare to IN1
IN4 (DINT) - third input to compare to IN1
IN5 (DINT) - fourth input to compare to IN1

Outputs: OK (BOOL) - execution complete
OUT (BOOL) - ON or OFF depending on result of comparison

```
<<INSTANCE NAME>>:G_GE_ALL(EN00 := <<BOOL>>, IN1 := <<DINT>>,
  IN2 := <<DINT>>, IN3 := <<DINT>>, IN4 := <<DINT>>, IN5 := <<DINT>>,
  OK => <<BOOL>>, OUT => <<BOOL>>);
```

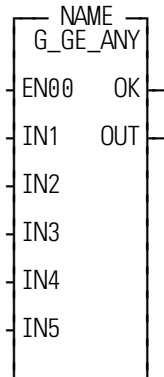
This function block compares the first input with all of the other inputs and sets an output if the first input is greater than or equal to all of the other inputs. If IN1 is greater than or equal to IN2, IN3, IN4, and IN5, then OUT will be ON.

If IN1 is less than IN2, IN3, IN4, or IN5, then OUT will be OFF.

G_GE_ANY

Greater than or equal to any

USER/G_EVAL



Inputs: EN00 (BOOL) -enables execution
IN1 (DINT) - this input will be compared to all the other inputs
IN2 (DINT) - first input to compare to IN1
IN3 (DINT) - second input to compare to IN1
IN4 (DINT) - third input to compare to IN1
IN5 (DINT) - fourth input to compare to IN1

Outputs: OK (BOOL) - execution complete
OUT (BOOL) - ON or OFF depending on result of comparison

```
<<INSTANCE NAME>>:G_GE_ANY(EN00 := <<BOOL>>, IN1 := <<DINT>>,
  IN2 := <<DINT>>, IN3 := <<DINT>>, IN4 := <<DINT>>, IN5 := <<DINT>>,
  OK => <<BOOL>>, OUT => <<BOOL>>);
```

This function block compares the first input with all of the other inputs and sets an output if the first input is greater than or equal to any of the other inputs.

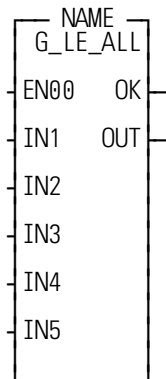
If IN1 is greater than or equal to IN2, IN3, IN4, or IN5, then OUT will be ON.

If IN1 is less than IN2, IN3, IN4 and IN5, then OUT will be OFF.

G_LE_ALL

Less than or equal to all

USER/G_EVAL



Inputs: EN00 (BOOL) -enables execution
IN1 (DINT) - this input will be compared to all the other inputs
IN2 (DINT) - first input to compare to IN1
IN3 (DINT) - second input to compare to IN1
IN4 (DINT) - third input to compare to IN1
IN5 (DINT) - fourth input to compare to IN1

Outputs: OK (BOOL) - execution complete
OUT (BOOL) - ON or OFF depending on result of comparison

```
<<INSTANCE NAME>>:G_LE_ALL(EN00 := <<BOOL>>, IN1 := <<DINT>>,
  IN2 := <<DINT>>, IN3 := <<DINT>>, IN4 := <<DINT>>, IN5 := <<DINT>>,
  OK => <<BOOL>>, OUT => <<BOOL>>);
```

This function block compares the first input with all of the other inputs and sets an output if the first input is less than or equal to all of the other inputs.

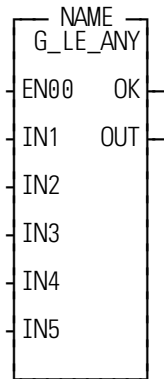
If IN1 is less than or equal to IN2, IN3, IN4, and IN5 then OUT will be ON.

If IN1 is greater than IN2, IN3, IN4 or IN5 then OUT will be OFF.

G_LE_ANY

Less than or equal to any

USER/G_EVAL



Inputs: EN00 (BOOL) - enables execution
IN1 (DINT) - this input will be compared to all the other inputs
IN2 (DINT) - first input to compare to IN1
IN3 (DINT) - second input to compare to IN1
IN4 (DINT) - third input to compare to IN1
IN5 (DINT) - fourth input to compare to IN1

Outputs: OK (BOOL) - execution complete
OUT (BOOL) - ON or OFF depending on result of comparison

```
<<INSTANCE NAME>>:G_LE_ANY(EN00 := <<BOOL>>, IN1 := <<DINT>>,
  IN2 := <<DINT>>, IN3 := <<DINT>>, IN4 := <<DINT>>, IN5 := <<DINT>>,
  OK => <<BOOL>>, OUT => <<BOOL>>);
```

This function block compares the first input with all of the other inputs and sets an output if the first input is less than or equal to any of the other inputs.

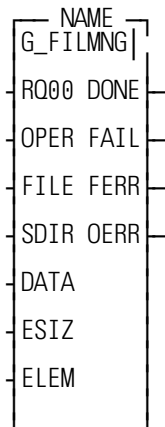
If IN1 is less than or equal to IN2, IN3, IN4, or IN5, then OUT will be ON.

If IN1 is greater than IN2, IN3, IN4, and IN5, then OUT will be OFF.

G_FILMNG

Data list file manager

USER/G_FILE



Inputs: RQ00 (BOOL) - one-shot to start any operation
OPER (USINT) - requested file operation
FILE (STRING) - file name to edit
SDIR (STRING) - the RAMDISK subdirectory where the file is located
DATA (STRUCT) - data to be edited or read
ESIZ (INT) - Size of DATA structure in bytes
ELEM (INT) - element number to read, replace, insert after, insert before or delete

Outputs: DONE (BOOL) - set if file operation completes successfully
FAIL (BOOL) - set if file operation is terminated due to an error condition
FERR (INT) - error code from OPEN, READ, or CLOSE function blocks
OERR (INT) - error code not from OPEN, READ, or CLOSE function blocks

```
<<INSTANCE NAME>>:G_FILMNG(RQ00 := <<BOOL>>, OPER :=  
  <<USINT>>, FILE := <<STRING>>, SDIR := <<STRING>>, DATA :=  
  <<MEMORY AREA>>, ESIZ := <<INT>>, ELEM := <<INT>>, DONE =>  
  <<BOOL>>, FAIL => <<BOOL>>, FERR => <<INT>>, OERR => <<INT>>);
```

The data list file manager is used for creating and editing recipes or part programs on the PiC900 RAMDISK.

Each part program or recipe is stored as an individual file on the RAMDISK. The file name is specified at the FILE input. The name can be up to eight characters, with an extension of up to three characters. The last character of the FILE string must be a \$00. EXAMPLE: '\FILENAME.EXT\$00

If the file is in a subdirectory on the RAMDISK, then the SDIR input defines the name of the subdirectory. The name can be up to eight characters, and can not have an extension. EXAMPLE: 'SUBDIR'.

NOTE: If you want the file to be placed in the main directory, then enter a string with no initial value at the SDIR input.

Each file consists of one or more elements.

Each element in the file must be the same size and have the same format. The format for an element is defined by creating a structure. That structure is then placed at the DATA input of this function block.

To request an operation the RQ00 input must be one-shot and the operation desired must be placed in the OPER input. A table of allowable operations is shown below:

OPER	Description
1	Create new file with one element.
2	Delete existing file.
20	Read element from file into DATA structure.
21	Replace element in file with new data from DATA structure.
22	Insert new element from DATA structure into file after element in ELEM.
23	Insert new element from DATA structure into file before element in ELEM.
24	Delete element specified by ELEM in file.

The data that is being inserted into the file, or read from the file, is always stored in the DATA structure. The DATA structure must be of a fixed format. The size of the DATA structure in bytes must always be present at the ESIZ input. This tells the function block how many bytes to read or write to the file.

The ELEM input tells the function which element in the file to insert after, insert before, read, replace, or delete. The elements of the file are numbered beginning with element 0. The DONE output will be cleared when RQ00 is energized, and then set if the requested operation completed successfully. The FAIL output will be cleared when RQ00 is energized, and then set if the requested operation is terminated due to an error condition.

The FERR output will be cleared when RQ00 is energized. If an error occurs during the execution of an OPEN, READ, SEEK, CLOSE, or WRITE function block that prevented the requested operation from completing, the error code will be stored in the FERR output. A complete listing of these errors can be found in the Appendix B of the PiC900 Software Manual.

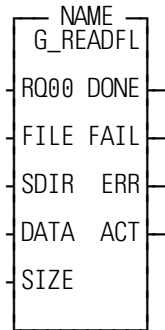
The OERR output will be cleared when RQ00 is energized. If an error is detected which prevents the requested operation from being attempted, OERR is set to a value indicating what error has been detected. A listing of these errors follows:

OERR	Description
0	No error.
1	OPER is not a valid number.
3	Seek past end occurred during element read.
4	File I/O error occurred during element insert operation.
10	New file operation requested and file already exists.

G_READFL

Reads file from RAMDISK

USER/G_FILE



Inputs: RQ00 (BOOL) - requests file read
FILE (STRING) - the file name
SDIR (STRING) - the RAMDISK subdirectory where the file is located
DATA (STRUCTURE OR ARRAY) - defines where the data read from the file will be placed
SIZE (INT) - defines the number of bytes to read

Outputs: DONE (BOOL) - reset when file read requested, set when read file is complete with no error
FAIL (BOOL) - reset when file read requested, set instead of the DONE output if file read failed
ERR (INT) - error number that occurred during file read
ACT (INT) - the number of bytes read

```
<<INSTANCE NAME>>:G_READFL(RQ00 := <<BOOL>>, FILE :=  
<<STRING>>, SDIR := <<STRING>>, DATA := <<STRING>>, SIZE :=  
<<INT>>, DONE => <<BOOL>>, FAIL => <<BOOL>>, ERR => <<INT>>,  
ACT => <<INT>>);
```

This function block reads a file from RAMDISK into a structure or array.

The RQ00 input must be one-shot to initiate the file read.

The file name can be up to eight characters with a three character extension.

EXAMPLE: FILENAME.EXT

If the file is in a subdirectory on the RAMDISK, then the SDIR input defines the name of the subdirectory. The name can be up to eight characters, and can not have an extension. EXAMPLE: 'SUBDIR'

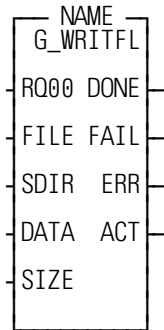
NOTE: If you want the file to be placed in the main directory, then enter a string with no initial value at the SDIR input.

If an error occurs in reading the file, an error code will be stored in the ERR output. A complete listing of error codes can be found in the Appendix B of the PiC900 software manual.

G_WRITFL

Writes file to RAMDISK

USER/G_FILE



Inputs: RQ00 (BOOL) - requests file write
FILE (STRING) - the file name
SDIR (STRING) - defines the subdirectory on the RAMDISK where the file is located
DATA (STRUCT or ARRAY) - the data to write to the RAMDISK
SIZE (INT) - defines the number of bytes to write to the file

Outputs: DONE (BOOL) - reset when file write is requested, set if file write completed successfully
FAIL (BOOL) - reset when file write is requested, set if an error occurred during file write
ERR (INT) - the error number that occurred
ACT (INT) - the number of bytes written to the file

```
<<INSTANCE NAME>>:G_WRITFL(RQ00 := <<BOOL>>, FILE :=  
  <<STRING>>, SDIR := <<STRING>>, DATA := <<MEMORY AREA>>,  
  SIZE := <<INT>>, DONE => <<BOOL>>, FAIL => <<BOOL>>, ERR =>  
  <<INT>>, ACT => <<INT>>);
```

This function block writes a file to the RAMDISK from a structure or array.

The RQ00 input must be one-shot to initiate the file write.

The file name can be up to eight characters with a three character extension.

EXAMPLE: FILENAME.EXT

If the file is in a subdirectory on the RAMDISK, then the SDIR input defines the name of the subdirectory. The name can be up to eight characters, and can not have an extension.

EXAMPLE: SUBDIR

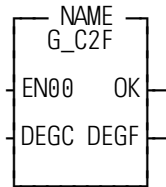
NOTE: If you want the file to be placed in the main directory, then enter a string with no initial value at the SDIR input.

If an error occurs in reading the file, an error code will be stored in the ERR output. A complete listing of error codes can be found in the Appendix B of the PiC900 software manual.

G_C2F

Celsius to Fahrenheit

USER/G_MISC



Inputs: EN00 (BOOL) - enables execution

DEGC (DINT) - the temperature in Celsius

Outputs: OK (BOOL) - execution complete

DEGF (DINT) - the temperature in degrees Fahrenheit

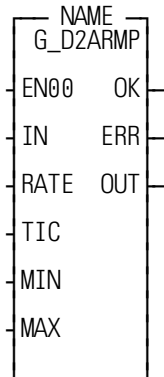
```
<<INSTANCE NAME>>:G_C2F(EN00 := <<BOOL>>, DEGC := <<DINT>>  
OK => <<BOOL>>, DEGF => <<DINT>>);
```

This function block converts temperature from Celsius to Fahrenheit.

G_D2ARMP

D/A with ramping

USER/G_MISC



Inputs: EN00 (BOOL) - enables execution
IN (INT) - the input command
RATE (INT) - the maximum number of counts that OUT can change by every time tick
TIC (TIME) - the time tick
MIN (INT) - the OUT value will never be allowed to be less than MIN
MAX (INT) - the OUT value will never be allowed to be greater than MAX

Outputs: OK (BOOL) - execution complete
ERR (INT) - error number
OUT (INT) -the output command

```
<<INSTANCE NAME>>:G_D2ARMP(EN00 := <<BOOL>>, IN := <<INT>>,
  RATE := <<INT>>, TIC := <<TIME>>, MIN := <<INT>>, MAX := <<INT>>,
  OK => <<BOOL>>, ERR => <<INT>>, OUT => <<INT>>);
```

This function block produces an output command based on the input command and the allowable rate of change of the input. The output of this function block would be tied directly to the input of the analog output function. The EN00 input of this function block should be set every scan.

This function will not execute and the OK output will not be set if any of the following error conditions are present: MAX is less than or equal to MIN, RATE is less than or equal to zero, TIC is equal to zero, or if an error occurs in the calculations for OUT. If the OK is not set, the ERR output will hold a code describing the error that occurred. A table of these errors is listed below:

ERR	Description
0	No error.
1	MAX is less than or equal to MIN.
2	Rate is less than or equal to zero.
3	TIC is zero.
4	An error occurred in calculating OUT.

EXAMPLES:

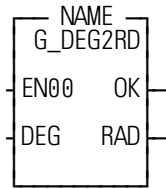
If OUT = 0 , RATE = 1000, TIC = 1 sec, MIN = -32767, and MAX = 32767 and IN is changed to 10000, then the value of OUT will be 1000 after 1 sec, 2000 after 2 sec, 3000 after 3 sec, and 10000 after 10 seconds have elapsed.

If OUT = 10000, RATE = 1000, TIC = 1 sec, MIN = -32767, and MAX = 32767, and IN is changed to 5600, then the value of OUT will be 9000 after 1 sec, 8000 after 2 sec, 7000 after 3 sec, 6000 after 4 sec, and then 5600 after 5 seconds have elapsed.

G_DEG2RD

Converts degrees to radians

USER/G_MISC

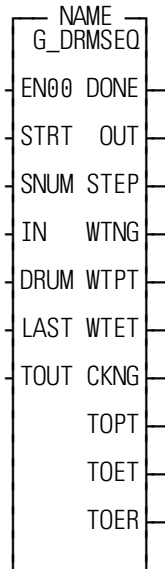


Inputs: EN00 (BOOL) - enables execution
DEG (REAL) - the angle in degrees

Outputs: OK (BOOL) - execution complete
RAD (REAL) - the angle in radians

```
<<INSTANCE NAME>>:G_DEG2RD(EN00 := <<BOOL>>, DEG :=  
<<REAL>>, OK => <<BOOL>>, RAD => <<REAL>>);
```

This function block converts an angle in degrees to radians.



Inputs: EN00 (BOOL) - enables execution
STRT (BOOL) - start sequence
SNUM (UINT) - starting step number
IN (DWORD) - user defined inputs
DRUM (ARRAY of STRUCT) - structure to define sequence pattern
LAST (UINT) - number of steps in sequence
TOUT (BOOL) - enables/disables timeout option

Outputs: DONE (BOOL) - sequence complete
OUT (DWORD) - user defined outputs
STEP (UINT) - active step number
WTNG (BOOL) - set while wait time elapses
WTPT (TIME) - preset wait time for this step
WTET (TIME) - elapsed wait time for this step
CKNG (BOOL) - set while checking input states
TOPT (TIME) - preset timeout time for this step
TOET (TIME) - elapsed timeout time for this step
TOER (BOOL) - timeout error occurred

```
<<INSTANCE NAME>>:G_DRMSEQ(EN00 := <<BOOL>>, STRT :=  
<<BOOL>>, SNUM := <<UINT>>, IN := <<DWORD>>, DRUM := <<MEM-  
ORY AREA>> LAST := <<UINT>>, TOUT := <<BOOL>>, DONE =>  
<<BOOL>>, OUT => <<DWORD>>, STEP => <<UINT>>, WTNG =>  
<<BOOL>>, WTPT => <<TIME>>, CKNG => <<BOOL>>, TOPT =>  
<<TIME>>, TOET => <<TIME>>, TOER => <<BOOL>>);
```

This function block sequences through an array of output patterns. Sequencing is accomplished by matching a user-defined input pattern and by a user-defined step timer.

This function block operates like a drum sequencer. A simple example of a drum sequencer is a rotary drum music box. A music box uses raised portions of the drum to play the notes of the song as the drum turns. The musical notes are the outputs of the sequence. The song is comprised of many steps. Each step of the song requires different output notes. Sequencing through the notes is accomplished by having the listener turn the crank on the side of the music box.

The crank is an input to the sequence. It determines when the music box advances to the next step of notes.

This function block operates in a similar fashion. The sequence is defined by the programmer. The programmer defines how many steps are in the sequence, the output for each step, and the input and/or time delay required to advance to the next step.

The function block input which defines each step is the DRUM input. The DRUM input is an array of structures. The DRUM structure has five members. It must be declared as follows:

DRUM	STRUCT(0..???)
.OUTPUTS	DWORD
.WAITTIME	TIME
.INPUTS	DWORD
.MASK	DWORD
.TIMEOUT	TIME

The number of elements in the DRUM array should match the number of steps in the sequence. For example, if the sequence has 35 steps, the DRUM array should have 35 elements.

The output for each step is defined in the .OUTPUTS member of the DRUM structure. For example, the output for step #2 would be defined in DRUM(2).OUTPUTS. The DRUM(2).OUTPUTS value will appear at OUT when step #2 is active.

The conditions required to advance to the next step in the sequence are defined in the .INPUTS, .MASK, .WAITTIME and .TIMEOUT members of the DRUM structure.

In order to advance to the next step of the sequence, two conditions must be met. First, the wait timer must elapse (the duration of the wait timer is defined by the .WAITTIME member of the DRUM structure). Second, the value at IN must satisfy the input conditions for the current step. NOTE: The input conditions will not be checked until the wait timer has elapsed.

After the wait time has elapsed, the inputs will be checked. If the TOUT input is energized, then the inputs will only be checked for the amount of time specified in the .TIMEOUT member of the DRUM structure. After this timeout time has elapsed, if the inputs are not in the correct state, then a timeout error will be set. If the TOUT input is not energized, then the inputs will be checked forever until the input conditions are satisfied.

The input conditions are defined by the .INPUTS and the .MASK members of the DRUM structure. The .MASK member defines which of the 32 bits of IN and the .INPUTS member must match. The .INPUTS member defines what the state of these bits should be.

For example, if step #1 of the sequence is active and the data for step #1 is...

```
DRUM(1)
.OUTPUTS = 16#FF00 0000 = 2#1111 1111 0000 0000 0000 0000 0000 0000
.WAITTIME = T#3s
.INPUTS = 16#0000 0009 = 2#0000 0000 0000 0000 0000 0000 0000 1001
.MASK = 16#0000 000F = 2#0000 0000 0000 0000 0000 0000 0000 1111
.TIMEOUT = T#5s
```

The DRUM(1).MASK value of 16#0000 000F indicates that the four least significant bits of DRUM(1).INPUTS must match the four least significant bits of IN. When these bits match the next step will be enabled.

So, when IN =16#0000 0029 =2#0000 0000 0000 0000 0000 0000 0010
1001

the four least significant bits match, and the next step will be enabled.

The STRT input is used to start or reset the sequence. The sequence will begin at the step number specified at the SNUM input.

The number of steps in the sequence is specified at the LAST input. When the last step of the sequence is completed the DONE output will be energized.

The 32-bit input DWORD is specified at IN.

The step number active is present at the STEP output.

The 32-bit output for the active step is present at the OUT output.

When a new step is begun, the outputs will be set to their new state and the WTNG output will be set until the wait time has elapsed. The preset wait time is present at the WTPT output, and the elapsed wait time is present at the WTET output.

After the wait timer has elapsed, the CKNG output will be set .

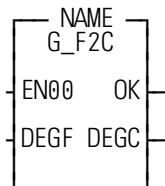
If the TOUT input is energized, the inputs will be checked for the amount of time in TOPT. If the inputs are not in the correct state by the time TOPT elapses, then the TOER (timeout error) output will be set. If the inputs are in the correct state before TOPT elapses, then the sequence will advance to the next step.

If the TOUT input is not energized, then the CKNG output will remain on until the inputs have satisfied the input condition specified for the active step. When the inputs are in the correct state, then the sequence will advance to the next step.

G_F2C

Fahrenheit to Celsius

USER/G_MISC



Inputs: EN00 (BOOL) - enables execution
DEGF (DINT) - the temperature in degrees Fahrenheit

Outputs: OK(BOOL) - execution complete
DEGC (DINT) - the temperature in Celsius

```
<<INSTANCE NAME>>:G_F2C(EN00 := <<BOOL>>, DEGF := <<DINT>>, OK => <<BOOL>>, DEGC => <<DINT>>);
```

This function block converts temperature in degrees Fahrenheit to Celsius.

G_HSTGRM

Contact histogram

USER/G_MISC

NAME	
G_HSTGRM	
EN00	OK
CLCT	COL1
SNAP	SNP1
IN1	COL2
IN2	SNP2
IN3	COL3
IN4	SNP3
IN5	SNP3
IN5	COL4
IN6	SNP4
IN7	COL5
IN8	SNP5
	COL6
	SNP6
	COL7
	SNP7
	COL8
	SNP8

Inputs: EN00 (BOOL) - enables execution
CLCT (BOOL) - data collection input
SNAP (BOOL) - data snapshot input
IN1 (BOOL) - first input to collect data for
IN2 (BOOL) - second input to collect data for
IN3 (BOOL) - third input to collect data for
IN4 (BOOL) - fourth input to collect data for
IN5 (BOOL) - fifth input to collect data for
IN6 (BOOL) - sixth input to collect data for
IN7 (BOOL) - seventh input to collect data for
IN8 (BOOL) - eighth input to collect data for

Outputs: OK (BOOL) - execution complete
COL1 (DWORD) - collected data for IN1
SNP1 (DWORD) - snapshot data for IN1
COL2 (DWORD) - collected data for IN2
SNP2 (DWORD) - snapshot data for IN2
COL3 (DWORD) - collected data for IN3
SNP3 (DWORD) - snapshot data for IN3
COL4 (DWORD) - collected data for IN4
SNP4 (DWORD) - snapshot data for IN4
COL5 (DWORD) - collected data for IN5
SNP5 (DWORD) - snapshot data for IN5
COL6 (DWORD) - collected data for IN6
SNP6 (DWORD) - snapshot data for IN6
COL7 (DWORD) - collected data for IN7
SNP7 (DWORD) - snapshot data for IN7
COL8 (DWORD) - collected data for IN8
SNP8 (DWORD) - snapshot data for IN8

```
<<INSTANCE NAME>>:G_HSTGRM(EN00 := <<BOOL>>, CLCT :=  
<<BOOL>>, SNAP := <<BOOL>>, IN1 := <<BOOL>>, IN2 :=  
<<BOOL>>, IN3 := <<BOOL>>, IN4 := <<BOOL>>, IN5 := <<BOOL>>,  
IN6 := <<BOOL>>, IN7 := <<BOOL>>, IN8 := <<BOOL>>, OK =>  
<<BOOL>>, COL1 => <<DWORD>>, SNP1 => <<DWORD>>, COL2 =>  
<<DWORD>>, SNP2 => <<DWORD>>, COL3 => <<DWORD>>, SNP3  
=> <<DWORD>>, COL4 => <<DWORD>>, SNP4 => <<DWORD>>,  
COL5 => <<DWORD>>, SNP5 => <<DWORD>>, COL6 =>  
<<DWORD>>, SNP6 => <<DWORD>>, COL7 => <<DWORD>>, SNP7  
=> <<DWORD>>, COL8 => <<DWORD>>, SNP8 => <<DWORD>>);
```

This function block collects a contact histogram for up to eight BOOL variables.

The EN00 input of this function block should be set every scan.

This function block can collect histogram data two different ways.

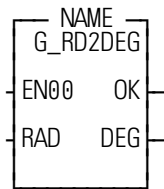
When the CLCT input is ON, data will be collected every scan for each of the eight inputs. The collected data is stored in the outputs COL1 to COL8. When this input is OFF, the values of COL1 through COL8 will remain unchanged. The least significant bit of each COL output is the most recent value.

When the SNAP input makes an OFF to ON transition, the state of all eight inputs will be saved for that scan and the next 31 scans in the outputs SNP1 to SNP8. The least significant bit of each SNP output is the value of the input on the 32nd scan.

G_RD2DEG

Converts radians to degrees

USER/G_MISC



Inputs: EN00 (BOOL) - enables execution
RAD (REAL) - the angle in radians

Outputs: OK (BOOL) - execution complete
DEG (REAL) - the angle in degrees

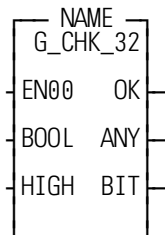
```
<<INSTANCE NAME>>:G_RD2DEG(EN00 := <<BOOL>>, RAD :=  
<<REAL>>, OK => <<BOOL>>, DEG => <<REAL>>);
```

This function block converts an angle in radians to degrees.

G_CHK_32

Return bit set in BOOL array, size=32

USER/G_SHFTRG



Inputs: EN00 (BOOL) - enables execution
BOOL (BOOL (0..31)) - array of BOOLs to check
HIGH (BOOL) - defines whether to return highest or lowest bit set

Outputs: OK (BOOL) - execution complete
ANY (BOOL) - set if any BOOL is ON
BIT (UINT) - the number of the BOOL which was found ON

```
<<INSTANCE NAME>>:G_CHK_32(EN00 := <<BOOL>>, BOOL :=  
<<BOOL>>, HIGH := <<BOOL>>, OK => <<BOOL>>, ANY => <<BOOL>>,  
BIT <<UINT>>);
```

This function block returns the number of the highest or lowest bit set in an array of 32 BOOLs.

The array of BOOLs at the BOOL input must be dimensioned to a size of 32.

If the HIGH input is OFF, then BIT will be the number of the lowest BOOL set in the array.

If the HIGH input is ON, then BIT will be the number of the highest BOOL set in the array.

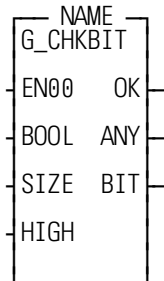
For example, if BOOL(0) and BOOL(31) are ON and BOOL(1) through BOOL(30) are all OFF, then if HIGH is OFF, BIT will be 0, but if HIGH is ON, BIT will be 31. In both cases, the ANY output will also be ON indicating that at least one BOOL in the array is ON.

If the ANY output is OFF, then none of the BOOLs in the array are ON.

G_CHKBIT

Returns bit set in BOOL array

USER/G_SHFTRG



Inputs: EN00 (BOOL) - enables execution
BOOL (BOOL (0..?)) - array of BOOLS to check
SIZE (UINT) - size of array at BOOL input
HIGH (BOOL) - defines whether to return highest or lowest numbered bit set

Outputs: OK (BOOL) - execution complete
ANY (BOOL) - set if any BOOL is ON
BIT (UINT) - number of highest or lowest BOOL energized

```
<<INSTANCE NAME>>:G_CHKBIT(EN00 := <<BOOL>>, BOOL :=  
  <<BOOL>>, SIZE := <<UINT>>, HIGH := <<BOOL>>, OK => <<BOOL>>,  
  ANY => <<BOOL>>, BIT <<UINT>>);
```

This function block determines the highest or lowest numbered energized BOOL in an array of BOOLS.

The size of the array at the BOOL input must be a multiple of 32. This size is entered at the SIZE input.

If the HIGH input is OFF, then BIT will be the lowest numbered BOOL set in the array. If the HIGH input is ON, then BIT will be the highest numbered BOOL set in the array. BIT will be a number from 0 to SIZE - 1.

This function block will not execute and the OK will not be set if SIZE is not a multiple of 32.

The ANY output will be set if at least one of the BOOLS in the array is ON.

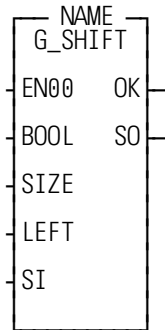
EXAMPLE:

If SIZE is 64, and BOOL(0) and BOOL(63) are ON, and BOOL(1) through BOOL(62) are all OFF, then if HIGH is OFF, BIT will be 0, but if HIGH is ON, BIT will be 63. In both cases, the ANY output will be ON.

G_SHIFT

Shift array of BOOLs left or right

USER/G_SHFTRG



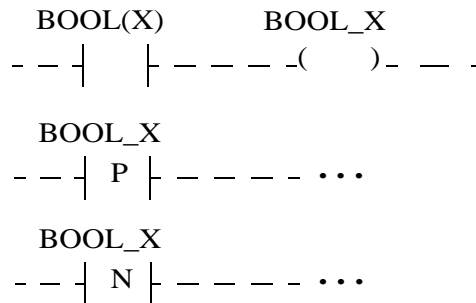
- Inputs:**
- EN00 (BOOL) - enables execution
 - BOOL (BOOL (0..?)) - array of BOOLs to shift
 - SIZE (UINT) - size of array at BOOL input
 - LEFT (BOOL) - defines direction to shift
 - SI (BOOL) - defines the new value to shift into the array
- Outputs:**
- OK (BOOL) - execution complete
 - SO (BOOL) - value shifted out of the array

```
<<INSTANCE NAME>>:G_SHIFT(EN00 := <<BOOL>>, BOOL :=
  <<BOOL>>, SIZE := <<UINT>>, LEFT := <<BOOL>>, SI := <<BOOL>>, OK
  => <<BOOL>>, SO => <<BOOL>>);
```

This function block performs a shift left or shift right on an array of BOOLs. The size of the array at the BOOL input must be a multiple of 32. This size is entered at the SIZE input. If the LEFT input is ON, then the array will be shifted to the left. The value that was in BOOL(0) will be moved to BOOL(1), the value that was in BOOL(1) will be moved to BOOL(2), etc.. The value that was in BOOL(SIZE - 1) will be moved into the SO output and the value from SI will be moved into BOOL(0).

If the LEFT input is OFF, then the array will be shifted to the right. The value that was in BOOL(1) will be moved to BOOL(0), the value that was in BOOL(2) will be moved to BOOL(1), etc.. The value that was in BOOL(0) will be moved into the SO output and the value from SI will be moved into BOOL(SIZE - 1). This function block will not execute and the OK will not be set if SIZE is not a multiple of 32.

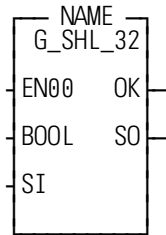
IMPORTANT: Do not use a positive or negative transitional contact in your LDO with the BOOL array for the shift register ASFBs. If it is necessary to set up a transitional contact with a Boolean in the BOOL array, use subsequent Boolean for the transitional contact as shown in the example below.



G_SHL_32

Shifts array of 32 BOOLs left

USER/G_SHFTRG



Inputs: EN00 (BOOL) - enables execution
BOOL (BOOL (0..31)) - array of BOOLs to shift.
SI (BOOL) - 'ON' or 'OFF' to shift into BOOL (0)

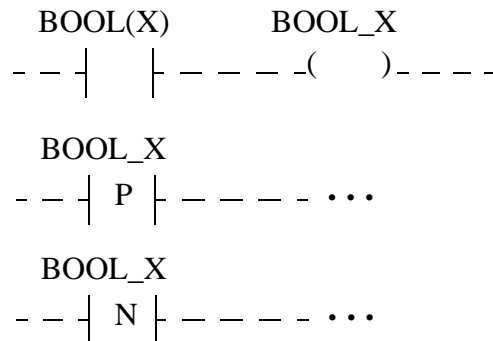
Outputs: OK (BOOL) - execution complete
SO (BOOL) - 'ON' or 'OFF' shifted out of
BOOL (31)

```
<<INSTANCE NAME>>:G_SHL_32(EN00 := <<BOOL>>, BOOL :=  
<<BOOL>>, SI := <<BOOL>>, OK => <<BOOL>>, SO => <<BOOL>>);
```

This function block performs a shift left on an array of 32 BOOLs.

The value that was in BOOL(0) will be moved to BOOL(1), the value that is in BOOL(1) will be moved to BOOL(2), etc.. The value that was in BOOL(31) will be moved into the SO output and the value from SI will be moved into BOOL(0).

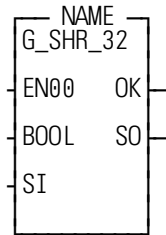
IMPORTANT: Do not use a positive or negative transitional contact in your LDO with the BOOL array for the shift register ASFBs.
If it is necessary to set up a transitional contact with a Boolean in the BOOL array, use subsequent Boolean for the transitional contact as shown in the example below.



G_SHR_32

Shifts array of 32 BOOLs right

USER/G_SHFTRG



Inputs: EN00 (BOOL) - enables execution

BOOL (BOOL (0..31)) - array of BOOLs to shift.

SI (BOOL) - 'ON' or 'OFF' to shift into BOOL (31)

Outputs: OK (BOOL) - execution complete

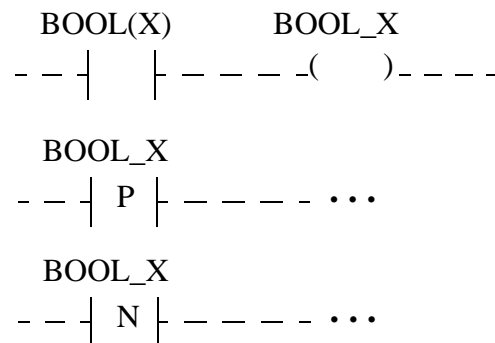
SO (BOOL) - 'ON' or 'OFF' shifted out of
BOOL (0)

```
<<INSTANCE NAME>>:G_SHR_32(EN00 := <<BOOL>>, BOOL :=  
<<BOOL>>, SI := <<BOOL>>, OK => <<BOOL>>, SO => <<BOOL>>);
```

This function block performs a shift right on an array of 32 BOOLs.

The value that was in BOOL(1) will be moved to BOOL(0), the value that is in BOOL(2) will be moved to BOOL(1), etc.. The value that was in BOOL(0) will be moved into the SO output and the value from SI will be moved into BOOL(31).

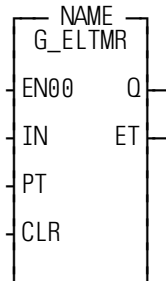
IMPORTANT: Do not use a positive or negative transitional contact in your LDO with the BOOL array for the shift register ASFBs.
If it is necessary to set up a transitional contact with a Boolean in the BOOL array, use subsequent Boolean for the transitional contact as shown in the example below.



G_ELTMR

Elapse time timer

USER/G_TIMER



Inputs: EN00 (BOOL) - enables execution
IN (BOOL) - the input to time
PT (TIME) - defines the preset time
CLR (BOOL) - clear input

Outputs: Q (BOOL) - timer output
ET (TIME) - elapsed time

```
<<INSTANCE NAME>>:G_ELTMR(EN00 := <<BOOL>>, IN := <<BOOL>>,
PT := <<TIME>>, CLR := <<BOOL>> Q => <<BOOL>>, ET => <<TIME>>);
```

This function block energizes an output after an input has been energized for a period of time. If the input goes off and then on, the timer will resume timing where it left off. There is a clear input that will reset the elapsed time, and start timing over again.

The EN00 input of this function should be set every scan.

This function block will keep track of the total time that IN has been energized. The elapsed time will be stored in the ET output. If IN goes off and then comes back on, the elapsed time (ET) will continue counting where it left off.

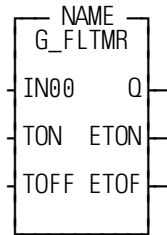
When ET equals PT, then the Q output will be energized.

To reset the elapsed time, energize the CLR input.

G_FLTMR

Flash timer

USER/G_TIMER



Inputs: IN00 (BOOL) - when set, the output will pulse according to the duty cycle entered
TON (TIME) - defines the time that the Q output will remain on
TOFF (TIME) - defines the time that the Q output will remain off

Outputs: Q (BOOL) - timer output
ETON (TIME) - the elapsed time that Q has been on
ETOF (TIME) - the elapsed time that Q has been off

```
<<INSTANCE NAME>>:G_FLTMR(IN00 := <<BOOL>>, TON := <<TIME>>,
  TOFF := <<TIME>>, Q => <<BOOL>>, ETON => <<TIME>>, ETOF =>
  <<TIME>>);
```

This function block pulses an output on and off when an input is energized.

When IN00 is ON, Q will be ON for the amount of time in TON, then OFF for the amount of time in TOFF, then ON for the amount of time in TON, etc.. The ETON output shows the amount of time that Q has been ON. The ETOFF output shows the amount of time that Q has been OFF.

If IN00 is OFF, Q will be OFF.

Index

A

angle

degrees to radians 2-32

radians to degrees 2-39

ASFB 1-1

using 1-3

B

BOOL

pack to WORD 2-12

return bit set 2-41

shift array 2-42

shift left 2-43

shift right 2-44

to DWORD 2-10

to WORD 2-10

BYTE

return one bit 2-13

C

Celsius

to Fahrenheit 2-30

Communications ASFBS 2-1

contact histogram 2-38

D

data list file manager 2-26

Data Type Conversion ASFBS 2-1

degrees

to radians 2-32

DINT

greater than or equal 2-22, 2-23

in between 2-21

less than or equal 2-24, 2-25

to string 2-18

drum sequencer 2-33

DWORD

return one bit 2-14

to binary formatted 2-9

to BOOLs 2-15

to hex string 2-17

E

Evaluation ASFBS 2-2

F

Fahrenheit

to Celsius 2-36

G

G_BETWN 2-21

G_BN2STR 2-9

G_BOO2DW 2-10

G_BOO2WD 2-12

G_BY2BIT 2-13

G_C2F 2-30

G_CHK_32 2-40

G_CHKBIT 2-41

G_COMM.LIB 2-1

G_COMMEX.LDO 2-1

G_CONFIG 2-5, 2-7, 2-8

G_D2ARMP 2-31

G_DATTYP.LIB 2-1

G_DEG2RD 2-32

G_DRMSEQ 2-33

G_DTYPEX.LDO 2-1

G_DW2BIT 2-14

G_DW2BOO 2-15

G_ELTMR 2-45

G_EVAL.LIB 2-2

G_EVALEX.LDO 2-2

G_F2C 2-36

G_FILE.LIB 2-2

G_FILEEX.LDO 2-2

G_FILMNG 2-26

G_FLTMR 2-46

G_GE_ALL 2-22

G_GE_ANY 2-23

G_HSTGRM 2-37

G_HX2STR 2-17

G_LE_ALL 2-24

G_LE_ANY 2-25

G_MISC.LIB 2-3

G_MISCEX.LDO 2-3

G_NM2STR 2-18

G_RCVSTR 2-5, 2-7

G_RD2DEG 2-39

G_READFL 2-28

G_SHFTEX.LDO 2-3

G_SHFTRG.LIB 2-3

G_SHIFT 2-42

G_SHL_32 2-43
G_SHR_32 2-44
G_SNDSTR 2-5, 2-8
G_TIMER.LIB 2-4
G_TMREX.LDO 2-4
G_WD2BIT 2-19
G_WD2BOO 2-20
G_WRITFL 2-29

I

Installation 1-1

M

Miscellaneous ASFBS 2-3

R

radians

to degrees 2-39

RAMDISK 2-26

read file 2-28

write file 2-29

RAMDISK File Manipulation ASFBS 2-2

revision

history 1-2

range 1-2

S

serial port

configure 2-5

receive string 2-7

send string 2-8

Shift Register ASFBS 2-3

T

temperature

Celcius to Fahrenheit 2-30

Fahrenheit to Celsius 2-36

timer

elapse time 2-45

flash 2-46

Timer ASFBS 2-4

W

WORD

return one bit 2-19

to BOOLs 2-20