



*Open Architecture Control Software*

Using the COMGENIE TLI Instructions

Because of the variety of uses of the information described in this application note, the users of, and those responsible for applying this information must satisfy themselves as to the acceptability of each application and use of the information. In no event will SoftPLC Corporation be responsible or liable for its use, nor for any infringements of patents or other rights of third parties which may result from its use.

SOFTPLC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

SoftPLC Corporation reserves the right to change product specifications at any time without notice.

No part of this document may be reproduced by any means, nor translated, nor transmitted to any magnetic medium without the written consent of SoftPLC Corporation.

SoftPLC and TOPDOC are registered trademarks of SoftPLC Corporation. SoftWIRES is a trademark of SoftPLC Corporation.

© Copyright 1994 - 2002 SoftPLC Corporation  
ALL RIGHTS RESERVED

First Printing: November, 1994  
Last Printing: February, 2003

**SoftPLC Corporation**  
25603 Red Brangus Drive  
Spicewood, TX 78669 USA  
Telephone: 512/264-8390 or 1-800-SoftPLC  
Fax: 512/264-8399  
WWW: <http://www.softplc.com>  
Email: [info@softplc.com](mailto:info@softplc.com)

# TABLE OF CONTENTS

<u>SECTION</u>	<u>PAGE</u>
1. INTRODUCTION .....	1
2. COMGENIE OVERVIEW .....	1
3. INSTALLING/CONFIGURING THE TLM .....	2
3.1 COMGENIE PARAMETERS .....	3
3.2 MULTIPLE SERIAL PORT SUPPORT .....	4
3.3 USE UNDER TOPDOC .....	4
4. THE STRING TABLE .....	5
5. THE TLI'S .....	7
5.1 COMRCVCLEAR .....	7
5.2 COMXMITCLEAR .....	7
5.3 COMRCVSTS .....	7
5.4 COMPRINT .....	8
5.5 COMSCAN .....	12
6. COMGENIE EXAMPLES .....	16
7. DIGIBOARD SETTINGS .....	19
8. 8 PORT PC/104 BOARD SETTINGS .....	20

# 1. INTRODUCTION

SoftPLC Corporation's SoftPLC product employs a unique technology which lets C/C++ or Java language programmers add new ladder logic instructions to the instruction set. These loadable instructions are called TLI's. TOPDOC, the ladder logic programming package which supports SoftPLC, automatically learns about the new TLI's as it logs into the SoftPLC. Additionally, simple entries may be placed into an ASCII text file to inform TOPDOC about the TLI's so they may be used during offline editing as well. Then the TLI's can even be used with complete access to the offline emulation and debugging capabilities of SoftPLC Corporation's SoftWIRES.

TLI's may be developed by any competent 'C' programmer who has access to the SoftPLC 'C' or Java Programmer's Toolkits, products readily available from SoftPLC Corporation. There are a number of Systems Integrators who are SoftPLC Partners who possess the requisite expertise. End users may also have this capability.

This document describes a number of TLI's, all which reside in a SoftPLC Corporation Loadable Module (TLM) that is provided at no charge with all SoftPLC licenses. The TLM is called **COMGENIE** and resides in a file with the name COMGENIE.TLM. This TLM was developed and is maintained by SoftPLC Corporation. **COMGENIE** can be used to perform powerful bi-directional ASCII communications to devices via RS-232, RS-422, or RS-485 links. **COMGENIE** supports up to 36 serial ports in a SoftPLC system, each of which has separate input and output buffers.

## 2. COMGENIE OVERVIEW

**NOTE:** Throughout this document, the term "character" should be taken to mean the same thing as byte.

The TLI's included in **COMGENIE.TLM** are:

COMRCVCLEAR	Clears the input buffer.
COMXMITCLEAR	Clears the output buffer.
COMRCVSTS	Tells how many characters are in the input buffer.
COMPRINT	Outputs a packet of characters which is dynamically assembled using powerful string formatting.
COMSCAN	Receives a packet of characters and automatically decomposes it into useful data fields located in INTEGER, FLOAT and/or ASCII datatable words.
MEMSCAN	Works like COMSCAN, but uses a datatable resident string instead of characters in a PORT. Automatically decomposes a string into useful data fields located in INTEGER, FLOAT and/or ASCII datatable words.
MEMPRINT	Works like COMPRINT, but uses a datatable resident destination string instead of using a Port. Outputs a string of characters using powerful string formatting.
MEMCOMPARE	Compares a range of bytes in the datatable.

All the TLI's take a parameter called Port. The Port number specifies which logical serial COM channel a TLI is to work on. Since **COMGENIE** supports 36 Ports, legal values for Port are 0 through 35. The Port number may be provided either as a constant or as a PLC datatable word containing the value. All supported Ports may be concurrently active.

### 3. INSTALLING/CONFIGURING THE TLM

**COMGENIE.TLM** uses a lower level driver module called COMM.DLL. COMM.DLL should always be in the same directory as softplc.exe, which is where it is installed by default from the factory.

Use TOPDOC NexGen's Module Editor to load and configure the COMGENIE.TLM for use with SoftPLC, as shown in Figure 1. Clicking Configure opens the driver configuration file, COMGENIE.LST, for editing.

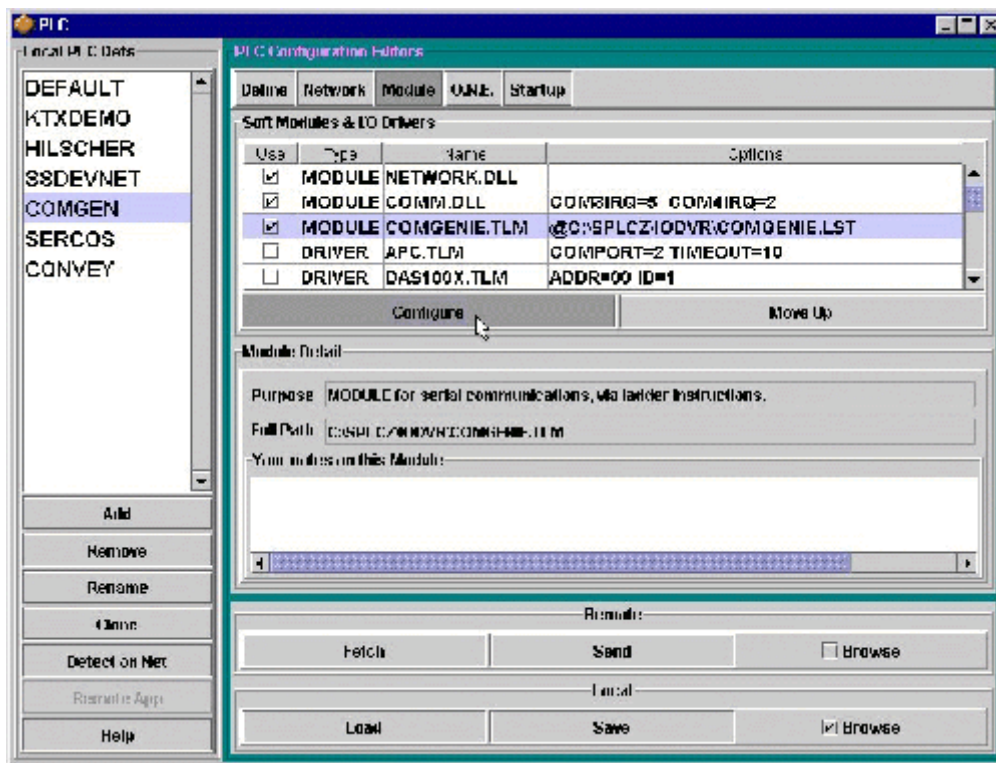


Figure 1 - Loading the Driver

**COMGENIE.TLM** can accept several configuration parameters, which are defined in a separate text file. That text file is referenced by default in the options field through TOPDOC NexGen's Module editor. For example:

```
@\SPLCZ\IODVR\COMGENIE.LST
```

Each line of COMGENIE.LST is used to list the configuration parameters for each port, as in the example below. One is mandatory, others are optional. Each optional parameter has a default value that is used if the parameter is not specified. The available parameters and their default values are shown below.

**Mandatory:** You must specify one or more PORT<n> parameters, otherwise the TLM will prevent SoftPLC from starting.

**Example COMGENIE.LST file:**

```
STRINGS=\SPLCZ\STRINGS.TXT
PORT0=COM2(9600,N,8,1) TIMEOUT=5 XMITSIZE=1000
PORT1=COM5(9600,N,8,1) TIMEOUT=5 XMITSIZE=1000
PORT2=COM25(9600,N,8,1) TIMEOUT=5 XMITSIZE=1000
```

**Note:** The COMGENIE.LST file cannot include any blank lines.

## 3.1 COMGENIE PARAMETERS

### STRINGS=<filename>

where <filename> is the name of an ASCII text file which contains string constants used by all ports for the TLM. Default is \SPLCZ\STRINGS.TXT.

### PORT*i*=COM*j*(<baud rate>, <parity>, <data bits>, <stop bits>)

where: *i* in PORT*i* = 0 to 35, indicating Logical port 0 through 35  
*j* in COM*j* = 1 through 36, indicating COM1 through COM36  
baud rate = 100 or greater  
parity = 'N' for None, 'O' for Odd, 'E' for Even, 'S' for Space, 'M' for Mark  
data bits = 5, 6, 7, or 8  
stop bits = 1 or 2

If not specified, PORT '*i*' is not useable from the ladder. This option lets you remap the meaning of logical PORT '*i*' without having to change your ladder logic, and it also tells the TLM that you intend to use this Port.

### TIMEOUT=<seconds>

where <seconds> is the number of seconds that a COMSCAN TLI will wait before giving up. The default is 1 second. You may use a decimal point when specifying <timeout>. For example, 0.3 is 3/10 of a second.

### RCVSIZE=<numBytes>

where <numBytes> is the number of bytes to allocate for the receiver ring buffer. Default is 1000. Minimum allowed is 100, maximum is 30,000.

### XMITSIZE=<numBytes>

where <numBytes> is the number of bytes to allocate for the transmit ring buffer. Default is 300. Minimum allowed is 100, maximum is 30,000. The XMITSIZE should be set greater than or equal to the largest string that you will be sending out a PORT with the COMPRINT TLI.

**NOTE:** When specifying values for the TIMEOUT, RCVSIZE, and XMITSIZE parameters, they must follow the port definition parameter PORT*i*=<comport>, and they pertain only to that port. For example, say that in your COMGENIE.LST file you have:

```
PORT0=COM2(9600,N,8,1) TIMEOUT=5  
PORT1=COM1(2400,E,7,1) TIMEOUT=1
```

The first PORT definition maps logical port 0 to COM2 and initializes COM2 to 9600 Baud, no parity, 8 data bits, 1 stop bit and a timeout value of 5 seconds. The second PORT definition maps logical port 1 to COM1 and initializes COM1 to 2400 baud, even parity, 7 data bits, 1 stop bit and a timeout value of 1 second.

## 3.2 MULTIPLE SERIAL PORT SUPPORT

COM1 and COM2 are found on the SoftPLC's CPU and are defined as physical ports using these standard hardware resources:

<u>IOPORT</u>	<u>IRQ</u>	<u>PortName</u>
{ 0x3F8,	4 },	// COM1
{ 0x2F8,	3 },	// COM2
{ 0x3E8,	5 },	// COM3
{ 0x2E8,	2 },	// COM4

COM3-COM36 can only be used if you have one or more supported serial port expansion cards installed in the system. Serial port expansion cards available from SoftPLC with (2) additional ports for COM3 and COM4 use the IRQ's above. The IRQ's and I/O ports for cards which provide COM5 and up, along with setup information for these cards, are in sections 7 and 8 of this document.

## 3.3 COMGENIE.TLM USE UNDER TOPDOC

There is a separate and different COMM.DLL for use under TOPDOC/SoftWIRES. That different COMM.DLL allows COMGENIE to print and scan under Windows 95/98/NT/2000. It must also be listed in TOPDOC's MODULE.LST file as the first module listed, followed by COMGENIE.TLM. This form of COMM.DLL normally resides in the \TDZ directory.

**HINT:** Using SoftWIRES for COMGENIE program development/testing can sometimes be faster than using the target SoftPLC, since you have ready access to the STRINGS.TXT file in another Win32 session. Then, after editing STRINGS.TXT, you can quickly restart TOPDOC to get the modified STRINGS.TXT file to load again. Only COM1-4 have been tested under Win32 operating systems. It is possible that COM5-COM36 may also work with the add-on hardware and associated driver for that operating system installed, but this has not been tested.

## 4. THE STRING TABLE

When **COMGENIE.TLM** is loaded, it loads a string table into RAM that it owns privately. The string table is a read-only array of strings. As read-only, the strings may not be modified at run time. Strings from the string table are used solely as format specifications. You create the string table offline with any ASCII text editor and save it to a file (with the default name of STRINGS.TXT, or the filename you specified in COMGENIE.LST with the STRINGS parameter).

Strings in the RAM resident string table are numbered from 1 upwards. There is no string 0. A string table may contain up to 30,000 strings, or up to the limits of available memory, whichever comes first. Each string takes 8 bytes of memory overhead in addition to the number of characters in the string. A single string may not exceed 255 characters in length.

When loading the RAM resident string table from the STRINGS.TXT file, each line of text within the file is scanned for a string delineated with a starting and ending double quote character. Text which is not enclosed in double quotes is ignored. Only the first double quoted string per line is used. Text outside double quotes may be used to document or comment the associated string(s). If no double quotes are found on a given line of text, then that string will be equivalent to the null string at runtime. The null string is the same thing as "".

If a string needs to make use of the " character itself, it must be preceded with the escape delineation character \. The \ character is also used to specify non-ASCII (extended) bytes and control codes. If the \ character itself is needed within a string, it must be preceded with another instance of \. The full list of supported escape sequences is given in the following table. Except for \n which maps to carriage return AND line feed characters (two bytes), all escape sequences are translated to 1 byte at the time of string table load. In other words, these sequences look like two characters, but represent only one (except for \n). In addition, an arbitrary byte-sized bit pattern can be specified by \ooo, where ooo is one to three octal digits (0...7) or by \xhh, where hh is one or two hexadecimal digits (0...9, a...f, A...F).

The complete set of escape sequences is:

<code>\a</code>	alert (bell character), same as <code>\x07</code>
<code>\b</code>	backspace, same as <code>\x08</code>
<code>\f</code>	formfeed, same as <code>\x0C</code>
<code>\n</code>	newline, same as <code>\x0D\x0A</code>
<code>\r</code>	carriage return, same as <code>\x0D</code>
<code>\t</code>	horizontal tab, same as <code>\x09</code>
<code>\v</code>	vertical tab, same as <code>\x0B</code>
<code>\\</code>	backslash
<code>\"</code>	double quote
<code>\ooo</code>	byte value given in octal
<code>\xhh</code>	byte value given in hexadecimal

## EXAMPLE:

The following is an example STRINGS.TXT file:

```
1 "Enter your \"name:\""           prompt the user for his name
2 "Enter your access code:"       prompt the user for his access code.
3 Tell user thank you.            "Thank you for being there."
4 "\x1B[%d;%dH"                  ANSI terminal cursor position code,
5                                 which is ESC [#;#H
6 "\x10\x06"                     Data Highway DLE ACK control codes.
7 "NK %S\0"                      Tool Coordinator Not Acknowledge
8 "HM %3d %d %S\0"              Tool Positioner Home command.
"\b\b\b"                          9      3 backspaces to erase last input.
"First line\nsecond line" 10     Output two lines using \n.
```

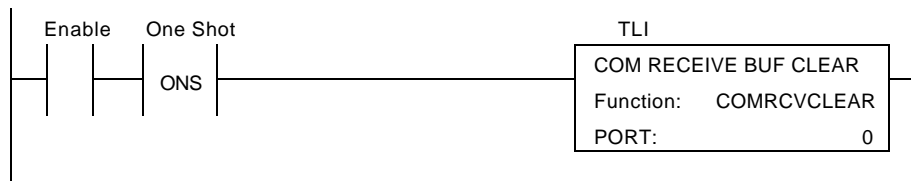
The above file will generate 10 strings, numbered 1 to 10 in the RAM resident string table. Again, only the first characters enclosed in double quotes encountered on each line will be retained in the RAM resident string table. String 5 will be the null string, since line 5 has no double quoted text (line 5 was used as a continuation of a comment relating to string 4).

The above example deliberately shows how the order of entries within a line do not matter to COMGENIE, which looks only for the first set of characters enclosed in double quotes (see lines 3, 9 and 10).

## 5. THE TLI's

### 5.1 COMRCVCLEAR

This TLI throws away any characters which are in the receiver ring buffer for the specified Port. After this instruction is energized, no further input is possible with COMSCAN until more characters actually arrive through the Port.



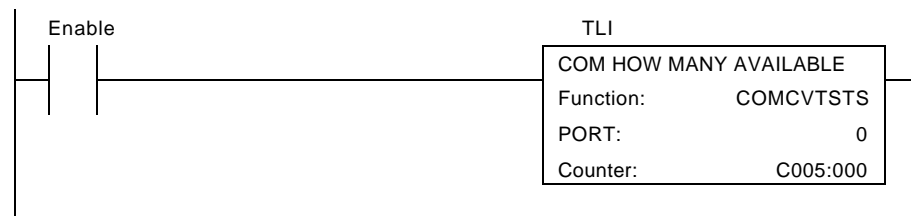
### 5.2 COMXMITCLEAR

This TLI throws away any characters in the transmitter ring buffer for the specified Port. After this instruction is energized, no further output will take place until a COMPRINT instruction is energized for this Port.



### 5.3 COMRCVSTS

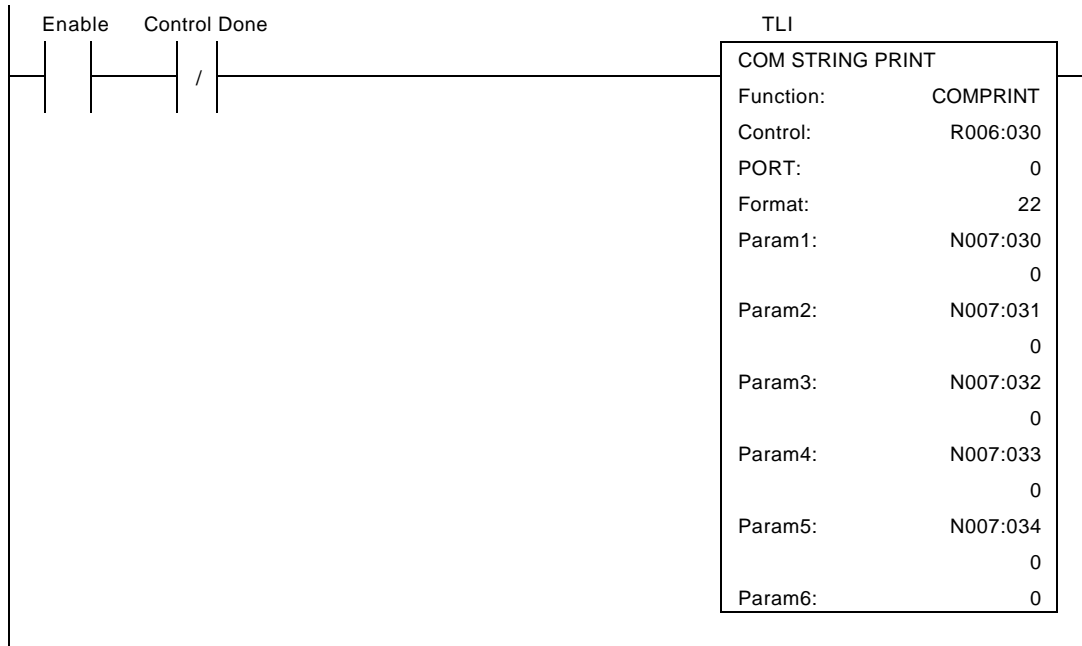
This TLI determines the number of characters available within the receive ring buffer for a particular Port.



The PRE field of the Counter is set to the size of the receive buffer which was set on the COMGENIE.TLM command line. The ACC word is set to the received characters in the buffer. The overflow bit (OV) of the counter is set if there has been a receive buffer overflow. This internal flag is automatically reset during the next COMSCAN which removes enough characters to correct the overflow condition.

## 5.4 COMPRINT

COMPRINT outputs a packet of characters which is dynamically assembled using powerful string formatting. COMPRINT is a general purpose output formatting function, similar in behavior to the C programming language's **printf()** function, but extended with automatic checksum generation capabilities. Unlike printf(), COMPRINT format strings may have embedded zeros in them, because format string lengths are implicitly given by the trailing " character in STRINGS.TXT.



COMPRINT converts and writes output to the Port under control of the string table string given by Format. The format string contains two types of objects: (a) ordinary characters, which are copied to the output Port, and (b) conversion specifications, each of which causes conversion and output of the next successive Param given to COMPRINT. Each conversion specification begins with the character % and ends with a conversion character. To output a %, use a %%. Conversion specifications consist of:

```
'%'{flag}[width][ '.' precision][ 'l' or 'L' ]conversion_char
```

where a % always signifies the beginning of a conversion specification.

Between the % and the conversion\_char there may be, in order:

**Flags** (in any order), which modify the specification:

- which specifies left adjustment of the converted argument in its field.
- + which specifies that the number will always be output with a sign.
- space if the first character is not a sign, a space will be prefixed.
- 0 for numeric conversions, specifies fill in the field with leading zeros up to the width.
- # specifies an alternate output form. For o, the first digit will be zero. For x or X, 0x or 0X will be prefixed to a non-zero result. For e, E, f, g, and G, the output will always have a decimal point; for g and G, trailing zeros will not be removed. For S, a two's complement form (negated form) of checksum is generated instead of a simple positive summation.

**Width** is a sequence of decimal digits. Or it may be the character \*, in which case the actual value is taken from the next integer Param and that Param is consumed. The interpretation of width varies based on conversion character:

S or R: See the S and R conversion characters below for details.

All others: minimum width of the field. The converted Param will be printed in a field at least this wide, and wider if necessary. If the converted Param has fewer characters than the field width it will be padded on the left (or right, if left adjustment has been requested) to make up the field width. The padding character is normally space, but is 0 if the zero padding flag is present.

**Period** "." is used to separate the width from the precision.

**Precision** is a sequence of decimal digits. Or it may be the character \*, in which case the actual value is taken from the next integer Param and that Param is consumed. If no digits appear after the ., then the precision is taken as 0. If the . does not appear, no precision may be specified and it defaults to 1. If the precision is the \* character, then the value is taken from the next Param which must be supplied as type integer, and this next Param is thusly consumed. The interpretation of precision varies based on conversion character:

e, E, or f: the number of digits to be printed after the decimal point.

g or G: the number of significant digits.

c or C: the number of consecutive characters to output starting with the supplied Param.

o, i, b, u, or d: minimum number of digits to be printed (leading 0s will be added to make up the necessary width).

s: the maximum number of characters to be printed from a string.

S or R: see the conversion characters S and R below for details.

**Length** "L" when used with the o, u, x, X, i, d, or b conversion characters indicates that the corresponding Param is to be output as a 32 bit long integer in displayable form. For example, if N7:0 passed as the Param for this conversion specification "%Ld", two integer words are used and interpreted as a 32 bit long integer N7:0 and N7:1, with the least significant 16 bits of the long word being N7:0. For all other conversion characters "L" is ignored.

**Conversion character** may be one of the following, and must match up with the Param type indicated.

Param type: Converted to

b integer; output in displayable **unsigned binary**.

d integer; output in displayable **signed decimal**.

i integer; output in displayable **signed decimal**.

o integer; output in displayable **unsigned octal**.

u integer; output in displayable **unsigned decimal**.

x integer; output in displayable **unsigned hexadecimal**, lowercase hex letters will be used.

X integer; output in displayable **unsigned hexadecimal**, uppercase hex letters will be used.

(cont'd) Param type: Converted to

- s string; The Param is the start of an ASCII character sequence which contains a zero terminator. The characters are output until a \0 character is encountered or the number of characters specified in precision are output. The terminating \0 is not output. The precision defaults to 32767.
- c integer; single character taken from the least significant 8 bits of the Param (little endian). If the precision is specified as greater than 1, then multiple characters are output continuing with the most significant 8 bits of the Param, then the least significant 8 bits of the word following the Param, then the most significant 8 bits of the word following Param, and so on, until the precision requirements are met. For example, if you wanted to send 6 bytes of binary information stored at N7:10 through N7:12, you would use the format specification "%.6c" and a Param of N7:10. This is similar to the s conversion character except that the Param data may contain embedded \0 characters.
- C integer; single character taken from the most significant 8 bits of the Param (big endian). If the precision is specified as greater than 1, then multiple characters are output continuing with the least significant 8 bits of the Param, then the most significant 8 bits of the word following the Param, then the least significant 8 bits of the word following Param, and so on, until the precision requirements are met. For example, if you wanted to send 6 bytes of binary information stored at N7:10 through N7:12, **and do it most significant byte first for each word**, you would use the format specification "%.6C" and a Param of N7:10. This is similar to the c conversion character except that the Param data is output most significant byte first, then least significant byte.
- f float; decimal string of the form [-]ddd.dddd. The number of digits after the decimal point is given by the precision, which defaults to 6. If the precision is 0, no fractional digits or decimal points appear.
- e,E float; string using scientific notation in the form [d.dddddde+-dd. There is one digit before the decimal point and precision digits after. The precision defaults to 6. If the precision is 0, the decimal point is not written. E is used for the exponent instead of e if the E conversion character was specified. A minimum of two digits will appear in the exponent.
- g,G float; string using either f or e (or E if G was specified) format, depending on the value of the Param. e will be used if the exponent is < -3 or > the precision. The precision gives the number of significant digits; it defaults to 6. The decimal point appears if followed by a digit; trailing 0s are truncated.
- % The % character is printed.
- S no Param consumed; The output is a **binary checksum** on all the characters output from this COMPRINT starting from the character at offset "width" and continuing up to and including the character "precision" characters before the place in the output stream where the first character of the checksum will go. Precision defaults to 1, and 1 means include the character just before the checksum. If precision were 2, then the character before the checksum would be excluded, etc. A sixteen bit checksum is used (two bytes) unless the "h" Length modifier is present indicating a one byte checksum. The output from this conversion is not likely to be displayable.

(cont'd) Param type: Converted to

- R no Param consumed; The output is a **binary Cyclic Redundancy Check (CRC-16)** on all the characters output from this COMPRINT starting from the character at offset "width" and continuing up to and including the character "precision" characters before the place in the output stream where the first character of the checksum will go. Precision defaults to 1, and 1 means include the character just before the checksum. If precision were 2, then the character before the checksum would be excluded, etc. A sixteen bit checksum is used. The output from this conversion is not likely to be displayable.

Any other characters cause an error.

#### **STATUS BITS/TROUBLESHOOTING COMPRINT:**

The COMPRINT TLI uses a Control parameter to execute asynchronously. The TLI starts its operation on a low to high rung condition transition. Unless and until the output buffer can hold the entire COMPRINT output, no partial output is performed. When the output is entered into the buffer, the DN bit comes on within the Control and the LEN is set to the number of bytes actually output. Note that this may happen before the characters are actually transmitted out the physical Port. The characters are in a ring buffer initially and the first one is or soon will be on its way out the physical Port. If the rung conditions go false before the TLI can get access to the sufficiently depleted Port buffer such that the entire output will fit, no output is performed.

There is no timeout associated with COMPRINT. At most 6 conversion Params and therefore at most 6 conversion specifications (of the type requiring a Param) may be specified in this TLI. The XMITSIZE command line parameter must provide a large enough buffer size to accommodate the largest single chunk of output characters generated by any one given COMPRINT invocation.

COMPRINT can fail to print only if:

- a) you never energize the rung
- b) you have a bad format string INDEX
- c) you have not allowed the rung to go false from the last time you fired it and the DN bit came on
- d) you have a bad port number
- e) you have an illegal format string (for example, "%k")

If any of the above are true, then the ER bit is set AND the DN bit is set.

COMPRINT can alternatively be delayed if:

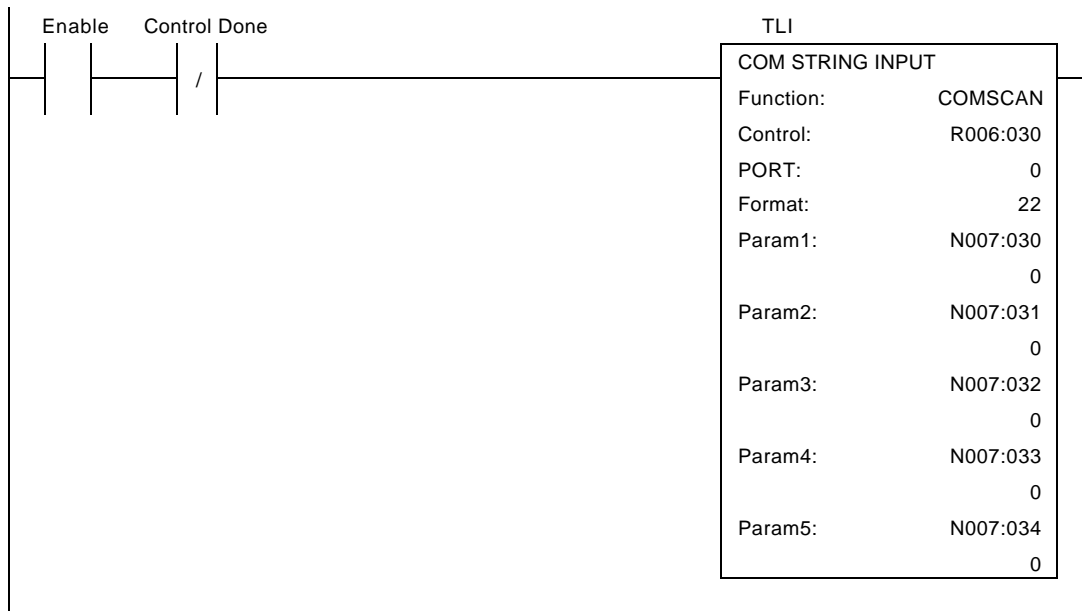
- f) there is insufficient room in the output buffer. This is indicated by the DN bit not coming on.

If none of the conditions (a) through (f) happen, then you expect the DN bit to come on and the ER bit will be reset by the instruction. You do not ever have to reset the ER bit on COMPRINT because successful completion clears the ER bit.

Additionally, a false rung condition clears both DN and ER [condition (a) above].

## 5.5 COMSCAN

COMSCAN reads characters from the receive buffer. Characters read are converted according to a format string and the values created are stored in the TLI's integer, float, and/or ASCII datatable Params. COMSCAN is a general purpose input formatting function, with very powerful and wide ranging capabilities.



The format string consists of:

1. Spaces, tabs and newlines which cause input to be skipped up to the next character which is not a white space. (White space characters are blank, tab, newline, carriage return, vertical tab, and formfeed.)
2. Other characters, except for % which are matched against the input.
3. A conversion specification.

```
'%['*']['#']['width']['.' precision']['L']conversion_char
```

A conversion specification determines the conversion of the next input field. The result of the conversion is placed in the next Param. Params are thus consumed one after another in this fashion. If there are less Params than conversion specifications an error is returned. Excess Params are ignored. If the \* is present, the conversion is performed, but no Param is consumed.

An input field is normally a string of non-white space characters; it extends either to the next white space character or until the field width, if specified, is exhausted. This implies that COMSCAN may read across line boundaries to find its input, since newlines are white space.

The conversion\_char indicates the interpretation of the input field. A % always signifies the beginning of a conversion specification. Between the % and the conversion\_char there may be, in order:

- \* is the assignment suppression indicator. Conversion is attempted as a pattern matching verification, but no Param is consumed.
- # is the alternate form indicator. For S, a two's complement form (negated form) of checksum is generated instead of a simple positive summation.

**Width** is a sequence of decimal digits. It may NOT be the '\*' character. The interpretation of width varies based on conversion character:

S or R: See the S and R conversion characters below for details.

All others: ignored.

**Period** "." is used to separate the width from the precision.

**Precision** is a sequence of decimal digits. It may NOT be the '\*' character. If the . does not appear, no precision may be specified and it defaults to 1. The interpretation of width varies based on conversion character:

S or R: See the S and R conversion characters below for details.

c or C: Specifies the number of bytes to be copied from the raw input stream.

All others: ignored.

**Length** "L" indicates that the corresponding argument is to be printed as a 32 bit long integer when used with the o, u, x, X, i, d or b conversion character. For all other conversion characters, it is ignored.

**Conversion character.** May be one of the following, and must match up with the Param type indicated.

- Param type: Converted from
- b integer; unsigned binary.
  - d integer; signed decimal.
  - i integer; if the field starts with an 0, it is interpreted as octal, if it starts with 0x or 0X it is interpreted as hexadecimal, otherwise it is interpreted as decimal.
  - o integer; unsigned octal.
  - u integer; unsigned decimal.
  - x,X integer; unsigned hexadecimal.
  - c integer; single character put into the least significant 8 bits of the integer Param. If the precision is greater than 1, then the fill continues with the most significant byte of the integer, then the least significant byte of the next integer word, etc., until precision characters have been copied to the block of words given by the Param. The characters are not NUL terminated. (for example: "%.30c" would convert 30 characters including spaces).
  - C integer; single character put into the most significant 8 bits of the integer Param. If the precision is greater than 1, then the fill continues with the least significant byte of the integer, then the most significant byte of the next integer word, etc., until precision characters have been copied to the block of words given by the Param. The characters are not NUL terminated.
  - e,E,f,g,G float; floating point number is created.
  - % The % character is matched and no Param is consumed.

(cont'd) Param type: Converted to

- s string; The Param is the start of an ASCII word address which will receive a NUL terminated string of characters. The characters are converted until a white space character is encountered. (If white space is to be included, use "c" with precision, for example: "%.30c")
  
- [ string; The Param is the start of an ASCII word address which will receive a number of characters, NUL terminated. Between the [ and a closing ] will be the characters acceptable to the string. If the [ is immediately followed by a ^, the acceptable characters for the string are all those except the ones between the ^ and the ]. A NUL character is appended to the string.
  
- S no Param consumed; The width defaults to 0 and the precision defaults to 1. The output is a binary checksum on all the characters received (NOT the same as converted) from this COMSCAN starting from the character at offset "width" up to and including the character "precision" characters before this field. The calculated value is compared to the two bytes in the input (or 1 byte if the 'h' modifier is given).
  
- R no Param consumed; The width defaults to two and the precision defaults to 0. The output is a CRC-16 checksum on all the characters received (NOT the same as converted) from this COMSCAN starting from the character at offset "width" up to and including the character "precision" characters before this field. The calculated value is compared to the two bytes in the input.

Other characters cause an error.

The COMSCAN TLI uses a Control parameter to execute asynchronously. The TLI starts its operation on a low to high rung condition transition. The TLI finishes when the format string is exhausted. It finishes with an error if the raw received character data does not match the format specification. If the amount of raw input data is insufficient to process the entire format string, the instruction waits for several scans until enough data is available or until the TLM wide TIMEOUT value has run out. The timeout is measured from the time the rung was first energized. If the rung is de-energized before completion, no data is removed from the receive buffer. Only complete format strings are processed, and if in any given scan there is an insufficient amount of data in the receive buffer, NO data is removed from the buffer until there is sufficient data.

#### **STATUS BITS/TROUBLESHOOTING COMSCAN:**

Upon completion the Control's DN bit will be set, and the ER bit may also be set if there was a format string mismatch or a timeout error. In summary, there are three possibilities when the DN bit finally comes on:

- 1) The format string was successfully matched and there were no errors. In this case the LEN word of the Control is set to the total number of characters processed. The POS word of the Control will be set to the number of format conversions performed. The conversion Params will be updated in the datatable.
  
- 2) There were sufficient received characters to determine a mismatch. The received characters are consumed up til the first mismatch. The ER bit is set as well as the DN bit and the LEN field of the Control is set to the zero based character offset into the format string where the mismatch occurred. Any Params which did happen to match before the mismatch occurred will be modified.
  
- 3) There were insufficient received characters to exhaust the format string. The TIMEOUT expired. **No received characters are consumed.** The ER bit, EM bit and DN bits are all set. The LEN field of the Control is set to the number of characters which were looked at on the last scan in an effort to match with the format string. This is also the number of characters in the receive buffer at that time.

COMSCAN can fail to scan only if:

- a) you never energize the rung
- b) you have a bad format string INDEX, or
- c) you have not allowed the rung to go false from the last time you fired it and the DN bit came on
- d) you have a bad port number
- e) you have an illegal format string (for example, "%k")
- f) you have a mismatch in the number of parameters indicated in the format string relative to the number supplied to the instruction, or a type mismatch on any parameter. (Eg: "%u" and then supply a float parameter to the TLI instruction, where %u indicates an integer.)
- g) a mismatch on the inbound data occurred, meaning the serial line data did not agree with the format string
- h) a timeout occurred. This sets the EM, ER, and DN bits.

**If any of the above are true, then the ER bit is set AND the DN bit is set.**

- i) there is insufficient room in the input ring buffer. This will go unnoticed by COMSCAN. You can get into a state where the receive ring buffer is full and all subsequent COMSCANs might mismatch the data on the front end of the buffer. In this state the buffer remains full and all subsequent input will be lost. Use COMRCVSTS to get the overflow condition, when close to full you can do a COMCLEAR to toss all bytes that are in the input ring buffer.

## 6. COMGENIE EXAMPLES

### Example 1:

Assume you are implementing a communications protocol which needs to send out the following command packet in ASCII:

Char	Parameter	Description
0	H	Home Acronym
1	M	Home Acronym
2	Blank	Separator
3	Device ID	3 digit decimal number
4	Device ID	
5	Device ID	
6	Blank	Separator
7	Axis #	Axis 0 - 5
8	Blank	Separator
9	Checksum	1st of a two byte checksum of chars 0-8
10	Checksum	2nd "
11	NULL	Message Delimiter

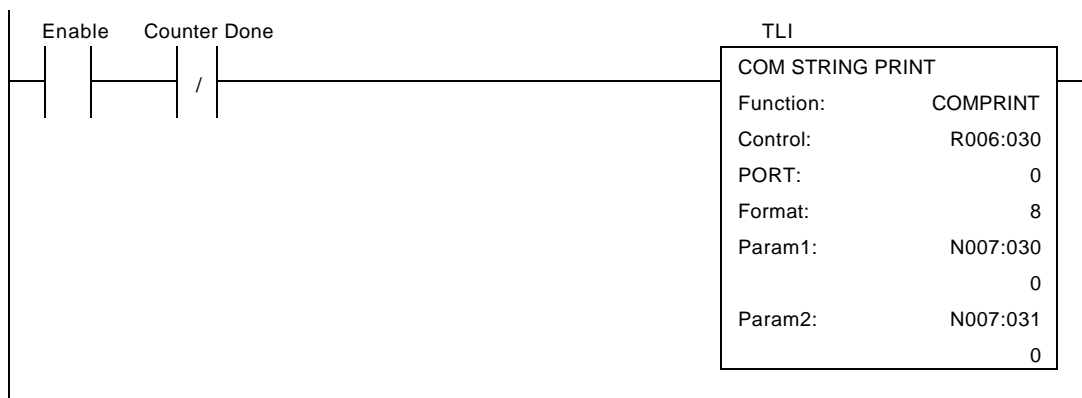
In STRINGS.TXT on line 8 we put:

```
8 "HM %3d %d %S\0"                Tool Positioner Home command.
```

Assume the following integer words contain:

Address	Meaning
N7:30	Device ID
N7:31	Axis number

Then in our ladder we place the following rung. Notice Param 3, 4, 5 and 6 are not used so they are set to 0. Only two conversion Params are needed, even though there are 3 % characters in the format string. The last (3rd) conversion specification is to compute the checksum automatically, and so no Param is required for that. The following rung will send out the necessary 12 characters and set bit R6:30.DN when the string is in the transmit buffer.



## Example 2:

Assume you are implementing a communications protocol which needs to receive the following reply packet in ASCII:

CHAR	PARAMETER	DESCRIPTION	CHAR	PARAMETER	DESCRIPTION
0	A	Acknowledge Home Acronym	11	Blank	Separator
1	K	Acknowledge Home Acronym	12	Status	Home Status 0 or 1
2	Blank	Separator	13	Blank	Separator
3	H	Home Acronym	14	Status	1=com fail, 2=data fail, 3=Drive not avail
4	M	Home Acronym	15	Blank	Separator
5	Blank	Separator	16	Status	Future
6	Device ID	3 digit decimal number	17	Blank	Separator
7	Device ID		18	Status	Future
8	Device ID		19	Blank	Separator
9	Blank	Separator	20	Checksum	Sum of characters 0-19
10	Axis #	Axis 0 - 5	21	Checksum	Sum of characters 0-19
			22	NULL	Message Delimiter

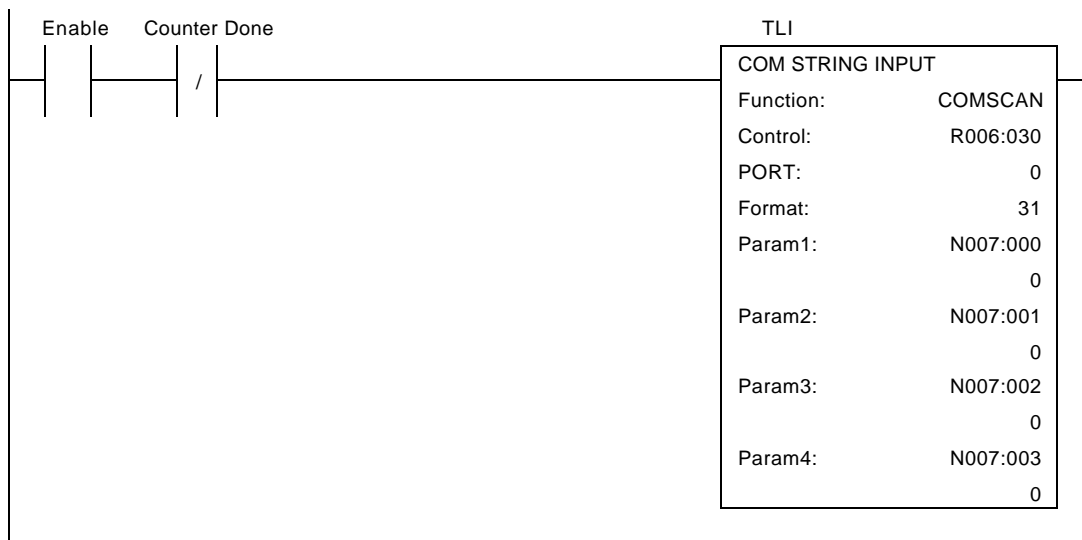
In our STRINGS.TXT on line, say for example 31, we put:

```
31 "AK HM %d %d %d %d %*5c %S\0"          Tool Positioner Home AK.
```

Assume the following integer words contain:

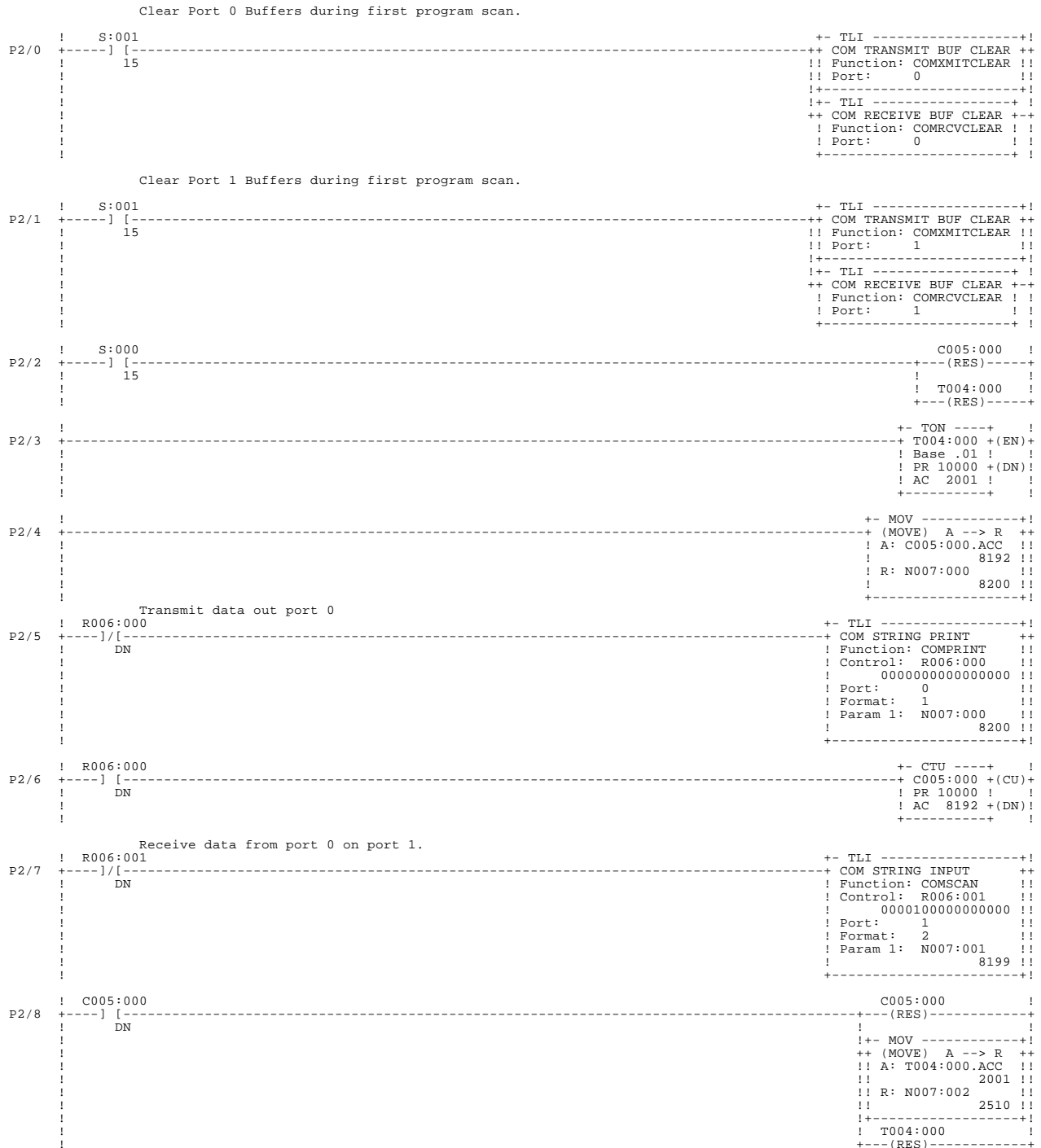
ADDRESS	MEANING	ADDRESS	MEANING
N7:0	Device ID	N7:2	Home status
N7:1	Axis number	N7:3	Command status

Then in our ladder we place the following rung. Notice Param 3, 4, 5 and 6 are not used so they are set to 0. Only four conversion Params are needed, even though there are 6 % characters in the format string. The 5th conversion specification uses assignment suppression and so consumes no Param. The 6th conversion specification is to compute the checksum automatically, and so no Param is required for that. The following rung will receive this packet and set the DN bit when completed ok with checksum validation, timeout, or packet mismatch is detected.



### EXAMPLE 3:

The following ladder logic illustrates how a transfer of data from COM1 to COM2. Through this example, you can deduce both how to send and receive data using COMGENIE. The last rung is used to clear the control bits in case of an error.



## 7. DIGIBOARD SETTINGS

This section only applies if you are using any of the ports in the range COM5 through COM36 with a DigiBoard Classic8 PC/8 card. The hard coded (non-adjustable) settings for COM1 to COM36 are hereby defined as follows:

<u>IOPORT</u>	<u>IRQ</u>	<u>PORTNAME</u>	<u>IOPORT</u>	<u>IRQ</u>	<u>PORTNAME</u>
// COM5 - COM12 all on one digiboard			// COM21 - COM28 all on one digiboard		
// control port = 0x1340			// control port = 0x1540		
{ 0x1300,	11 },	// COM5	{ 0x1500,	9 },	// COM21
{ 0x1308,	11 },		{ 0x1508,	9 },	
{ 0x1310,	11 },		{ 0x1510,	9 },	
{ 0x1318,	11 },		{ 0x1518,	9 },	
{ 0x1320,	11 },		{ 0x1520,	9 },	
{ 0x1328,	11 },		{ 0x1528,	9 },	
{ 0x1330,	11 },		{ 0x1530,	9 },	
{ 0x1338,	11 },	// COM12	{ 0x1538,	9 },	// COM28
// COM13 - COM20 all on one digiboard			// COM29 - COM36 all on one digiboard		
// control port = 0x1440			// control port = 0x1040		
{ 0x1400,	14 },	// COM13	{ 0x1000,	10 },	// COM29
{ 0x1408,	14 },		{ 0x1008,	10 },	
{ 0x1410,	14 },		{ 0x1010,	10 },	
{ 0x1418,	14 },		{ 0x1018,	10 },	
{ 0x1420,	14 },		{ 0x1020,	10 },	
{ 0x1428,	14 },		{ 0x1028,	10 },	
{ 0x1430,	14 },		{ 0x1030,	10 },	
{ 0x1438,	14 },	// COM20	{ 0x1038,	10 },	// COM36

Digiboards must be setup according to one of the above supported configurations. All ports on a digiboard card should use the same IRQ. This is done using the CFG.EXE program from Digi, and setting

```
PnP mode = No
Control Register
Int Mode: Digi
I/O Address: <one of the above control ports>
IRQ: <one of the above IRQ's>
No. Waits: 0
```

The IRQ setting for each channel should show blank. Be sure and write the profile to the board.

## 8. PC/104 8 PORT BOARD SETTINGS

This section only applies if you are using any of the ports in the range COM5 through COM36 with the Cat. No. ICO-SER104-8, 8 port PC/104 serial card supplied from SoftPLC. The hard coded (non-adjustable) settings for COM5 to COM36 are hereby defined as follows:

<u>IOPORT</u>	<u>IRQ</u>	<u>PORTNAME</u>	<u>IOPORT</u>	<u>IRQ</u>	<u>PORTNAME</u>
// COM5 - COM12 for Xtreme-8/104			// COM21 - COM28 for Xtreme-8/104		
// control port = 0x240			// control port = 0x190		
{ 0x200,	11 },	// COM5	{ 0x150,	15 },	// COM21
{ 0x208,	11 },		{ 0x158,	15 },	
{ 0x210,	11 },		{ 0x160,	15 },	
{ 0x218,	11 },		{ 0x168,	15 },	
{ 0x220,	11 },		{ 0x170,	15 },	
{ 0x228,	11 },		{ 0x178,	15 },	
{ 0x230,	11 },		{ 0x180,	15 },	
{ 0x238,	11 },	// COM12	{ 0x188,	15 },	// COM28
// COM13 - COM20 for Xtreme-8/104			// COM29 - COM36 for Xtreme-8/104		
// control port = 0x1E0			// control port = 0x140		
{ 0x1A0,	14 },	// COM13	{ 0x100,	10 },	// COM29
{ 0x1A8,	14 },		{ 0x108,	10 },	
{ 0x1B0,	14 },		{ 0x110,	10 },	
{ 0x1B8,	14 },		{ 0x118,	10 },	
{ 0x1C0,	14 },		{ 0x120,	10 },	
{ 0x1C8,	14 },		{ 0x128,	10 },	
{ 0x1D0,	14 },		{ 0x130,	10 },	
{ 0x1D8,	14 },	// COM20	{ 0x138,	10 },	// COM36

**Notes:**

1. The standard product shipped by the vendor of this card must be ordered with the above port settings configured by the factory. The SoftPLC part number for the specially configured version of this card is: ICO-SER104-8. We strongly recommend that you order the part from SoftPLC to ensure that you will receive proper configuration.

2. A unique version of the COMM.DLL is also required for this card. The standard COMM.DLL is hard-coded to support the DIGIBoard Classic 8 only. As a result, the DLL named COMMCTI.DLL must be renamed to COMM.DLL to enable the above settings for the CTI Xtreme-8/104 card. SoftPLC provides this DLL as part of the ICO-SER104-8