ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Honeywell 627-1002R* at our website: *Click HERE*

Lines 1 and 8 contain the PULL from the MiniCOP communication function in a Skip block that is executed only on command (that is, when transitional contact is true).

Line 2 unconditionally enables control bit D5, which directs the pointer to increment from its current location when data is PULLed from the MiniCOP's data table.

Line 3 unconditionally SETs control bit D0 which directs the MiniCOP to transfer data in the specified number of 16-bit register locations to its most significant I/O address (A7). **Line 3** provides the MiniCOP with eight 16-bit data values.

Line 4 is a time delay of at least 258.8µsec that allows the MiniCOP to retrieve all the requested data before receiving further instructions from the programming block.

Line 5 PULLs the data from the MiniCOP and stores it into the 620 Platform Controller's data registers

Lines 6 and 7 unlatch (CLEAR or disable) control bits D0 and D5.

NOTE

When multiple PULLs of data from the MiniCOP (more than 8 data words) are required, repeat **Lines 3 through 5** as necessary.

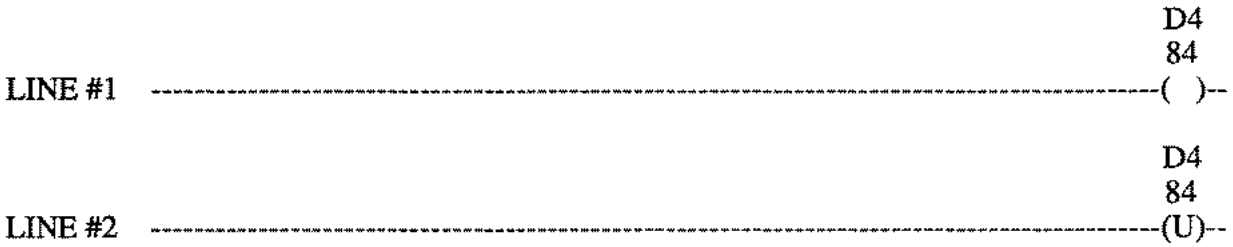
ATTENTION

When using this program block in a model 620-12, 620-1633, or 620-36 processor, refer to the *Parallel Programming Block Time Delays* in this section of the manual.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Reset the Data Table Pointer (Parallel I/O)
Enable Bit D4

This communication function determines the position of the pointer after a PUSH data to the MiniCOP or PULL data from the MiniCOP operation. If control bit D4 is enabled, the pointer is returned to the location specified by the last Set the Data Table Pointer function after the transfer has been completed. If disabled, the pointer does not reset after each data transfer. Subsequent data transfers would then occur from the current position of the pointer. If control bit D4 is not included in the ladder logic the MiniCOP will default to the disable operation during data transfers. Typically, the following two lines of logic will be included in either data transfer function when the pointer is to reset.



Line 1 should be placed in the PUSH Data to the MiniCOP or PULL Data from the MiniCOP programming block before the line of logic that actually performs the transfer operation.

Line 2 is placed in the programming block after the line of logic that performs the data transfer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Set the Pointer Direction (Parallel I/O)
Enable Bit D5

This control bit determines the direction of movement from the current data table location that the pointer will move during a PUSH to or PULL from the MiniCOP operation. If enabled, control bit D5 causes the pointer to increment. If disabled, control bit D5 causes the pointer to decrement. This control bit does not require its own programming block, but should be included in the PUSH data to the MiniCOP or PULL data from the MiniCOP blocks. The Set Pointer Direction function will default to the decrement mode of operation if its control bit is neither enabled or disabled by ladder logic. Refer to the PUSH Data to the MiniCOP and PULL Data from the MiniCOP program blocks for control bit D5 examples.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Reset the MiniCOP (Parallel I/O)
Enable Bits D3 and D0

The MiniCOP can be remotely reset by a programming block in the 620 Platform Controller's ladder logic. Executing this programming block will have the same effect on the MiniCOP as a hardware reset (pressing the MiniCOP's toggle switch when configured for reset operation) or a software reset (issuing the OS-9 command RESET on the SHELL command line).

```
                RESET
                  5
LINE #1   --]/\[-------------------------------------------------------------------------------NSKR--
                                                                                                5

                                                                                               D3
                                                                                               83
LINE #2   ------------------------------------------------------------------------------------(  )--

                                                                                               D0
                                                                                               80
LINE #3   ------------------------------------------------------------------------------------(  )--

LINE #4   ----------------------(Minimum Time Delay of 258.8µsec)-----------------------------

                                                                                               D0
                                                                                               80
LINE #5   ------------------------------------------------------------------------------------(U)--

                                                                                               D3
                                                                                               83
LINE #6   ------------------------------------------------------------------------------------(U)--

LINE #7   -------------------------------------------------------------------------------EOS--
                                                                                                5
```

Lines 1 and 7 contain the Reset the MiniCOP communication function in a Skip block that is executed only on command (that is, when transitional contact is true).

Lines 2 and 3 unconditionally SET control bits D3 and D0, which direct the MiniCOP to perform a reset operation. Line 2 provides the MiniCOP with the necessary time and date data to set the real-time clock.

Line 4 is a time delay of at least 258.8µsec that allows the MiniCOP to execute the control bit's request prior to receiving further instructions from the programming block.

Lines 5 and 6 unlatch (CLEAR or disable) control bits D0 and D3.

ATTENTION When using this program block in a model 620-12, 620-1633, or 620-36 processor, refer to the *Parallel Programming Block Time Delays* in this section of the manual.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SERIAL PROGRAMMING BLOCK DESCRIPTIONS

The serial programming blocks each consist of one or two skip blocks. The following list of serial programming blocks are those that consist of two skip blocks:

- Read the real-time clock
- Set the real-time clock
- Set the data table pointer
- PUSH data to the MiniCOP
- PULL data from the MiniCOP

In these the first skip block provides a method to maintain the correct sequence of serial functions. This block consists of ladder logic interlocks allowing only one of these five functions to be active at any given time. The second skip block contains the ladder logic required to perform the specified communication function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Read the Real-time Clock (Serial I/O)
Enable Control Bits D2 and D0

The data contained in the real-time clock can be pulled from the MiniCOP by invoking this function. The error count and current data table pointer location can also be accessed using this sequence of control bits. The Read the Real-time Clock communication function PULLs the formatted information from address A7 as listed in Table 22.

TABLE 22 - READ THE REAL-TIME CLOCK DATA FORMAT

MiniCOP Address	Data Transfer Format
A7	Second
A6	Minute
A5	Hour
A4	Day
A3	Month
A2	Year
A1	Error Count
A0	Current Data Table Pointer Location

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                    SET       DTP       PUSH      PULL       D0        D1
                     0       2049      2050      2051       2052       80        81
LINE #1   --]/\[--------] / [--------] / [--------] / [--------] / [--------] / [----------NSKR--
                                                                                          0

                                                                                         D2
                                                                                         82
LINE #2   ------------------------------------------------------------------------------(L)--

                                                                                         D0
                                                                                         80
LINE #3   ------------------------------------------------------------------------------(L)--

                                                                                       READ
                                                                                       2048
LINE #4   ------------------------------------------------------------------------------(L)--

LINE #5   ------------------------------------------------------------------------------EOS--
                                                                                          0

                    READ      D1
                    2048      81
LINE #6   --]  [--------]  [-------------------------------------------------------------NSKR--
                                                                                         10

                      A7                                                                TIME
                      87                                                               4107
LINE #7   --[PUL]-------------------------------------------------------------------(PSH)--
                       8                                                                 8

                                                                                         D0
                                                                                         80
LINE #8   ------------------------------------------------------------------------------(U)--

                                                                                         D2
                                                                                         82
LINE #9   ------------------------------------------------------------------------------(U)--

                                                                                       READ
                                                                                       2048
LINE #10  ------------------------------------------------------------------------------(U)--

LINE #11  ------------------------------------------------------------------------------EOS--
                                                                                         10
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Lines 1 through 5 comprise the first skip block:

- **Line 1** contains the serial programming block monitor and sequence maintenance logic.
- **Lines 2 and 3** unconditionally SET control bits D2 and D0, which direct the MiniCOP to provide the current time, date, error count, and data table pointer location to its most significant I/O address (A7).
- **Line 4** sets (latches) coil 2048, which requests the Read the Real-time Clock programming block to execute.

Lines 6 through 11 contain the Read the Real-time Clock communication function in a Skip block that is executed only on command (that is, Line 6 N/O Contact 2048 "READ" and N/O Contact 81 "D1" must both be true).

- **Line 7** performs a PULL of eight 16-bit data words from A7, MiniCOP I/O address 95. This is the same information that was written to this location by the MiniCOP as directed by control bits D2 and D0 in lines 2 and 3. The PUSH instruction stores this information in the 620 Platform Controller's data registers for use by your ladder logic program.
- **Lines 8, 9, and 10** unlatch (CLEAR or disable) control bits D0 and D2, and the READ request coil 2048.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Set the Real-time Clock (Serial I/O)
Enable Control Bits D3 and D1

The MiniCOP's real-time clock can be set by ladder logic in the 620 Platform Controller. In this example the programming block sets the real-time clock using Constant instructions. In reality, the ladder logic program would use Bring In or Pull instructions to obtain the time and date data to be sent to the MiniCOP. This data may be provided from internal ladder logic routines, operators station/panel, dedicated process reference clock, TDC 3000 Universal Station, etc.

```
                    READ     DTP      PUSH     PULL      D0        D1
                     1       2048     2050     2051      2052      80        81
LINE #1   --]/[--------] / [--------] / [--------] / [--------] / [--------] / [----------NSKR--
                                                                                          1

                                                                                         A7
                    N/A      N/A      YR       MO       DAY       HR        MIN       SEC       87
LINE #2   --[K2]-----[K2]-----[K2]-----[K2]-----[K2]-----[K2]-----[K2]-----[K2]-------(PSH)--
                     0        0        91       07        09        10        36        12        8

                                                                                         D3
                                                                                         83
LINE #3   ------------------------------------------------------------------------------(L)--

                                                                                         D1
                                                                                         81
LINE #4   ------------------------------------------------------------------------------(L)--

                                                                                        SET
                                                                                       2049
LINE #5   ------------------------------------------------------------------------------(L)--

LINE #6   ------------------------------------------------------------------------------EOS--
                                                                                          1

                    SET      D0
                   2049      80
LINE #6   --]  [--------]  [-------------------------------------------------------------NSKR--
                                                                                         11

                                                                                         D1
                                                                                         81
LINE #8   ------------------------------------------------------------------------------(U)--

                                                                                         D3
                                                                                         83
LINE #9   ------------------------------------------------------------------------------(U)--

                                                                                        SET
                                                                                       2049
LINE #10  ------------------------------------------------------------------------------(U)--

LINE #11  ------------------------------------------------------------------------------EOS--
                                                                                         11
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Lines 1 through 6 comprise the first skip block.

- **Line 1** contains the serial programming block monitor and sequence maintenance logic.

- **Line 2** provides the MiniCOP with the necessary time and date data to set the real-time clock. Note that the Set the Real-time Clock programming block requires a PUSH of 8 to execute the function. The first two values PUSHed must have a value of zero; these values do not affect the operation of the MiniCOP or its real-time clock.

- **Lines 3 and 4** unconditionally SET control bits D2 and D0, which direct the MiniCOP to provide the current time, date, error count, and data table pointer location to its most significant I/O address (A7).

- **Line 5** sets (latches) coil 2048 which requests the Read the Real-time Clock programming block to execute.

Lines 7 through 11 contain the Set the Real-time Clock communication function in a Skip block that is executed only on command (that is, Line 7 N/O Contact 2048 "SET" and N/O Contact 80 "D0" must both be true).

- **Lines 8, 9, and 10** unlatch (CLEAR or disable) control bits D1 and D3 and the SET request coil 2049.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Set the Data Table Pointer (Serial I/O)
Enable Control Bits D2 and D1

The MiniCOP's data table pointer and PUSH/PULL quantity can be set by ladder logic in the 620 Platform Controller. Two 16-bit values are PUSHed to the MiniCOP's most significant I/O address (A7). The first value sets the data table pointer to a specific location. This value must be a valid data table register location (0-511). In this example the programming block sets the real-time clock using Constant instructions. In reality, your ladder logic program could also use Bring In or Pull instructions to obtain the time and date data to be sent to the MiniCOP. This nonconstant data can be provided by internal ladder logic routines, the operator's station/panel, dedicated process reference clock, TDC 3000 Universal Station, etc.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
                    READ     SET     PUSH     PULL      D0       D1
                     2       2048    2049     2051      2052      80       81
LINE #1  --]/\[--------] / [--------] / [--------] / [--------] / [--------] / [----------NSKR--
                                                                                          2

                                                                                         A7
                    DTP     QTY                                                          87
LINE #2  --[K2]-----[K2]-------------------------------------------------------------(PSH)--
                     32      8                                                           2

                                                                                         D2
                                                                                         82
LINE #3  ---------------------------------------------------------------------------(L)--

                                                                                         D1
                                                                                         81
LINE #4  ---------------------------------------------------------------------------(L)--

                                                                                    DTP-LCH
                                                                                    2050
LINE #5  ---------------------------------------------------------------------------(L)--

LINE #6  ---------------------------------------------------------------------------EOS--
                                                                                          2

                    DTP      D0
                    2050     80
LINE #7  --] [--------] [-------------------------------------------------------------NSKR--
                                                                                         12

                                                                                         D1
                                                                                         81
LINE #8  ---------------------------------------------------------------------------(U)--

                                                                                         D2
                                                                                         82
LINE #9  ---------------------------------------------------------------------------(U)--

                                                                                    DTP
                                                                                    2050
LINE #10 ---------------------------------------------------------------------------(U)--

LINE #11 ---------------------------------------------------------------------------EOS--
                                                                                         12
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Lines 1 through 6 comprise the first skip block.

- **Line 1** contains the serial programming block monitor and sequence maintenance logic.

- **Line 2** provides the MiniCOP with two 16-bit values. The first value is used as the register location to which the data table pointer is to be set. The second value determines the number of data table registers that can be transferred into or out of the data table by a single PUSH or PULL instruction.

- **Lines 3 and 4** unconditionally SET control bits D2 and D1, which direct the MiniCOP to use the data just received at its most significant I/O address (A7) as the location at which to set the data table pointer and the maximum quantity of data table locations that can be transferred.

- **Line 5** sets (latches) coil 2050, which requests the Set the Data Table Pointer programming block to execute.

Lines 7 through 11 contain the Set the Data Table Pointer communication function in a Skip block that is executed only on command (that is, Line 7 N/O Contact 2050 "DTP" and N/O Contact 80 "D0" must both be true).

- **Lines 8, 9, and 10** unlatch (CLEAR or disable) control bits D1 and D2 and the DTP request coil 2050.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PUSH Data to the MiniCOP (Serial I/O)
Enable Control Bits D1 and D5

This communication function PUSHes data to the MiniCOP's data table through its most significant I/O address (A7). The PUSH data to the MiniCOP programming block can only transfer the number of values specified by the Set the Data Table Pointer function. Control bit D5, which determines the direction the data table pointer will move, is included in this programming block. A PUSH of data to the MiniCOP will place this data in the data table starting from the current position of the pointer.

```
                    READ     SET      DTP      PULL      D0       D1
                     3       2048     2049     2050      2052     80       81
LINE #1  --]/[--------] / [--------] / [--------] / [--------] / [--------] / [----------NSKR--
                                                                                          3

                                                                                         A7
                                                                                         87
LINE #2  --[K2]-----[K2]-----[K2]-----[K2]-----[K2]-----[K2]-----[K2]-----[K2]-------(PSH)--
           800      700      600      500      400      300      200      100          8

                                                                                         D5
                                                                                         85
LINE #3  ------------------------------------------------------------------------------(L)--

                                                                                         D1
                                                                                         81
LINE #4  ------------------------------------------------------------------------------(L)--

                                                                                        PSH
                                                                                       2051
LINE #5  ------------------------------------------------------------------------------(L)--

LINE #6  ------------------------------------------------------------------------------EOS--
                                                                                          3

                    PSH      D0
                   2051      80
LINE #7  --] [--------] [-------------------------------------------------------------NSKR--
                                                                                         13
                                                                                         D1
                                                                                         81
LINE #8  ------------------------------------------------------------------------------(U)--

                                                                                         D5
                                                                                         85
LINE #9  ------------------------------------------------------------------------------(U)--

                                                                                        PSH
                                                                                       2051
LINE #5  ------------------------------------------------------------------------------(U)--

LINE #10 ------------------------------------------------------------------------------EOS--
                                                                                         13
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Lines 1 through 6 comprise the first skip block:

- **Line 1** contains the serial programming block monitor and sequence maintenance logic.
- **Line 2** provides the MiniCOP with eight 16-bit data values.
- **Line 3** unconditionally enables control bit D5, which directs the pointer to increment from its current location when data is PUSHed to the MiniCOP's data table.
- **Line 4** unconditionally SETs control bit D1 which directs the MiniCOP to store the data just received at its most significant I/O address (A7) into the data table.
- **Line 5** sets (latches) coil 2051, which requests the PUSH data to the MiniCOP programming block to execute.

Lines 7 through 11 contain the PUSH Data to the MiniCOP communication function in a Skip block that is executed only on command (that is, Line 7 N/O Contact 2051 "PSH" and N/O Contact 80 "D0" must both be true).

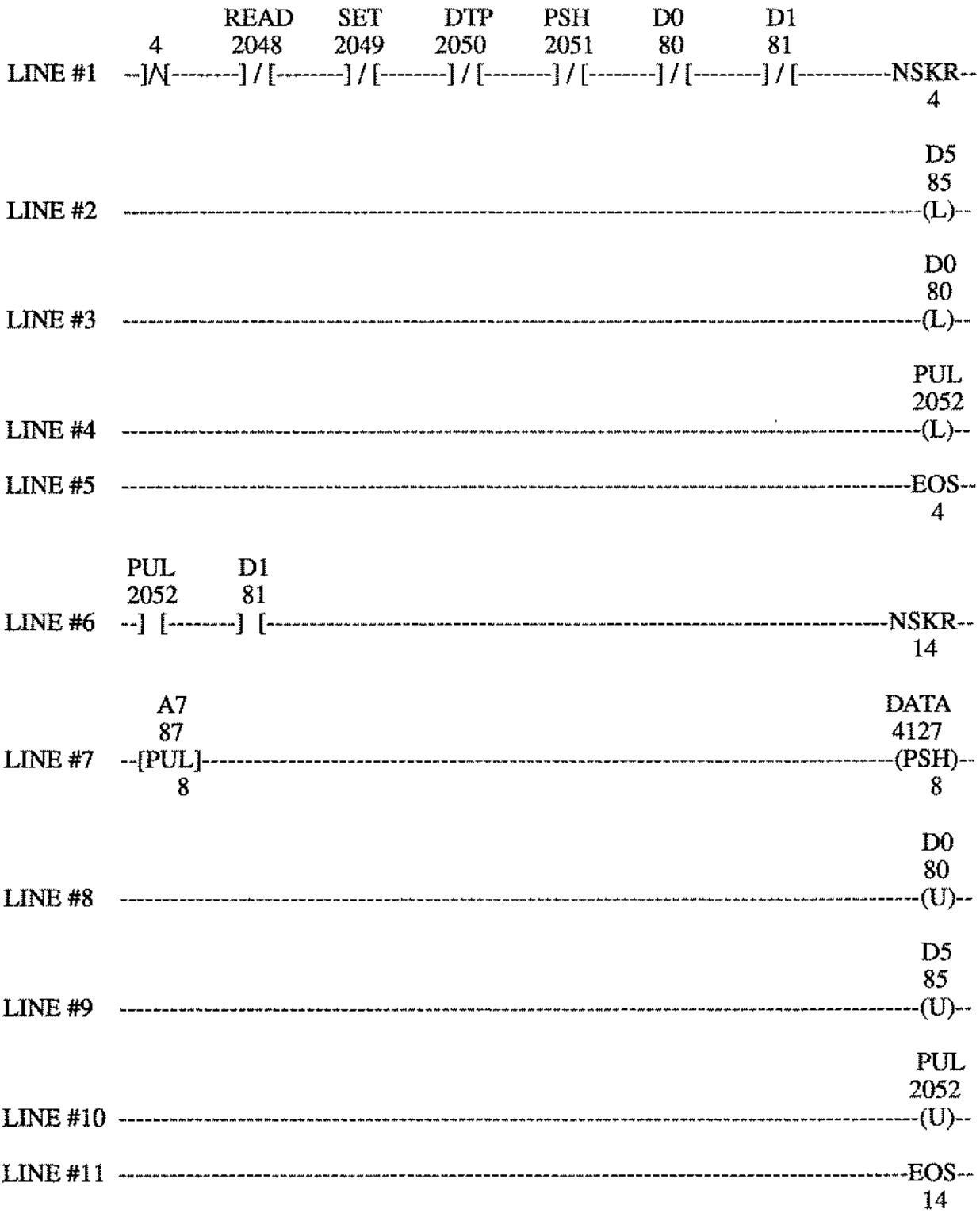
- **Lines 8, 9, and 10** unlatch (CLEAR or disable) control bits D1 and D2 and the PSH request coil 2051.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PULL Data from the MiniCOP (Serial I/O)
Enable Control Bits D0 and D5

This communication function PULLs data from the MiniCOP's data table through its most significant I/O address (A7). The PULL Data from the MiniCOP programming block can only transfer the number of values specified by the Set the Data Table Pointer function. Control bit D5, which determines the direction the data table pointer will move, is included in this programming block. A PULL of data from the MiniCOP will retrieve data from the data table starting from the current position of the pointer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Lines 1 through 5 comprise the first skip block.

- **Line 1** contains the serial programming block monitor and sequence maintenance logic.
- **Line 2** unconditionally enables control bit D5, which directs the pointer to increment from its current location when data is PULLed from the MiniCOP's data table.
- **Line 3** unconditionally enables control bit D0, which directs the MiniCOP to transfer the data in the specified number of 16-bit register locations to its most significant I/O address (A7).
- **Line 4** sets (latches) coil 2052, which requests the PULL data from the MiniCOP programming block to execute.

Lines 6 through 11 contain the PUSH data to the MiniCOP communication function in a Skip block that is executed only on command (that is, Line 6 N/O Contact 2052 "PUL" and N/O Contact 81 "D1" must both be true).

- **Line 7** PULLs the data from the MiniCOP and stores it into the 620 Platform Controller's data registers.
- **Lines 8, 9, and 10** unlatch (CLEAR or disable) control bits D0 and D5 and the PUL request coil 2052.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Reset the Data Table Pointer (Serial I/O)
Enable Bit D4

This communication function determines the position of the pointer after a PUSH Data to the MiniCOP or PULL Data from the MiniCOP operation. If control bit D4 is enabled, the pointer is returned to the location specified by the **last** Set the Data Table Pointer function, after the transfer has been completed. If disabled, the pointer does not reset after each data transfer. Subsequent data transfers would then occur from the current position of the pointer. If control bit D4 is not included in the ladder logic the MiniCOP will default to the disable operation during data transfers. Typically, the following two lines of logic will be included in either data transfer function when the pointer is to reset.

```
                                                                                                    D4
                                                                                                    84
LINE #1 ------------------------------------------------------------------------------------------( )--

                                                                                                    D4
                                                                                                    84
LINE #2 ------------------------------------------------------------------------------------------(U)--
```

Line 1 should be placed in the PUSH Data to the MiniCOP or PULL Data from the MiniCOP programming block before the line of logic that actually performs the transfer operation.

Line 2 is placed in the programming block after the line of logic that performs the data transfer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

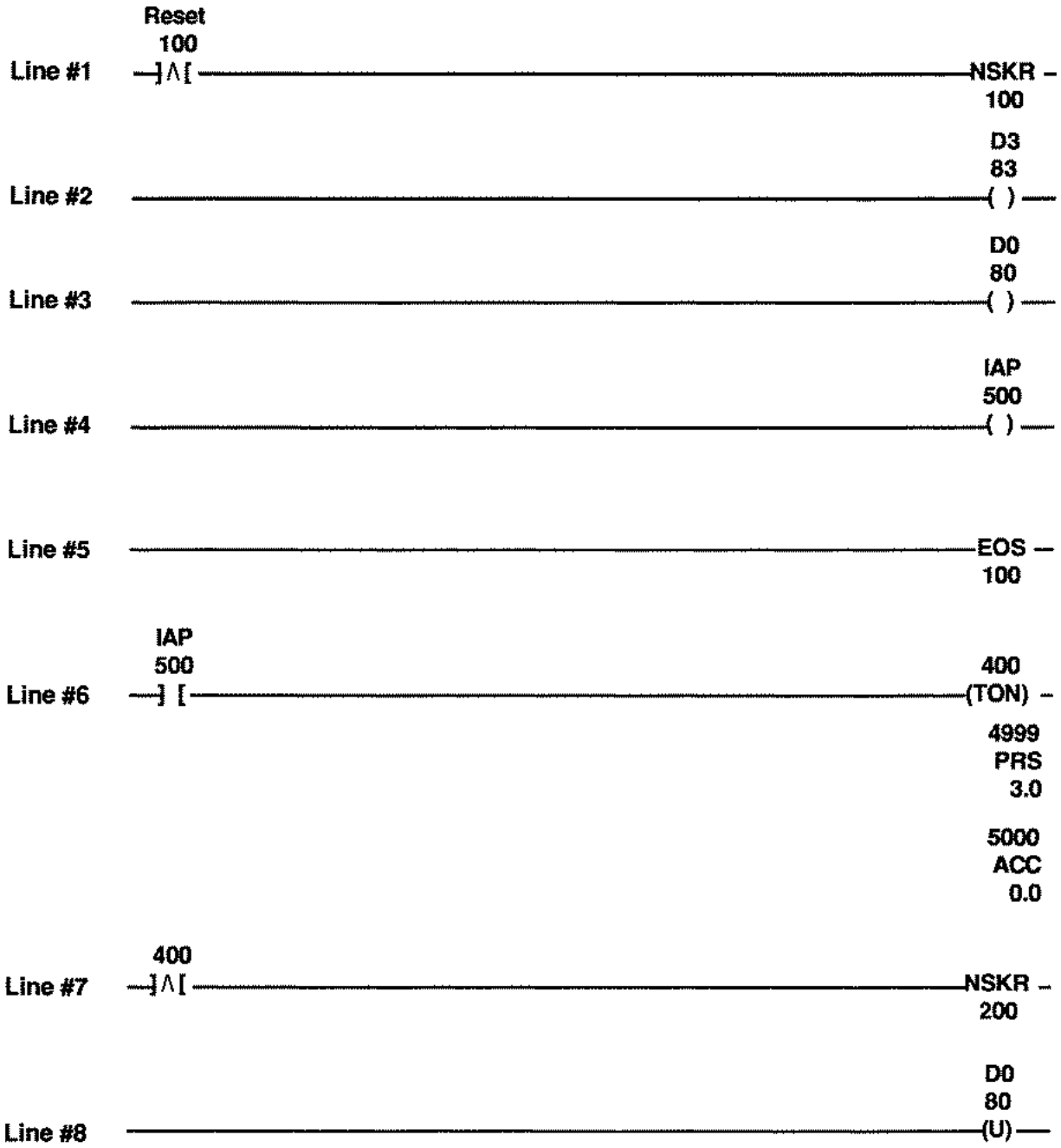
Set the Pointer Direction (Serial I/O)
Enable Bit D5

This control bit determines the direction of movement from the current data table location that the pointer will move during a PUSH to or PULL from the MiniCOP operation. If enabled, control bit D5 causes the pointer to increment. If disabled, control bit D5 causes the pointer to decrement. This control bit does not require its own programming block, but should be included in the PUSH Data to the MiniCOP or PULL Data from the MiniCOP blocks. The Set Pointer Direction function will default to the decrement mode of operation if its control bit is neither enabled or disabled by ladder logic. Refer to the PUSH Data to the MiniCOP and PULL Data from the MiniCOP program blocks for control bit D5 examples.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

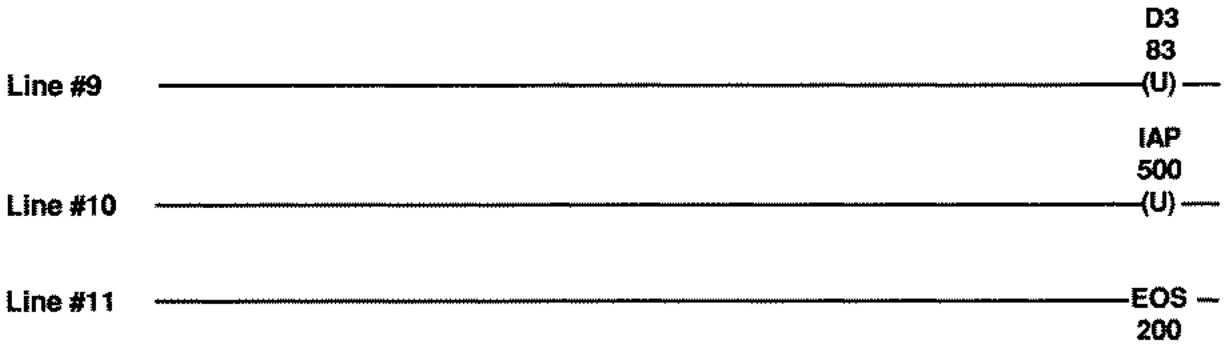
Reset the MiniCOP (Serial I/O)
Enable Bits D3 and D0

The MiniCOP can be remotely reset by a programming block in the 620 Platform Controller's ladder logic. Executing this programming block will have the same effect on the MiniCOP as a hardware reset (pressing the MiniCOP's toggle switch when configured for reset operation) or software reset (issuing the OS-9 command RESET on the SHELL command line).



continued on next page...

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Lines 1 and 5 contain the Reset the MiniCOP communication function in a Skip block that is executed only on command (that is, when transitional contact is true).

- **Lines 2 and 3** unconditionally SET control bits D3 and D0 which directs the MiniCOP to perform a reset operation. **Line 2** also provides the MiniCOP with the necessary time and date data to set the real-time clock.

- **Line 4** unconditionally SETs an internal address point (IAP).

Line 6 is a time delay, set to several tenths of a second, that allows the MiniCOP to execute the control bit's request prior to receiving further instructions from the programming block.

Lines 7 through 11 comprise a Skip block that unlatches (CLEARs or disables) control bits D0 and D3.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PARALLEL PROGRAMMING BLOCK TIME DELAYS

Time delays provide sufficient time for valid data to reach its destination and permit the MiniCOP to complete the previous operation before receiving further control bit commands from the 620 Platform Controller's ladder logic. When the MiniCOP is installed in a parallel I/O rack, proper timing maintains the integrity of the data in the communication functions.

A time delay consists of one or more lines of ladder logic that execute in a known amount of time. Different 620 Platform Controller models execute ladder logic at different speeds. The specific execution time per ladder logic instruction for a given 620 Platform Controller is provided in that processor's user manual. These times are provided as minimum and maximum execution times in microseconds. Each parallel programming block contains one or more blank lines of ladder logic to indicate the position of the required time delay. You must replace this blank line with the appropriate line or lines of time delay logic as determined by the 620 Platform Controller model.

PUSH/PULL instructions are used to create the necessary time delays when the MiniCOP is used with the 620-06/15/25 or 35 CPMs. The Delay instruction can be used in place of the PUSH/PULL instruction when the MiniCOP is used with the 620-11/12/14/1631/1633/or 36 CPMs.

Time Delays in the 620-06/15/25 or 35 CPMs

When using the 620-06/15/25 or 35 CPMs, PUSH/PULL instructions are used to create the necessary time delays. The following line of ladder logic provides the format for a PUSH/PULL delay line. Use this format combined with the information listed in Table 23 to create any necessary time delays.



REGADDR = Any unused 620 Platform Controller data register location. Use the same address for both the PUSH and PULL instructions. The same address can be repeated in every delay line for all time delay operations.

N = The number of registers to be PULLed and PUSHed (1-8)*.

NOTE
Some time delays cannot be achieved in one line of logic. In these instances simply repeat the delay logic, setting each line's N to equal the values given in Table 23.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 23 - PARALLEL PROGRAMMING BLOCK TIME DELAYS

620 Platform Controller	Delay Timing Values
620-06 LCS 620-15 LCS Series 9000	50µsec delay, N = 1 200µsec delay, N = 4 300µsec delay, N = 7
620-25 LCS 620-35 LCS Series 9100 Series 9200 Logic Manager	50µsec delay, N = 1 200µsec delay, N = 8 300µsec delay, N = 8 and N = 3 (requires two lines)

For example, the PUSH Data to the MiniCOP programming block uses two delay lines -- Lines 4 and 6. **Line 4** requires a minimum time delay of 40µsec while **Line 6** requires a minimum time delay of 289.2µsec. If the MiniCOP is installed in the parallel or local I/O slots of a 620-06 or 620-15 LC, the two lines of delay logic would appear as follows:

```
          DELAY                                                    DELAY
            4300                                                     4300
LINE #4   --[PUL]-----------------------------------------------------(PSH)--
              1                                                        1

          DELAY                                                    DELAY
            4300                                                     4300
LINE #6   --[PUL]-----------------------------------------------------(PSH)--
              7                                                        7
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Time Delays in the 620-11/12/14/1631/1633 or 36 CPMs

When using the 620-11/12/14/1631/1633 or 36 CPMs, the Delay instruction is used to create the necessary time delays. The following line of ladder logic provides the format for the required delay line. Use this format combined with the necessary time delay indicated in each parallel programming block description.

```
                                                    I/OADDR
--[DLA]-----------------------------------------------------(    )
  N
```

I/OADDR = Any unused 620 Platform Controller internal I/O address. The same address can be repeated in every delay line thereby using only one I/O address for all time delay operations.

N = The amount of the time delay in microseconds from 0 to 65530 in increments of 10.

CAUTION

Unlike the Push/Pull instruction based time delays, the Delay instruction does not permit processor interrupts to occur while the delay is in progress. When using the Delay instruction for communications operations with a MiniCOP residing in parallel I/O, multiply the minimum time delay given in the ladder logic examples by a factor of 1.5.

For example, the PUSH Data to the MiniCOP programming block uses two delay lines -- Lines 4 and 6. **Line 4** requires a minimum time delay of 40µsec while **Line 6** requires a minimum time delay of 289.2µsec. If the MiniCOP is installed in a parallel or local I/O slot of a 620-11/12/14/1631/1633 or 36 LCS, the two lines of delay logic would appear as follows:

```
            DELAY                                                  D0
                                                                   80
LINE #4   --[DLA]-----------------------------------------------------(    )--
              60

            DELAY                                                  D0
                                                                   80
LINE #6   --[DLA]-----------------------------------------------------(    )--
              440
```

NOTE

To establish time delays in this manner, simply repeat setting the delay time N to equal the appropriate values given in the parallel programming block descriptions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MiniCOP/PERSONALITY PROCEDURES TO PLCs

The personality procedures are programs (drivers) that are called from BASIC09 procedures. They provide a mechanism for the MiniCOP to communicate via its ASYNC ports to nonhost PLCs. In this use the term nonhost simply means that the MiniCOP is not installed in a manner such that it can communicate directly on the PLC's I/O bus. Personality procedures allow the MiniCOP to interface with Allen-Bradley and Modicon PLCs. Using a personality procedure, the MiniCOP can also interface with 620 Platform Controllers.

MiniCOP to 620 PLATFORM CONTROLLER PERSONALITY PROCEDURE

The 620 LCS Personality Procedure (Model No. 626-1012) allows the MiniCOP to communicate with any 620 Platform Controller. Communication is carried out from any of the MiniCOP's serial ASYNC ports to a serial communications port in the 620 Platform Controller. There are four possible serial ports in the 620 System which the MiniCOP can use to communicate. Two of these are option modules: the 620-0048 Single Port Data Collection Module (DCM), and the 620-0052 Dual Port Data Collection Module (DCM). The MiniCOP can also communicate with the serial port that is onboard the processor module of the 620-11/ -12/ -14/ -16/ -36 LCs, and the S9000e Loop and Logic Controllers.

Figures 87, 88, and 89 illustrate several example configurations where the MiniCOP communicates to a 620 Platform Controller via the 620 LCS Personality Procedure.

CAUTION

When calling a Personality Procedure from a BASIC09 Procedure, care must be taken in the allocation of sufficient workspace for operation of any communication buffers required. For specific detailed information regarding BASIC09 and the Personality Procedures, refer to the *BASIC09 and Personality Procedures* section of *Appendix I* .

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In Figure 87, the MiniCOP is installed in the local I/O slots of a 620-36 LCS. Through an ASYNC port, the MiniCOP is interfaced to a S9200e. The MiniCOP can monitor the operations in either or both devices while also allowing them to share data. The unused MiniCOP ASYNC ports can also be interfaced to other field devices.

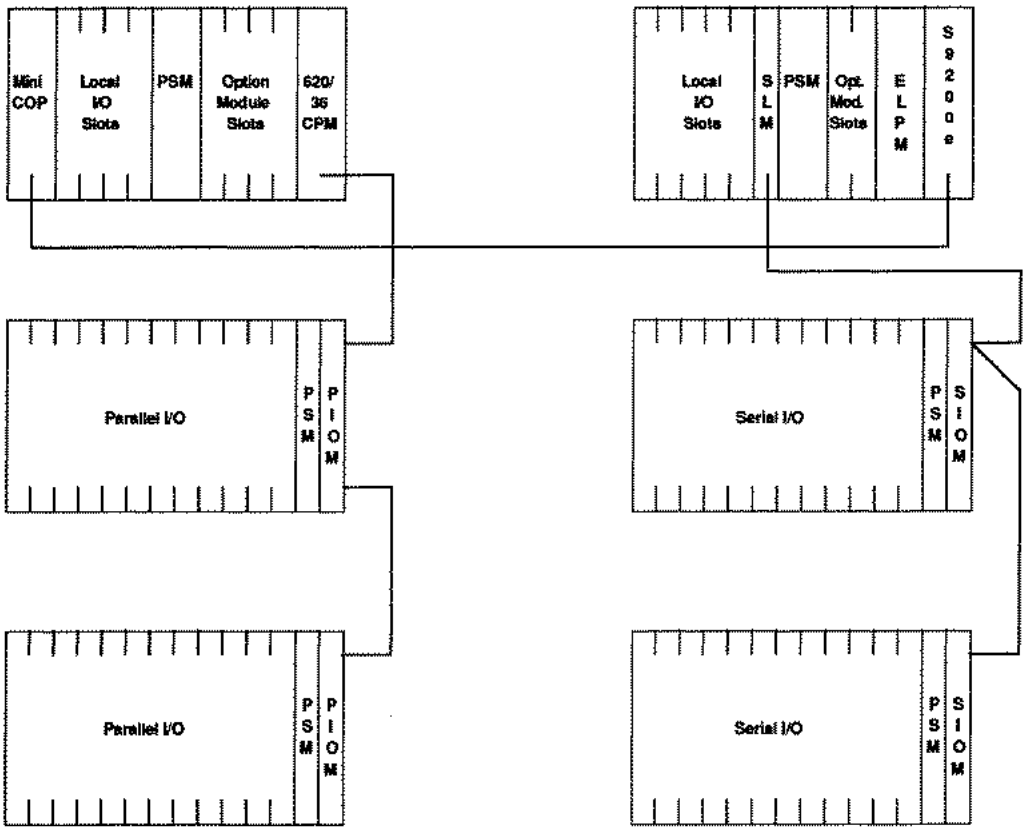


FIGURE 87 - MINICOP TO 620 PLATFORM CONTROLLER — EXAMPLE #1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In Figure 88, the MiniCOP is installed in the first parallel I/O rack of a Logic Manager. Through an ASYNC port, the MiniCOP is interfaced to a 620-35 LCS and a S9100E. The MiniCOP can monitor the operations in all three devices while also allowing them to share data. The unused MiniCOP ASYNC ports can also be interfaced to other field devices.

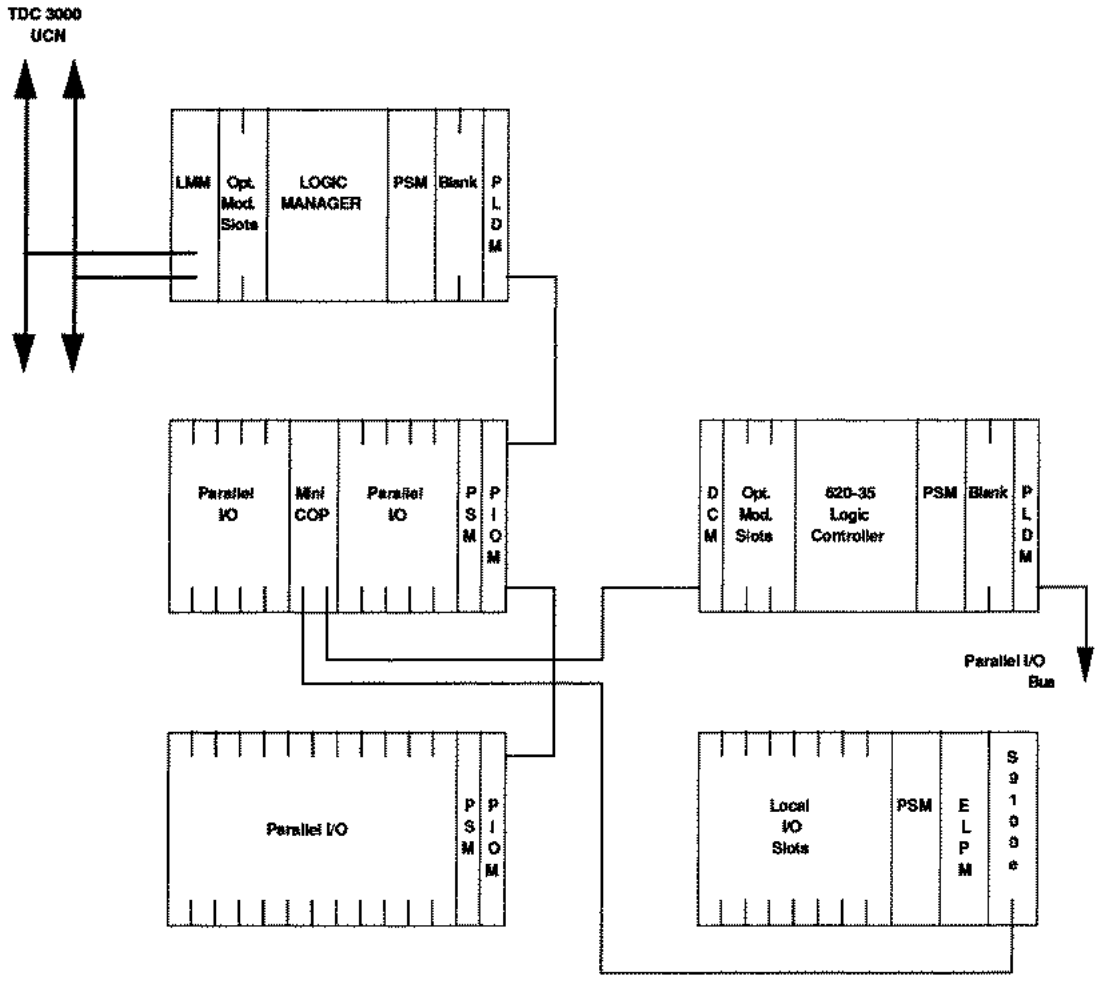


FIGURE 88 - MINICOP TO 620 PLATFORM CONTROLLER — EXAMPLE #2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In Figure 89, the MiniCOP is installed in a serial I/O rack of a 620-35 LCS. Through an ASYNC port the MiniCOP is interfaced to its host through a DCM. The unused MiniCOP ASYNC ports can also be interfaced to other field devices. In this manner the MiniCOP by-passes the I/O bus in favor of the personality procedure.

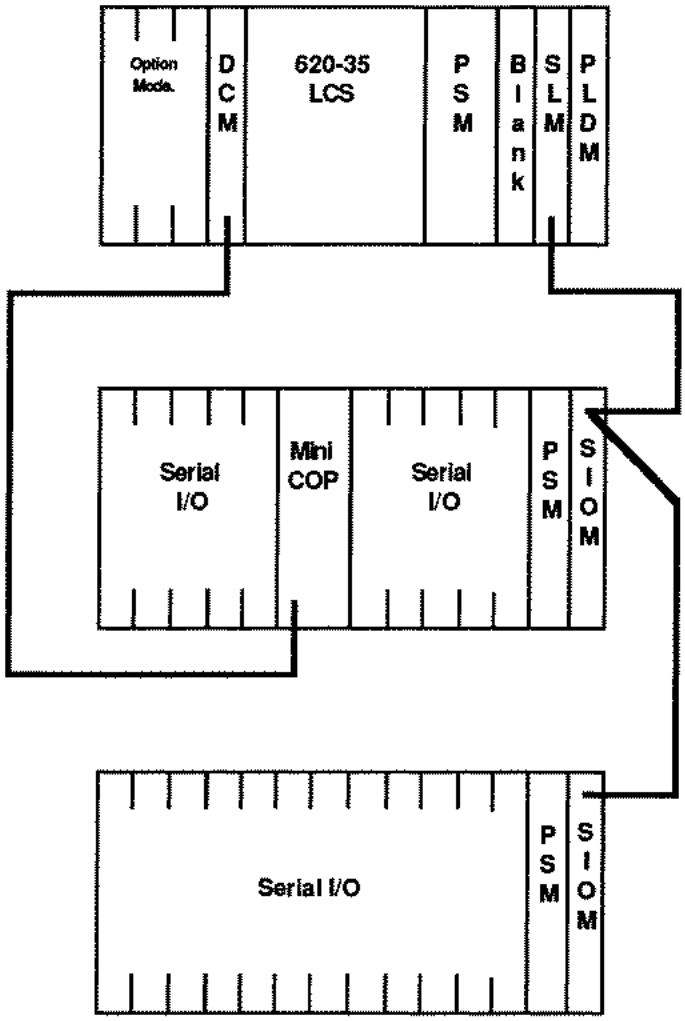


FIGURE 89 - MINICOP TO 620 PLATFORM CONTROLLER — EXAMPLE #3

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 24 provides a comparison of the MiniCOP to Host 620 Platform Controller using a personality procedure versus the I/O bus. Use this table as an outline to compare the two possibilities. The communications choice for any application will involve factors such as: amount and location of the data to be transferred, physical I/O location of the MiniCOP, ladder logic scan time, programming preferences or expertise, etc.

TABLE 24 - MINICOP TO HOST 620 PLATFORM CONTROLLER COMMUNICATION COMPARISON

	Using the I/O Bus	Using the Personality Procedures
Maximum number of 16-bit registers transferred per operation	8	256
Effect on ladder logic scan time	Each line of program block ladder logic in the communication functions adds to the overall scan time. Because of the small number of registers that can be handled per PUSH/PULL, many lines may be needed. Delay lines required when the MiniCOP is installed in parallel I/O also add time. Ladder logic instruction execution is typically measured in microseconds.	The CIM, DCM or MiniCOP (directly connected to a third generation processor) is given a window on the processor backplane at the start of each scan. During this time all data transfer operations take place. This "load" is typically measured in milliseconds per scan.
Ease of use	This requires you to program in ladder logic to build the necessary communication functions. These functions can only access the MiniCOP's data table. Programming in BASIC09 is also required to allow the MiniCOP to call and execute the IPC621 driver utility. IPC621 allows the MiniCOP to access its data table.	This requires you to program in BASIC09. All data transfer operations are performed via the personality procedure.
Speed of data transfer*	Approximately 5 to 7 transactions per second can be achieved.	Approximately 3 to 5 transactions per second can be achieved.

* Due to variables external to the MiniCOP, precise figures are not possible.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Eighteen operations are available through the 620 personality procedure. They allow reading and writing of sequential I/O addresses, sequential data registers, scattered I/O, and scattered data registers. Data can be PUSHed to or PULLed from selected 621 I/O Modules. Program header information and CIM or DCM status can be monitored by the MiniCOP. When the personality procedure is called and executed from a BASIC09 procedure, specific parameters must be passed. The parameters are validated, and if there are no errors, the operation is performed.

The 620 LCS Personality Procedure (Model No. 626-1012) disk contains two programs -- PPINT and IPC620. The programs are loaded to the module directory using the TLOAD command and can be copied to COPRAM if desired. PPINT is a generic program that controls the execution of the personality procedure and its parameters. A standard RUN statement within a BASIC09 procedure performs both the READ and WRITE operations. The RUN statement is constructed with a series of eleven parameters that must be programmed in a specific order. The statement appears as:

RUN PPINT (PP$, CP, PN$, NN, CC, QT, AD, DA(1), FM, PM, ER)

The parameters PP$ and PN$ are string variables -- DA(1) is a numeric array (type: integer or real) and the remaining parameters are numeric values (type: integer or real). Parameter definitions are as follows:

PP$ = Driver Name; "**IPC620** " (a [SPACE] must always follow the zero).

CP = Communication Parameter*; "**0**".

PN$ = Path Name; the port descriptor used to communicate with the DCM (for example, T1, T3, etc.).

NN = Slave or Node Number; the slave or node number of the called DCM (0-31).

CC = Command Code; the number corresponding to the desired READ/WRITE operation (0-17).

QT = Quantity; corresponds to the value "N" of the READ/WRITE operation command codes.

AD = PLC Address; used for sequential READ/WRITE operations.

DA(1) = Data Array; holds data and/or address for WRITE operations and data for READ operations.

FM = Flag Modifier; "**0**" (for future use).

PM = Packing Modifier; "**1**" specifies packed (16 I/O addresses), "**0**" specifies unpacked (single I/O address).

ER = Error Code; holds any error code generated by the operation.

*This parameter was required by the COP, but is not required by the MiniCOP. Enter a zero (0) to satisfy the syntax.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 25 lists the eighteen READ/WRITE operations available with the 620 LCS Personality Procedure, along with the command code, quantity, and absolute address limits. Actual upper address limits are determined by the particular model of 620 Platform Controller being accessed.

NOTE

For specific detailed information regarding the 620 LCS Personality Procedure (Model No. 626-1012), refer to the *IPC 620 Personality Procedure for MiniCOP User Manual* .

TABLE 25 - 620 LCS PERSONALITY PROCEDURE OPERATIONS

Code	Operation	Quantity	Address Limits
0	Read N System Status Registers	1-192	2049-2431
1	Read N Unsigned Registers	1-256	4096-8191
2	Read N Signed Registers	1-256	4096-8191
3	Read 16N I/O Addresses	1-256	0-8176
4	Write N Outputs	1-256	0-8191
5	Write 16N Outputs	1-256	0-8176
6	Write N Unsigned Registers	1-256	4096-8191
7	Write N Signed Registers	1-256	4096-8191
8	Read DCM Status	N/A	N/A
9	Read 16N Scattered I/O	1-64	0-8176
10	Read N Scattered Registers	1-64	4096-8191
11	Write N Scattered Outputs	1-64	0-8191
12	Write N Scattered Registers	1-64	4096-8191
13	PULL 16N I/O Addresses	1-8	0-2047
14	PUSH 16N I/O Addresses	1-8	0-2047
15	Read Program Date	N/A	N/A
16	Read Programmer	N/A	N/A
17	Read Program Title	N/A	N/A

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MiniCOP to ALLEN-BRADLEY PLCs PERSONALITY PROCEDURE

The Allen-Bradley Personality Procedure (Model No. 626-1014) allows the MiniCOP to communicate with any Allen-Bradley PLC that can be used with the KC, KD, KE, or KG Communication Controller Modules (CCM). Communications are carried out from any of the MiniCOP's serial ASYNC ports to a CCM connected to the Allen-Bradley PLC.

Figures 90, 91, and 92 illustrate several example configurations where the MiniCOP communicates to an Allen-Bradley PLC via the Allen-Bradley Personality Procedure.

Seven operations are available through the personality procedure. These operations allow reading and writing of bits, data, and diagnostics. When the personality procedure is called and executed from a BASIC09 procedure, specific parameters must be passed. The parameters are validated, and if there are no errors, the operation is performed.

CAUTION

When calling a Personality Procedure from a BASIC09 Procedure, care must be taken in the allocation of sufficient workspace for operation of any communication buffers required. For specific detailed information regarding BASIC09 and the Personality Procedures, refer to the *BASIC09 and Personality Procedures* section of *Appendix I* .

In Figure 90, the MiniCOP is installed in the local I/O slots of a 620-36 LCS. Through an ASYNC port the MiniCOP is interfaced to an Allen-Bradley PLC with the KG Communication Controller Module. The MiniCOP can monitor the operations in either or both devices while also allowing them to share data. The unused MiniCOP ASYNC ports can also be interfaced to other field devices.



FIGURE 90 - MINICOP TO ALLEN-BRADLEY PLC — EXAMPLE #1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In Figure 91, the MiniCOP is installed in the first parallel I/O rack of a Logic Manager. Through an ASYNC port the MiniCOP is interfaced to a KD Communications Controller Module. The KD CCM is connected to the Allen-Bradley Data Highway, which in turn is connected to two Allen-Bradley PLCs through KA CCMs. The MiniCOP can monitor the operations in all three devices while also allowing them to share data. The unused MiniCOP ASYNC ports can also be interfaced to other field devices.

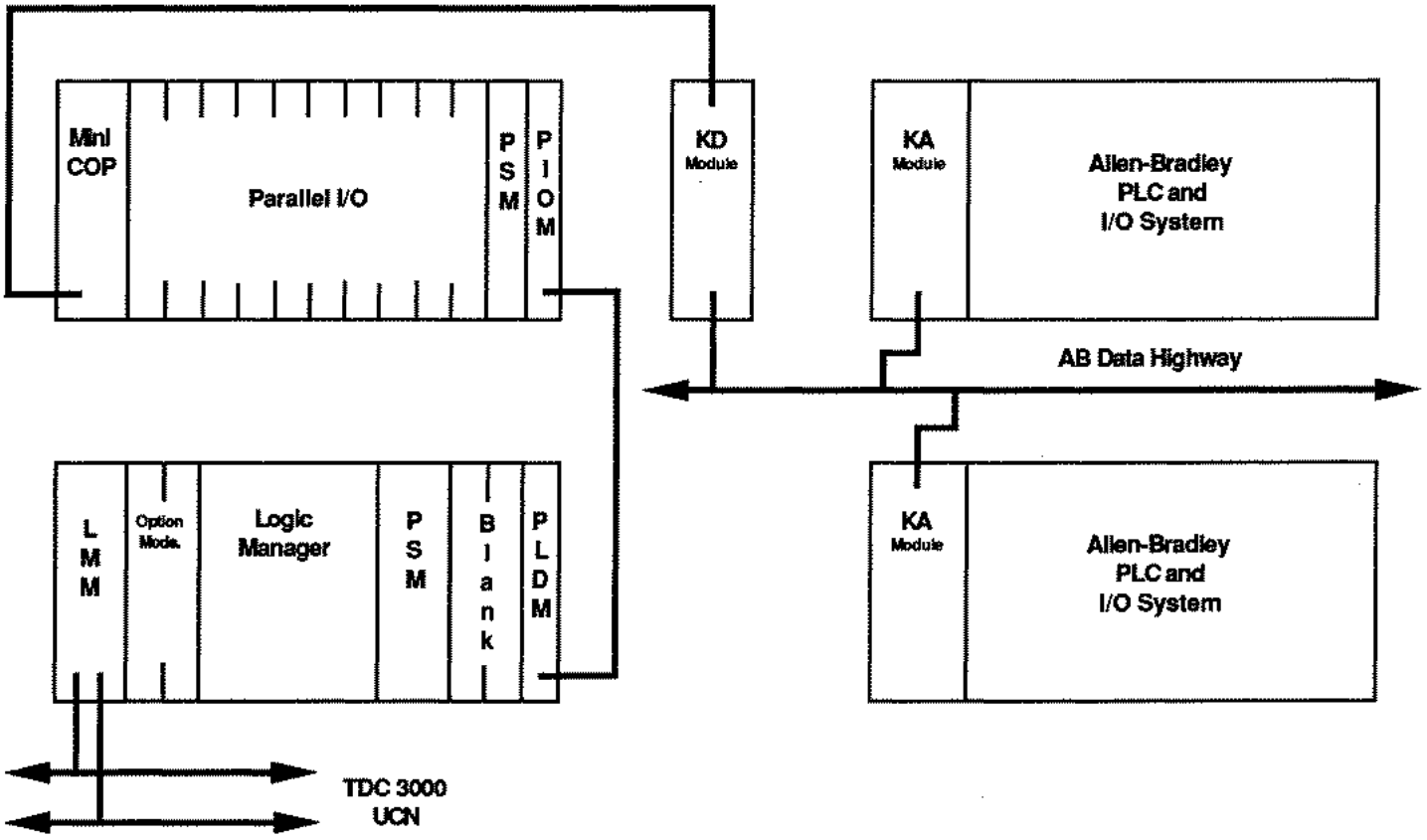


FIGURE 91 - MINICOP TO ALLEN-BRADLEY PLC — EXAMPLE #2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In Figure 92, the MiniCOP is installed in a serial I/O rack of a 620-35 LCS system. Through the ASYNC port, the MiniCOP is interfaced to a KC Communications Controller Module that is installed in an Allen-Bradley I/O rack. This KC CCM is connected to the Allen-Bradley data highway providing access to an Allen-Bradley PLC through its KA CCM.

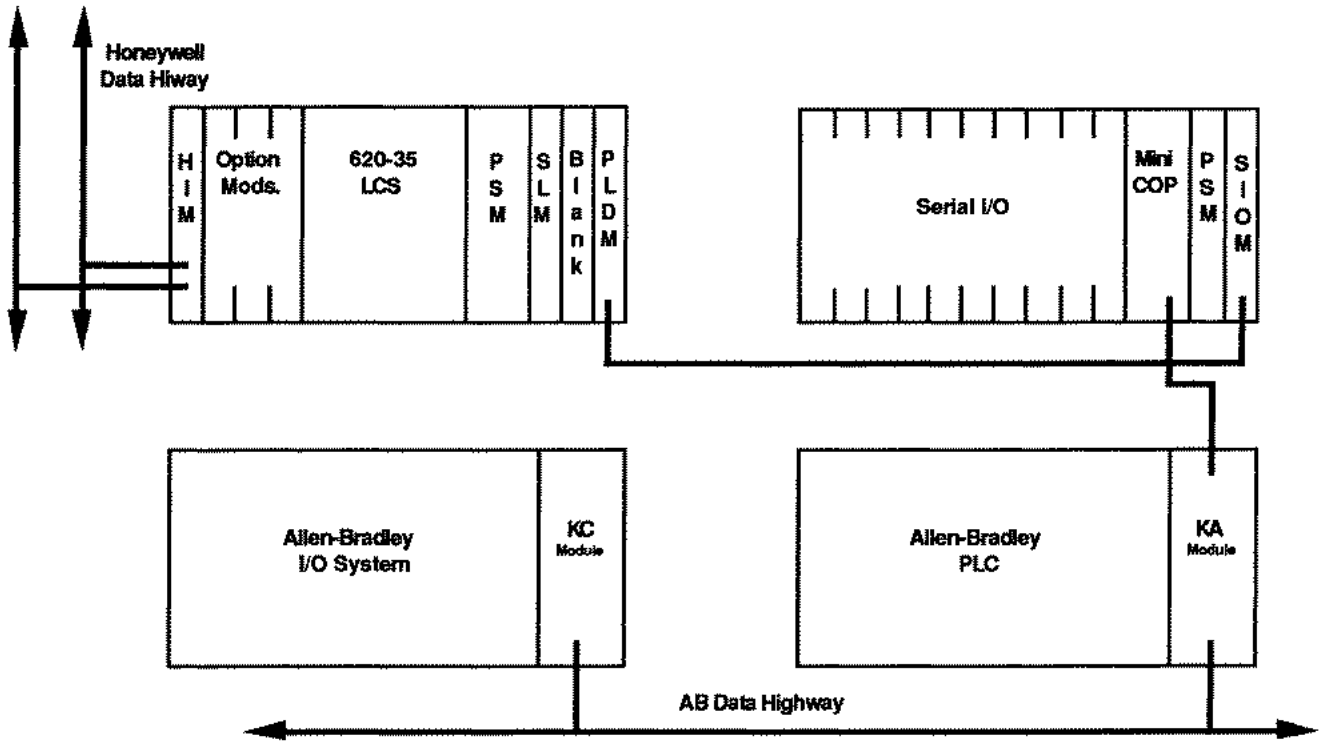


FIGURE 92 - MINICOP TO ALLEN-BRADLEY PLC — `EXAMPLE #3

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Allen-Bradley Personality Procedure (Model No. 626-1014) disk contains two programs -- PPINT and MINIABDH. The programs are loaded to the module directory using the TLOAD command and can be copied to COPRAM if desired. PPINT is a generic program that controls the execution of the personality procedure and its parameters. A standard RUN statement within a BASIC09 procedure performs both the READ and WRITE operations. The RUN statement is constructed with a series of eleven parameters that must be programmed in a specific order. The statement appears as:

RUN PPINT (PP$, CP, PN$, NN, CC, QT, AD, DA(1), FM, PM, ER)

The parameters PP$ and PN$ are string variables -- DA(1) is a numeric array (type: integer or real) and the remaining parameters are numeric values (type: integer or real).

Parameter definitions are as follows:

PP$ = Driver Name; "**MINIABDH** " (a [SPACE] must always follow the zero).

CP = Communication Parameter; "**0**".

PN$ = Path Name; the port descriptor used to communicate with the CCM (for example, T1, T3, etc.).

NN = Slave or Node Number; the slave or node number in octal of the called CCM.

CC = Command Code; the number corresponding to the desired READ/WRITE operation (0-6).

QT = Quantity; corresponds to the value "N" of the READ/WRITE operation command codes.

AD = PLC or CCM Address; used for sequential READ/WRITE operations.

DA(1) = Data Array; holds data and/or address for WRITE operations and data for READ operations.

FM = Flag Modifier; its various uses include the bit position within the specified word where an operation is to begin, 3 or 4 digit BCD format, and diagnostic command codes.

PM = Packing Modifier; "**1**" specifies packed (16-bit format), "**0**" specifies unpacked (single-bit format).

ER = Error Code; holds any error code generated by the operation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 26 lists the seven READ/WRITE operations available with the Allen-Bradley Personality Procedure, along with the command code, quantity, and flag modifier information.

NOTE

For detailed information regarding the Allen-Bradley Personality Procedure (Model No. 626-1014), refer to the *IPC 620 A-B Personality Procedure for MiniCOP User Manual.*

TABLE 26 - ALLEN-BRADLEY PERSONALITY PROCEDURE OPERATIONS

Code	Operation	Quantity	Flag Modifier
0	Status Check	bytes returned	N/A
1	Bit Read	1-64 bits	bit position within address word
2	16 (N) Bit Read	1-64 words	N/A
3	Register Read	1-64 words	0 = 3-digit BCD, non-zero = 4-digit BCD
4	Bit Write	1-bit:	bit position within addressed word
		16 bits:	N/A
5	Register Write	1-64 words	0 = 3-digit BCD, non-zero = 4-digit BCD
6	Diagnostics	command dep.	diagnostic command code

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MiniCOP to MODBUS PERSONALITY PROCEDURE

The MODBUS Personality Procedure (Model No. 626-1016) allows the MiniCOP to communicate with Modicon PLCs (Models 984, 584, 484, 384,and 184) and the J475 or J347 Communication Interface Modems (CIM). When communicating with a Modicon 484 PLC, the MiniCOP is first cabled into a Modicon J475 CIM that is cabled to the 484. If a Modicon 384 or 184 PLC is used, the MiniCOP is first cabled into a J347 CIM that is cabled to either PLC. The Modicon 984 and 584 PLCs have built-in CIM ports that can be cabled directly to the MiniCOP's ASYNC ports.

NOTE

The Modbus Personality Procedure supports the **Modbus ASCII Protocol**. It does not support the Modbus RTU Protocol.

Seven operations are available through the personality procedure. They allow reading and writing of input and output registers, data registers, and diagnostics. When the personality procedure is called and executed from a BASIC09 procedure, specific parameters must be passed. The parameters are validated, and if there are no errors, the operation is performed.

CAUTION

When calling a Personality Procedures from a BASIC09 Procedure, care must be taken in the allocation of sufficient workspace for operation of any communication buffers required. For specific detailed information regarding BASIC09 and the Personality Procedures, refer to the *BASIC09 and Personality Procedures* section of *Appendix I* .

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

In Figure 93, the MiniCOP is installed in the local I/O slots of a 620-36 LCS. Through an ASYNC port the MiniCOP is interfaced to a Modicon J475 Communication Interface Modem. The J475 CIM is interfaced to a Modicon 484 PLC. The MiniCOP can monitor the operations in either or both devices while also allowing them to share data. The unused MiniCOP ASYNC ports can also be interfaced to other field devices.

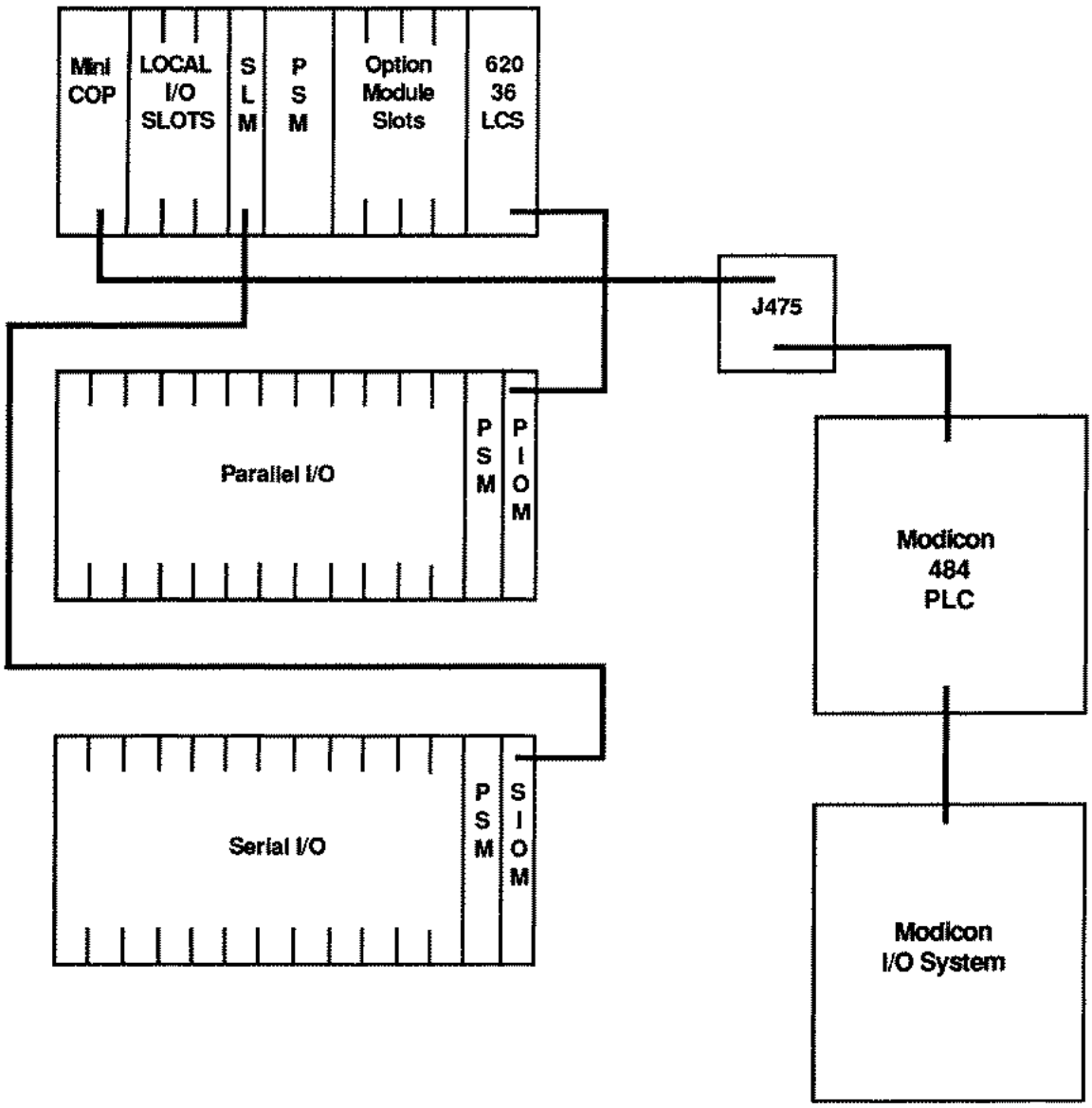


FIGURE 93 - MINICOP TO MODBUS EXAMPLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The MODBUS Personality Procedure (Model No. 626-1016) disk contains two programs -- PPINT and MINIMMBUS. The programs are loaded to the module directory using the TLOAD command and can be copied to COPRAM if desired. PPINT is a generic program that controls the execution of the personality procedure and its parameters. A standard RUN statement, within a BASIC09 procedure, performs both the READ and WRITE operations. The RUN statement is constructed with a series of eleven parameters that must be programmed in a specific order. The statement appears as:

RUN PPINT (PP$, CP, PN$, NN, CC, QT, AD, DA(1), FM, PM, ER)

The parameters PP$ and PN$ are string variables, DA(1) is a numeric array (type: integer or real) and the remaining parameters are numeric values (type: integer or real). Parameter definitions are as follows:

PP$ = Driver Name; "**MINIMMBUS** " (a [SPACE] must always follow the zero).

CP = Communication Parameter; "**0**".

PN$ = Path Name; the port descriptor used to communicate with the MODBUS device (for example, T1, T3, etc.).

NN = Slave or Node Number; the slave or node number of the called MODBUS device.

CC = Command Code; the number corresponding to the desired READ/WRITE operation (0-6).

QT = Quantity; corresponds to the value "N" of the READ/WRITE operation command codes.

AD = MODBUS Device Address; used for sequential READ/WRITE operations.

DA(1) = Data Array; holds data and/or address for WRITE operations and data for READ operations.

FM = Flag Modifier; "**0**".

PM = Packing Modifier; "**1**" specifies packed (16 bit format), "**0**" specifies unpacked (single bit format).

ER = Error Code; holds any error code generated by the operation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 27 lists the seven READ/WRITE operations available with the MODBUS Personality Procedure, along with the command code, quantity, and high-low address.

NOTE

For specific detailed information regarding the Modbus Personality Procedure (Model No. 626-1016), refer to the *Modbus Personality Procedure for MiniCOP User Manual.*

TABLE 27 - MODBUS PERSONALITY PROCEDURE OPERATIONS

Code	Operation	Quantity	High-Low Address*
0	Status Check	N/A	—
1	Read Input Registers	1-2048	1001 – 2000 10001 – 20000
2	Read Output Registers (Coils)	1-2048	1 – 1000 1 – 10000
3	Read Registers	1-125	3001 – 5000 30001 – 50000
4	Write Outputs (Coils)	1-800	1 – 1000 1 – 10000
5	Write Registers	1-100	4001 – 5000 40001 – 50000
6	Diagnostics	N/A	—

*The first entry high-low address for each command applies to PC Model Numbers 184, 384, and 484. The second entry applies only to Model Number 584.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

UPDATING 627 COP BASIC09 APPLICATION PROGRAMS

BASIC09 application programs that were written for use in the 627-7000 COP and the COP Personality Procedures can be adapted for use with the 627 MiniCOP and its personality procedures. Simply change the COP Personality Procedure driver name parameter in the RUN PPINT statement to the driver name used by the MiniCOP personality procedure. To do this, load the source code version of the BASIC09 application program into the BASIC09 workspace, enter the edit mode, and type in the following command:

C*/"<COP driver name>"/"<MiniCOP driver name>" [RETURN]

This will change all references to the <COP driver name> to the <MiniCOP driver name> found in the BASIC09 applications program. Check for and make any other application-dependent changes as needed. Finally save and/or pack the new version.

CAUTION

When calling a Personality Procedures from a BASIC09 Procedure, care must be taken in the allocation of sufficient workspace for operation of any communication buffers required. For specific detailed information regarding BASIC09 and the Personality Procedures, refer to the *BASIC09 and Personality Procedures* section of *Appendix I*.

MiniCOP TO OTHER TARGET DEVICES

You may choose to interface a MiniCOP to some target device other than those capable of communicating with the Personality Procedures. This is entirely possible provided that all electrical and protocol specifications can be met. In this case, you may choose to develop your own custom driver using the OS-9 MAKDESC command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 5.
BASIC09 Programming Language

About This Section

This section presents a programming background for the BASIC09 Programming Language. A BASIC09 commands and statements listing is also provided, along with a BASIC09 tutorial.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 OPERATIONAL OVERVIEW

INTRODUCTION

BASIC09 is a structured BASIC programming language developed for the Motorola 6800 and 68000 microprocessors by Microware Inc. The Honeywell 627 MiniCOP uses the Motorola 6809 microprocessor.

In addition to the typical or standard statements and commands found in most versions of BASIC, BASIC09 includes many useful elements of the PASCAL programming language. This allows programs to be modular, well-structured, and to be able to use sophisticated data structures. BASIC09 permits full access to most OS-9 system commands and functions.

BASIC09 uses an interactive compiler which provides the fast execution speed of a compiled language along with the ease of use and memory efficiency of an interpreted language.

Included in BASIC09 is a powerful text editor, a multipass compiler, a run-time interpreter, a high-level interactive debugger, and a system executive. These individual components have been integrated to provide a user-friendly, highly interactive programming resource.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PROGRAMMING FEATURES

BASIC09 like OS-9 incorporates a number of features to assist you in real-world applications, including:

- **REENTRANT PROGRAMMING MODULES** - these modules, once loaded, can be used by several different processes at the same time reducing memory requirements and increasing execution speed.

- **UNIFIED I/O SYSTEM** - provides a single interface standard for all devices, allowing them to be used interchangeably.

- **POWERFUL TYPE COMMAND** - allows you to develop data bases for use in BASIC09 procedures with multiple fields per record.

- **OPTIONAL LINE NUMBERS** - BASIC09 encourages structured, modular programs where line numbers are not required, except when using subroutines or similar statements.

- **INTERMEDIATE COMPILING** - as each line of a procedure is entered when creating BASIC09 procedures it is compiled, which greatly reduces the debug phase of program development.

- **PACK COMMAND** - packing a BASIC09 procedure performs an additional compile that removes all unnecessary or nonexecutable statements, reducing memory usage, and providing security against unauthorized program changes.

- **INTERACTIVE DEBUGGER** - allows you to set breakpoints and trace program execution step-by-step.

- **FIVE DATA TYPES** - REAL, INTEGER, BYTE, STRING, AND BOOLEAN data types are supported by BASIC09, which also provides for automatic type conversion to allow you to process data more efficiently.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GENERAL PRACTICE

General practice when working in BASIC09 is to first create a new workspace by calling BASIC09. Procedures (BASIC09 programs) are created and edited only within the workspace and are then saved to COPRAM and/or floppy or hard disk via the Terminal Emulator. These saved procedures can be in the form of source code (I-code) or packed code. When a process or application is to be executed, individual procedures can be loaded back into the workspace. Usually all procedures needed in a specific application are grouped together under one name, but a library of procedures may also be used. This library might contain "generic" procedures that can be used by several different applications. In this manner the programmer needs only create the procedures unique to a specific application, the remainder can be made of these prebuilt library procedures. Typically, these library procedures are stored somewhere in COPRAM and are called and executed when required by any application currently running in the MiniCOP.

MODES OF OPERATION

BASIC09 has four modes of operation: BASIC09 mode, edit mode, execution mode, and debug mode. Each mode of operation has a particular role in procedure development and execution. Figure 94 illustrates the four BASIC09 modes and their relationship to each other, and to the OS-9 operating system. Also included in this figure are the commands and keyboard interrupts that are effective in each mode. The arrows indicate mode changes that occur when that particular command or keyboard interrupt is executed. Keyboard interrupts are enclosed in brackets ([ ]).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

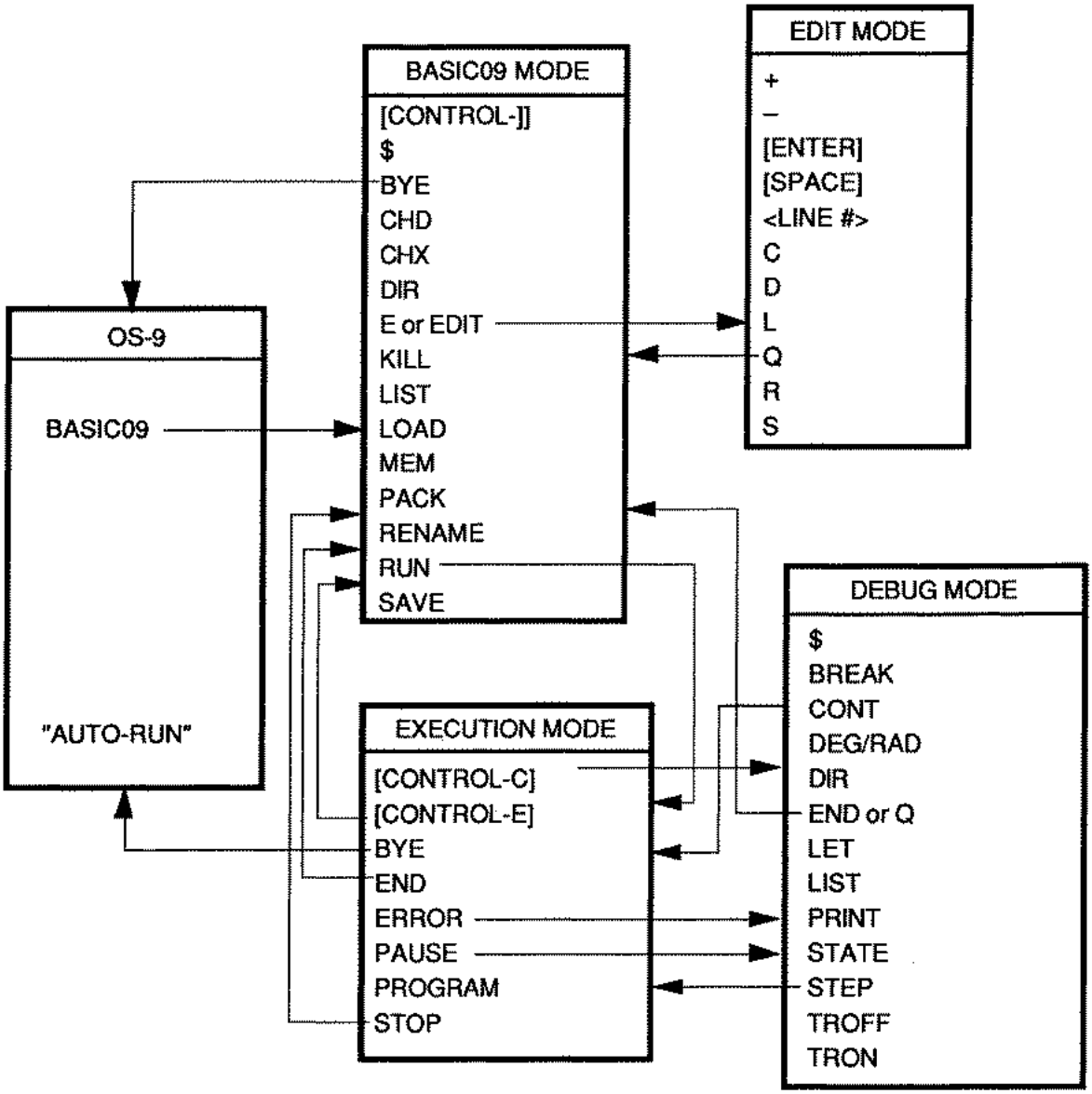


FIGURE 94 - BASIC09 MODE RELATIONSHIPS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 MODE

The BASIC09 mode is the first encountered when entering BASIC09. System commands, such as SAVE and LOAD, are executed in this mode. Also executed in this mode are commands that allow BASIC09 to interact with the OS-9 operating system. The BASIC09 mode is automatically entered when exiting any other mode of operation, except when the BYE command is used to exit the execution and BASIC09 modes.

BASIC09 mode commands are processed by a command interpreter, which always identifies itself by displaying the B: prompt. Commands can be entered in either upper or lower case characters. Commands can be made to operate on a specific procedure or on all procedures within the workspace with the * option. If a command is given without a procedure being named, the current working procedure is assumed. The current working procedure is the last procedure executed or worked on. It is always preceded in a listing of the working directory by an asterisk. Table 28 provides a summary listing of the BASIC09 mode commands and keyboard interrupts (control characters).

TABLE 28 - BASIC09 MODE COMMANDS SUMMARY

Command with Syntax	Operation
[CONTROL-]]	End of file, keyboard interrupt (control character code).
$[cmd]	Calls OS-9 SHELL and executes [cmd].
BYE	Exits BASIC09 and returns to OS-9, unsaved procedures are lost.
CHD <pathlist>	Changes current data directory to <pathlist>.
CHX <pathlist>	Changes current execution directory to <pathlist>.
DIR	Displays all procedures in the workspace.
E or EDIT [proc name]	Calls the edit mode and the [proc name] creates a new procedure if the name is not found.
KILL	Erases named procedure from the workspace.
LIST	List the contents of the current or named procedure.
LOAD	Loads a procedure from COPRAM to the workspace.
MEM	Displays, changes or de-allocates the current workspace memory size.
PACK	Compiles the named procedure or procedures.
RENAME	Changes a procedure name.
RUN	Executes a designated or current procedure.
SAVE	Saves a source code procedure to COPRAM or MiniCOP port.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EDIT MODE

The edit mode is the one mode in which all BASIC09 procedures are developed and edited. It is entered from the BASIC09 mode by means of the E or EDIT command. The edit mode is identified by the E: prompt. BASIC09 procedures are stored in memory in a compiled, intermediate form called I-code. I-code is compact, can be executed rapidly, and can be reconstructed to the original source code form. Compiling and decompiling are performed automatically by BASIC09 and are transparent to the user. Compiling allows BASIC09 to report syntax errors and reduces the amount of memory required to store the procedure. Commands can be made to operate all occurrences of a string or line number workspace with the *, rather than a single occurrence within the specific or current procedure. Table 29 provides a summary listing of the edit mode commands and keyboard interrupts (control characters).

TABLE 29 - EDIT MODE COMMANDS SUMMARY

Command with Syntax	Operation
[RETURN]	Move the edit pointer forward by one line.
+	Move the edit pointer forward by one line.
+[#]	Move the edit pointer forward by [#] of lines.
+*	Move the edit pointer to the end of the procedure.
–	Move the edit pointer backward by one line.
–[#]	Move the edit pointer backward by [#] of lines.
–*	Move the edit pointer to the beginning of the procedure.
[SPACE]	Insert unnumbered line ahead of edit pointer.
[line#] <text>	If [line#] exists, replace line with new <text>. If [line#] does not exist, insert numbered line ahead of edit pointer or next higher [line#].
c/[old]/[new]	Change [old] string to [new] string.
c*/[old]/[new]	Change all occurrences of [old] string to [new] string, from pointer position.
d[#]	Delete [#] of lines from edit pointer.
d*	Delete all lines in the procedure.
l[#]	List [#] of lines from the edit pointer.
l*	List all lines in the procedure.
q	Quit edit mode and return to BASIC09 mode.
r[start#,increment#]	Renumber all line numbers from edit pointer.
r*[start#,increment#]	Renumber all line numbers in the procedure.
s[string]	Search for [string] from the edit pointer.
s*[string]	Search entire procedure for [string].

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXECUTION MODE

The execution mode is entered via the RUN command from the BASIC09 mode or as a result of the auto-run feature. Technically, this mode involves the execution of one or more procedures currently inside the BASIC09 workspace or stored in COPRAM. A BASIC09 procedure can call other procedures and can nest in the same manner as subroutines. The RUN command can pass parameters to a procedure if such parameters are expected.

The auto-run feature is called from OS-9. It allows you to call the BASIC09 module and execute a specific procedure or procedures in one simple step. For example, suppose you are currently in OS-9 and wish to run a procedure called **palletizer**. The following two examples illustrate how both the step-by-step and auto-run methods perform this action:

STEP-BY-STEP EXAMPLE

OS9:BASIC09

B:RUN palletizer

AUTO-RUN EXAMPLE

OS9:BASIC09 palletizer

Once placed in the execution mode, BASIC09 remains in that mode until one of five of the following events occur:

1. An END, STOP, or BYE statement is executed by the BASIC09 procedure while executing.

2. The keyboard interrupt, [CONTROL-E], is entered to exit the currently-executing procedure and return to the BASIC09 mode.

3. The keyboard interrupt, [CONTROL-C], is entered forcing BASIC09 to exit the currently-executing procedure and enter the debug mode.

4. A run-time error occurs, which suspends the currently executing BASIC09 procedure and enters the debug mode.

5. A PAUSE statement is encountered in the BASIC09 procedure, which suspends the currently-executing BASIC09 procedure and enters the debug mode.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The execution mode displays no prompt of its own and is therefore transparent to you as the user except for the generation of any output that might be directed to a monitor, printer, etc. Like the other modes of operation, the execution mode also accepts commands and keyboard interrupts. A major difference is that only the keyboard interrupts can be entered when you are in this mode. The commands are programmed into a BASIC09 procedure and are executed when the procedure is run. Table 30 provides a summary listing of the execution mode commands and keyboard interrupts (control characters).

TABLE 30 - EXECUTION MODE COMMANDS SUMMARY

Command with Syntax	Operation
[CONTROL-C]	Halt execution of current running procedure and go to the Debug Mode, keyboard interrupt (control character code).
[CONTROL-E]	Exit current running procedure, keyboard interrupt (control character code).
BYE	Exits BASIC09 and returns to OS-9; unsaved procedures are lost.
END	Returns execution to calling procedure.
ERROR	Generates an error code.
PAUSE	Suspends execution and enters debug mode.
STOP	Terminates procedure execution.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DEBUG MODE

The BASIC09 debug mode offers a powerful set of debugging commands. These commands are used to examine the inner workings of the procedure. The debug mode lets you watch the procedure execute in "slow motion", and allows testing using the actual procedure names and statements.

The debug mode is entered from the execution mode as the result of an error, a user-entered [CONTROL-C] keyboard interrupt, or a user-programmed PAUSE statement in the procedure. The debug mode is identified by the D: prompt. In addition, the debug mode suspends and displays the name of the BASIC09 procedure currently being executed. For example, assume a BASIC09 procedure called **xmit.data** is currently executing. During execution, if one of the previously mentioned actions occurs, the following message will be displayed:

```
BREAK: PROCEDURE xmit.data
D:
```

From this point the debug commands can be used to: execute the procedure one line at a time, terminate the procedure, or resume normal execution after temporarily ignoring the error. Table 31 provides a summary listing of the debug mode commands and keyboard interrupts (control characters).

TABLE 31 - DEBUG MODE COMMANDS SUMMARY

Command with Syntax	Operation
$ [cmd]	Calls OS-9 SHELL and executes [cmd].
BREAK <proc name>	Sets a breakpoint in the named procedure.
CONT	Continues normal procedure execution.
DEG	Evaluates angle in degree units.
DIR	Displays all procedures in the workspace.
END or Q	Exits debug mode and returns to BASIC09 mode.
LET <var> = <expr>	Same as LET statement; not applicable on user-defined data structures.
LIST	Lists suspended procedure.
PRINT [#<expr>]	Same as PRINT statement; not applicable on user-defined data structures.
RAD	Evaluates angles in radian units.
STATE	Lists calling order of all active processes.
STEP[#]	Executes [#] line from current location.
TROFF	Trace off; deactivates trace mode.
TRON	Trace on; activates trace mode.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 WORKSPACE

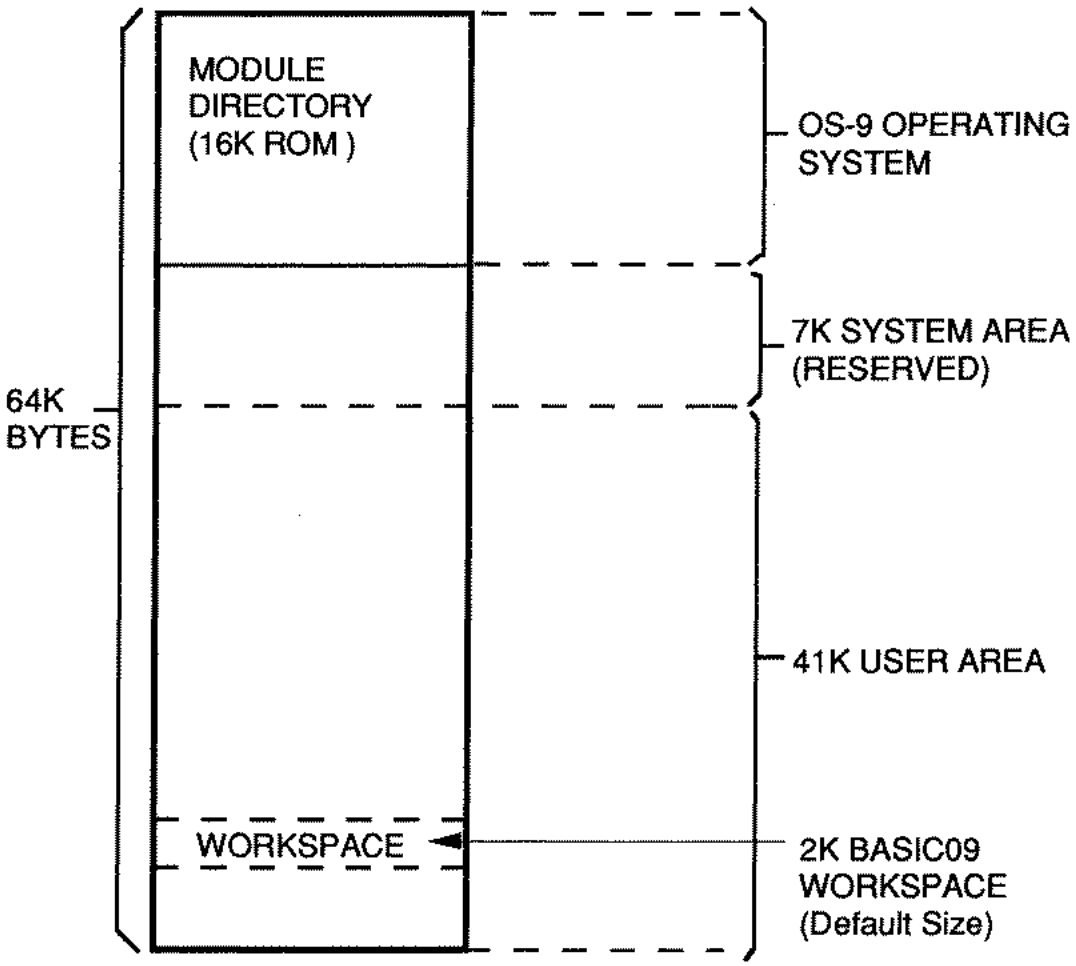
Every time BASIC09 is called, a block of memory, known as the BASIC09 workspace, is set aside to provide an area for the creation and editing of procedures. This BASIC09 workspace is created in the MiniCOP's main memory. The workspace concept allows you to group together a number of processes that relate to a single process or application in a logical manner. Modular program structure and the workspace combine to make the programmer's task easier while making efficient use of the system's memory resources. A workspace is created each time BASIC09 is called. Entering BASIC09 at the OS9: prompt will create a BASIC09 workspace and cause the BASIC09 mode to be entered. The following illustrates the messages that are displayed when BASIC09 is entered at the OS9: prompt:

```
OS9:BASIC09
Copyright 1980 by Motorola and Microware
Reproduced Under License
BASIC09
Ready
B:
```

Each new workspace will be created with the default size of 2Kbytes. Approximately 1.2Kbytes are reserved for internal use and the remaining 767 bytes are available for temporary storage of BASIC09 procedures and procedure variables. Figure 95 illustrates the MiniCOP's main memory after one BASIC09 workspace has been created at the default size of 2Kbytes.

Using the DIR command from the BASIC09 mode will display the listing of procedures in the workspace by name, proc-size, and data-size. Proc-size indicates the actual source code size of the procedure while data size indicates the amount of memory allocated for the procedure's variables. At the end of this listing the amount of free, unused workspace memory is displayed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



NOTE: Workspace sizes are approximate.

FIGURE 95 - MAIN MEMORY DETAIL

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHANGING THE WORKSPACE SIZE

There are two basic methods that can be used to increase the BASIC09 workspace from its default size. With either method the total amount of free main memory will govern the maximum size of a workspace.

METHOD 1

In this method, you can allocate a specific amount of main memory for the workspace at the same time it is being created. When calling BASIC09, you can accept the default size of 2Kbytes, request a number of 256 byte pages, or request the amount of memory in Kbytes. The following examples illustrate these three possible options for increasing the workspace size using Method 1.

OS9:BASIC09	Loads and executes BASIC09 with the 2Kbyte default size (2048).
OS9:BASIC09 #16	Loads and executes BASIC09 and requests 16 pages of main memory to be allocated to the workspace (16 x 256 = 4096).
OS9:BASIC09 #16K	Loads and executes BASIC09 and requests 16Kbytes of main memory to be allocated to the workspace (16384).

This method can also be combined with the auto-run feature.

METHOD 2

In this method you can change the specific amount of main memory allocated to the workspace after it has been created. This is performed from the BASIC09 mode using the MEM command. The MEM command allows you to increase or decrease the size of the workspace. The MEM command, when entered with no parameter, will display the current amount of memory assigned to a workspace. Adding a parameter to the command will increase or decrease the workspace size based on the amount specified. This parameter specifies the number of bytes to be allocated to the workspace and can be entered as a decimal or hexadecimal value when preceded by a **$**. If the value specified is not a multiple of 256, BASIC09 will automatically round up to the next highest multiple. The following examples illustrate the three possible options for increasing the workspace size using Method 2.

B:MEM	Will display the size of the current workspace.
B:MEM 4095	Requests a change in workspace size to the decimal value of 4095 bytes.
B:MEM $4000	Requests a change in workspace size to the hexadecimal value of 4000 bytes (16384 decimal).

NOTE

When using either Method 1 or 2 and requesting a specific amount of BASIC09 workspace memory, if the MiniCOP responds with the message "What?", the specified amount is greater than the available memory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NESTED WORKSPACES

Nested or multiple BASIC09 workspaces can exist in main memory at the same time. These workspaces can either be interleaved with multiple SHELLs or exist paralleled in the same SHELL. Nested workspaces can be used creatively. Multitasking permits several procedures in several workspaces to be executed concurrently. Nesting workspaces are one way of organizing procedures by application. Because of the reentrant nature of OS-9 program modules, several procedures in different workspaces can access the same data files or utility programs.

INTERLEAVED WORKSPACES

To interleave workspaces BASIC09 calls the OS-9 SHELL, which in turn can be used to call BASIC09. Each time this occurs, a new workspace and a new SHELL are created. Entering $BASIC09 at the B: prompt: exits the workspace ($ calls the SHELL from BASIC09), creates a new OS-9 SHELL, and creates a new workspace. The original or all preceding workspaces will still reside in memory. Figure 96 illustrates a block diagram where four OS-9 SHELLs and three BASIC09 workspaces are interleaved in memory.

SHELL 1 is created when the MiniCOP is first initialized or powered-up.

Workspace 1 and SHELL 2 are created by entering BASIC09 at the OS9: prompt.

Workspace 2 and SHELL 3 are created by the command $BASIC09 being entered at the B: prompt in workspace 1.

Workspace 3 and SHELL 4 are created by the command $BASIC09 being entered at the B: prompt in workspace 2.

Each new workspace has no contact or common files with any of its predecessors.

To exit the last (current) BASIC09 workspace and OS-9 SHELL, use either the BYE command or the [CONTROL-]] keyboard interrupt at the B: prompt. Be sure to SAVE any and all BASIC09 procedures to COPRAM before exiting each workspace. BYE and [CONTROL-]] will completely remove the workspace and all procedures it contained from main memory.

NOTE

The OS-9 Command, RESET, and the MiniCOP's toggle switch, when configured for reset operation, will de-allocate all multiple workspaces.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

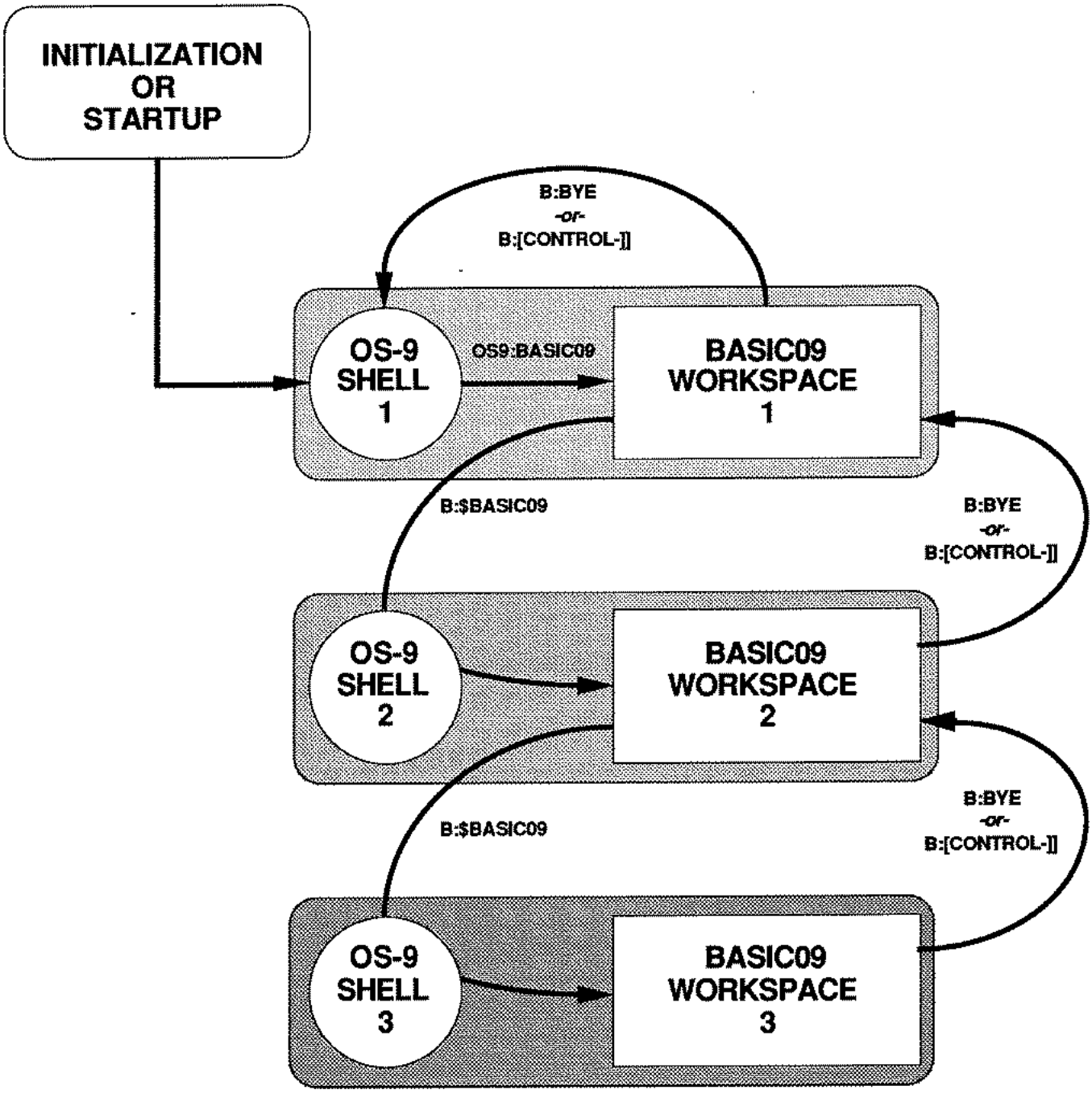


FIGURE 96 - INTERLEAVED BASIC09 WORKSPACES

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PARALLEL WORKSPACES

Multiple BASIC09 workspaces can be nested in parallel (concurrently), without creating additional SHELLs, using the command $EX BASIC09 at the B: prompt. $ calls the SHELL from BASIC09 and EX is a built-in OS-9 command that, in this case, creates a new workspace from the original SHELL. Figure 97 illustrates a block diagram where three BASIC09 workspaces have been created in memory under the same SHELL.

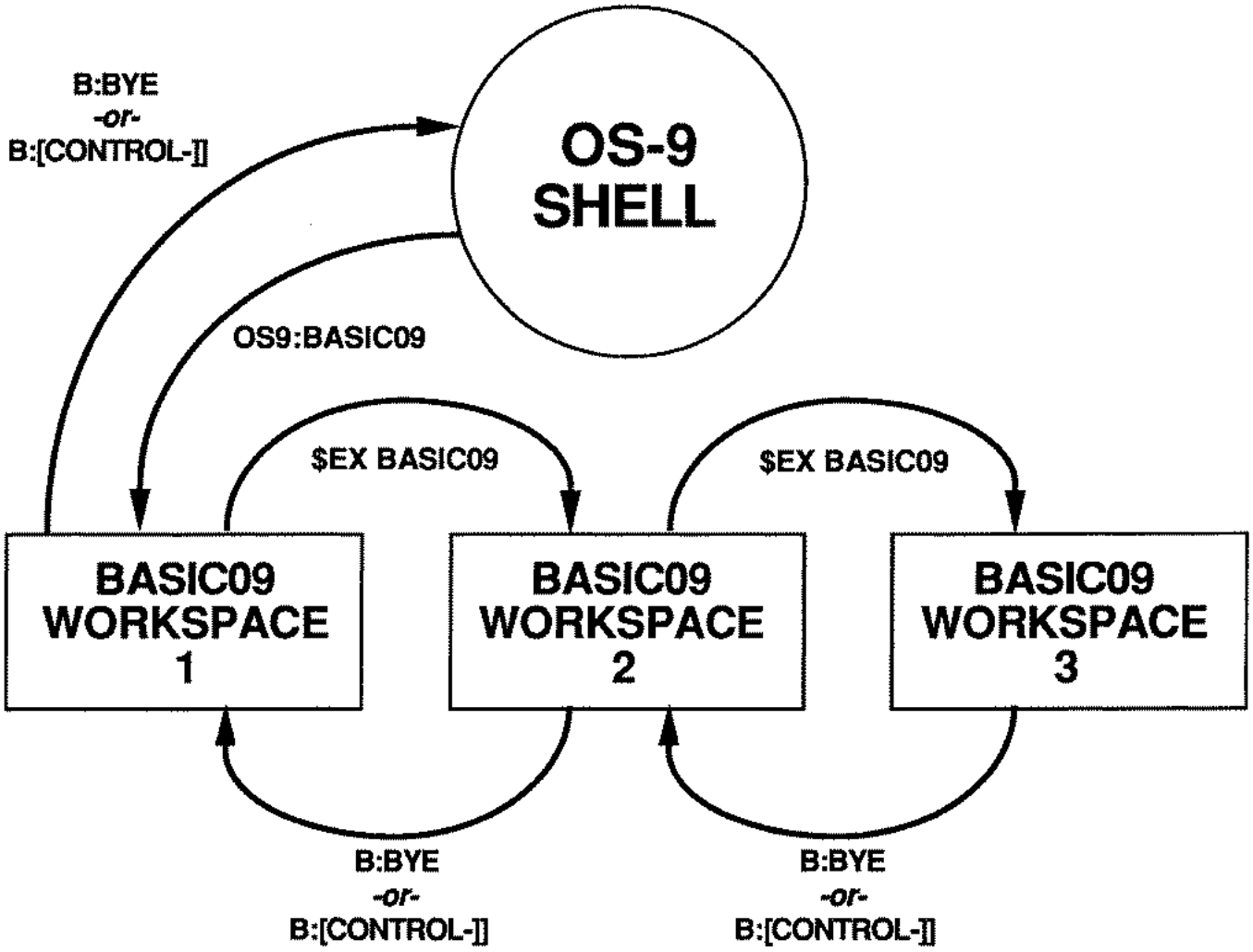


FIGURE 97 - PARALLEL BASIC09 WORKSPACES

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To exit the last (current) BASIC09 workspace and OS-9 SHELL, use either the BYE command or the [CONTROL-]] keyboard interrupt at the B: prompt. Be sure to SAVE any and all BASIC09 procedures to COPRAM before exiting each workspace. BYE and [CONTROL-]] will completely remove the workspace and all procedures it contained from main memory.

CAUTION

Use care when moving between BASIC09 and the OS-9 SHELL to prevent the creation of unnecessary workspaces. Each workspace and SHELL will reserve portions of main memory, which will reduce the memory available for other uses such as creating BASIC09 procedures.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MODULAR PROGRAMMING

Modular programming involves the breaking down of a complicated process into its constituent parts. It means creating a number of small, specialized procedures instead of one large procedure. These procedures can call upon one another and be used as building blocks. The modular approach simplifies program development, testing, and debugging. Modular programming will also allow long complicated procedures (too large to be loaded into memory) to be broken down into smaller manageable pieces.

An example of modular programming could be a group of procedures that COLLECT data from an industrial process by sampling a field device connected to a MiniCOP port every 30 seconds. Next, the procedures might be programmed to CONVERT this data from binary to decimal integers every hour, then CALCULATE the median value and standard deviation, and finally create a coordinate PLOT for this set of values. Additionally, a fifth procedure may be used as an ERROR recovery operation that shifts resultant data into a storage buffer. Figure 98 illustrates a simplified flowchart for this modular program.

These five procedures perform standard functions that could be used by many applications. If these procedures reside in the workspace or COPRAM, they can be called upon by any number of procedures and given specific application-related data to process. A library of such procedures can then be stored in COPRAM or on floppy- or hard-disk via the Terminal Emulator.

The following, **application1**, is a unique procedure that calls (or can call) each of the five previously-described procedures from a library. This procedure could be multitasked so that it is running concurrently with other procedures. Being modular in design, **application 1** reduces the amount of memory required by the procedure at any given time.

application1

```
        DIM X, Y, Z, M, N:INTEGER
        DIM PATH:BYTE
        DIM data.array(120):REAL
10      ON ERROR GOTO 100
20      RUN COLLECT              (calls and executes COLLECT)
        X=100
30      RUN CONVERT              (calls and executes CONVERT)
        Y=250
40      RUN CALCULATE            (calls and executes CALCULATE)
        Z=1000
50      RUN PLOT                 (calls and executes PLOT)
100     RUN ERROR(M, N)          (calls and executes the ERROR recovery
                                  procedure)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

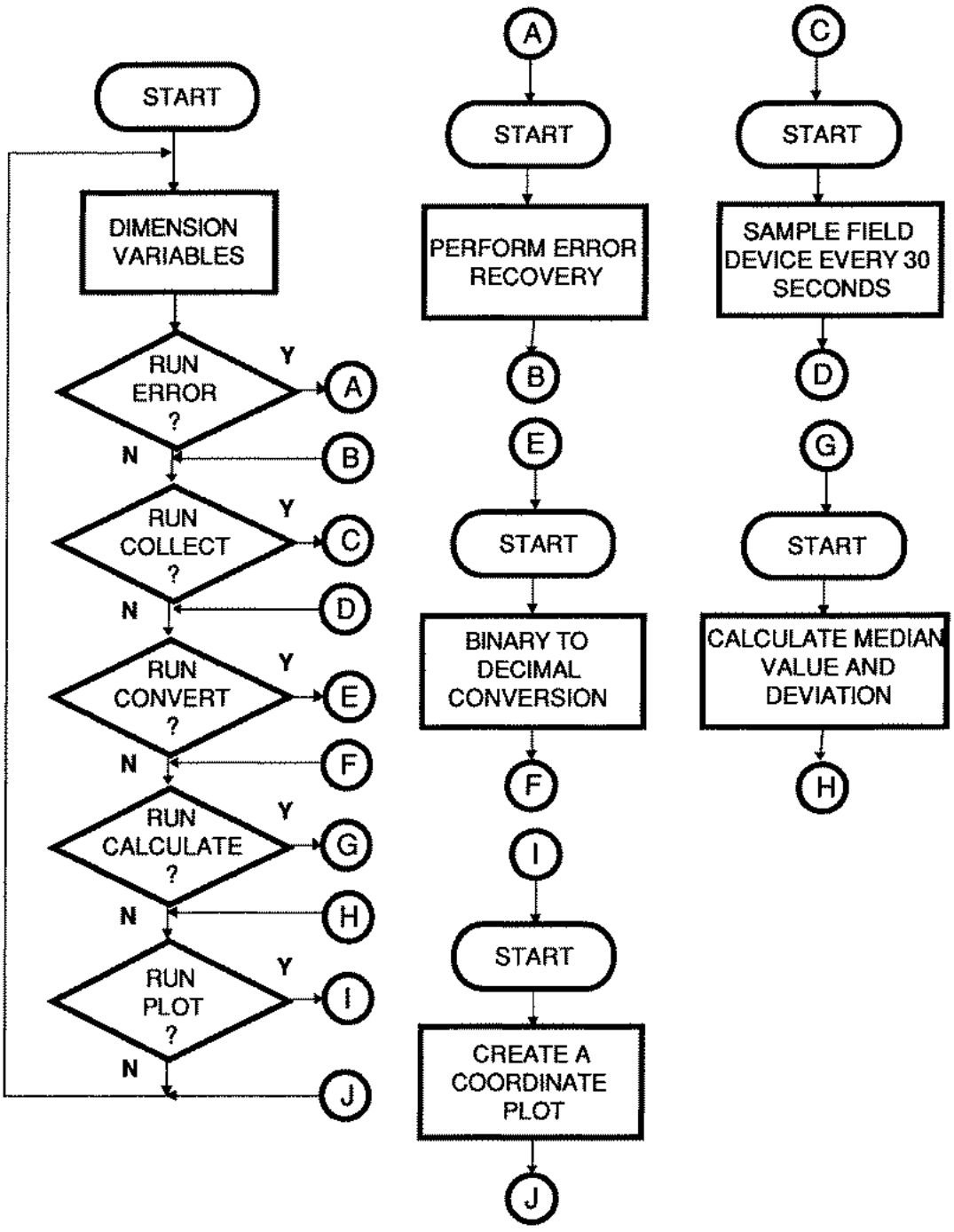


FIGURE 98 - SIMPLIFIED MODULAR PROGRAM FLOWCHART

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 PROGRAMMING BACKGROUND

PROCEDURE NAMES

The same rules that govern for the naming of OS-9 files apply to BASIC09 procedure names.

NOTE
Refer to *Naming OS-9 Files* in the *Working with OS-9 Files* subsection of the *OS-9 Operating System* section of this manual.

LINE NUMBERS

Program line numbers are optional in BASIC09. Structured programming languages like BASIC09 seldom require numbered lines, except to accompany the GOTO and GOSUB statements. Since line numbers occupy valuable memory and are not necessary, it is normally prudent to use them sparingly.

In BASIC09 line numbers can range from 1 to 32767 and are local to each procedure. That is, the same line number may appear in different procedures even when multitasking. The line number is the first element of the line (precedes the BASIC09 statement), and must be separated from the statement by a [SPACE] character.

If line numbers are not used the first character on the line must be a [SPACE]. Entering a statement that is not proceeded by a space will display the message "What?" indicating that the compiler did not understand the last entry. The following example illustrates a portion of a BASIC09 procedure as it is entered in the edit mode. Line numbers are included on some lines and not on others.

```
E: INPUT X
E: IF X>200 THEN GOSUB 100:
"

additional BASIC09 code

"
E: 100 PRINT "ERROR 'VALUE MUST BE FROM 0 TO 200'"
E: PRINT "PLEASE TRY AGAIN."
E: RETURN
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

COMMANDS AND STATEMENTS

The BASIC09 instruction set includes both commands and statements. Statements are used within a BASIC09 procedure to institute an action or to direct a program execution sequence. Commands are used in the BASIC09 and debug modes to perform operations on the procedure. Statement examples will include DIM, FOR/NEXT, PAUSE, INPUT, etc., while command examples include SAVE, LIST, STEP, PACK, etc. Some operations, such as PRINT and RUN, can be performed both as statements and as commands.

NOTE

For a complete listing of the BASIC09 commands and statements, refer to the *BASIC09 Commands and Statements Listing*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RESERVED WORDS

Certain words have special meaning within BASIC09. These reserved words include all BASIC09 commands, statements, functions, and operators. Reserved words cannot be used as variable names, array names, or other procedure arguments.

Reserved words should be separated from data or other parts of a BASIC09 statement (as syntax allows) by special characters known as delimiters. The [SPACE], comma, semicolon, and colon are a few examples of BASIC09 delimiters.

Table 32 contains a complete list of reserved BASIC09 words. It is also possible to create additional reserved words. For example, if the OS-9 MAKDESC (make descriptor) command was used to create a device descriptor, which is unique to a specific field device, then that descriptor's name should be included in the BASIC09 reserved words for that specific application.

TABLE 32 - BASIC09 RESERVED WORDS

ABS	DO	KILL	PRINT	STRING
ACS	ELSE	LAND	PROCEDURE	SUBSTR
ADDR	END	LEFT$	PUT	T1
AND	ENDEXIT	LEN	RAD	T2
ASC	ENDIF	LET	READ	T3
ASN	ENDLOOP	LNOT	REAL	T4
ATN	ENDWHILE	LOG	REM	TAB
BASE	EOF	LOG10	REPEAT	TAN
BOOLEAN	ERR	LOOP	RESTORE	THEN
BYE	ERROR	LOR	RETURN	TO
BYTE	EXEC	LXOR	RIGHT$	TP
CHAIN	EXITIF	MID$	RND	TRIM$
CHD	EXP	MOD	RUN	TROFF
CHR$	FALSE	NEXT	SEEK	TRON
CHX	FIX	NOT	SGN	TRUE
CLOSE	FLOAT	ON	SHELL	TYPE
COS	FOR	OPEN	SIN	UNTIL
CREATE	GET	OR	SIZE	UPDATE
DATA	GOSUB	PARAM	SQ	USING
DATE$	GOTO	PAUSE	SQR	V4
DEG	IF	PEEK	SQRT	VAL
DELETE	INPUT	PI	STEP	WHILE
DIM	INT	POKE	STOP	WRITE
DIR	INTEGER	POS	STR$	XOR

SYNTAX NOTATION

The description of each BASIC09 statement or command includes a syntax listing. A special notation is used to identify the types of modifiers needed for the individual command or statement.

NOTE

Refer to the *BASIC09 Commands and Statements Listing* for additional details regarding syntax notation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DATA REPRESENTATION, STORAGE, AND STRUCTURE

Data can be represented in many forms. It can be numbers used for counting or measurement. Data in the form of variables and constants can be composed of letters, numbers, and punctuation. Even within the same classification or type of data there are many different methods of representation. BASIC09 provides three different ways to represent numbers -- REAL, BYTE, or INTEGER -- each of which has its own advantages. The choice depends on the type of procedure being written. When it comes to storing data variables, BASIC09 gives you five standard data types: REAL, BYTE, INTEGER, STRING, and BOOLEAN. BASIC09 also allows you to create new customized data types.

A data structure permits storage for one or more data items under a single name. Data structures are often the most practical and convenient ways to organize large amounts of similar data. The simplest kind of data structure is the array, which is a table of values all of the same data type. Arrays can be one, two, or three dimensional. BASIC09 allows you to create complex data structures that use combinations of data types.

CONSTANTS

Constants are used to assign actual values to variables. There are three types of constants: numeric, string, and Boolean.

NUMERIC CONSTANTS

Numeric constants can be of the data types REAL, BYTE, or INTEGER.

REAL

REAL constants use a decimal point or the scientific floating point "E" notation to represent exponents of 10. If a constant is entered with a decimal point, it is automatically typed (assigned) REAL. Numbers without a decimal point or exponent, but which are too large to be represented as INTEGER constants, are automatically typed REAL. Following are some examples of REAL constants:

1.0	**9.8433218**
-.01	**-999.990099**
100000000	**5655.34532**
1.95E+12	**-89786.7E-33**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Byte

BYTE constants contain INTEGER values with a range from 0 to 255. BYTE constants can also be in hexadecimal form with a range from $00 to $FF. Hex numbers must be preceded by the "$" to indicate to BASIC09 that the value is hexadecimal. Following are some examples of decimal and hexadecimal BYTE constants:

9	**30**
213	**0**
4	**29**
255	**156**

Integer

INTEGER constants can be in the signed value range of -32768 to 32767. INTEGER constants can also be in hexadecimal form with a range from $0000 to $FFFF. Hex numbers must be preceded by the "$" to indicate to BASIC09 that the value is hexadecimal. Following are some examples of decimal and hexadecimal INTEGER constants:

12	**-3000**
-51111	**0**
$6D	**$20**
$BDF0	**$FFFF**

STRING CONSTANTS

A STRING constant is a sequence of characters enclosed in double quotation marks. The null STRING "" represents a string with no characters and is analogous to the numeric zero. Quotation marks used **within** a string must take the form of single quotes. Following are some examples of STRING constants:

"WARNING! REACTOR TEMPERATURE NEARING 'CRITICAL' RANGE"
"ENTER RECIPE SELECTION"
"SHIFT PRODUCTION DATA"
"FLOW RATE = "

BOOLEAN CONSTANTS

A BOOLEAN constant has two possible values -- TRUE or FALSE. BOOLEAN constants cannot be used for numerical computation, but are used for logical flags and expressions.

NOTE
For additional information regarding the data types used by numeric, string, and Boolean constants refer to the *Data Types* subsection.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VARIABLES

Variables are used to represent values in BASIC09. There are two types of variables -- NUMERIC and STRING.

A NUMERIC variable always represents a value that is a number and is used for computation and comparison operations. Variables can be of any length.

A STRING variable represents alphanumeric characters that are not used in computation. STRING variables can be used in comparison operations.

NOTE

BASIC09 reserves certain words for its own use. For additional information and a listing of reserved words, refer to *Reserved Words* in the *Commands and Statements* portion of this section.

WARNING

In BASIC09 variables are NOT initialized to zero but may contain random values. You as the user MUST program any initialization routines.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DECLARING VARIABLES

Variables must be declared in your user's program by using the DIM, PARAM, or LET statements. Variables may also be declared through other BASIC09 routines (such as the result of an arithmetic operation or a FOR/NEXT routine).

DIM

The DIM statement is used to dimension variables according to type: REAL, BYTE, INTEGER, STRING, or BOOLEAN.

NOTE
For additional information regarding data types, refer to the *Data Types* portion of this section.

PARAM

The PARAM statement is used when parameters (variables) are being passed from one procedure to another in modular programming. PARAM allows new variable names to be assigned to the passed values and their data types to be selected.

NOTE
For additional information regarding parameter passing and data types, refer to *Parameter Passing and Data Types* in the *BASIC09 Programming Background* subsection.

LET

The LET statement is used to implicitly assign a numeric or string value to a variable name. The following examples assign the numeric value **230** to a variable named x and the string value **Main Menu** to the variable **m$**.

```
LET x=230
LET m$="Main Menu"
```

The LET statement itself is optional when implicitly defining a variable name and value in this manner. The following two examples perform the same function as the previous examples. The absence of a statement in this case is interpreted as LET.

```
x=230
m$="Main Menu"
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 ROUTINES

Values can also be assigned to variable names as the result of BASIC09 routines. These routines can take many forms. Examples might include: INPUT statements requesting a value from a user, FOR/NEXT loops, or the result of an arithmetic operation. The following three examples illustrate these routines.

INPUT "Enter Recipe Code# ",RC	(**RC** = the numeric value entered by the user).
FOR x=1 TO 10 **PRINT x** **NEXT x**	(the value of **x** will increment form 1 to 10. and print out during each loop).
Val=(Q+6)/4095*1000	(**Val** = the value of Q plus 6, which is then divided by 4095 and multiplied by 1000).

Variables and variable names are local to the procedure in which they are defined. This means that the same variable names can be used independently in two different procedures, even if they are part of a modularized program. It is also possible to allow two or more procedures to share the same variable name when passing parameters (variables) in a modularized program.

VARIABLE STORAGE

Variables are stored in a portion of the BASIC09 workspace, which is allocated when the procedure is called. After the procedure is exited, the variable storage space is returned to the BASIC09 workspace and all variables stored there are lost. A procedure is capable of calling itself through a process called "recursion". This process will allocate a separate variable storage space in each incarnation of the procedure.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PARAMETER PASSING

Variables can be passed as parameters to a called procedure by being included with the RUN statement or command. The called procedure must be prepared to receive these values and will need to include the PARAM statement. If a procedure is called and a PARAM statement is executed, but no parameters were passed, an **Error #56 - Parameter Error** will be generated

Parameter variables can be passed to a called procedure when it is RUN from the B: prompt, BASIC09 mode as shown in Figure 99.

In this particular example, the procedure **calc** is called from the BASIC09 mode. The programmer has included two parameter values (9 and 6). When **calc** is executed, the PARAM statement takes these values and assigns them to the variable names **A** and **B**. They are further typed as REAL. The remainder of the procedure **calc** may use these variables as necessary.

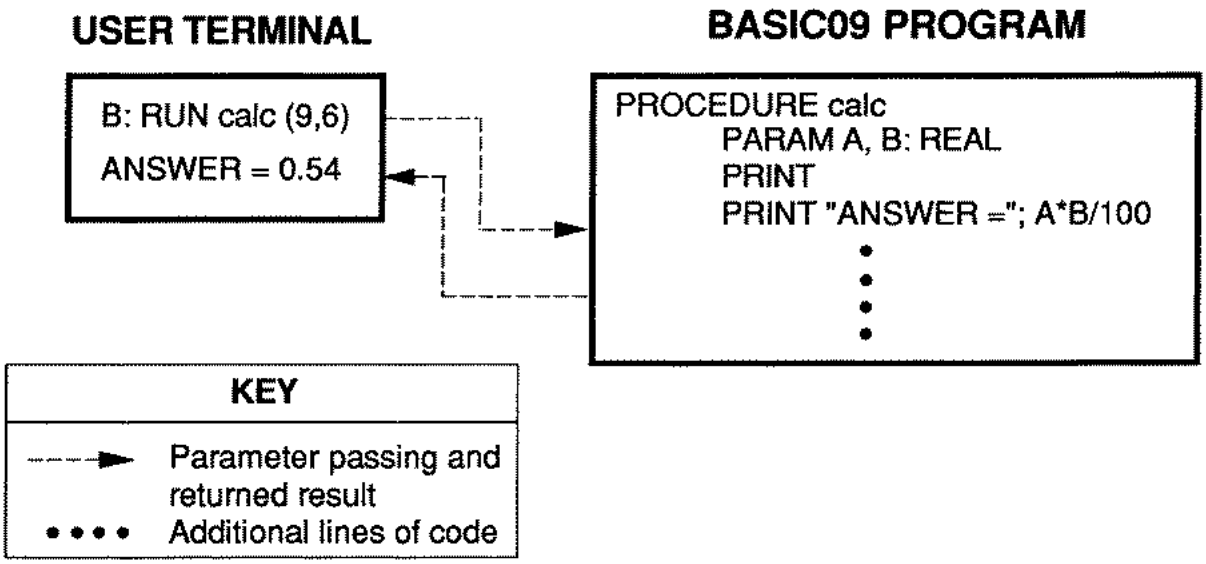


FIGURE 99 - PARAMETER PASSING EXAMPLE #1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

One procedure can also pass parameters to another if the procedure currently being executed uses the RUN statement to call another procedure as shown in Figure 100.

In this example, the procedure **report** is currently running when the RUN statement is executed. The procedure **report** is suspended and **plot** is called for execution. Two parameter variables are also passed to plot (**50** and **V**). The PARAM statement in **plot** assigns these values to **a** and **b**, and also types them as INTEGER. The remainder of the procedure **plot** may use these variables as necessary.

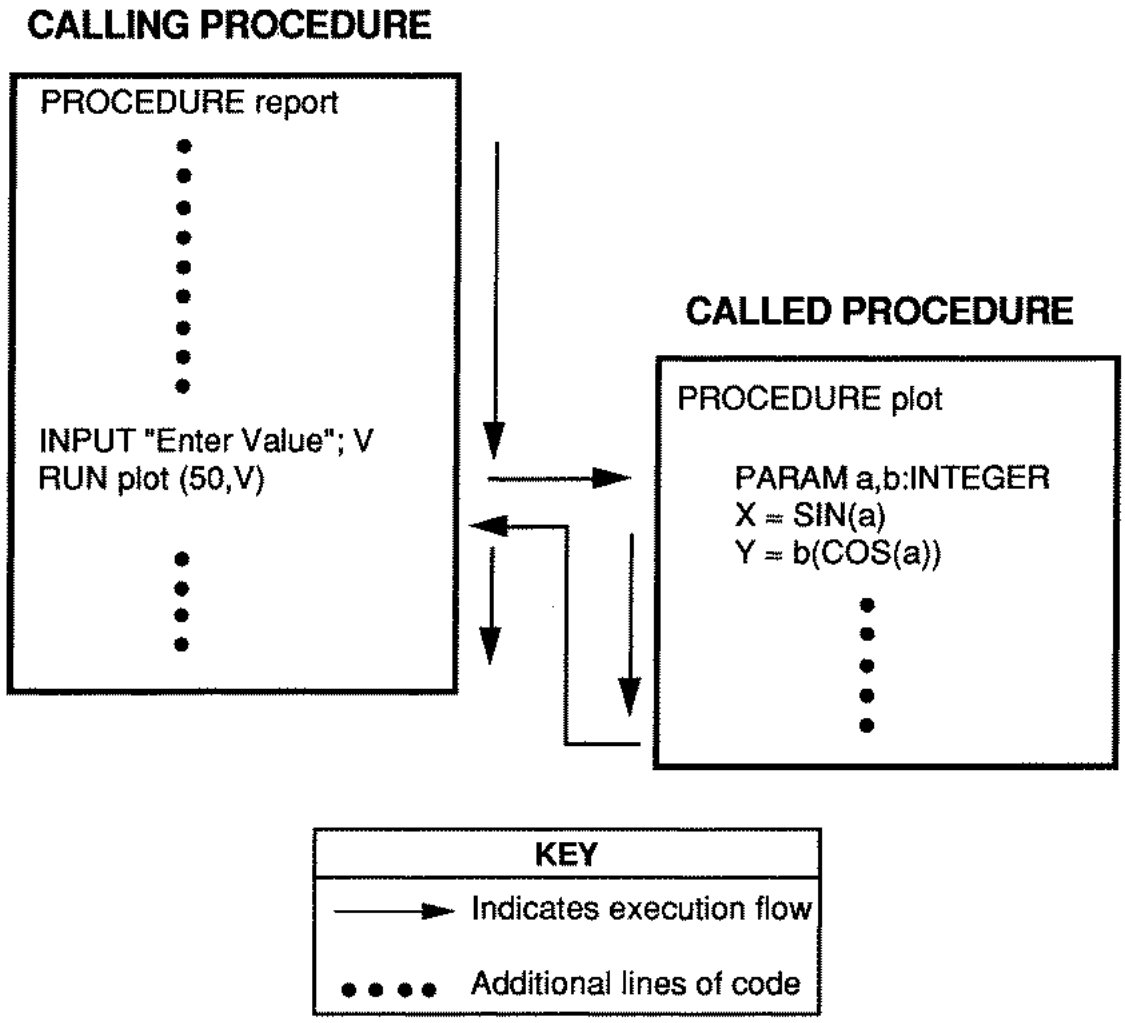


FIGURE 100 - PARAMETER PASSING EXAMPLE #2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Parameters may also be passed by reference or by value.

PASSING PARAMETERS BY REFERENCE

This type of parameter passing allows the called procedure to return values. This is done by enclosing the parameters in parentheses as part of the RUN statement. The parameters are associated, in order, with names in the PARAM statement appearing in the called procedure.

PASSING PARAMETERS BY VALUE

This type of parameter passing creates a temporary value when the expression is evaluated. This is the value passed to the called procedure. Passing by value keeps the value in the calling procedure from being changed by the called procedure. Numeric variables must be followed by a plus zero (+ **0**), and string variables must be followed by a plus null string (+ **""**).

For example:

```
RUN test (X + 0)

RUN xfer (word$ + "")
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DATA TYPES

BASIC09 offers five atomic data types: REAL, BYTE, INTEGER, STRING, and BOOLEAN. REAL, BYTE, and INTEGER are used to represent numeric values. STRING is used to represent character data, and BOOLEAN to represent TRUE and FALSE values. Arrays can be constructed from any data type in one, two, or three dimensions. Table 33 provides an overview of the characteristics of each data type.

Each numeric data type has its own advantages and disadvantages, as follows:

- **REAL** When used with a decimal point, it is the most versatile and has the greatest range of values; it is also the slowest by a factor of four.
- **BYTE** Slightly faster than INTEGER and requires less memory; useful as a building block for complex data structures; has the smallest range of values.
- **INTEGER** Best choice for loop counters and indexing arrays; uses less memory than REAL but more than BYTE; wider range of values than BYTE but smaller than REAL.

CAUTION

If a variable or data structures data type is not declared, BASIC09 will automatically assign the default type of REAL. Additionally, data structures and variables are NOT initialized to zero, but may contain random values. You MUST program any initialization routines.

TABLE 33 - DATA TYPE OVERVIEW

Type	Value Range	Memory Usage
REAL	Floating Point $\pm 1 \times 10^{38}$	Five Bytes
BYTE	Whole Numbers 0 to 255	One Byte
INTEGER	Whole Numbers –32768 to 32767	Two Bytes
STRING	Alphanumerics (letters, numbers, punctuation)	One Byte per Character
BOOLEAN	TRUE or FALSE	One Byte

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

REAL

The REAL data type is the default for any undeclared variables. Variables can also be explicitly declared as REAL. REAL values are printed with a decimal point, and only those constants entered with a decimal point are stored as REAL.

REAL values are stored in five consecutive memory locations (bytes). As shown in Figure 101, the first byte (byte 0) holds the exponent, if any. The next 31 bits (bytes 1, 2, 3 and the three most significant bits of byte 4) hold the mantissa. The last bit (least significant bit of byte 4) holds the sign bit of the mantissa.

The internal range of exponent values ranges from 2^{-128} to 2^{127}. REAL variables can represent values on the number line in increments as small as .0000000005. Operations that result in values outside of the REAL range cause overflow or underflow errors, which can be automatically handled by programming a error trap/handling routine using the ON ERROR command.

Floating point arithmetic is inherently inexact, and a sequence of operations can produce a cumulative error. Proper rounding of results, as done by BASIC09, will reduce this effect but not eliminate it. Use caution when performing strict comparisons (=, <, >) on REAL quantities as the exact value may not occur during program execution.

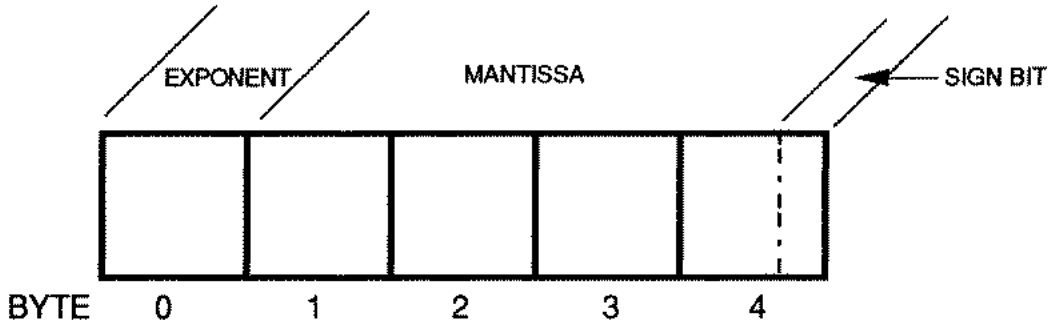


FIGURE 101 - REAL STRING CONTENTS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BYTE

BYTE variables hold integer values ranging from 0 to 255, each stored in a single byte of memory. BYTE variables require only half the storage used by INTEGER values and only a fifth of that used by REAL values.

If an integer greater than 255 is stored in a BYTE variable, only the least significant eight bits will be stored. No error will be generated. Byte constants can also be represented in the range of 0 to 255 ($00 to $FF hex) for logical operations involving the manipulation of individual bits.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INTEGER

INTEGER variables require two bytes of memory and can hold values ranging from -32768 to 32767. Decimal points are not allowed. INTEGER constants can also be represented in the range of 0 to 65535 ($000) to $FFFF hex) for address calculations. INTEGER arithmetic is faster and requires less storage than equivalent REAL operations.

Arithmetic results outside of the INTEGER type's normal range do not cause run-time errors, but instead wrap around from positive to negative and vice versa. For example:

32767 + 1 = -32768

or

-32768 - 1 = 32767

INTEGER division produces quotients only, discarding all remainders. When comparing two or more values where at least one of these values is greater than 32767, the operation identifies that value as negative. It is advisable to limit such comparisons to test for equality. In addition, certain functions (LAND, LNOT, LOR, LXOR) use INTEGER values but produce results in a bit-by-bit manner.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

STRING

A STRING is a variable length sequence of alphanumeric characters. A variable can be identified as a STRING either explicitly using the DIM command or by adding the $ at the end of the variable name.

For example:

DIM title:STRING (the variable name, **title,** is explicitly defined as a STRING)

title$="DATA DISPLAY MENU"** (the **$ identifies the variable name, **title**, as a STRING)

The maximum default length of a string is 32 characters. A string length can be user-defined as larger or smaller than 32 characters with the DIM command. Dimensioning the size of a string only sets the maximum length.

For example:

DIM short:STRING[4] (sets the maximum length at 4 characters)

DIM long:STRING[2880] (sets the maximum length at 2880 characters)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

When alphanumeric characters are assigned to a STRING variable, the string is copied into a storage area byte-by-byte. The first character of the string is stored in byte 0. If the number of characters assigned to a string is greater than the defined (or default) length, those characters exceeding the length are ignored or truncated. No error is generated. If the number of characters assigned to a string is less than the defined (or default) length, BASIC09 writes the hexadecimal value $FF (255 decimal), as a terminator, into the first available byte after the last character. Should the number of characters assigned to a string be equal to or greater than the defined (or default) length, no terminator is used.

For example, assume that the variable, **name**, has been dimensioned as a STRING[6] in a BASIC09 procedure that requests the operator's first name. Figure 102 illustrates the contents of the string's six bytes when three different operators (SAM, ROBERT, MARGARET) "log-in".

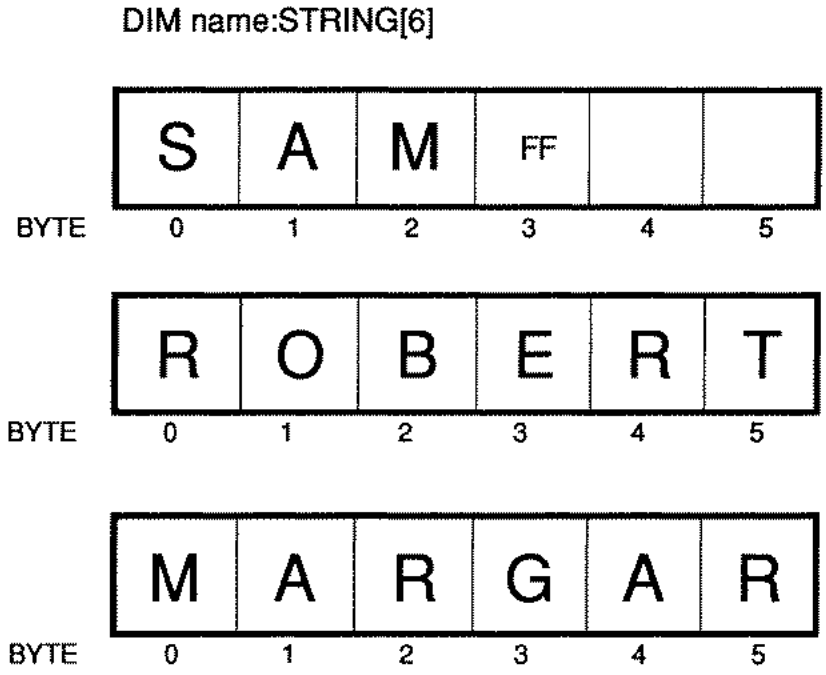


FIGURE 102 - STRING LENGTH ASSIGNMENTS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BOOLEAN

A BOOLEAN variable has only two possible values -- TRUE or FALSE. A variable can be explicitly identified as BOOLEAN using the DIM command. BOOLEAN quantities are stored as single byte values, but they cannot be used for numeric computation. They print out as character strings "TRUE" and "FALSE".

BOOLEAN values can result from relational and logical comparisons and are used as logical flags for expressions.

For example, in the following, RESULT does not equal a number but the value TRUE or FALSE.

```
DIM RESULT:BOOLEAN
RESULT = X AND Y AND Z
```

Once the variable has been dimensioned as BOOLEAN, it cannot accept a non-Boolean value. Attempting to do so will result in a run-time error.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

COMPLEX

Complex data structures allow you to store and manipulate values of many types, to create new types, and to create data structures of unusual shapes or sizes. Access to a complex data structure is somewhat faster than access to an array. The elements of a complex structure can be copied to another structure using the single operator "=". An entire complex data structure can be written to or read from mass storage as a single entity using the PUT command.

The TYPE statement can be used to define a new complex data type that can be a combination of REAL, BYTE, INTEGER, STRING, and BOOLEAN values.

For example:

```
TYPE SHIFTSTRUCTURE=name:STRING[10];id:INTEGER;start:
STRING[17];finish:STRING[17];quantity:REAL;operator:
STRING[15]
```

In this example, the complex data type, SHIFTSTRUCURE, is a one-dimensional array (vector) made up of six fields -- name, id, start, finish, quantity, and operator. The various elements take the form of STRING, INTEGER, and REAL data types. They are accessed by field names, rather than array coordinates. For example, the following creates a new array, **shift(15)** defined as being of the SHIFTSTRUCTURE type and being capable of having 15 records. Each field is then explicitly assigned to shift(1).

```
DIM shift(15):SHIFTSTRUCTURE
shift(1).name="V6 ENGINE BLOCKS"
shift(1).id=1257
shift(1).start="91/07/19/07/00/00"
shift(1).finish="91/07/19/15/30/00"
shift(1).quantity=12
shift(1).operator="J. Smith"
```

Eventually, all record entries will be assigned and the resulting data structure will be complete. The values for each shift element can be stored in COPRAM for later retrieval These elements can be displayed or printed in various report formats. The information can be sorted and searched in a number of ways. The following example searches the data structure for batch quantities greater than 10 and displays the quantity, operator's name, and id number.

```
FOR x=1 TO 15
IF shift(x).quantity>10 THEN PRINT shift(x).quantity,
shift(x).operator, shift(x).id
ENDIF
NEXT x
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

AUTOMATIC TYPE CONVERSIONS

The REAL, BYTE, and INTEGER type expressions, which can mix numeric data types, are converted to the largest type necessary to retain accuracy. Other BASIC09 functions perform type conversions as needed. This means that numeric types can be used together in most cases. This type conversion does take time, and whenever possible it is advisable to use such expressions with like data types.

If a type-mismatch should occur, it will be signaled at the second compiler pass upon exiting the edit mode.

For example:

EQ	(Q quits or exits the edit mode; returns to BASIC09 mode).
ERROR #46	(indicates an operand type mismatch).
B:	(BASIC09 mode prompt).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ARRAYS

An array is a group or table of values defined under a single name. Individual values may be composed of atomic data types, complex data types, or other arrays. To create an array use the DIM statement and define the array's name, the number of elements it will contain, and its data type.

The are two basic types of arrays: vectors and matrices.

VECTOR ARRAYS

Vector arrays are one-dimensional. They are created using a DIM statement with a single numeric value in parentheses after the name. The following example creates a vector array called TESTDATA that can hold 30 REAL values:

```
DIM TESTDATA(30):REAL
```

MATRIX ARRAYS

Matrix arrays can be two- or three-dimensional. They are created using a DIM statement with two or three numeric values, separated by commas, in parentheses after the name. The following example creates a two-dimensional matrix array called PLOT containing 100 rows and 3 columns of INTEGER values.

```
DIM PLOT(100,3):INTEGER
```

The following example creates a three-dimensional matrix array called **WEEKLY PRODUCTION** containing 25 rows, 6 columns, and 5 pages of COMPLEX values.

```
DIM WEEKLYPRODUCTION(25,6,5):BATCHSTRUCTURE
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INDEXING ARRAYS

BASIC09 allows two modes for indexing arrays -- BASE 0 and BASE 1. Indexing a two-dimensional array provides for a two digit reference for each array element; three-dimensional array elements are identified using a three digit reference. In BASE 0 indexing, the first element in a two-dimensional array has an index of 0,0; or 0,0,0 in a three-dimensional array. In BASE 1 indexing, the first element in a two dimensional array has an index of 1,1; or 1,1,1 in a three dimensional array.

Figure 103 illustrates two, three-dimensional arrays that are indexed BASE 0 and BASE 1. The first digit of the array index indicates the element's row; the second digit indicates the column. In three-dimensional arrays the third digit represents the page; a two dimensional array will only have one page, otherwise, they are identical.

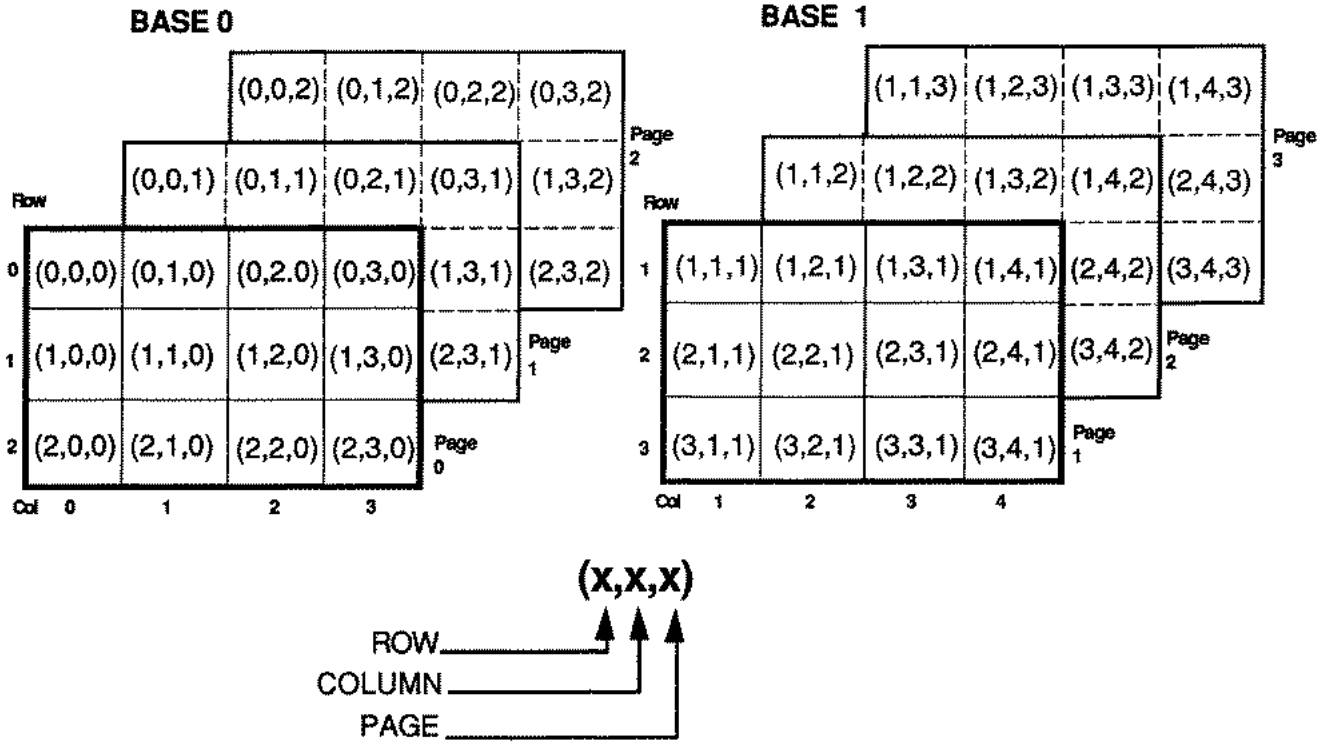


FIGURE 103 - ARRAY INDEXING MODES

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

All arrays created in BASIC09 default to the BASE 1 mode. Array modes are changed with the BASIC09 statement BASE followed by the mode. For example:

BASE 0 (changes or sets the array index mode to 0)

BASE 1 (changes or sets the array index mode to 1)

In the following example BASIC09 procedure, **data$** is defined as a two-dimensional INTEGER array. Since this array's index mode is undefined, the default of BASE 1 is applied.

```
DIM data$(2,3):INTEGER          (defines array).
FOR x=1 to 2                    (the two FOR/NEXT loops cycle through the array
    FOR y=1 to 3                zeroing each element).
        data$(x,y)=0
    NEXT y
NEXT X
data$(2,2)=99                   (defines element 2,2 as 99).
FOR x=1 to 2                    (the two FOR/NEXT loops print out each array
    FOR y=1 to 3                element).
        PRINT data$(x,y);
    NEXT y
    PRINT
NEXT X
```

PRINT OUT EXAMPLE

```
0    0    0
0    99   0
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXPRESSIONS

BASIC09 statements often evaluate expressions. An expression is a combination of constants, variables, functions, and operators used to produce a value. The resultant value can be of any data type (REAL, BYTE, INTEGER, STRING, or BOOLEAN).

OPERATORS

Operators perform mathematical or logical operations on numeric values, and logical operations on string values.

FUNDAMENTAL OPERATORS

There are three categories into which the fundamental operators can be classified -- arithmetic, logical, or relational. Table 34 lists the fundamental BASIC09 operators, their function, operand type, and result type. Operand refers to the data type that the function works on, while result type refers to the data type that the function produces. A NUMERIC result type refers to either the REAL, BYTE, or INTEGER data types.

TABLE 34 - FUNDAMENTAL OPERATORS

Category	Operator	Function	Operand Type	Result type
ARITHMETIC	-	Negation	NUMERIC	NUMERIC
	^ or **	Exponentiation	NUMERIC	NUMERIC
	*	Multiplication	NUMERIC	NUMERIC
	/	Division	NUMERIC	NUMERIC
	+	Addition	NUMERIC	NUMERIC
	-	Subtraction	NUMERIC	NUMERIC
LOGICAL	NOT	Logical negation	BOOLEAN	BOOLEAN
	AND	Logical AND	BOOLEAN	BOOLEAN
	OR	Logical OR	BOOLEAN	BOOLEAN
	XOR	Logical XOR	BOOLEAN	BOOLEAN
RELATIONAL	=	Equal to	ANY	BOOLEAN
	<> or ><	Not equal to	ANY	BOOLEAN
	<	Less than	NUMERIC STRING*	BOOLEAN
	<= or =>	Less than or equal to	NUMERIC STRING*	BOOLEAN
	>	Greater than	NUMERIC STRING*	BOOLEAN
	>= or +>	Greater than or equal to	NUMERIC STRING*	BOOLEAN

*When comparing strings, the following ASCII collating sequence is used:

0<1<2< .....<8<9<A<B<.....<Y<Z <a<b<.....<y<z

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ARITHMETIC OPERATORS

The arithmetic operations performed by BASIC09 are similar to those performed in general mathematics, algebra, or other versions of the **B**eginners **A**ll-Purpose **S**ymbolic **I**nstruction **C**ode (BASIC). The only two operations whose symbols may appear unusual from general mathematics or algebra is multiplication and exponentiation. The following examples illustrate the manner in which multiplication and exponentiation are symbolized in each form:

MULTIPLICATION

General Mathematics	Algebra	BASIC09
2 x 190	2•190	2*190
	AB *or* A•B	A*B
	Z100 *or* Z•100	Z*100

EXPONENTIATION

General Mathematics	Algebra	BASIC09
5×10^{12}	5^{12}	5^12 *or* 5**12
36×10^{3}	36^{3}	36^3 *or* 36**3

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LOGICAL OPERATORS

Logical operators perform logical or Boolean operations on two values. The operands in this type of operation must be data type BOOLEAN (TRUE or FALSE). Logical operators can also be used to make decisions affecting program flow.

A logical operation takes TRUE and FALSE data and returns a TRUE or FALSE result. There are four operations available in BASIC09 -- NOT, AND, OR, and XOR (Exclusive OR). Tables 35 through 38 list the truth tables for each of these operators where T =TRUE and F = FALSE.

TABLE 35 - NOT TRUTH TABLE

NOT	
X	NOT X
T	F
F	T

TABLE 36 - AND TRUTH TABLE

AND		
X	Y	X and Y
T	T	T
T	F	F
F	T	F
F	F	F

TABLE 37 - OR TRUTH TABLE

OR		
X	Y	X or Y
T	T	T
T	F	T
F	T	T
F	F	T

TABLE 38 - XOR TRUTH TABLE

XOR		
X	Y	X and Y
T	T	F
T	F	T
F	T	T
F	F	F

The following examples illustrate typical uses of logical operators:

IF X<25 AND Y>15 THEN 200 (both comparisons must be TRUE for the total expression to be true).

IF A=99 OR B=10 THEN 150 (if either comparison is TRUE, the expression is TRUE).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RELATIONAL OPERATORS

Relational operators compare two values. The values can be numerics or strings. The result is either TRUE or FALSE and is typically used to direct or condition program flow.

NUMERIC

In a numeric comparison any arithmetic operations are always performed prior to the comparison as in the following example.

A+B<(X-4)/Y	(A+B and (X-4)/Y are performed first, then the two results are compared with the Boolean <).

The following two examples illustrate the use of relational operators to control program flow.

IF X<100 THEN 200	(when X<100 = TRUE go to line 200, FALSE continue).
WHILE Q>501 DO **PRINT "NO ERROR"** **ENDWHILE**	(while Q>501 = TRUE print NO ERROR, FALSE exit loop and continue).

STRING

Strings can be compared alphabetically. One string is "less than" another if the first would proceed the second using the following ASCII Collating Sequence:

0<1<.....<8<9<A<B<.....<Y<Z<a<b<.....<y<z

Each character of the two strings is compared. If the two characters are identical, they are equal or TRUE. If the two characters differ, they are FALSE. At the first miscompare (FALSE result), the ASCII collating sequence is used to determine which of the two strings is less than the other.

For example:

"ABC"<"ABc"	(since **C** is less than **c**, this comparison is TRUE).
"1test">"2test"	(since **1** is less than **2**,this comparison is FALSE).
"XYZ"<"XY"	(since **Z** is greater than no character, this comparison is FALSE).
"Start"="Start"	(since all characters are equal, this comparison is TRUE).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OPERATOR PRECEDENCE

Operators are evaluated in a specific order with some taking precedence over others. In BASIC09 parentheses override the natural precedence of operators. The operation in a set of parentheses is solved first . Multiple layers of parentheses can also be used. The inner-most set is solved first, and BASIC09 works outward from that set. Table 39 lists the operator's order of natural precedence from highest to lowest. Operators listed in parallel have no direct natural precedence over any other in its group. BASIC09 will, however, solve these equal precedence operators (except exponentiation) from left to right as they appear in the expression. Exponentiation is solved right to left.

TABLE 39 - OPERATOR ORDER OF PRECEDENCE

Precedence	Operator	Notes
Highest	NOT or -	Negate
	^ or **	Exponentiation
	*,/	Multiplication, Division
	+,-	Addition, Subtraction
	>,<,=<>,<=,+>	
	AND	
Lowest	OR and XOR	

BASIC09 will also attempt to shorten expressions that contain operators by eliminating any unnecessary parentheses and by reordering operators when required. In the following examples, **"Equivalent Form"** refers to the expression as you might create it and **"BASIC09 Form"** refers to the expression after it has been shortened.

EQUIVALENT FORM	BASIC09 FORM
B+((C^2)/D)	B+C^2/D
((l>m) AND (n>o)) OR m=o	l>m AND n>o OR m=o
((W+X)+Y)/Z	(W+X+Y)/Z
(q^(r^s))*t	q^r^s*t

When each line of source code is entered, the compiler will automatically translate the expression into the shortest BASIC09 form possible.

NOTE
Raising a NEGATIVE number to a power CANNOT be performed in BASIC09.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SPECIAL FUNCTION OPERATORS

BASIC09 also offers a number of special function operators. These functions require one or more arguments and perform a specific operation on them. The argument may be an expression, constant, or variable. Special function operators differ from the fundamental operators in that they are actually pre-built routines which perform the same task as several fundamental operators combined. For example, instead of creating an expression out of the fundamental operators to calculate the square root of a number the SQR special function operator can be used. Special function operators can themselves be used as part of a user-defined expression. Table 40 lists the special function operators available in BASIC09.

TABLE 40 - BASIC09 SPECIAL FUNCTION OPERATORS

Operator	Function	Result
SIN(<num>)	Trigonometric sine of <num>.	These functions return REAL results. Accuracy of transcendental functions is 8+ decimal digits. Angles can be either degrees or radians (see DEG/RAD statement descriptions).
COS(<num>	Trigonometric cosine of <num>.	
TAN(<num>)	Trigonometric tangent of <num>.	
ASN(<num>)	Trigonometric arcsine of <num>.	
ASC	Trigonometric arcosine of <num>.	
ATN(<num>)	Trigonometric arctangent of <num>.	
LOG(<num>)	Natural logarithm (base e) of <num>.	
LOG10(<num)	Logarithm (base 10) of <num>.	
EXP(<num>)	e (2.71828)raised to a positive power.	
FLOAT(<num>)	<num> converted to type REAL.	
INT(<num>)	Largest whole number <num>.	
PI	The constant 3.14159265.	
SQR(<num>)	Square root of positive <num>.	
SQRT (<num>)	Same as SQR.	
RND(<num>)	If <num>=0 returns random x, 0<x<1. If <num>>0 returns random x, 0<x<<num>. If <num><0 use absolute value as random number seed.	
ABS(<num>)	Absolute value of <num>.	Functions return ANY numeric type, depending on the type of input parameters.
SGN(<num>)	Signum of <num>.	
SQ (<num>)	<num> squared.	
VAL(<str>)	Convert type string to type numeric.	

(Continued)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 40 - BASIC09 SPECIAL FUNCTIONS OPERATORS *(Continued)*

Operator	Function	Result
FIX(<num>	Round REAL number and convert to INTEGER.	These functions return results of type INTEGER or BYTE.
MOD (<num1>,<num2>)	Modulus (remainder) function.	
ADDR(<num>)	Absolute memory address of variable, array, or structure named <name>.	
SIZE(<name>)	Storage size in bytes of <name>.	
ERR	Error code of most recent error, automatically resets to zero when referenced.	
PEEK(<int>)	Value to byte memory address <int>.	
POS	Current character position of print buffer.	
ASC(<str>)	Numeric value.	
LEN(<str>)	Length of string.	
SUBSTR(<str>,<str>)	Substring search -- returns starting position of first occurrence of <str> in <str2>, 0 if not found.	
LAND(<num>,<num>)	Logical AND.	Functions perform bit-by-bit logical operations that return INTEGER results.
LOR(num>,<num>)	Logical OR.	
LXOR	Logical Exclusive OR.	
LNOT(<num>)	Logical NOT.	
CHR$(<int>)	ASCII character equivalent of <int>.	These functions return STRING values.
DATES$	Date and time: format yy/mm/dd hh :mm:ss.	
LEFT$(<str>, <int>)	Leftmost <int> character of <str>.	
RIGHT$(<str>,<int.>)	Rightmost <int> character of <str>.	
MID$(<str>,<int>,int2>)	Middle <int,2> characters of <str> starting at character position <intg 1>.	
STR$(<num>)	Converts numeric type <num> to displayable characters of type STRING representing the number converted.	
TRIM$(<str>)	<str > with trailing spaces removed.	
TRUE	Always returns TRUE.	These functions return BOOLEAN values.
FALSE	Always returns FALSE.	
EOF(#<num>)	End-of-file test on COPRAM filepath <num>; returns TRUE if end-of-file condition.	

<num> -- a numeric = result expression.
<str> -- a string = result expression.
<int> -- an integr-result expression.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CONCATENATION

Joining two strings together with the plus (+) sign is called concatenation. This allows multiple strings to be printed together and in different orders. The two following procedures **concat1** and **concat2** are identical in every way except in the order of string concatenation in the PRINT statement. Assume that the user's name is entered as John Q. Public.

concat1

```
INPUT "ENTER FIRST NAME";F$
INPUT "ENTER MIDDLE INITIAL";M$
INPUT "ENTER LAST NAME";L$
PRINT F$+M$+L$
```

concat2

```
INPUT "ENTER FIRST NAME";F$
INPUT "ENTER MIDDLE INITIAL";M$
INPUT "ENTER LAST NAME";L$
PRINT L$+F$+M$
```

EXAMPLE PRINT OUT

John Q. Public

Public John Q.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INPUT/OUTPUT OPERATIONS

Any BASIC09 statement can be used with any I/O device. Hardware I/O devices used by OS-9 work like files and the I/O operations are performed in a unified manner. This means that all I/O facilities (COPRAM and the ASYNC Ports) use the same statements and the same interface standard regardless of the type of device involved.

The unified I/O approach, used by OS-9, benefits programming -- file operations can be debugged by communicating with a terminal or printer instead of a storage device. Also, procedures that normally communicate with a terminal can be tested by exchanging data with a storage device. BASIC09 works with two types of files -- sequential and random-access.

NOTE

For additional information regarding the operation and characteristics of unified I/O system and related material, refer to *OS-9 Input/Output Operations* in the *OS-9 Operating System* section.

I/O PATHS

A path is a channel through which data flows between a procedure and an I/O device. In order for data to be exchanged with a device, an applicable device driver must be resident in OS-9. When a path is created, OS-9 returns a number to identify the path in subsequent file operation. This path number is used by the I/O statements to specify the file to be used. Three path numbers identify standard I/O paths between BASIC09 procedures and your terminal. These paths are automatically opened and should not be closed. The standard I/O paths are:

0 - Standard Input Path (keyboard)
1 - Standard Output Path (monitor)
2 - Standard Error/Status (monitor)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SEQUENTIAL FILE OPERATIONS

A sequential file sends and receives (READ/WRITE) text and data in a sequential order. Once transmission begins from an external device it cannot be controlled. A printer is an example of an I/O device that uses sequential files.

To transfer data between a file or device and the BASIC09 procedure, a channel must be opened using the CREATE or OPEN statement. READ and WRITE operations will be performed on this open channel. When the transfer is complete, the channel must be closed via the CLOSE statement.

A sequential file contains only valid ASCII characters. The READ and WRITE commands perform format conversions similar to that done automatically by the INPUT and PRINT commands. Each WRITE command sends a complete sequential file record, which is an arbitrary number of characters followed by a [RETURN] character. A READ command reads all characters up to the next [RETURN].

The following two BASIC09 procedures provide examples of the WRITE and READ operations:

PROCEDURE write

```
DIM PATH:BYTE                (path must be dimensioned as BYTE).
DIM message:STRING[80]       (message is dimensioned as STRING).
OPEN #PATH,"text.store"      (opens a channel to text.store).
PRINT "Enter Text"
INPUT message                (test string is entered by the user).
WRITE #PATH, message         (writes text as ASCII into a text.store).
CLOSE #PATH                  (closes channel to text.store).
```

An alternative to the OPEN statement:

```
CREATE #PATH "text.store"    (if text.store exists a channel is opened to it, if it does
                              not exist it created first).
```

PROCEDURE read

```
DIM PATH:BYTE                (path must be dimensioned as BYTE).
DIM message:STRING[80]       (message is dimensioned as STRING).
OPEN #PATH,"text.store"      (opens a channel to text.store).
READ #PATH, message          (reads the contents of character string message).
PRINT message                (prints the contents of character string message).
CLOSE #PATH                  (closes channel to text.store).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RANDOM ACCESS FILE OPERATIONS

A random access file sends and receives (PUT/GET) data in binary form. It is possible to PUT and GET individual bytes, or a data structure of many bytes. The SEEK command is used to move to a particular byte in a random access file. A new file is created on a storage device by means of the CREATE command. Once a file exists, the OPEN command can establish a channel to the desired I/O device. The channel number is used in file access operations (READ, WRITE, GET, PUT, SEEK, etc.). File access is terminated by the CLOSE command.

The following two BASIC09 procedures provide examples of the PUT and GET operations:

PROCEDURE put

```
DIM PATH:BYTE                (path must be dimensioned as BYTE).
DIM record(15):STRING[10]    (record is dimensioned as STRING).
OPEN #PATH,"storage"         (opens a channel to storage).
FOR x=1 TO 15
   INPUT record(x)
NEXT x
PUT #PATH, record            (puts array record into COPRAM file storage).
CLOSE #PATH                  (closes channel to storage).
```

An alternative to the OPEN statement:

```
CREATE #PATH "storage"       (if storage exists a channel is opened to it; if it does not
                              exist, it is created first).
```

PROCEDURE get

```
PARAM a:INTEGER              (expects a parameter to be passed to it).
DIM PATH:BYTE                (path must be dimensioned as BYTE).
DIM record(15):STRING[10]    (record is dimensioned as STRING).
OPEN #PATH,"storage"         (opens a channel to storage).
GET #PATH, record            (gets the 15 element array record).
PRINT record                 (prints the array element corresponding to a).
CLOSE #PATH                  (closes channel to storage).
```

The following command line entered at the B: prompt calls and executes the previous procedure **get** while passing the parameter 5. 5 is assigned to variable a, which is later used to retrieve element 5 from the 15 element array record.

```
B:RUN get(5)
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WORKING IN THE BASIC09 MODE

The BASIC09 mode of operation is the first mode encountered when BASIC09 is entered at the OS9: prompt. From the BASIC09 mode you can enter commands that interact with OS-9, manipulate the procedures within the workspace, or change to either the edit or execution modes. It is also possible to exit BASIC09 and return to OS-9.

All procedures are stored in a compiled form of their source code called intermediate code or I-code. I-code requires approximately 30% less memory than the true source code form. In I-code, BASIC09 statements are condensed into single character representation. For example, the I-code single character ? (question mark) represents (or is equivalent to) the source code PRINT statement. I-code is converted back to source code when listing to a monitor. Compiling and decompiling are automatic and transparent to you as the user. Syntax errors are detected as each line is entered (compiled).

BASIC09 can create procedures with or without line numbers. Normally line numbers are used only when GOSUB or GOTO statements are required. When using line numbers for all lines, each can be entered in any order. BASIC09 will automatically sequence the line in numerical order upon leaving the edit mode.

The E or EDIT command is entered from the BASIC09 mode, allowing you to create a new procedure or edit an old one that already exists in the workspace. This also changes BASIC09's current mode of operation to the edit mode, which is identified by the E: prompt.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CREATING NEW BASIC09 PROCEDURES

The E or EDIT command is entered from the BASIC09 mode allowing you to create a new procedure in the current workspace. This also changes BASIC09's current mode of operation to the edit mode. Use either of the following syntax examples to create a new procedure:

B:E <proc name>

B:EDIT <proc name>

The parameter, <proc name>, represents the name of the procedure you are about to create. If no procedure name is specified, the default name PROGRAM is assigned to the procedure by BASIC09. Once the procedure has been named, the edit mode is used to develop its contents.

LISTING BASIC09 PROCEDURES

A listing of the names of all procedures in the current workspace can be displayed to the monitor from the BASIC09 mode by entering the DIR command or pressing [RETURN] at the B: prompt.

The LIST command can be used to display the contents of BASIC09 procedures. When the LIST command is entered at the B: prompt, the contents of the specified procedure name will be displayed to the monitor. If no name is specified, the current procedure will be displayed. Multiple procedures can be displayed by including each name after the LIST command. The procedure names should be separated by a [SPACE] character. By including an asterisk (*) after the command, all procedures in the current workspace are displayed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WORKING IN THE EDIT MODE

The edit mode is the one mode in which all BASIC09 procedures are developed and edited. It is entered from the BASIC09 mode by means of the E or EDIT command. The edit mode is identified by the E: prompt. The edit mode can only be entered from the BASIC09 mode.

EDITING EXISTING BASIC09 PROCEDURES

The edit mode is entered from the BASIC09 mode using the E or EDIT command. Use either of the following syntax examples to edit an existing procedure:

B:E <proc name>

B:EDIT <proc name>

The parameter, <proc name>, represents the name of the procedure you wish to edit. If no procedure name is specified, the current (last procedure worked on) is used. Once in the edit mode you can enter new lines of code anywhere in the procedure and change existing lines by moving the edit pointer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MOVING THE EDIT POINTER

The edit pointer is used to edit lines of code. It keeps track of which program line is currently being edited. The current working line is preceded by an asterisk (*) when the procedure is listed. As line or string editing commands are given, the edit mode will perform their operations beginning with the line to which the edit pointer indicates.

The plus (+) and minus (-) keys are used to move the pointer. The plus key will increment the pointer by one line while the minus key decrements it through the lines of code. By entering a number following the plus or minus key the pointer is directed to move that number of lines from its current position. If the asterisk is entered following the plus or minus key, the pointer will move to the last or first line in the procedure respectively. The pointer will also increment by one line each time the [RETURN] key is pressed. If line numbers have been included in the current procedure, an additional option becomes available for moving the edit pointer. Entering the line number of an existing line of code will move the pointer to that location. The following examples illustrate the entries, when made at the E: prompt, that will move the edit pointer.

E:+	(increments the pointer by one line).
E:+5	(increments the pointer by five lines).
E:+*	(increments the pointer to the last line).
E:-	(decrements the pointer by one line).
E:-3	(decrements the pointer by three lines).
E:-*	(decrements the pointer to the first line).
E:[RETURN]	(increments the pointer by one line).
E:20	(moves the pointer directly to line 20).

CAUTION

Do NOT enter any characters other than the line number when using this method of moving the edit pointer. Doing so will delete the existing line and replace it with those characters, if they are recognized as a valid BASIC09 statement or command. If not, an error will be generated.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LISTING BASIC09 PROCEDURES

The L command is used from the edit mode to display the contents of BASIC09 procedures. The L command followed by a number will display the specified number of lines from the current edit pointer position. When the command L* is entered at the E: prompt, the entire contents of the current procedure name will be displayed to the monitor. The following examples illustrate the entries, when made at the E: prompt, that will list all or part of the current procedure.

E:L*	(lists the entire contents of the current procedure).
E:L7	(lists seven lines beginning with the current edit pointer position).

Figure 104 shows an example listing of a BASIC09 procedure. Its constituent parts are also identified.

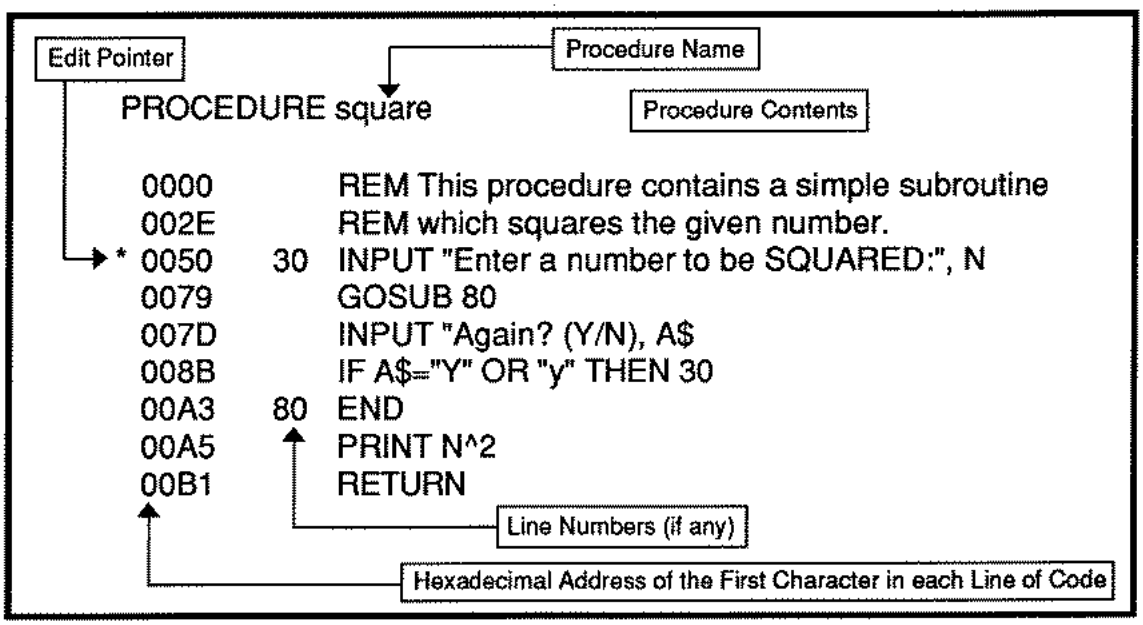


FIGURE 104 - BASIC09 PROCEDURE LISTING

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LINE EDITING COMMANDS

Line oriented editing commands allow you to change, insert, or delete lines of BASIC09 code. A line number editing command is also available.

The d command will delete the current working line, which is identified by the edit pointer. By including a number following the d command, that number of lines will be deleted, again beginning with the current working line. The command d* will delete all lines in the current procedure.

To insert a line, that does not use a line number, move the edit pointer to the line ahead of which the new line is to be placed. Next enter the line as required, it will automatically be inserted ahead of the current working line. When inserting a line that does use a line number, it will be automatically placed immediately prior to the next higher line number. If no line numbers or no higher line numbers currently exist in the procedure, the line will be inserted ahead of the current working line.

The r command is used to renumber existing line numbers. Two numeric parameters are typically associated with the renumber command -- starting line number and increment factor. When the renumber command is issued the operation will begin at edit pointer. The first line number beginning at the pointer will be changed to the value specified in the starting line number parameter. The starting value will be increased by the increment factor and the next line number found will be changed. This continues until the last line number has been changed. The renumber operation will NOT wrap around to the beginning of the procedure. If an asterisk (*) is included following the r command and prior to the start and increment parameters, all line numbers in the procedure will be changed regardless of the edit pointer position. The r* command with no start or end parameters will default to a starting line number of 100 and an increment factor of 10. It is also possible to specify the start line number parameter and not the increment factor, or vice versa. In both cases, the unspecified parameter will utilize its default value. The following examples illustrate the entries, when made at the E: prompt, that will renumber all or part of the current procedure.

E:r 110,5*	(from the pointer, renumbers all line numbers in the current procedure starting with 110 and incrementing by 5).
E:* 10, 10	(from the beginning, renumbers all line numbers in the current procedure starting with 10 and incrementing by 10).
E:*	(from the beginning, renumbers all line numbers in the current procedure starting with the default 100 and incrementing by 10).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

STRING EDITING COMMANDS

String-oriented editing commands allow you to search for and/or change all or a portion of a line of code. A string is a sequence of one or more characters that can include letters, numbers, and punctuation. String editing commands allow a string of characters within a line of code to be identified and changed without the need to reenter the entire line. This makes procedure editing both faster and easier.

The s command locates the occurrence of the specified string from the current edit pointer position. When found, the entire line in which the string appears will be displayed. The edit pointer will also increment to this line. The string to be searched for must be preceded by a delimiter (space, comma, semicolon, colon, slash, etc.). While the slash is most commonly used, the only restriction is that the delimiter does not also appear within the string.

For example:

E:s/pb.14 (searches for character string "**pb.14**").

E:s;test/1 (searches for character string "**test/1**").

By including the asterisk with the search command (s*), all occurrences of the specified string will be located and displayed. The pointer will move to the last line containing the string. In both cases if the specified string cannot be located, the message **CAN'T FIND "(string)"** is displayed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The c command is used to substitute one string for another. String substitution eliminates the need to retype the entire line. The c command locates, removes, and replaces the string as specified. The string to be searched for is specified first, followed by the replacement string. The command and each string must be separated by a delimiter. Without additional parameters, the c command will only search for the string in the current working line.

For example, assume that the string FLOW is to be changed to FLOWDATA:

***02A6**	**DIM FLOW:INTEGER**	(original line).
	E:c/FLOW/FLOWDATA	(searches and exchanges FLOW for FLOWDATA).
***02A6**	**DIM FLOWDATA:INTEGER**	(new line).

CAUTION

If the string is not found in the line at the current edit pointer position, the change command will continue to search each successive line until a match is found or the end of the procedure is reached.

By including the asterisk with the change command (c*) all occurrences of the specified string from the current pointer position will be located and displayed. The pointer will move to the last line containing the string. In both cases if the specified string cannot be located, the message **CAN'T FIND "(string)"** is displayed.

CAUTION

A string of characters can be inadvertently changed even if it appears in a larger string. For example, changing the string "NO" to "YES" can also change the string "NORMAL" to "YESRMAL". This is most likely to occur and go unnoticed when using the asterisk (*) parameter to search for and change all occurrences. It is advisable to include characters that surround the string to be changed to make it unique.

In the following example, the string **"NUM"** is to be changed to **"VAL"**. To perform this operation without also affecting the same characters in the string **"NUMBER"**, you can do the following:

***043B**	**PRINT "NUMBER = ";NUM**	(original line)
	E:c/;NUM/;VAL	(searches and exchanges; NUM for ;VAL)
***043B**	**PRINT "NUMBER = ";VAL**	(newline)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WORKING IN THE EXECUTION MODE

The execution mode displays no prompt of its own and is therefore transparent to you as the user except for the generation of any output that might be directed to a monitor, printer, etc. When the execution mode is active, the MiniCOP is running a BASIC09 procedure or group of procedures. Unless prompted for data input, you can only enter keyboard interrupt commands from the Terminal Emulator. Any other commands need to be programmed into the procedure.

RUNNING BASIC09 PROCEDURES

The execution mode can be entered using the RUN command or the auto-run feature. The RUN command and a procedure name is entered at the B: prompt (BASIC09 mode). BASIC09 immediately enters the execution mode and carries out the tasks specified in the procedure. If no procedure is named, BASIC09 will execute the current working procedure. The RUN command can also be used as a statement from within one procedure to call and execute another.

When a procedure is named for execution, BASIC09 searches memory in the following order:

1. The current BASIC09 workspace.
2. The remainder of main memory (other workspaces, module directory).
3. COPRAM's current execution directory.
4. COPRAM's current data directory.

If multiple versions of the same procedure name exist, the first encountered with this order of priority will be executed. When the procedure is located outside the current workspace it will automatically be loaded (copied into the current workspace) and executed. Once in the execution mode, BASIC09 remains there until one of the following events occurs:

1. An END or STOP statement is executed.*
2. A [CONTROL - E] keyboard interrupt is entered.*
3. A run-time error occurs. **
4. A [CONTROL - C] keyboard interrupt is entered.**
5. A PAUSE statement is executed.**
6. A BYE statement is executed.***

---

* These events return the MiniCOP to the BASIC09 mode.

** These events send the MiniCOP to the debug mode.

*** Aborts BASIC09 and the MiniCOP to OS-9.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Parameters can be passed to the called procedure. These parameters can be defined with the RUN command and procedure name at the B: prompt. They can also be defined when the RUN statement is used by one procedure to call another. This feature allows you to initialize procedure variables.

NOTE

For additional detailed information relating to the parameter passing, refer to the *Parameter Passing* portion of this section.

In the auto-run method, a procedure's name is included as a parameter when BASIC09 is called from the OS9: prompt. Once BASIC09 is loaded, it immediately calls and executes the named procedure or procedures.

A third alternative to entering the execution mode relates to a MiniCOP that has been set-up to operate in RUNB. RUNB is similar to BASIC09 except that all nonexecution essential functions, such as editing and debugging have been stripped away. RUNB can only execute packed BASIC09 procedures.

NOTE

For additional detailed information relating to the RUNB operation, refer to *Appendix E*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WORKING IN THE DEBUG MODE

The debug mode is entered from the execution mode as the result of a run-time error, a [CONTROL - C] keyboard interrupt, or a user-programmed PAUSE statement. The debug mode is identified by the D: prompt. The debug mode offers several commands that can be used to locate the source of problems.

When a procedure fails, use the PRINT command to display procedure variables and their values. Also use the LIST command to display a listing of the complete procedure. Studying these two displays will assist you in uncovering the more obvious errors. Check to see that variables are properly initialized and that loop counters have incremented.

If this check does not uncover the source of all errors, the next step is to temporarily place a PAUSE statement in the procedure where a problem is suspected. Then try executing it again.

The [CONTROL-C] keyboard interrupt will cause BASIC09 to exit the execution mode and enter DEBUG, which is identified by the D: prompt. If the procedure being executed is short or near its end, it may terminate prior to the interrupt's execution.

The PAUSE statement, when encountered, will cause the program to suspend execution and places BASIC09 in the debug mode. From this point the trace mode can be turned on with the TRON command. The TROFF command turns the trace mode off.

The purpose of the trace mode is to execute the program one line at a time so that each line's result can be monitored in "slow motion". As the [RETURN] key is pressed the next line is displayed, any expressions are evaluated, and their current values are displayed.

In some cases, a loop may take several passes before the task is complete. The STEP command combined with the number of steps can be used in these instances. The command STEP 50 will have the same effect as pressing the [RETURN] key fifty times. The procedure can be stepped through with the trace on or off.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The debug mode is local to one procedure only. If the procedure under investigation calls another procedure, debug will not be active until execution returns to the parent (calling) procedure. In order to check line-by-line execution in the called procedure, the BREAK command or a PAUSE statement can be inserted into that procedure.

The BREAK command can also be used to enter the debug mode while a procedure is executing. To use the BREAK command you must initiate with the [CONTROL-C] keyboard interrupt. Then, BREAK <procedure name> is entered at the D: prompt. If the procedure named is found to be in the current RUN stack, BASIC09 will respond with the message **ok**. The BREAK command sets a breakpoint in the named procedure where execution was interrupted by the [CONTROL-C] keyboard interrupt. The CONT command will resume normal execution of the procedure. When the breakpoint is next encountered during normal execution, the debug mode is automatically entered.

Nested procedures (where one procedure calls another, which calls another, which calls another, etc.) can be quite confusing and difficult to debug. One aid available is the STATE command, which lists the current nesting order of all **active** procedures. The procedure currently being executed is listed first, with each calling procedure listed in order.

To exit the debug mode and return to the BASIC09 mode, use the q command at the D: prompt.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MANIPULATING BASIC09 PROCEDURES

Once a procedure has been developed and debugged, it can be copied from the workspace to COPRAM or permanent storage (disk) at the Terminal Emulator. Procedures can be saved in both a source code or packed format.

SAVING SOURCE CODE PROCEDURES TO COPRAM

Unless otherwise directed, the SAVE command will save the specified source code procedure to the current data directory in COPRAM. To save a procedure to the current data directory no path name is needed. If it is necessary to save the procedure to a directory other than the current data directory, simply include its full path list. By including an asterisk (*) with the SAVE command, all procedures in the workspace can be packed and saved under a single specified name.

For example:

B:SAVE initialize	(saves the source code procedure, **initialize**, to the current data directory).
B:SAVE* initialize	(saves all source code procedure in the workspace under the name, **initialize**).
B:SAVE /D0/TEMP/initialize	(saves the source code procedure, **initialize**, to the **TEMP** directory).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SAVING SOURCE CODE PROCEDURES TO THE TERMINAL EMULATOR

Source code procedures can also be saved to a permanent storage medium (such as floppy or hard disk) in the microcomputer that is running the Terminal Emulator software. This procedure is slightly more involved than saving to COPRAM and will require that the Terminal Emulator's receive function be prepared for the file transfer.

For example:

B:SAVE initialize >/TP	(sends the source code procedure, **initialize**, to the Terminal Emulator via the MiniCOP's tape port, **/TP**; > is a redirection modifier).

Note that the [RETURN] key is not pressed after the SAVE command is entered at the B: prompt. The following steps should actually be issued next from the Terminal Emulator:

1. Press [F4] Rec to display the receive file from tape port selection.

2. Change drive and/or path name to the desired destination, including the name of the file being received.

3. Press [RETURN] twice to initiate the transfer.

4. After transfer is complete, use the [F7] Dir and/or [F5] selection to verify that the file (procedure) has properly transferred.

NOTE

For additional detailed information regarding the use and operation of the Terminal Emulator, refer to the *Terminal Emulator* section and the *623 Terminal Emulator User Manual* .

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PACKING SOURCE CODE PROCEDURES TO COPRAM

Once a procedure has been saved it can be packed. When the PACK command is executed, an additional compiler pass is made on the source code (I-code) procedure or procedures. All names, line numbers, nonexecutable statements, etc. are removed from the procedure. The result is a smaller, faster procedure.

Unsaved source code procedures in the workspace will be lost when packed. Packed procedures cannot be edited or debugged, but can be executed by BASIC09. PACKing destroys the source code form of the procedure in the workspace. Unless otherwise directed, procedures are automatically saved to the current execution directory when packed. By including an asterisk (*) with the PACK command, all procedures in the workspace can be packed and saved under a single specified name.

For example:

B:PACK initialize	(packs and saves the source code procedure, **initialize**, to the current execution directory).
B:PACK* initialize	(packs and saves all source code procedures in the workspace under the name, **initialize**).
B:PACK /D0/PACKDIR/initialize	(packs and saves the source code procedure, **initialize**, to the PACKDIR directory).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PACKING SOURCE CODE PROCEDURES TO THE TERMINAL EMULATOR

Source code procedures can also be packed and saved to a permanent storage medium (such as floppy or hard disk) in the microcomputer that is running the Terminal Emulator software. This procedure is slightly more involved than packing to COPRAM and will require that the Terminal Emulator's receive function be prepared for the file transfer.

For example:

B:PACK initialize >/TP	(packs and sends the source code procedure, **initialize**, to the Terminal Emulator via the MiniCOP's tape port, **/TP**; > is a redirection modifier).

The [RETURN] key would not be pressed after the PACK command is entered at the B: prompt. Next the following steps would be issued from the Terminal Emulator:

1. Press [F4] Rec to display the receive file from the tape port selection.
2. Change drive and/or path name to the desired destination, including the name of the file being received.
3. Press [RETURN] twice to initiate the transfer.
4. After the transfer is complete, use the [F7] Dir and/or [F5] selection to verify that the file (procedure) has properly transferred.

NOTE

For additional information regarding the use and operation of the Terminal Emulator, refer to the *Terminal Emulator* section and the *623 Terminal Emulator User Manual*.

CAUTION

A packed file cannot be directly loaded into the BASIC09 workspace, and it cannot be edited or debugged. Always save the source code form of the procedure prior to packing. Packing will destroy the source procedure in the workspace.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LOADING SOURCE CODE PROCEDURES FROM COPRAM

Source procedures that do not currently reside in the workspace can be transferred from either COPRAM or permanent storage at the Terminal Emulator with the LOAD command. The names of all procedures specified are displayed as they are loaded. Procedures can be loaded from the current data directory by simply specifying their name in the LOAD command, or from other directories by including the proper pathname. When loading files from the Terminal Emulator, the file redirection modifier and port driver (/TP or /T2) must also be included.

For example:

B:LOAD initialize	(loads the source code procedure, **initialize**, from the current execution directory).
B:LOAD /D0/TEMP/initialize	(loads the source code procedure, **initialize**, from the **TEMP** directory).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LOADING SOURCE CODE PROCEDURES FROM THE TERMINAL EMULATOR

Source procedures can also be loaded from a permanent storage medium (such as floppy or hard disk) from the microcomputer that is running the Terminal Emulator software. This procedure is slightly more involved than loading fro COPRAM and will require that the Terminal Emulator's send function be prepared for the file transfer.

For example:

B:LOAD >/TP initialize	(loads the source code procedure, **initialize**, from the Terminal Emulator via the MiniCOP's tape port, **/TP**; > is a redirection modifier).

The file name does not need to be specified on the command line when loading from the Terminal Emulator, the next file read at the tape port will be used.

When loading from the Terminal Emulator, the [RETURN] key would not be pressed after the LOAD command is entered at the B: prompt. Next the following steps would be issued from the Terminal Emulator:

1. Press [F4] Rec to display the receive file from tape port selection.

2. Change the drive and/or path name to the desired destination, including the name of the file being received.

3. Press [RETURN] twice to initiate the transfer.

4. After the transfer is complete, use the [F7] Dir and/or [F5] selections to verify that the file (procedure) has properly transferred.

NOTE

For additional detailed information regarding the use and operation of the Terminal Emulator, refer to either the *Terminal Emulator* section or the *623 Terminal Emulator User Manual*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LISTING SOURCE CODE PROCEDURES TO A PRINTER VIA THE TERMINAL EMULATOR

Source procedures in the BASIC09 workspace can be listed to a serial printer, which can be connected to one of the MiniCOP's ASYNC ports. Typically, port 3 is chosen since during initialization and programming it is unused by the Terminal Emulator. Assuming the printer is connected to port 3 and port 3 is configured to the printer's transmission characteristics, the following example might be used to LIST a procedure:

B:LIST initialize >/T3	(provides a line-by-line listing of the source code procedure, **initialize**, from the workspace)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PROGRAMMING TIPS

TIPS TO REDUCE MEMORY USAGE

One characteristic of a good program is an efficient use of memory. There are ways to minimize memory usage without altering the effectiveness of the program. Some of these methods involve reduced documentation and should only be used when space is severely limited.

1. **USE THE PACK COMMAND.** Once a procedure has been edited, debugged, and saved, the PACK command will produce a compressed, more memory-efficient form. Line number and nonexecutable statements are automatically removed from a packed procedure.

2. **USE INTEGER, NOT REAL, ARRAYS.** INTEGER arrays require 60% less memory per element than REAL (2 bytes/elements vs. 5 bytes/element) and should be used only when the REAL data types are not critical to the application.

3. **USE VARIABLES INSTEAD OF CONSTANTS.** Each constant requires memory when it is defined in a BASIC09 procedure. Variables need only to be defined once and can change to represent other values. Variables are stored in a portion of the workspace which is automatically reserved for their use.

4. **ELIMINATE END STATEMENTS.** END statements are not required by BASIC09 procedures and generally only serve to use up memory.

5. **REUSE VARIABLE NAMES.** If a variable is used to store a temporary value (as in FOR/NEXT) loops at the beginning of the procedure, it can be used again later.

6. **REDUCE ARITHMETIC FUNCTIONS.** Expressions consisting of only constants should be reduced to their simplest form. For example, the expression...

   X=X+SQRT(100)/2

   ...is the same as...

   X=X+5

7. **USE SUBROUTINES.** Subroutines are ideal for performing identical, repetitive tasks in a procedure.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TIPS TO INCREASE PROCEDURE EXECUTION

There are some easy ways to decrease the execution time of a procedure without changing its function. One possible method is to reduce memory usage (which is always a good idea). Another is to structure the program statements so that they are executed most efficiently.

1. **USE VARIABLES INSTEAD OF CONSTANTS.** This tip alone can increase the execution speed of expression by a factor of 10. It requires much more time to convert an INTEGER constant into its REAL representation than it does to retrieve the value of a REAL variable from the temporary storage area.

2. **ASSIGN VARIABLES AT THE BEGINNING OF A PROCEDURE.** A statement such as:

   A$=CHR$(27)

   should be executed early in the procedure. This is especially true in procedures that constantly repeat or cycle to a point near the beginning of the program, where it is not necessary for all or most variables to be reassigned.

3. **USE INTEGER, INSTEAD OF REAL, VALUES IN EXPRESSIONS.** The first of the following expressions, for example, requires about ten times more time to execute as opposed to the second. Both perform the same function.

   VAL=VAL*2.0

   VAL=VAL*2

   It should be noted that the INTEGER division discards any remainders. Table 41 lists the relative speed of arithmetic functions for both REAL and INTEGER values.

TABLE 41 - EXECUTION SPEED OF INTEGER AND REAL ARITHMETIC FUNCTIONS

Operation	Integer	Real
Addition	150	440
Subtraction	440	540
Multiplication	240	990
Division	960	3870
Square Root		7360
Logarithm		20400
Eponentiation		20400
Sine		32500
Cosine		32500
Power (^)		39200

NOTE:			
	X+X	faster than	x*2
	X*X or SQR(X)	faster than	X^2
	X*0.5	4 times faster than	X/2.0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4. **USE OPERANDS AND OPERATORS OF THE SAME DATA TYPE WHENEVER POSSIBLE.** Type conversions can be time-consuming.

5. **USE INTEGER TYPE VARIABLES FOR LOOP COUNTERS INSTEAD OF REAL.** Again INTEGER data type values execute about ten times faster than REAL.

6. **USE READ/WRITE STATEMENTS INSTEAD OF GET/PUT.** READ/WRITE statements execute faster than GET/PUT. When reading or writing, one line or record at a time is also inherently faster than one character at a time.

7. **COMPUTE VALUES BEFORE ENTERING A LOOP.** When executing loops each line may be evaluated many times before the loop ends. Any value that can be computed before entering a loop will increase the loop speed.

8. **USE THE LET STATEMENT TO COPY IDENTICALLY SIZED DATA STRUCTURES.** This feature is faster than copying arrays or other structures element-by-element inside a loop. The LET statement can also be dropped from the expression since BASIC09 will assume this function when one item is said to be equal to another.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 COMMANDS AND STATEMENTS LISTING

This section consists of a list of the BASIC09 commands and statements in alphabetical order. Each listing includes an outline of its syntax structure and a few helpful examples.

BASIC09 commands and statements can be organized according to their function. Some cause direct system action, while others are used to perform internal program functions. There are seven classifications of BASIC09 commands and statements:

- **BASIC09 mode commands**
- **Debug mode commands**
- **Control statements**
- **Declarative statements**
- **I/O statements**
- **Internal data statements**
- **Formatted output statements**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CLASSIFICATIONS

BASIC09 MODE COMMANDS

Commands available in the BASIC09 mode are used to save, load, and examine procedures. There are commands that interact with the OS-9 operating system and other commands that control the workspace environment.

BASIC09 commands include:

[CONTROL-]]	CHD	KILL	PACK
[CONTROL-E]	CHX	LIST	RENAME
$	DIR	LOAD	RUN
BYE	E *or* EDIT	MEM	SAVE

DEBUG MODE COMMANDS

Commands available in the debug mode allow procedures to be tested and manipulated using the actual BASIC09 statements as they were programmed. The program can be executed (one or multiple lines at a time), terminated, and resumed.

Debug mode commands include:

$	DIR	PRINT	STEP
BREAK	END	Q	TROFF
CONT	LET	RAD	TRON
DEG	LIST	STATE	

CONTROL STATEMENTS

Control statements affect the sequential execution of other procedure statements. They are used to construct loops or make decisions that alter execution flow.

Control statements include:

BASE 0	ERROR	LOOP/ENDLOOP	REPEAT/UNTIL
BASE 1	FOR/NEXT	ON ERROR GOTO	RUN
BYE	GOTO	ON GOSUB	SHELL
CHAIN	GOSUB/RETURN	ON GOTO	STOP
END	IF	PAUSE	WHILE/DO
EXITIF	KILL		

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

I/O STATEMENTS

Input/output statements are used within a procedure to transfer data between itself and a data file or I/O device. Table 42 lists the I/O statements along with the types of devices and files that can be used with them.

TABLE 42 - I/O STATEMENTS

Statement	Device Used With	Data File Type
INPUT	Keyboard (interactive input)	Text (sequential)
PRINT	Terminals, Printer	Text (sequential)
OPEN	COPRAM, I/O Devices	Any
CREATE	COPRAM	Any
CLOSE	COPRAM, I/O Devices	Any
DELETE	COPRAM	Any
SEEK	COPRAM	Binary (random)
READ	COPRAM	Text (sequential)
WRITE	COPRAM	Text (sequential)
GET	COPRAM and I/O Devices	Binary (random)
PUT	COPRAM and I/O Devices	Binary (random)

INTERNAL DATA STATEMENTS

Internal data statements are used to build constant tables within a procedure. The DATA statement provides values, the READ statement assigns values, and the RESTORE statement defines whether the READ or DATA is executed next.

FORMATTED OUTPUT STATEMENT

The PRINT USING statement provides an output editing capability. It is used to create a formatted output for report generation and other applications.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SYNTAX

Each BASIC09 Command and Statement includes a specific syntax. Syntax refers to all rules that specify precisely how each command or statement and their parameters are to be given so that BASIC09 can recognize and process the command correctly. Some parameters are optional in a command or statements syntax while others are required.

[ ]	=	brackets indicate that the enclosed item or items are optional.
{ }	=	braces indicate that the enclosed item or items can be either omitted or repeated.
<procname>	=	represents any legal module name.
<pathlist>	=	represents any legal pathlist.
<#d>	=	represents a decimal value.
<#h>	=	represents a hexadecimal value.
<cmd>	=	represents an OS-9 command.
<text>	=	represents alphanumeric characters entered via the terminal.
<expr>	=	represents any legal BASIC09 expression.
<boolexpr>	=	represents any legal BASIC09 Boolean expression.
<stringexpr>	=	represents any legal BASIC09 string expression.
<intexpr>	=	represents an internal expression (PATH, PORT).
<accessmode>	=	represents file accessibility (WRITE, UPDATE, EXEC).

COMMANDS

This section consists of a list of BASIC09 Commands and Statements in alphabetical order. Each command or statement description includes a syntax listing. To the right of each name is an indication of its us; Command, Statement, or both. If used as a command, you will also find the BASIC09 Mode(s)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

$

CALLS OS-9 SHELL	COMMAND
SYNTAX: $ [<cmd>]	Command Modes: DEBUG, BASIC09

This command calls the OS-9 SHELL command interpreter to process an OS-9 command or to run another procedure. If the $ is followed by a <cmd>, the SHELL is called to process the command as a single OS-9 command line. After the command is executed, BASIC09 is immediately reentered (except in the case of built-in SHELL commands, in which case BASIC09 is not reentered).

If no <cmd> is given, BASIC09 is suspended and the OS-9 SHELL is called to process multiple command lines individually entered from the Terminal Emulator keyboard. Control is returned to BASIC09 when an end-of-file character ([CONTROL-]]) is entered. The contents of the BASIC09 workspace are not disturbed. This is a convenient way to temporarily leave BASIC09 to manipulate files or perform general system housekeeping.

This command is the gateway to OS-9 from within BASIC09. It allows access to any OS-9 command or other programs. It also permits the creation of concurrent processes and other real-time functions.

EXAMPLES:

B:$COPY proc1 proc1A	(calls OS-9 COPY command).
B:$DIR	(displays current COPRAM data directory).
B:$ **OS9:**	(suspends BASIC09 and enters OS-9).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASE 0
BASE 1

ESTABLISHES ARRAY INDEX AS BASE 0 OR BASE1	STATEMENT
SYNTAX: BASE <#d>	

These statements indicate whether a particular procedure's lowest array or data structure index (subscript) is 0 or 1. The default, <#d>, is 1, BASE 1. These statements do not affect the string operations (RIGHT$, MID$, or LEFT$). In these cases the beginning character is always indexed as one.

EXAMPLES:

BASE 0	(sets next array index to BASE 0).
DIM A(2,3)	(creates a two-dimensional array A; first element is A(0,0), last is A(1.2)).
BASE 1	(sets next array index to BASE 1).
DIM B(2,3)	(creates a two-dimensional array B; first element is B(0,0), last is B(1.2)).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BREAK

SET PROCEDURE BREAKPOINT	COMMAND
SYNTAX: BREAK [,<procname>]	Command Mode: DEBUG

This command sets up a breakpoint position in the named procedure. This command is used when procedures call each other, and provides a way to reenter the debug mode when returning to a specific procedure. To illustrate how the BREAK command works, suppose there are three procedures in the workspace: **proc1**, **proc2**, and **proc3**. Assume that **proc1** calls **proc2** which in turn calls **proc3**. While **proc3** is executing, you as the user enter the program interrupt ([CONTROL-C]) to enter the debug mode.

To establish a breakpoint at **proc1** the following is entered:

EXAMPLE:

D:BREAK proc1	(sets a breakpoint at **proc1**).
ok	(an **ok** indicates the procedure was found).
D:CONT	(resumes execution).

The response of **ok** indicates that the procedure named was found in the active RUN stack and that a breakpoint was assigned to it. The STATE command can be used to verify that all procedures are nested as expected. Execution of **proc3** is resumed with the CONT command. After **proc3** terminates, control is passed to **proc2**, which in turn terminates and returns control to **proc1**. As soon as this occurs, the breakpoint is recognized, execution of **proc1** is suspended, and the debug mode is called.

Three notable characteristics of the BREAK command are:

1. The breakpoint is removed as soon as it occurs.
2. Only one breakpoint can be established for each active procedure.
3. A breakpoint cannot be set for active procedures. The procedure named must have already been called, suspended, and not yet returned to.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BYE

EXITS BASIC09 WORKSPACE	COMMAND/STATEMENT
SYNTAX: BYE	Command Mode: BASIC09

BYE can be used as a command from the B: prompt or as a statement from within a procedure. Both uses provide the same function and produce the same result -- the termination of BASIC09 and the return to OS-9. Any open files are closed, but unsaved procedures in the workspace are lost. The BYE command is particularly useful for creating packed BASIC09 procedures and/or procedures that are to be called from OS-9 procedure files. The BYE statement can be used to exit a packed procedure, placing the system in OS-9 rather than BASIC09.

EXAMPLE:

```
PRINT "This is the end of the procedure."
PRINT "Exiting to OS-9
BYE                                    (BYE as a statement).

B:BYE                                  (BYE as a command).
```

WARNING

The BYE command causes BASIC09 to abort. All unsaved procedures in the workspace will be destroyed. Always save needed procedures prior to issuing the BYE command. Remember, packed BASIC09 procedures have had all their source codes removed and cannot be edited, debugged, or converted back to a source code format.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAIN

PASSES PARAMETERS TO AN OS-9 MODULE	STATEMENT
SYNTAX: CHAIN <string expr>	

The CHAIN statement performs an OS-9 chain operation on the SHELL, passing the specified string as an argument. The string must be enclosed in quotation marks. This causes BASIC09 to be exited, unlinked, and its memory to be returned to OS-9. The string should evaluate the name of an executable module (such as DIR), and pass parameters if appropriate.

CHAIN can begin execution of any module. It executes the module indirectly through the SHELL to take advantage of the SHELL's parameter processing. This has the side-effect of leaving an extra incarnation of the SHELL active. Programs that repeatedly chain to each other will eventually find that all available memory has been filled with unused available SHELLs. This can be prevented by using the EX option of the SHELL.

Files that are open when a CHAIN is executed are not closed. However when creating a new process, OS-9 will only pass the standard I/O paths (0, 1, 2) to a child process. Therefore, if it is necessary to pass an open path to another procedure, the EX option must be used.

EXAMPLE:

CHAIN "EX BASIC09 menu"	(exits the current workspace and creates a new workspace without creating a new SHELL, loads **menu**).
CHAIN "BASIC09 #10K sort(22,5)"	(exits the current workspace and creates a new workspace with 10Kbytes of memory; loads **sort** and passes it two parameters **22** and **5**).
CHAIN "DIR /D0"	(exits the workspace and displays the /D0 (root) directory).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHD

CHANGES CURRENT COPRAM DATA DIRECTORY	COMMAND
SYNTAX: CHD <pathlist>	Command Mode: BASIC09

Changes the current COPRAM data directory pointer to the specified pathlist. The pathlist must be a valid directory file. Several BASIC09 commands and statements default to the current data directory to complete their operation if a specific path is not otherwise indicated. The SAVE and LOAD commands, for example, default to the current data directory.

EXAMPLES:

B:CHD /D0/CMDS	(changes the current working data directory to **CMDS**).
B:CHD /D0	(changes the current working data directory to **/D0**, root directory).
B:CHD /D0/APPLICS/SOURCE	(changes the current working data directory to **SOURCE**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHX

CHANGES CURRENT COPRAM EXECUTION DIRECTORY	COMMAND
SYNTAX: CHX <pathlist>	Command Mode: BASIC09

This command changes the current COPRAM execution directory pointer to the specified pathlist. The pathlist must be a valid directory file. BASIC09 uses the current execution directory as a default to store packed source coed procedures if a specific path is not otherwise indicated.

EXAMPLES:

B:CHD /D0

(changes the current working data directory to **/D0**, root directory)).

B:CHD /D0/APPLICS/PACK

(changes the current working data directory to **PACK**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CLOSE

CLOSES AN OPEN I/O PATH	STATEMENT
SYNTAX: CHAIN #<intexpr> {,#<intexpr>}	

The CLOSE statement notifies OS-9 that one or more I/O paths are no longer needed. A path is specified by its number or name. If the path being closed uses a nonsharable device (such as a printer), the device is released and can be assigned to another user. The path must have been previously established by the OPEN or CREATE statements.

EXAMPLES:

B:CLOSE #PATH	(close the previously opened path).
B:CLOSE #PORT	(close the previously opened port).
B:CLOSE #PATH, #PORT	(close the previously opened path and port).
B:CLOSE #5	(close the previously opened path 5).
B:CLOSE #6, #7	(close the previously opened paths 6 and 7).

CAUTION

Paths #0, #1, and #2 (the standard output paths) **should never be closed** unless you as the programmer immediately open a new path to assume the standard path number (0 = keyboard input; 1 = display output; 2 = display error output).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CONT

RESUME NORMAL PROCEDURE EXECUTION	COMMAND
SYNTAX: CONT	Command Mode: DEBUG

This command, used only in the debug mode, causes procedure execution to continue at the next statement. It may be used to resume procedures suspended by [CONTROL-C] keyboard interrupts, PAUSE statements, BREAK command breakpoints, or after a nonfatal run-time error.

EXAMPLE:

BREAK PROCEDURE test	(suspended procedure name is displayed).
D:STEP 5	(debug operations are performed).
D:CONT	(exits debug and resumes normal execution).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CREATE

CREATE A NEW FILE IN COPRAM	STATEMENT
SYNTAX: CREATE <intrexpr>, <stringexpr> [:<access mode]	

The CREATE statement is used to create a new file in COPRAM. The variable name is used to receive the path number assigned by OS-9 and must be dimensioned as either type BYTE or INTEGER. The STRING expression is evaluated and passed to OS-9 to be used as the descriptive pathlist.

The access mode defines the direction of subsequent I/O transfers and should be either WRITE or UPDATE. The UPDATE mode allows the file to be read or written to. UPDATE is the default mode for the operation.

OS-9 has a single file type that can be accessed in either the sequential or random access mode. Files are byte-addressed, so no explicit record length need be given. When a new file is created, it has an initial length of zero. Files are expanded automatically by PRINT, WRITE, or PUT statements that write beyond the current end-of-file marker.

EXAMPLE:

```
DIM PATH,TRANS,SPOOL,OUTPUT:BYTE
```

CREATE #PATH, "storage"	(creates COPRAM file **storage** in the data directory, defaults to UPDATE mode).
CREATE #TRANS, "transactions:UPDATE	(creates a COPRAM file **transactions** that can be read from and written to).
CREATE #SPOOL, "/DATA/recipe1":WRITE	(creates a COPRAM file **recipe1** under DATA that can be written to but not read).
CREATE #OUTPUT, NAME$:UPDATE+EXEC	(creates a COPRAM file **name$** that can be read from, written to, and executed).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DATA

PROVIDES USER VALUES FOR A CONSTANT TABLE	STATEMENT
SYNTAX: DATA <expr>;{<expr>}	

The DATA statement (along with the READ and RESTORE statements) is used to build constant tables within a procedure. The DATA statement provides values. It has one or more expressions separated by commas, and can be located anywhere in the procedure. The expressions are evaluated each time the DATA statement is read. Any of the five atomic data types can be used.

EXAMPLES:

DATA 1.1, 1.5, 9999, "HI", "LOW"	(creates a data table that holds five elements).
DATA SIN(temp/25), COS(temp/pi)	(creates a data table that contains two formulas as expressions).
DATA TRUE,FALSE, TRUE, TRUE, FALSE	(creates a data table that contains five elements).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DEG/RAD

EVALUATES ANGLES IN DEGREES OR RADIANS	COMMAND/STATEMENT
SYNTAX: DEG RAD	Command Mode: DEBUG

DEG and RAD can be used as statements within a BASIC09 procedure and as commands within the debug mode. They have the same function in both cases. These statements allow angles to be evaluated in degrees or radians in such functions as SIN, COS, TAN, ACS, ASN, and ATN. A state flag is set and this flag applies only to the currently-active procedure. The default state is radians, RAD.

STATEMENT EXAMPLES:

```
DEG                    (angle values are evaluated in degrees).
X=SIN(PI/2)            (X will equal approximately .0274 degrees).

RAD                    (angle values are evaluated in degrees).
X=SIN(PI/2)            (X will equal approximately 1 radian).
```

COMMAND EXAMPLES:

```
D:DEG                  (angle values are evaluated in degrees).
X=SIN(PI/2)            (X will equal approximately .0274 degrees).
=.0274                 (result is displayed in the debug mode).

D:RAD                  (angle values are evaluated in degrees).
X=SIN(PI/2)            (X will equal approximately 1 radian).
=1                     (result is displayed in the debug mode).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DELETE

DELETES A COPRAM FILE	STATEMENT
SYNTAX: DELETE <stringexpr>	

This statement is used to delete COPRAM files from BASIC09 procedures. The file's name is removed from the directory and all its storage space is de-allocated. All data contained in the file is lost. The string expression is evaluated and passed to OS-9 as the descriptive pathlist of the file. If the file name and no pathlist is given, then the file is deleted from the current data directory. You must have write permission for the file in order to delete it.

EXAMPLES:

DELETE "test"	(deletes **test** from the current data directory in COPRAM).
DELETE "/D0/SOURCE/temp"	(deletes **temp** from **/D0/SOURCE**).
LET file$="daily.history"** **DELETE file$	(deletes **daily.history** from the current data directory in COPRAM).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DIM

DECLARES VARIABLES, ARRAYS, AND COMPLEX DATA STRUCTURES	STATEMENT
SYNTAX: DIM <var>, {<var>}[:type>]; {<var>, {<var>}[:type>]}	

The DIM statement is used to declare simple variables, arrays, or complex data structures of the five atomic data types, or any user-defined data type. During compilation, BASIC09 assigns storage required for all variables declared in DIM statements.

Declaring Simple Variables

Simple variables are declared by using the variable name in a DIM statement without a subscript. If variables are not explicitly declared they are made type REAL, or STRING[32] if the name ends with a $ character. All simple variables of other types must be explicitly declared. In addition, several types can be declared in a single DIM statement by using a semicolon to separate the different types.

EXAMPLES:

DIM logical:BOOLEAN	(declares the variable **logical** to be type BOOLEAN).
DIM a, b, c:STRING	(declares the variables **a**, **b**, **c** to be type STRING).
DIM a, b, c:INTEGER; n,m:REAL; x:BOOLEAN	(declares the variables **a**, **b**, **c** to be type INTEGER; **n**, **m** to be type REAL; and **x** to be type BOOLEAN).
DIM name:STRING[40]; id:STRING	(declares the variable name to be type STRING at 40 characters maximum and id as type STRING with a default of 32 characters maximum).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Array Declarations

Arrays can have one, two, or three dimensions. The DIM statement format is the same as for simple variables except that each name is followed by a subscript to indicate its size and number of dimensions (1, 2, or 3). The maximum subscript size is 32767. Simple variable and array declarations can be mixed in the same DIM statement.

EXAMPLES:

DIM x(10)	(declares the array **x** to be one-dimensional containing 10 elements).
DIM y:(5, 5)	(declares the array **y** to be two-dimensional containing 25 elements).
DIM z(10,20,5)	(declares the array **z** to be three-dimensional containing 1000 elements).
DIM test(20, 60); error(12)	(declares the array **test** to be two-dimensional with 1200 elements and **error** to be one dimensional with 12 elements).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DIR

DISPLAYS WORKSPACE DIRECTORY	COMMAND
SYNTAX: DIR [<pathlist>]	Command Modes: DEBUG, BASIC09

This command displays the name, size (proc-size), and variable storage (data-size) requirement of each procedure presently in the workspace. The current working procedure is preceded by an asterisk. Any packed procedures will also display a minus sign prior to their name. The available free memory within the workspace is also given. If a pathlist is given, the DIR commands listing will be directed to that location -- COPRAM, MiniCOP port, etc. The debug mode uses the DIR command in the same manner as the BASIC09 mode.

If proc-size is preceded by a question mark (?), the workspace does not have sufficient free memory to execute the procedure.

Pressing the **[RETURN]** key at the B: prompt has the same effect as entering the DIR command in BASIC09 mode.

EXAMPLE:

B:DIR (display the contents of the workspace).

	Name	Proc-Size	Data-Size	
*	file001	127	42	(current working procedure, "*").
-	file002	366	109	(a packed procedure, "-").
	file003	? 2796	1107	(insufficient workspace memory to execute).

2239 free (available workspace memory).

NOTE

This command should not be confused with the OS-9 DIR command. The two appear similar, but have completely different meanings.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DO (see WHILE/DO)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

E/EDIT

ENTERS THE EDIT MODE		COMMAND
SYNTAX:	E [<procname>]	Command Mode: BASIC09
	EDIT [<procname>]	

The EDIT command exits the BASIC09 mode and enters the edit mode of operation. This allows you to create and edit BASIC09 procedures. If no procedure name is specified and no existing workspace procedure is tagged as current, BASIC09 will create a new procedure called **Program**. If no procedure name is specified and a procedure in the workspace is tagged as current, BASIC09 will call that procedure. If a name is specified and a procedure using that name already exists in the workspace, BASIC09 will call that procedure. If a name is specified and it does not already appear in the workspace, BASIC09 will create a new procedure using that name. Either the entire edit command's name, EDIT, or E can be used.

EXAMPLES:

EDIT	(calls the current workspace procedure if one exists; if not, creates a procedure called **Program**).
E monitor	(calls the procedure **monitor** if it exists; if not, it will be created).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

END

RETURNS BASIC09 TO ITS CALLING PROCEDURE	COMMAND/STATEMENT
SYNTAX: END [<outputlist>]	Command Mode: DEBUG

This statement ends the execution of the procedure and returns to the calling procedure, or to the BASIC09 mode if it was the highest level procedure. If an **<outputlist>** is given, it will be displayed to the monitor as it would by a PRINT statement. END is an executable statement and can be used several times in the same procedure. The END statement is optional -- it is not required at the end of a procedure. END used as a command in the debug mode has the same results as END used as a statement.

EXAMPLES:

END	(executing procedure terminates).
END "Current process complete."	(executing procedure terminates and the message **Current process complete** is displayed).
END "Final result = ", val	(executing procedure terminates and the message **Final result** = is displayed along with the value of the variable **val**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ENDEXIT (see LOOP/ENDLOOP)

ENDIF (see IF (VERSION 2))

ENDLOOP (see LOOP/ENDLOOP)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ERROR

GENERATES AN ERROR CODE	STATEMENT
SYNTAX: END <intexpr>	

This statement generates an error having the error code specified by the result of evaluation of the expression. ERROR is often used for testing error routines.

EXAMPLES:

ERROR 215	(generates the **Bad Path Name** error).
ERROR 207	(generates the **Memory Full** error).
ERROR 205	(generates the **Illegal Module Header** error).

NOTE

For additional information regarding the ERROR statement, refer to the *On Error GOTO Statement* description in this section of the manual and *Appendix F*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXITIF (see LOOP/ENDLOOP)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FOR/NEXT

CREATE A CONTROLLED PROGRAM LOOP	STATEMENT
SYNTAX: FOR <var> = <expr1> TO <expr2> [STEP <expr3>] NEXT <var>	

This statement creates a loop that executes a given number of times while automatically incrementing or decrementing a specified counter variable, <var>. The <expr1> expression is evaluated and the result is stored in <var>, which must be a simple INTEGER or REAL variable. The <expr2> expression is then evaluated and stored in a temporary variable established by BASIC09. If the STEP clause **is not** used, the value of <var> is incremented by 1. If the STEP clause **is** used: the value of <var> is used; the <expr3> expression is evaluated; and the value of <var> is incremented or decremented by its result (the STEP expression can be positive or negative).

Statements between the FOR and NEXT statements are executed. When the NEXT statement is executed, if <var> is less than <expr2>, then the loop is repeated beginning at the FOR statement. If <var> is equal to or greater than <expr2>, the loop is terminated and normal sequential execution begins at the next line of code.

EXAMPLES:

```
FOR x=1 TO 100                          (defaults to a STEP of 1).
  PRINT ER$                             (prints the text in ER$).
NEXT x                                  (if x=>100 terminate loop, if x<100 repeat).

FOR num=1000 TO 1 STEP -1               (decrements num by 1 each loop).
  PRINT num                             (prints the value of num).
NEXT num                                (if num<=1 terminate loop, if num>1 repeat).

FOR counter=1 TO limit STEP.5           (increments counter by .5 each loop).
PRINT "Current Count = ";counter        (prints the message and value of counter).
NEXT counter                            (if x=limit terminate loop, if x<>limit repeat).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GET/PUT

READ AND WRITE DATA RECORDS TO COPRAM FILES OR I/O DEVICES	STATEMENT
SYNTAX: GET #<expr>, <structname> PUT #<expr>, <structname>	

The GET and PUT statements read and write fixed-size binary data records to COPRAM files or I/O devices. These are the primary I/O statements used for random-access input and output operations.

The <expr> is evaluated and used as the number of the I/O paths, which must have previously been opened by an OPEN or CREATE statement. Paths used by PUT statements must have been opened in WRITE or UPDATE access modes, and paths used by GET statements must be in the READ or UPDATE mode.

The statement uses exactly one name, which can be the name of a variable, array, or complex data structure. Data is written from, or read into, the variable structure named. The data is transferred in BASIC09's internal binary format without conversion which affords very high throughput as compared to READ and WRITE statements. Data is transferred beginning at the current position of the path's file pointer, which is updated automatically. The SEEK statement is used to control the file pointer address.

OS-9's file system does not impose record structures on random-access files. All files are considered to be continuous sequences of addressable binary bytes. A byte or group of bytes located anywhere in the file can be read or written in any order. You as the programmer are free to use the file access system to create any record structure.

Record I/O in BASIC09 is associated with data structures defined by DIM and TYPE statements. The GET and PUT statements write entire data structures or parts of data structures. A PUT statement, for example, can write a simple variable, an entire array, or a complex data structure in one operation.

GET and PUT statements affect the entire structure. In the following example all 32 (default) elements contained in the variable char will be transferred (PUT) to I/O path #1 and written into a data structure also called **char**.

```
DIM char:STRING
char="1,2,3"
PUT #1,char
```

The combination of complex data structures and random-access I/O statements has the following characteristics:

1. It is self-documenting -- what a procedure does is clearly seen since data structures have descriptive, named substructures.

2. It is extremely fast.

3. Procedures are simplified and typically require fewer statements to perform I/O functions.

4. It is versatile -- by creating appropriate data structures almost any kind of data on any file can be read or written, including files created by other procedures.

These advantages are possible because a single GET and PUT statement can move any amount of data organized in any manner desired.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NOTE

In many cases, the I/O path number that follows the GET and PUT statement will be one of the three standard I/O paths (0 = keyboard input path; 1 = display output path; 2 = display error output path).

GET EXAMPLES:

```
REM This procedure will GET a character coming in the standard
REM input path 1 and print it to the monitor screen.
DIM char:STRING[1]
GET #0, char
PRINT char

REM This procedure demonstrates the use of the BASIC09 GET statement.  In
REM this procedure 10 characters are gathered from the MiniCOP's Port 3 (/T3).
DIM port3:BYTE
OPEN #port2, "/T2":READ
FOR x=1 TO 10
  GET #port3,char
  PRINT #port3,char,CHR$(char)                    (feedback to port 3)
NEXT x
CLOSE #port2
```

PUT EXAMPLES:

```
REM This procedure will PUT the characters found in char$
REM to the standard output path 1.
char$="01234567890123456789012345678901"
PUT #1, char$

REM This procedure demonstrates the use of the BASIC09 PUT statement.  In
REM this procedure digits 1 through 9 and 0 are sent to the device at the MiniCOP's
REM Port 1 (/T1).
DIM port1:BYTE
DIM nums:STRING[10]
OPEN #port1, "/T1":WRITE
nums="1234567890"
CLOSE #port1
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GOTO

TRANSFERS EXECUTION TO A DESIGNATED LINE NUMBER	STATEMENT
SYNTAX: GOTO <#d>	

The GOTO statement **unconditionally** transfers execution flow to the line having the specified line number. The line number, <#d>, must be a constant value and cannot be an expression or variable.

EXAMPLE:

GOTO 1000	(unconditionally transfers program execution to line 1000).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GOSUB/RETURN

TRANSFERS EXECUTION TO A SUBROUTINE AND RETURNS	STATEMENT
SYNTAX: GOSUB <#d>	

The GOSUB statement transfers execution flow to a subroutine starting at the specified line number, <#d>.The subroutine is executed until a RETURN statement is encountered, which causes execution to resume at the statement immediately following the GOSUB. Subroutines may be nested to any depth.

EXAMPLE:

```
        REM This procedure contains a simple subroutine
        REM which provides the square of the given number.
10      INPUT "Enter a number to be SQUARED: ";num
        GOSUB 20
        GOTO 10
20      PRINT num;" SQUARED = ";num^2
        RETURN
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IF (VERSION 1)

TRANSFERS EXECUTION AFTER A TRUE EXPRESSION	STATEMENT
SYNTAX: IF <boolexpr> THEN <#d>	

This form of the IF statement causes execution to be transferred to the statement used on the specified line number, <#d>, when the <boolexpr> is TRUE. Otherwise, the next sequential statement is executed. No ENDIF statement is required with this version.

EXAMPLE:

IF temp < hilimit THEN 400

(if the Boolean expression **temp => hilimit** is TRUE then execution continues at line 400; otherwise execution continues at the next statement).

If x=100 THEN 250

(if the Boolean expression **x =100** is TRUE then execution continues at line 250; otherwise execution continues at the next statement).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

IF (VERSION 2)

TRANSFERS EXECUTION AFTER A TRUE EXPR. OR ELSE STATEMENT	STATEMENT
SYNTAX: IF <boolexpr> THEN <statement> [ELSE <statement>] ENDIF	

This form of the IF statement uses the ELSE and ENDIF statements. In this form, the IF structure evaluates the <boolexpr> and:

If the result is true: the <statement> following THEN is executed; if an ELSE statement exists, statements between the ELSE and ENDIF are skipped.

If the result is false: execution control is transferred to the first statement following the ELSE, if present, or otherwise to the statement following the ENDIF.

EXAMPLES:

```
IF a<b THEN
    PRINT a;" is less than ";b
ELSE
    IF a=b THEN
        PRINT a;" equals ";b
    ELSE
        PRINT a;" is greater than ";b
    ENDIF
ENDIF

10  INPUT "ENTER YOUR PASSWORD",PW$
    IF PW$="123ABC" THEN
      PRINT "THANK YOU, ACCESS GRANTED"
    ELSE
      PRINT "SORRY, INVALID PASSWORD.  ACCESS DENIED"
      GOTO 10
    ENDIF
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INPUT

ACCEPTS INPUT DURING PROCEDURE EXECUTION	STATEMENT
SYNTAX: INPUT [#<expr>,] ["<prompt>",] <var>, {<var>}	

This statement accepts input during execution of a procedure. The input is normally read from the standard input path (#0 - keyboard) unless an option path number, #<expr>, is given. If a path other than the default (standard input path) is used, the path should be opened in the UPDATE mode. Extreme care should be taken when using the INPUT statement on COPRAM files.

The INPUT statement can act as both an input and output statement. When the INPUT statement is encountered, procedure execution is suspended and a "?" prompt is displayed. If the optional <prompt> string is given, it is displayed in place of the default "?" prompt. The INPUT statement can also be used to display messages relating to the required input.

The data entered is assigned to the variable names (<var>) in the order in which they appear in the input list. The variable must be separated by commas. They can be of any atomic data type, and the input data provided must be of the same or compatible type. There must be at least as many input items provided as variables in the input list.

The line is terminated by a carriage return. The input line cannot exceed 256 characters. If any errors, such as type mismatch, insufficient amount of data, etc. occur, the message:

****INPUT ERROR RETYPE****

is displayed followed by a new prompt. The entire input line must then be reentered.

The INPUT statement uses OS-9's line input function, which performs line editing such as backspace, delete, end-of-file, etc. To perform input without editing (that is, to read pure binary data), use the GET statement.

EXAMPLES:

```
INPUT number, name$, location

INPUT #path,x y, z

INPUT "Enter RECIPE Code Number";rcn
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

KILL

ERASES PROCEDURES FROM THE WORKSPACE	COMMAND/STATEMENT
SYNTAX: KILL [<procname> {[<procname>]} KILL*	Command Mode: BASIC09

KILL can be used as a BASIC09 mode command and as a statement within a procedure. Both uses have the same function, to remove or unlink a named procedure. The command can be used with a named procedure or procedures, or it can be followed by an asterisk (*) in which case the entire workspace is erased.

The statement is used to unlink an external procedure (possibly returning system memory) and to remove it from BASIC09's procedure directory. If the procedure is inside the workspace, nothing happens and no error is generated. KILL can be used along with autoloading PACKed procedures as an alternative to the CHAIN statement when procedure overlay is desired.

Typically, when using KILL as a statement, the RUN statement is also used. In this case, the RUN statement normally specifies a procedure name that does not exist in the workspace, but can be found either in the current data directory (source procedure) or the current execution directory (packed procedure). If the procedure is not currently in the workspace it will be loaded from COPRAM and then executed by the RUN statement. After it has terminated its execution, KILL will remove it from the workspace. This method of loading, executing, and killing procedures helps to manage the workspace memory.

WARNINGS

Three important factors must be taken into consideration when killing a procedure:

1. It can be **fatal** to OS-9 to **kill** a procedure that is still **active**.
2. When used together with a RUN statement, the RUN statement **must** use the same string variable that contains the name of the procedure.
3. It is **not legal** to use a string constant for a procedure name (refer to the first statement example).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

COMMAND EXAMPLES:

B:KILL testproc1	(erases **testproc1** from the workspace).
B:KILL testproc1, testproc2, testproc3	(erases **testproc1**, **testproc2**, and **testproc3** from the workspace).
B:KILL*	(erases all procedures from the workspace).

STATEMENT EXAMPLES:

LET procname$="average"	(assigns the string average to procname$).
RUN "average"	(this operation is **ILLEGAL**).
KILL "average"	(this operation is **ILLEGAL**).
LET procname$="average"	(assigns the string average to **procname$**).
RUN procname$	(loads and runs **procname$**, **"average"**).
KILL procname$	(kills **procname$**, **"average"**).
INPUT "Enter TEST to be executed";test$	(assigns the string average to **procname$**).
RUN test$	(loads and runs **test$**).
KILL test$	(kills **test$**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LET

ASSIGNS THE RESULT OF AN EXPRESSION TO A VARIABLE	COMMAND/STATEMENT
SYNTAX: LET <var>:=<expr> LET <var>=<expr> LET <struct1>:=<struct2> LET <struct1>=<struct2>	Command Mode: DEBUG

This statement evaluates an expression and stores the result in <var>, which may be a simple variable or data structure element. The result of the expression must be of the same or compatible type as <var>. BASIC09 will accept either "=" or ":=" as the assignment operator. The second form, ":=", distinguishes the assignment operation from a comparison test. The ":=" is identical to that used by the PASCAL programming language.

Another use of the assignment statement is to copy the entire contents of an array or complex data structure to another array or complex data structure. The data structures do not have to have the same type or shape. The only restriction is that the size of the destination (<struct2>) be equal to or larger than the source (<struct1>). This type of assignment can be used to perform unusual type conversions. For example, a string variable of 80 characters can be copied to a one-dimensional array of 80 bytes.

The LET statement can also be used in the debug mode, but not with user-defined data structures.

The LET statement itself is optional. BASIC09 treats any expression separated by "=" or ":=" as though it were preceded by a LET. Dropping the LET will save some memory.

COMMAND EXAMPLES:

B:LET A=0.1	(assigns the value **0.1** to **A**).
B:LET BATCHCODE$=DATE$+SHIFT$	(assigns the contents of **DATE$** and **SHIFT$** to **BATCHCODE$**).
VAL=temp/sin(x)	(assigns the expressions result to **VAL**).
DIM HI, LO:BOOLEAN	(declares **HI** and **LO** as type BOOLEAN).
HI:=x<1000	(**HI**=TRUE or FALSE).
LO:=y>200	(**LO**=TRUE or FALSE).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LIST

PRINTS A PROCEDURE LISTING		COMMAND
SYNTAX: LIST	[<procname> {, <procname>}][><pathlist>]	Command Modes: DEBUG, BASIC09
LIST*	[<pathlist>]	

Prints a formatted listing of one or more source code procedures that reside in the workspace. The listing includes each line's relative I-code storage addresses (in hexadecimal) in the first column. The second column is reserved for line numbers, if used.

If a pathlist is given, the listing is sent to that file or device. This option is commonly used to print hardcopy listings on a printer connected to one of the MiniCOP's ASYNC ports. The LIST, SAVE, and PACK commands all have similar syntax, except that LIST will print to the standard output path (#1 display output) as a default if no <pathlist > is given. When a <pathlist> is specified, it must be proceeded by the redirection modifier, >.

The LIST command can also be used in the debug mode. However, only the current procedure can be listed. The asterisk (*) is not applicable in this mode.

BASIC09 MODE EXAMPLES:

B:LIST testproc	(displays a listing of the procedure **testproc** to the standard output path).
B:LIST*	(displays a listing of ALL procedures in the workspace to the standard output path).
B:LIST extruder >T3	(sends a listing of the procedure extruder to the device at port 3).
B:LIST* >T3	(sends a listing of all procedures in the workspace to the device at port 3).

DEBUG MODE EXAMPLE

D:LIST	(displays a listing of the procedure currently being debugged to the standard output path).
B:LIST >T3	(sends a listing of the procedure currently being debugged to the device at port 3).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LOAD

LOADS A PROCEDURE FROM COPRAM OR THE TERMINAL EMULATOR	COMMAND
SYNTAX: LOAD <pathlist>	Command Mode: BASIC09

This command loads procedures from the specified COPRAM directory or Terminal Emulator. As procedures are loaded their names are displayed. If any of the procedures being loaded have the same name as a procedure already in the workspace, the existing procedure is erased and replaced with the new.

Both the PACK and SAVE command can be used to store multiple procedures under one file name. When this type of file is loaded, each procedure it contains is loaded individually in the order it was stored.

If the workspace fills before the last procedure in the file is loaded, an **Error #32 - Memory Full**, is given. In this case not all the procedures may have been loaded, and/or the last procedure being loaded when the error occurred may not have completed its load operation. The last procedure should be erased with the KILL command and the MEM command should be used to increase the workspace memory size. Another alternative to the MEM command is to also KILL any unnecessary procedures from the workspace until sufficient memory space is achieved. In either case, reLOAD the procedure(s).

When LOADing files from the Terminal Emulator, it will be necessary to interface with its functions to send the required files to the MiniCOP's tape port.

EXAMPLES:

B:LOAD functions	(loads the file **functions** into the workspace).
B:LOAD /TP B09file01	(loads the file **B09file01** from the tape port).
B:LIST /TP	(loads the next file sent from the tape port into the workspace).

NOTE

Procedures stored in source code form will be compiled as they are loaded into the workspace. There may be a relatively long delay when loading a large source code procedure.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LOOP/ENDLOOP/EXITIF/ENDEXIT

CONSTRUCTS LOOPS WITH INTERNAL TEST POINTS		STATEMENTS
SYNTAX:	LOOP EXITIF <boolexpr> THEN <statement> ENDIF ENDLOOP	

These related types of statements can be used to construct loops with test points located in the body of the loop. The LOOP and ENDLOOP statements define or enclose the loop's body. EXITIF clauses can be inserted anywhere inside the loop, causing the loop to be exited when TRUE.

The EXITIF clause evaluates an expression to a Boolean result. If the expression is FALSE, the statement following the ENDEXIT is executed next. Otherwise, the statement or statements between the EXITIF and ENDEXIT are executed, control is then transferred to the statement following the end of the body of the loop, ENDLOOP. This exit clause is often used to perform some specific function upon termination of the loop, which depends on where the loop terminated.

EXITIF statements are almost exclusively used with the LOOP/ENDLOOP statements, but they can also be used in any other type of BASIC09 loop statement structure, such as FOR/NEXT, REPEAT/UNTIL, and WHILE/DO.

CAUTION

The EXITIF clause **must** be included somewhere within the loop. If not an unending or continuous loop will occur.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXAMPLES:

```
x=0                                          (defines the value of x).
LOOP                                         (start of loop body).
EXITIF x>10 THEN ENDEXIT                     (if TRUE, exit loop; if FALSE
                                             continue).
    PRINT x
    x=x+1
ENDLOOP                                      (end of loop body).
PRINT "LOOP COMPLETE"                        (execution resumes here when EXITIF
                                             is TRUE).


count=0                                      (defines the value of count).
LOOP                                         (start of loop body).
EXITIF count>25 THEN                         (if TRUE, exit loop; if FALSE
                                             continue).
    PRINT "NOW LEAVING LOOP!"
ENDEXIT
    PRINT count
    count=count+5
ENDLOOP                                      (end of loop body).
PRINT "LOOP COMPLETE"                        (execution resumes here when EXITIF
                                             is TRUE).


INPUT "Enter two INTEGERS: ";X, Y            (defines X and Y).
LOOP                                         (start of loop body).
    PRINT
EXITIF X>0 THEN                              (if TRUE, exit loop; if FALSE
                                             continue).
    PRINT "X became zero first"
ENDEXIT
    X=X-1
EXITIF Y<0 THEN PRINT "Y became zero first"  (if TRUE, exit loop; if FALSE
                                             continue).
    Y=Y-1
ENDLOOP                                      (end of loop body).
PRINT "LOOP COMPLETE"                        (execution resumes here when EXITIF
                                             is TRUE).


WHILE TRUE DO                                (start of loop body).
INPUT "Type [C] to exit loop: ";char$
EXITIF char$="C" THEN                        (if TRUE, exit loop; if FALSE
                                             continue).
    PRINT "You have elected to leave the loop"
ENDEXIT
ENDWHILE                                     (end of loop body).
PRINT "LOOP COMPLETE"                        (execution resumes here when EXITIF
                                             is TRUE).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MEM

DISPLAY AND OR CHANGE WORKSPACE MEMORY SIZE	COMMAND
SYNTAX: MEM [<#d>] MEM [$<#h>]	Command Mode: BASIC09

The MEM command, when used without a numeric parameter, will display the present total workspace memory size in decimal bytes. If a decimal value is provided as <#d>, BASIC09 requests that OS-9 expand the workspace memory to that size. The MEM command can also accept hexadecimal values, <#h>, so long as the "$" is included. Workspace memory size can be increased or decreased depending on the current workspace size and the size requested. BASIC09 changes workspace memory in OS-9 page boundaries. Each page contains 256 bytes. If the requested value is not a multiple of 256, BASIC09 will round the value up to the next higher value. Workspace memory size can also be changed when BASIC09 is executed from OS-9.

If the MEM command responds with the message **"What?"**, one of three things has occurred:

1. A request has been made for more memory than is available.
2. A return of too much memory was requested. The workspace memory must be sufficient to store all current procedures.
3. An invalid number was specified.

EXAMPLES:

MEM	(displays the memory size of the current workspace).
MEM 4095	(requests a change in size to 4095 bytes, decimal).
MEM $4000	(requests a change in size to 4000 bytes, hexadecimal (16383 decimal)).

NOTE

For additional information regarding the changing of the workspace memory size, refer to *Changing the Workspace Size* in the *BASIC09 Programming Background* subsection.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

NEXT (see FOR/NEXT)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ON ERROR GOTO

SETS AN ERROR TRAP	STATEMENT
SYNTAX: ON ERROR [GOTO <#d>]	

This statement sets an error trap that transfers control to the line number, <#d>, given when a nonfatal run-time error occurs. The statement must be entered before the position in the program where an error is likely to occur. If no ON ERROR GOTO has been executed in the procedure before an error occurs, the procedure will stop and enter the debug mode. The error trap can be turned OFF by executing ON ERROR without the GOTO.

This statement is often used in conjunction with the ERR function, which returns the specific error code, and the ERROR statement, which artificially generates errors. A procedure with an ON ERROR GOTO statement can be executed in either BASIC09 or RUNB. Both will allow the trapping of nonfatal run-time errors.

CAUTION

In BASIC09, the ON ERROR GOTO statement will not trap **Error #001 – Unconditional Abort**; **Error #002 – Keyboard Abort**; or **Error #003 – Keyboard Interrupt**. In RUNB the ON ERROR GOTO statement will trap all errors including Errors #001, #002, and #003.

EXAMPLES:

```
    REM This procedure prevents a fault on invalid input.
    ON ERROR GOTO 20
10  INPUT "ENTER A NUMBER: ",num$
    X=VAL(num$)
    PRINT X
    GOTO 10
20  PRINT "That was NOT a number!  TRY AGAIN"
    GOTO 10

    REM The following procedure will list a given file until it reaches
    REM the end-of-file marker or some other error occurs.
    DIM path, errnum:INTEGER; name$:STRING[45]; line:STRING[80]
    ON ERROR GOTO 10
    INPUT "File Name? ", name
    OPEN #path, line
    LOOP
        READ #path, line
        PRINT line
    ENDLOOP
10  errnum=ERR
    IF errnum=211 THEN                          (Error #211 - End-of- File)
        PRINT "Listing Complete."
        CLOSE #path
        PRINT "That was NOT a number!  TRY AGAIN"
        END
    ELSE                                        (other error codes)
        PRINT "Error Number: ";errnum
        END
    ENDIF
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CONDITIONALLY TRANSFER EXECUTION TO ANOTHER STATEMENT	STATEMENT
SYNTAX: ON <expr> GOSUB <#d> {, <#d>}	
ON <expr> GOSUB <#d> {, <#d>}	

These statements evaluate an integer expression and use the result to select a corresponding line number, <#d>, from the ordered list. Control is then transferred to that line number. In ON GOSUB statements, this transfer takes place as a subroutine. In ON GOTO statements, this transfer occurs unconditionally redirection of program flow.

The expression must evaluate to a positive INTEGER type result having a value between 1 and n, inclusive. n is equivalent to the number of line numbers in the list. If the evaluation has any other result, no line number is selected and the next sequential statement is executed. No error is generated in this case.

EXAMPLE:

```
    REM This procedure displays the written form of the digits 0 - 9
    DIM digit:INTEGER
    A$="One digit only please, 0 - 9"
    INPUT "Enter a single digit, 0 - 9: ", digit
    ON digit+1 GOSUB 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
    PRINT A$
    STOP
    REM The following are the ten subroutines that print the written
    REM of the integer value entered
10  A$="ZERO"
    RETURN
11  A$="ONE"
    RETURN
12  A$="TWO"
    RETURN
13  A$="THREE"
    RETURN
14  A$="FOUR"
    RETURN
15  A$="FIVE"
    RETURN
16  A$="SIX"
    RETURN
17  A$="SEVEN"
    RETURN
18  A$="EIGHT"
    RETURN
19  A$="NINE"
    RETURN
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OPEN

OPENS A CHANNEL TO AN EXISTING COPRAM FILE OR ASYNC PORT	STATEMENT
SYNTAX: OPEN <intvar>, <stringexpr> [:<accessmode>]	

This statement issues a request to OS-9 to open an I/O path to an existing COPRAM file or one of the MiniCOP's ASYNC Ports. The <stringexpr> is evaluated and passed to OS-9 as the descriptive pathlist. The variable name specified must be dimensioned as type INTEGER or BYTE and is used to receive the path number assigned to the path by OS-9. This path number is used to reference the specific file or device in subsequent I/O statements.

The OPEN statement may also specify the path's desired access mode which can be READ, WRITE, UPDATE, EXEC, or DIR. This defines the direction in which I/O transfers will occur. If no access mode is specified, UPDATE is assumed and both reading and writing are permitted. The DIR mode allows OS-9 directory-type files to be accessed but should not be used in combination with the WRITE or UPDATE modes. The EXEC mode causes the current execution directory to be used instead of the current data directory, which is the default setting.

EXAMPLES:

```
DIM P:BYTE
OPEN #P, "/T2":WRITE
PRINT #P, message$
CLOSE #P
```

```
DIM printer.path:BYTE; name:STRING[24]
name="/T3"
OPEN #printer.path,name:WRITE
PRINT #printer.path,"SYSTEM ON-LINE"
CLOSE #path
```

```
REM This procedure opens a path to a file called mailbox
REM and writes a message into it. The file is then closed.
DIM path:BYTE
OPEN #path, "mailbox":WRITE
message$="HELLO"
WRITE #path, message$
PRINT message$
CLOSE #path
```

```
REM This procedure opens a path to mailbox, reads a message
REM from it and closes the file.
DIM path:BYTE
OPEN #path, "mailbox":READ
READ #path, message$
PRINT message$
CLOSE #path
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PACK

COMPILES A PROCEDURE	COMMAND
SYNTAX: PACK [<procname>] {[<procname>]}[<pathlist>] PACK* [<pathlist>]	Command Mode: BASIC09

This command causes an extra compiler pass on the procedure or procedures specified. This extra compiler pass removes names, line numbers, nonexecutable statements, etc. The result is a smaller, faster procedure. PACKed procedures **cannot** be edited or debugged. PACKed procedures can only be executed. The packing procedure **destroys** the source code version of the procedure in the workspace. If a <pathlist> is not given, a copy of the packed procedure is automatically sent to COPRAM for storage in the current execution directory. If a <pathlist> is specified, a copy of the packed procedure will be redirected to that destination.

If no procedure is named the current working procedure will be packed. By including the asterisk (*) after the PACK command, all procedures in the workspace will be packed and saved under the <procname> specified. If multiple procedures are named in one PACK operation, all procedures listed will be packed and saved under the first <procname> given. In this case, the first <procname> can be a unique name not found in the workspace.

EXAMPLES:

B:PACK calculate	(packs the procedure **calculate** and sends a copy to COPRAM's execution directory).
B:PACK /D0/PACKDIR/calculate	(packs the procedure **calculate** and sends a copy to directory called **PACKDIR**).
B:PACK calculate >/T1	(packs the procedure **calculate** and sends a copy to the device at ASYNC Port 1, /T1).
B:PACK mapping collect calculate plot	(packs the procedures **collect, calculate, plot,** and **mapping** (if it exists) under the name **mapping** and sends a copy to COPRAM's execution directory).
B:PACK* initialize	(packs all procedures in the workspace and sends a copy to COPRAM's execution directory).
B:PACK calculate	(packs the procedure calculate and sends a copy to COPRAM's execution directory).

WARNING

Since the PACK command destroys the workspace (source code) version of the procedure it is imperative that it be saved to COPRAM and/or permanently stored at the Terminal Emulator first. A PACK procedure cannot be directly loaded in the BASIC09 workspace, and cannot be edited or debugged.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PARAM

DECLARES PASSING VARIABLES	STATEMENT
SYNTAX: PARAM <declseq> {; <declseq>}	Command Mode: BASIC09

PARAM is similar to the DIM statement, but it does not create variable storage. Instead, it describes what parameters the called procedure expects to receive from the calling procedure.

You as the programmer must ensure that the total size of each parameter, as evaluated by the RUN statement in the calling procedure, conforms to the amount of storage expected for each parameter as specified by the PARAM statement. BASIC09 checks the size of each parameter to prevent accidental access to storage, but does not check type. In most cases, you should ensure that the parameters evaluated in the RUN statement and sent to the called procedure agree with the PARAM statement specification with respect to: the number of parameters, their size, shape, and type.

A parameter passing operation can perform conversions of identically-sized data structures. For example a string of 80 characters can be passed to a procedure expecting a byte array of 80 elements. The receiving procedure assigns the numeric value of each character string to the corresponding element of the byte array.

EXAMPLES:

PARAM I:INTEGER	(a single parameter is expected from the calling procedure and will be named variable **I** and typed INTEGER).
PARAM X, Y:REAL;A, B:INTEGER	(four parameters are expected: the first two will be called variables **X** and **Y** and will be typed REAL; the second two will be called **A** and **B** and will be typed INTEGER).
PARAM pv, sp:REAL;char:STRING[10];port:BYTE	(four parameters are expected: the first two will be called variables **pv** and **sp** and will be typed REAL; the third will be called **char** and will be typed STRING[10]; the last will be called **port** and will be typed BYTE).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PAUSE

SUSPENDS EXECUTION OF A PROCEDURE	STATEMENT
SYNTAX: PAUSE [<outputlist>]	

The PAUSE command suspends execution of the current procedure and causes BASIC09 to enter the debug mode. When the PAUSE is encountered the message BREAK IN PROCEDURE <procname> will be displayed. If an <outputlist> is given, the PAUSE statement will also act as a PRINT statement and print the list.

The debug mode command CONT can be used to resume procedure execution at the first statement following the PAUSE.

In the following examples assume that the PAUSE statement as shown is executed in a procedure called **testproc**.

EXAMPLES:

PAUSE	(procedure execution is suspended and BASIC09 enters the debug mode).
D:BREAK PROCEDURE testproc	(name of procedure is displayed).
PAUSE "Now outside the Main Loop"	(procedure execution is suspended and BASIC09 enters the debug mode).
Now outside the Main Loop	(the **<outputlist>** is displayed).
D:BREAK PROCEDURE testproc	(name of procedure is displayed).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

POKE

STORES DATA AT A SPECIFIED ADDRESS	STATEMENT
SYNTAX: POKE <intrexpr>, <bytexpr>	

This statement allows a procedure to store data at a specific memory address. The <intexpr> is used as the absolute address to store the type BYTE result of the <bytexpr>. The PEEK statement can be used to read data from a specific address. There is no assurance that a value which has been POKEed will exist later to be PEEKed.

EXAMPLES:

POKE 1200, 14	(places the number **14** into address **1200**).
POKE $1C00, $FF	(places the hex number **FF** into hex address **1C00**).
POKE ADDR(buffer)+5, ASC("a")	(places the numeric value of **A** into the address of variable **buffer**).
POKE pointer, PEEK(pointer+1)	(shifts the contents of **pointer+1** to **pointer**).

WARNING

OS-9 reserves the right to use all memory at its discretion. Using the POKE statement can possibly cause unforeseen changes to memory and can easily cause a fatal system error. **USE WITH GREAT CARE!**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PRINT

TRANSFERS DATA TO AN OUTPUT DEVICE	COMMAND/STATEMENT
SYNTAX: PRINT <outputlist>	Command Modes: DEBUG, BASIC09
PRINT #<intexpr>, <outputlist>	

This statement sends the <outputlist> to the standard output path (path #1 display output) unless otherwise directed by the <intexpr>.

The output consists of one or more items separated by commas or semicolons. Each item can be a constant, variable, or expression of any atomic data type. The PRINT statement evaluates each item and converts the result to corresponding ASCII characters which are then displayed. If the item is followed by a semicolon, the next item will be displayed without any spacing between itself and the previous item. If a comma is used, the next item will be displayed beginning at the next tab zone. In BASIC09 tab zones are 16 characters in length. If the line of code is terminated with a semicolon with no <outputlist>, then the carriage return is inhibited for that line and the cursor remains in the position immediately following the previous <outputlist>.

PRINT can also be used in the debug mode as a command to display the current value of a numeric or string variable.

STATEMENT EXAMPLES:

PRINT value, temp+(n/2.5), location$	(displays the current value of **value**, the result of **temp(n/2.5)**, and the current text value of **location$**).
PRINT #printer.path "The final result = ";n	(sends the message **The final result** = and the value of **n** to the device at ASYNC port described in **printer.path**).
PRINT X;" divided by ";Y;" = ";X/Y	(displays the current values of **X** and **Y**; the messages **divided by** and =; and the result of **X/Y**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PRINT TAB

TRANSFERS DATA TO AN OUTPUT DEVICE	COMMAND/STATEMENT
SYNTAX: **PRINT TAB(<expr>) <outputlist>**	**Command Modes: DEBUG, BASIC09**
PRINT TAB(<expr>) #<intexpr>, <outputlist>	
PRINT TAB(<expr>) <outputlist>; TAB(<expr>); <outputlist>	
PRINT <outputlist>; TAB(<expr>); <outputlist>	

This statement tabs or moves the cursor to the column position specified in **<expr>** and then sends the <outputlist> to the standard output path (path #1 display output) unless otherwise directed by the <intexpr>.

PRINT TAB is identical to the PRINT statement in every way except for the addition of the cursor positioning control (tab) prior to printing.

The output consists of one or more items separated by commas or semicolons. Each item can be a constant, variable, or expression of any atomic data type. The PRINT statement evaluates each item and converts the result to corresponding ASCII characters, which are then displayed. If the item is followed by a semicolon, the next item will be displayed without any spacing between itself and the previous item. If a comma is used, the next item will be displayed beginning at the next tab zone. In BASIC09, tab zones are 16 characters in length. If the line of code is terminated with a semicolon with no <outputlist>, then the carriage return is inhibited for that line and the cursor remains in the position immediately following the previous <outputlist>.

The TAB(<expr>) function can be placed anywhere in the statement after the PRINT including within the <outputlist>. When encountered the next item in the <output> list begins printing in the specified column. If the output line has already written past that position, the TAB is ignored. A related function POS can be used in the procedure to determine the current cursor (or output) position at any given time. Output columns are numbered from 1 to 255.

STATEMENT EXAMPLES:

PRINT TAB(25) "Valve Position is: ";vp$	(displays the message **Valve Position is:** and the string **vp$** at position 25).
PRINT TAB(x) #printer.path "The final result = ";n	(sends the message **The final result =** and the value of **n** to the device at ASYNC port described in **printer.path**; printing will begin at the position in variable **x**).
PRINT "Index "; ind; TAB(15);"value";val	(displays the message **Index** followed by the value of **ind**; position is tabbed to 15 ; the message **value** and the value of **val** is displayed).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PRINT USING

PROVIDES A FORMATTED OUTPUT	COMMAND/STATEMENT
SYNTAX: PRINT [#<expr>] USING <stringexpr>, <outputlist>	Command Modes: DEBUG, BASIC09

The PRINT USING statement is an output editing tool used in report generation or other applications where a formatted output is required. The optional output path number expression, [#<expr>], can be used to specify the path number of any output file or device. If it is omitted the <outputlist> is sent to the standard output path (path #1 display output).

PRINT USING can also be used in the debug mode as a command to display the current value of a numeric or string variable.

The <stringexpr> is evaluated and used as a format specification containing formatting directives for each item in the <outputlist>. The items in the list may be constants, variables, or expressions of any atomic data type. Blank spaces are not allowed in the <stringexpr>. The type of each item in the <outputlist> must be compatible with the formatting specification. If there are fewer format specifications than items in the <outputlist>, the last format specification will be repeated as many times as necessary.

A format string, <stringexpr>, has one or more format specifications that are separated by commas. There are two kinds of specifications -- those that control the output editing of an item from the <outputlist> and those that cause an output function by themselves (such as tabbing and spacing). There are six fundamental output editing directives, each corresponding to a one letter identifier:

R = real format.
E = exponential format.
I = integer format.
H = hexadecimal format.
S = string format.
B = Boolean format.

The identifier is followed by a constant number called the field width. This number indicates the exact number of print columns the output is to occupy and must allow for the data and overhead character positions. Overhead characters are those items that are not directly part of the numeric data itself, but are used to describe or identify it. For example, signs (+/-), decimal points, exponents, etc. are all overhead characters. Some formats have additional mandatory or optional parameters that control subfields or selected editing options. One of these options is justification, which specifies whether the output is to line up on the left side, right side, or center of the output field. Fields are commonly right-justified in reports because it arranges them into neat columns with decimal points aligned to the same position. The abbreviations and symbols used in the syntax specification are:

w	=	total field width	(1 to 255 print columns, inclusive).
f	=	fractional field	(1 to 9 digits to the right, inclusive).
j	=	optional justification	(left, right, or center).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

REAL FORMAT
Syntax: Rw.fj

This format can be used for numbers of the types REAL, INTEGER, or BYTE. The w specification must include two overhead positions for the sign and the decimal point. The f specifies how many fractional digits to the right of the point are to be displayed. If the number has more significant digits than the field allows, the undisplayed places are used to round the displayed digits. For example, if the value 12.349 is to be printed using the format R8.2 (where R = REAL, 8 = field width, and 2 = digits to the right of the point), then the value 12.35 will be printed.

The j specifies the justification mode. Right, left, and center justification are allowed in the real format. The justification modes used by the real format are:

<	=	left-justify, with leading sign and trailing spaces*.
>	=	right-justify, with leading sign and spaces.
^	=	center-justify, with leading spaces and trailing sign**.

EXAMPLES:

```
X=678.123
PRINT USING "R8.2 <",X          (left-justifies and prints 678.12).

X=12.3
PRINT USING "R8.2 >",X          (right-justifies and prints 12.30).

X=-555.9
PRINT USING "R8.2 >",X          (right-justifies and prints -555.90).

X=6722.4599
PRINT USING "R10.3 ^",X         (center-justifies and prints 6722.460).


X=6722.4599
PRINT USING "R10.3 ^",X         (center-justifies and prints 6722.460).
```

* Defaults if justification mode is omitted.
** Financial format.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXPONENTIAL FORMAT
Syntax: Ew.ff

This format prints numbers of types REAL, INTEGER, or BYTE in the scientific notation format using a mantissa and decimal exponent. The w specification must include six overhead positions for the mantissa, sign, decimal point, and exponent characters. The f specifies how many fractional digits to the right of the point are to be displayed. If the number has more significant digits than the field allows, the undisplayed places are used to round the displayed digits. The j specifies the justification mode. The justification modes used by the exponential format are:

<	=	left-justify, with leading sign and trailing spaces*.
>	=	right-justify, with leading sign and spaces.

EXAMPLES:

```
X=1234.567
PRINT USING "E12.3 <",X
```
(left justifies and prints 1.235E+03).

```
X=47309.2
PRINT USING "R13.4 >",X
```
(right justifies and prints 4.7309E+04).

* Default if justification mode is omitted.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

INTEGER FORMAT
Syntax: Iw.ff

This format can be used for numbers of the types REAL, INTEGER, or BYTE. The w specification must include ONE overhead position for the sign. The j specifies the justification mode. The justification modes used by the integer format are:

<	=	LEFT JUSTIFY, with leading sign and trailing spaces*.
>	=	RIGHT JUSTIFY, with leading sign and spaces.
^	=	CENTER JUSTIFY, with leading sign zeros.

EXAMPLES:

```
X=10
PRINT USING "I4 <",X
```
(left-justifies and prints 10 with two trailing spaces).

```
X=10
PRINT USING "I4 >",X
```
(right-justifies and prints 10 with two leading spaces).

```
X=10
PRINT USING "I4 ^",X
```
(center-justifies and prints 0010).

* Default if justification mode is omitted.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

HEXADECIMAL FORMAT
Syntax: Hwj

This format can be used to display the internal binary representation of any data type using hexadecimal characters. The w specifies the number of hexadecimal characters to be output. The j specifies the justification mode. The justification modes used by the hexadecimal format are:

<	=	left-justify, with trailing spaces.
>	=	right-justify, with leading spaces.
^	=	center-justify*.

Since the number of bytes of memory used to represent different data types varies, use the following as a guide to specifying the field width (w):

BOOLEAN	=	H2	(one byte, two hex characters).
BYTE	=	H2	(one byte, two hex characters).
INTEGER	=	H4	(two bytes, four hex characters).
REAL	=	H10	(five byte, ten hex characters).
STRING	=	Hn*2	(n equals length of string in characters).

EXAMPLES:

```
X=100
PRINT USING "H4",X                    (center-justifies and prints 00C4).

X=-1
PRINT USING "H4 >",X                  (center-justifies and prints FFFF).

X$="ABCD"
PRINT USING "H8",X                    (center-justifies and prints 41424344).
```

* Default if justification mode is omitted.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

STRING FORMAT
Syntax: Swj

This format is used to display string data of any length. The **w** specifies the total field size. If the string to be displayed is shorter than the field size, it is padded with spaces according to the justification mode. If it is too long, it will be truncated on the right regardless of the justification mode. The justification modes used by the string format are:

<	=	left-justify, with trailing spaces*.
>	=	right-justify, with leading spaces.
^	=	center-justify, with leading and/or trailing spaces.

EXAMPLES:

```
X$="TEST RESULTS"
PRINT USING "S20 <",X$
```

(left-justifies and prints **TEST RESULTS** with six trailing spaces).

```
X$="TEST RESULTS"
PRINT USING "S20 >",X$
```

(right-justifies and prints **TEST RESULTS** with six leading spaces).

```
X$="TEST RESULTS"
PRINT USING "S20 ^",X$
```

(right-justifies and prints **TEST RESULTS** with three leading and three trailing spaces).

* Default if justification mode is omitted.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BOOLEAN FORMAT
Syntax: Bwj

This format is used to display Boolean data. The result of the Boolean expression is converted to the string as TRUE or FALSE. The **w** specifies the total field size. If the string to be displayed is shorter than the field size, it is padded with spaces according to the justification mode. If it is too long, it will be truncated on the right, regardless of justification mode. The justification modes used by the Boolean format are:

<	=	left-justify, with trailing spaces*.
>	=	right-justify, with leading spaces.
^	=	center-justify, with leading and/or trailing spaces

EXAMPLES:

DIM bool:BOOLEAN **bool=TRUE**	(each example use these two lines).
PRINT USING "S6 <",bool	(left-justifies and prints **TRUE** with two trailing spaces).
PRINT USING "S6 >",bool	(right-justifies and prints **TRUE** with two trailing spaces).
PRINT USING "S6 ^",bool	(center-justifies and prints **TRUE** with one leading and one trailing space).

* Default if justification mode is omitted.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CONTROL SPECIFICATIONS

Control specifications are useful for horizontal formatting an <outputlist>. They are not matched with items in the <outputlist> and can be used freely. The control formats are:

Tn	=	tab to column n.
Xn	=	space n columns.
'str'	=	include constant string. The string must not include single or double quotes, backslash, or carriage return characters.

EXAMPLE:

```
PRINT USING "'addr', X2, H4, X2, 'data', H2", 1000, 100
```

This example will result in the following output:

```
addr  03E8  data  64
```

In this example the two strings inside the single quotes (addr and data) are constant string control specifications. The two references to X2 are space control specifications (Xn). H4 and H2 direct that the constants 1000 and 100 be printed using the hexadecimal format.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

REPEATED GROUPS

Many times identical sequences of specifications are repeated in format lists. The repeated groups can be enclosed in parentheses and preceded by a repeat count. These repeat groups can also be nested.

EXAMPLES:

PRINT USING "2(X2,R10.5)"	(equivalent to: "X2, R10.5, X2, R10.5").
PRINT USING "2(I2,2(X1, S4))	(equivalent to: "I2, X1, S4, X1, S4,I 2, X1, S4, X1, S4").

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PUT (see GET/PUT)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Q

EXITS THE EDIT OR DEBUG MODE	COMMAND
SYNTAX: Q	Command Modes: DEBUG, EDIT

This command is used to exit either the edit or debug mode and return to the BASIC09 mode. Any open paths are closed when the Q command is issued.

STATEMENT EXAMPLES:

E:Q	(edit mode is exited and system returns to BASIC09 mode).
D:Q	(debug mode is exited and system returns to BASIC09 mode).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RAD (see DEG/RAD)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

READ (External File Operation)

READS INPUT DATA FROM A COPRAM FILE OR A MiniCOP PORT	STATEMENT
SYNTAX: READ #<intexpr>, <inputlist>	

This statement causes input data, in ASCII character format, to be read from a COPRAM file or port. The <intexpr> specifies a path number which must have been previously opened by an OPEN or CREATE statement in either the READ or UPDATE access modes. The standard input path (#0 - Keyboard) is the only exception to this rule. Data is read starting at the path's current file pointer address, which is updated as the data is read.

This statement calls OS-9 to read a variable length ASCII record. Individual data items within the record are converted to BASIC09's internal binary format. These results are assigned in order to the variables given in the <inputlist>. The input data must match the number and type of variables in the <inputlist>.

The input data items in the input record are separated by ASCII null characters. Numeric items can also be delimited by commas or space characters. The input record is terminated by a [RETURN] character.

EXAMPLES:

READ #path,text$	(reads a line of text from the file associated with path and places it into **text$**).
READ #port,text$	(reads a line of text from the device associated with path and places it into **text$**).
READ #mail,message$	(reads a line of text from the file or device associated with mail and places it into **message$**).
READ #mail,message1, message2, message3	(reads three items from the file or device associated with mail and places it into **text$**).

```
REM This procedure opens a path to a COPRAM file called
REM 'mailbox' in the current data directory and reads a
REM message.  The file is then closed.
OPEN #path, "mailbox":READ
READ #path,message$
PRINT message$
CLOSE #path
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

READ (Internal Constant Operation)

RECEIVES INPUT FROM A DATA STATEMENT	STATEMENT
SYNTAX: READ <inputlist>	

This statement is used to read lists of expressions in the procedure. The READ statement has a list of one or more variable names. When executed, it gets "input" by evaluating the current expression in the current DATA statement. The result must match the type of the variable.

EXAMPLES:

```
REM These two statements will assign the values 1.0, 2.0, 3.0, and 4.0
REM to variables w, x, y, and z respectively
DATA a, b, c:BOOLEAN
READ w, x, y, z


REM These lines assign BOOLEAN values to a, b, and c.
DIM a, b, c:BOOLEAN
DATA TRUE,FALSE,TRUE
READ a, b, c


REM These initialize i, x, and char$ to values corresponding
REM to their respective data types.
DIM i:INTEGER; x:REAL; char$:STRING[10]
DATA 22, 2.2,"TWO"
READ i, x, char$
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

REM

APPLIES NON-EXECUTABLE COMMENTS TO PROCEDURES	STATEMENT
SYNTAX: REM <comment> ! <comment> \ <comment>	

The REM or remark statement is used to attach comments to a procedure. These comments are typically used to describe the operation of the procedure or a specific portion. They may also include items such as the programmer's name, project number, date created, etc. They can be placed practically anywhere in a procedure.

The first two forms of the statement, REM and !, are used to enter comments on separate lines. When each line is compiled into I-code during entry, the REM statement is actually converted to !.

Later, when the procedure is listed, the ! is decompiled back to REM. If the ! is used when entering a comment it cannot be compiled any further, but when the procedure is later listed it will be decompiled into REM for display.

The last form of the statement, \, is used to enter a comment to the end of a line of code. Spaces are not accepted in this form. To enter multiple word comments without running them all together, use a character such as the underline (_) to create easy-to-read text.

Comments are retained in source code versions of BASIC09 procedures, but are among the nonexecutable items removed by the PACK command.

EXAMPLES:

```
REM This is an example of a comment.  It can be viewed when the procedure is LISTed
REM to the display or to a printer.  It is ignored during normal execution.

! This is another form of the 'REM' statement.  It has all the same characteristics and
! uses as the 'REM'.

10  PRINT  "HELLO"    \This_is_the_final_form_of_the_'REM'_statement.
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RENAME

CHANGE A PROCEDURES NAME	COMMAND
SYNTAX: RENAME <procname1> <procname2>	Command Mode: BASIC09

This command is used to change the old name of <procname1> to a new name, <procname2>. It can be used to allow two copies of the same procedure to exist in the workspace under different names.

EXAMPLE:

B:RENAME testproc1 alarm (changes **testproc1** to **alarm**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

REPEAT/UNTIL

REPEATS A LOOP UNTIL A TRUE RESULT	STATEMENT
SYNTAX: REPEAT UNTIL <boolexpr>	

The REPEAT statement marks the start and the recycle point in the loop. The statement or statements within the loop are executed UNTIL the result of the <boolexpr> is TRUE. This is a loop that has its test at the end of the loop cycle. The body of this loop is always executed a minimum of one time.

EXAMPLES:

```
REPEAT
    INPUT "ENTER ACCESS CODE: ", code
UNTIL code=12345
PRINT "ACCESS GRANTED"


X=0
REPEAT
    PRINT X
    X=X+1
UNTIL X>10
PRINT "REPEAT/UNTIL LOOP ENDED"


REM Compute Factorial:n!
DIM temp, n:INTEGER
temp=1
INPUT "Factorial what number? ",n
REPEAT
    temp=temp*n
    n=n-1
UNTIL n<=1.0
PRINT 'The factorial is ";temp
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RESTORE

SETS NEXT DATA STATEMENT TO BE READ	STATEMENT
SYNTAX: RESTORE [<#d>]	

This command is used with other internal data statements, like READ and DATA, to indicate the next DATA statement to be read. If used without a line number (<#d>), it causes the first DATA statement in the procedure to be read next. Used with a line number (<#d>), the DATA statement at that line is read next.

EXAMPLES:

```
      DATA 1, 2, 3, 4
10    DATA 5, 6, 7, 8
      FOR N=1 TO X
          READ ARRAY(N)
      NEXT N
      RESTORE 10                    (line 10 is the next DATA statement read).
      READ A, B, C, D               (reads 5, 6, 7, 8 and assigns those
                                    values to A, B, C, D respectively).


      DATA 1, 2, 3, 4
      DATA 5, 6, 7, 8
      READ E, F, G, H
      RESTORE
      READ A, B, C, D               (reads 1, 2, 3, 4 and assigns those
                                    values to A, B, C, D respectively).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RETURN (see GOSUB/RETURN)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RUN

SETS NEXT DATA STATEMENT TO BE READ	COMMAND/STATEMENT
SYNTAX: RUN <procname> [(<parmpass>{,<parmpass>})]	Command Mode: BASIC09

RUN can be used as both a BASIC09 mode command or as a statement within a procedure. In either case, RUN has similar results and carries the same syntax.

RUN Command

The RUN command is used to execute procedures while in the BASIC09 mode. Technically, the MiniCOP leaves this mode and enters the execution mode. Procedures named in the <procname> can be of either the source or packed format. If the <procname> is not found inside the workspace, BASIC09 will search MiniCOP's memory by way of the following steps:

1. Main memory's module directory.
2. COPRAM's current execution directory.
3. COPRAM's current data directory.

If the <procname> is not found within the workspace, BASIC09 calls OS-9 to attempt to link a module in the module directory using the same name. If the <procname> is not found in the module directory, OS-9 then attempts to load a file of the same name from the current execution directory. If the <procname> is not found in the execution directory, OS-9 then attempts to load a file of the same name from the current data directory.

Parameters can be passed to the RUN procedure, but they must be constants or expressions without variables. They must also be expected, by the procedure which uses the PARAM statement, to identify and sign the parameters to variable names.

EXAMPLES:

B:RUN mastermix (runs procedure **mastermix**).

B:RUN add(4000,5) (runs and passes parameters to **add**).

B:RUN plot (runs and passes parameters to **plot**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RUN Statement

The RUN statement is used within a procedure to call (RUN) another procedure. When the RUN statement is executed, the calling procedure temporarily terminates while the called procedure is executed. When the called procedure ends, control is passed back to the calling procedure at the next statement following RUN. The RUN statement can call any procedure currently in the workspace. If the <procname> is not found within the workspace, an **Error #043 - Unknown Procedure** is generated.

Parameters can be passed to the RUN procedure by reference or value. The parameters included with a RUN statement may be constants or variables. They must also be expected, by the procedure which uses the PARAM statement, to identify and sign the parameters to variable names.

EXAMPLES:

```
IF x>100 THEN
    RUN overflow                    (calls a procedure called overflow).
ENDIF


FOR X=1 TO 10
    RUN profile(3,5)                (calls and passes two parameters to profile).
NEXT X


Y=2+X*28
RUN test(15, Y)                     (calls and passes two parameters to test).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SAVE

WRITE A PROCEDURE TO COPRAM OR MiniCOP Port	COMMAND
SYNTAX: SAVE [<procname> {,<procname>}, [><pathlist>] SAVE* [<pathlist>]	Command Mode: BASIC09

The SAVE command is used to write source code procedure, <procname>, COPRAM's current data directory, or a device at one of the MiniCOP's ASYNC ports. This command is similar to the LIST command except the output is not formatted and I-code addresses are not included. The <pathlist> is not used to specify the port through which to SAVE to an external device.

Multiple procedures can be saved by including the <procname> of each (separated by commas) in the command line. By including an asterisk (*) after the SAVE command all procedures in the workspace are saved. Additionally, all procedures can be saved under one name by including both an asterisk (*) and a <procname> after the SAVE command.

If a file of the same name already exists in the specified destination, BASIC09 will issue the prompt: **rewrite?** Answer Y, for yes, to continue with the operation. The existing file will be overwritten with the new procedure. Any other response will abort the SAVE operation and cancel the command.

EXAMPLES:

B:SAVE interlock	(saves **flow.control** to the current data directory).
B:SAVE pumpsta, valvepos	(saves **pumpsta** and **valvepos** to the current data directory).
B:SAVE*	(saves all workspace procedures to the current data directory).
B:SAVE* flow.control	(saves all workspace procedures to the current data directory under a procedure called **flow.control**).
B:SAVE gateway >/T2	(saves **gateway** to the device at ASYNC Port 2, **/T2**).

NOTE

This command should not be confused with the OS-9 SAVE command. Although the two seem similar, they have completely different meanings.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SEEK

CHANGES COPRAM FILE POINTER	STATEMENT
SYNTAX: SEEK #<intrexpr>, <realexpr>	MODE: BASIC09

The SEEK command changes the COPRAM file pointer address, which is the address of the next data byte or bytes to be read or written. This statement is essential for random access of file data using the GET and PUT statements.

The <intrexpr> specifies the path number of the file and must evaluate to a BYTE value. The second expression specifies the desired file pointer address and must evaluate to a REAL value in the range of 0 to 2,147,483,648, inclusive. Any fractional part of the result is truncated. The actual maximum file size depends on the capacity of the device.

Although SEEK is normally used with random-access files, it can be use to "rewind" sequential files.

For example:

SEEK#path, 0

This statement is the same as a rewind or restore function. This is the only form of the SEEK statement that is generally useful for files accessed by READ and WRITE statements. These statements use variable-length records, so it is difficult to know the address of any particular record in the file. The first byte of any record is always 0. Therefore if the 10th byte is to be located, a SEEK of 9 would be required.

The SEEK statement is also a quick and easy way to create an empty file of a predetermined size. To achieve this a new file must first be CREATEed in COPRAM, then a SEEK for the byte equivalent to the upper limit of the six is performed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXAMPLES:

```
SEEK #path, 0
```
(set the file pointer to the beginning of the file).

```
SEEK #path, 999
```
(set the pointer to the 1000th byte).

```
REM This line will set the file position pointer to
REM the 11th user-defined "batchstructure" in the
REM file associated with "data.file".
SEEK #data.file, 10*SIZE(batchstructure)
```

```
REM This procedure sets the file position pointer to
REM the 100th integer in the file 'processdata' and
REM GETs the next integer into a temporary integer
REM variable, 'temp'.
DIM temp:INTEGER
DIM data.path:BYTE
OPEN #data.path,"processdata":UPDATE
SEEK #data.path, 2*99
GET #data.path, temp
PRINT temp
CLOSE #data.path
```

```
CREATE #path, "TEST"
SEEK #path, 499
```
(creates an empty file **TEST**).
(sizes **TEST** at 500 bytes).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SHELL

GIVES BASIC09 PROCEDURES ACCESS TO OS-9 COMMANDS	STATEMENT
SYNTAX: SHELL <stringexpr>	

This statement allows BASIC09 procedures to execute any OS-9 commands or programs. This gives access to virtually any OS-9 function including multitasking, utility commands, terminal and I/O control, etc.

The SHELL statement requests that OS-9 create a new process, initially executing the SHELL, which is the OS-9 command interpreter. The SHELL can then call any procedure in the system, subject only to the normal security functions. The <stringexpr> is evaluated and passed to the SHELL to be executed as if it had been entered on the SHELL's OS-9 command line. If <stringexpr> is null, BASIC09 is temporarily suspended and the SHELL process displays prompts and accepts commands in its normal manner. When the SHELL process terminates, BASIC09 becomes active again and resumes execution at the statement immediately following the SHELL statement. However, if a built-in SHELL command is executed, the system does not return to the calling procedure.

EXAMPLES:

SHELL "COPY /D0/TEMPfile01 /D0/DATA/file01"	(copies **file01** from the **TEMP** directory to the **DATA** directory; sequential execution).
SHELL "COPY /D0/TEMPfile01 /D0/DATA/file01 &"	(copies **file01** from the **TEMP** directory to the **DATA** directory; concurrent execution).
N=5 **SHELL "KILL " +STR$(N)**	(kills process 5).
file$="/D0/batch.jobs"** **SHELL file$+"-p &"	(executes OS-9 file **batch.jobs** while inhibiting OS-9 prompts (**-P**); concurrent execution).
SHELL "TMODE upc"	(standard output mode set to uppercase characters only).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

STATE

LISTS PROCEDURE NESTING ORDER	COMMAND
SYNTAX: STATE	Command Mode: DEBUG

This command, available only in the debug mode, lists the calling (nesting) order of all active procedures. The highest-level procedure will always be the suspended procedure.

EXAMPLES:

```
D:STATE                    (requests nesting order of active procedures).
PROCEDURE delta            (suspended procedure).
called by gamma            (parent of delta, child of beta).
called by beta             (parent of gamma, child of alpha).
called by alpha            (parent of beta).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

STEP

STEPS THROUGH A SUSPENDED PROCEDURE	COMMAND
SYNTAX: STEP [<#d>]	Command Mode: DEBUG

This command, available only in the debug mode, allows the suspended procedure to be executed by one or more source statements at a time. [<#d>] indicates the number of source statements to be executed. Entering a [RETURN] at the D: prompt is equivalent to a STEP 1. The step command is most commonly used with the trace mode turned on (i.e., TRON). This displays the statement and the result of any expressions as they are executed.

Since compiled I-code contains each statement's actual memory addresses, the start and ending statements of loop structures are usually executed only once. For example in a FOR/NEXT loop the FOR statement is executed once when the loop is entered. The statement immediately following the FOR will appear to be the start of the loop when STEPing.

EXAMPLES:

D:STEP 10	(executes the next 10 statements).
D:STEP 100	(executes the next 100 statements).
D:[RETURN]	(executes the next statement; acts as a **STEP 1**).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

STOP

TERMINATES PROCEDURE EXECUTION	STATEMENT
SYNTAX: STOP [<outputlist>]	

This statement immediately terminates execution of all procedures and returns to the BASIC09 mode. If the procedure, which includes the STOP statement, is a child (called by another procedure), it and its parent will be terminated.

When the STOP is encountered, the [<outputlist>], if included, will be displayed. The message **"STOP Encountered"** will also be displayed prior to termination.

EXAMPLE:

```
STOP "The procedure STOPS here!"          (typical STOP statement in a procedure).

The procedure STOPS here!                 (result of [<outputlist>]).
STOP Encountered                          (standard STOP statement message).
Ready
B:                                        (returned to BASIC09 mode).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

THEN (see IF/THEN)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TRON/TROFF

TURNS DEBUGS TRACE MODE ON AND OFF	COMMAND/STATEMENT
SYNTAX: TRON TROFF	Command Mode: DEBUG

These statements or commands turn debug's trace mode function ON (TRON) and OFF (TROFF). When trace mode is turned ON, each statement is decompiled and printed before execution. The result of each evaluation encountered will be displayed as it occurs. The statement can be inserted in the BASIC09 to automatically turn the trace mode function ON when the procedure enters debug. Typically, this would be inserted into a procedure during development and can be removed after successful testing. The command can be used in debug to manually turn the trace mode ON and OFF.

STATEMENT EXAMPLE:

```
X=200
PAUSE                    (forces the procedure to enter debug mode).
TRON                     (automatically turns the trace mode ON).
FOR L= 1 TO X
    PRINT L
NEXT L
```

COMMENT EXAMPLE:

```
D:TRON                   (turns the trace mode ON).

D:TROFF                  (turns the trace mode OFF).
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TYPE

DEFINES NEW DATA TYPES	STATEMENT
SYNTAX: TYPE <typname>	

This statement is used to define new data types. New data types are defined as a one-dimensional array of previously-defined types. This structure differs from an array in that the various elements may be of different types and are accessed by field names instead of an array index.

Since the TYPE statement can include previously-defined types, very complex nonregular data structures can be created such as lists, trees, etc. This statement does not create any variable storage itself; the storage is created when the newly-defined type is used in a DIM statement. Elements can be accessed by index assignment or by field name.

The complex data structure allows the creation of data types appropriate to the job at hand by providing more natural organization and association of data. Unlike arrays, the position of the desired element is know and defined at compilation time and need not be calculated at run-time. Therefore, complex data structures can be accessed faster.

The total size of each record can be calculated by manually counting the number of bytes assigned for each type. Sometimes this can become a complicated, error-prone task, and any change in the TYPE definition could require recalculation. The built-in BASIC09 function, SIZE(<name>), will return the number of bytes assigned to any variable, array, or complex data structure thereby avoiding these problems. The SIZE function is often used in conjunction with the SEEK statement to position a file pointer to a specific records address.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

EXAMPLE:

The following example is based on a simple inventory system that requires a random access file having 100 records. Each record must include: the name of the item (25-byte STRING); the item's list price and cost (both typed REAL); and the quantity on hand (INTEGER). The TYPE statement describes a new data structure type called INVITEM, but does not cause variable storage to be assigned for it. Next two data structures are created -- an array of 100 records called inv_array and a single working record called work_rec. Both are dimensioned as type INVITEM.

```
REM The following procedure creates a file called "inventory"
REM and initializes it with zeros and nulls
TYPE INVITEM=name,:STRING[25]; list, cost:REAL;qty:INTEGER
DIM inv_array(100):INVITEM
DIM work-rec:INVITEM
DIM path:BYTE
work_rec.name=""
work_rec.list=0.
work_rec.cost=0.
work_rec.qty=0
FOR n= 1 TO 100
    PUT #path, work_rec
NEXT n
END
```

Notice that the assignment statement references each named field of work_rec by name, but the PUT statement references the record as a whole. The following routine could be a part of a data entry procedure used with the previous example. It asks for a record number, then asks for data to be entered. The data is written into the file at the specified record location.

```
INPUT "Record Number?",recnum
INPUT "Item Name?",work_rec.name
INPUT "List Price?",work_rec.price
INPUT "Cost?",work_rec.cost
INPUT "Quantity?",work_rec.qty
SEEK #path, (recnum-1)*SIZE(work-rec)
PUT #path, work_rec
```

The following routine could be a part of the same data entry procedure, that is, executed once all data entries are made. It will read the entire file into the array **inv_array**.

```
SEEK #path,0
FOR K=1 TO 100
    GET #path,inv_array(K)
NEXT K
```

Because data structures can be read , the FOR/NEXT loop can be eliminated and replaced by the following routine, which is smaller and faster.

```
SEEK #path,0
GET #path,inv_array
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

UNTIL (see REPEAT/UNTIL)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WHILE/DO

EXECUTES A CONDITIONAL LOOP	STATEMENT
SYNTAX: WHILE <boolexpr> DO until	

This statement is a loop constructed with the test at the top of the loop. The body of the loop will not be executed if the <boolexpr> is FALSE when the WHILE is first executed. In this case, execution will be passed to the statement immediately following the ENDWHILE. If the WHILE is TRUE, the loop will be entered and the statements within the loop will be executed as long as the <boolexpr> remains TRUE. If the WHILE becomes FALSE while the loop is executing, the loop will terminate the next time WHILE is executed. Execution will resume at the statement immediately following the ENDWHILE.

EXAMPLES:

```
WHILE a<b DO
    PRINT a
    a=a+1
ENDWHILE


WHILE count <= maximum DO
    RUN test
    count=count+1
ENDWHILE


REM Print the characters in word$ in reverse order of their entry
backward$=""
INPUT "Enter a string of up to 100 characters:",word$
While LEN(word$)>0 DO
    backward$=backward$+RIGHT$(word$,1)
    word$=LEFT$(word$, LEN(word$)-1)
ENDWHILE
word$=backward$
PRINT "Your entry in reverse order is:",word$


PAUSE                    (forces the procedure to enter debug mode).
TRON                     (automatically turns the trace mode ON).
FOR L= 1 TO X
    PRINT L
NEXT L
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

WRITE

WRITES ASCII DATA INTO A COPRAM FILE OR OUTPUT DEVICE **STATEMENT**
SYNTAX: WRITE <intrexpr>, <outputlist>

This statement writes data in ASCII character format into a COPRAM file. The <intrexpr> specifies the number of a path that was previously opened by an OPEN or CREATE statement in either WRITE or UPDATE modes. The <outputlist> consists of one or more expressions separated by commas. Each expression can evaluate to any expression type. The result is then converted to an ASCII character string and written on the specified path beginning at the present file pointer, which is updated as data is written.

If the <outputlist> contains more than one item, ASCII null characters($00) are written between each output string. The last item is followed by a carriage return character. This statement creates variable-length ASCII records.

EXAMPLES:

WRITE #path, text$	(writes the contents of text$ to the file or device associated with the path).
WRITE #port, text$	(writes the contents of text$ to the designated port).
WRITE #1, text$	(writes the contents of text$ to the standard output path display).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 TUTORIAL

INTRODUCTION

This tutorial is a step-by-step exploration of the basic operating characteristics and functionality of the BASIC09 programming language. It is intended for use with a personal computer running the Terminal Emulator software.

This tutorial includes hands-on instruction on how to:

- Initialize a BASIC09 workspace.
- Assign and change the workspace memory allocation.
- Create and edit BASIC09 procedures.
- Save and pack BASIC09 procedures.
- Delete and load BASIC09 procedures.

TUTORIAL REQUIREMENTS

SOFTWARE

To perform all of the operations in this tutorial, the following software is required:

- Terminal Emulator software (Model No. 626-6041)*, and
- OS-9 System Commands disk (Model No. 627-0006).

HARDWARE

To perform all of the operations in this tutorial, the following hardware is required:

- MS-DOS-based personal computer -or- 623 Loader/Terminal (Model No. 623-6100/-6150),
- 627 MiniCOP (Model No. 627-1002/-1003),
- 621 I/O rack**,
- 621 Power Supply Module***, and
- Serial cables as required per personal computer -or- 623 Loader/Terminal.

---

* The Terminal Emulator software can be purchased separately using Model No. 626-6041. It is also included with the 623 MS-DOS Loader software and is factory installed on both models of the 623-61 Loader/Terminal (623-6100, 623-6150).

** Any 621 I/O (standard, augmented, full-size, or half-size) is acceptable. A 620 processor rack with I/O slots is also acceptable and the processor module is not required (the MiniCOP uses PUSH/PULL instructions to communicate to a host processor via the I/O bus and is not typically installed in the 620-25 processor rack).

*** Use a 620 power supply module if using a processor rack (processor half-size racks use 621 power supplies).

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DOCUMENTATION

The following documentation is also required:

- *620 Installation and 621 I/O Communications User Manual* (Form 620-8986).
- *620 Installation and 621 I/O Communications for 620-12/1633/36 Logic Controller Systems User Manual* (Form 620-8986),
- *623 Terminal Emulator User Manual* (Form 623-8989), and
- *627 MiniCOP and BASIC09 User Manual* (Form 627-8991).

BASIC09 TUTORIAL PRELIMINARIES

In order to exercise the BASIC09 tutorial, it will be necessary to set-up and initialize a 627 MiniCOP system. Refer to *Hardware Configuration/Installation* and *MiniCOP Initialization* under *MiniCOP Set-up Tutorial* in the *OS-9 Operating System* section of this manual and perform all the required steps (1through 16) before proceeding to the following procedures.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BASIC09 FUNDAMENTAL OPERATIONS TUTORIAL

INITIALIZING BASIC09

In this part of the tutorial you will call BASIC09 from the OS9: prompt and change and define the workspace memory size.

1. From the OS9: prompt, call BASIC09 and accept the 2Kbyte workspace default size.

OS9:BASIC09 [RETURN]

2. Display the contents of the BASIC09 workspace. Observe the free memory indication.

B:DIR [RETURN]

Display the amount of memory assigned to the workspace.

B:MEM [RETURN]

Compare the amount of memory assigned to the workspace with the amount free.

3. Exit BASIC09 and return to OS-9.

B:BYE [RETURN]

4. Call BASIC09 and predefine the workspace size as sixteen 256-byte pages of memory.

B:BASIC09 #16 [RETURN]

5. Display the contents of the BASIC09 workspace and observe the free memory indication. Then display the amount of memory assigned to the workspace.

B:DIR [RETURN]

B:MEM [RETURN]

6. Exit BASIC09 and return to OS-9.

B:BYE [RETURN]

7. Call BASIC09 and predefine the workspace size as 16Kbytes of memory.

B:BASIC09 #16K [RETURN]

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8. Display the contents of the BASIC09 workspace and observe the free memory indication. Then display the amount of memory assigned to the workspace.

B:DIR [RETURN]

B:MEM [RETURN]

9. Decrease the size of the workspace to 4Kbytes in decimal terms. Observe the amount of memory assigned to the workspace and then display the available memory.

B:MEM 4000 [RETURN]

B:DIR [RETURN]

10. Increase the size of the workspace to 16Kbytes in hexadecimal terms. Observe the amount of memory assigned to the workspace and then display the available memory.

B:MEM $4000 [RETURN]

B:DIR [RETURN]

In each of the previous cases the free memory displayed by the DIR command (or by pressing E) and amount of memory actually assigned to the workspace differs. This is due to each BASIC09 reserving approximately 1.2Kbytes of each workspace for its own internal use.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

11. Call the edit mode and create a BASIC09 procedure.

```
B:E [RETURN]
PROCEDURE PROGRAM
*
E:
```

Calling the edit mode when no procedures exist in the workspace will create a procedure under the default name of **PROGRAM**. If procedures do exist in the workspace and one is identified as the current procedure (asterisk "*") it will be entered.

12. Quit the edit mode and return to the BASIC09 mode. Then display the contents of the workspace.

```
E:Q [RETURN]

B:DIR [RETURN]
```

A procedure called **PROGRAM** now resides in the workspace. Notice that the proc-size and data-size are also displayed. These represent the size of the procedure and the amount of variable storage respectively.

13. Create a procedure called **firstproc**:

```
B:E firstproc [RETURN]
PROCEDURE firstproc
*
E:
```

Enter the following lines of BASIC09 source code as they appear:

```
E:10 C = 0 [RETURN]
E:20 INPUT "Please Enter Your First Name: ";N$ [RETURN]
E:30 C = C + 1 [RETURN]
E:40 FOR C = 1 to 5 [RETURN]
E:50 PRINT  "This is ";N$;"'s first BASIC09 procedure."
[RETURN]
E:60 NEXT C [RETURN]
E:70 END [RETURN]
```

List the current procedure **firstproc** in the edit mode:

```
E:L* [RETURN]
```

Notice the asterisk (*) to the far left of the listing, this is known as the edit pointer and represents the current line in the procedure. It is used as a reference point for deleting, adding or changing lines of source code. The four digit (hexadecimal) value that proceeds each line is the byte address of the first statement in each line.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

14. Quit the edit mode and return to the BASIC09 mode. Then display the contents of the workspace:

E:Q [RETURN]

B:DIR [RETURN]

Two procedures, PROGRAM and firstproc, now reside in the workspace. In this case, the asterisk (*), represents the current workspace procedure. If a command is given that will affect a procedure in the workspace and its name is not specified, the current procedure is assumed (default).

15. List the current workspace procedure:

B:LIST [RETURN]

List the specific procedure **firstproc**:

B:LIST firstproc [RETURN]

In both cases, the procedure called **firstproc** was listed or displayed.

16. Run the current workspace procedure:

B:RUN [RETURN]

List the specific procedure **firstproc**:

B:RUN firstproc [RETURN]

In both cases, the procedure called **firstproc** was run or executed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

17. Create a new procedure called **secondproc**:

```
B:E secondproc [RETURN]
PROCEDURE secondproc
*
E:
```

Line numbers are optional in BASIC09. Normally they are not used since they are not required and use valuable memory. In some cases line numbers are useful. They can typically be found with the GOTO and GOSUB statements as well as some looping routines. If line numbers are not used the first character entered on each line must be a [SPACE].

Enter the following lines of BASIC09 source code as they appear:

```
E:[SPACE] C = 0 [RETURN]
E:[SPACE] INPUT "Please Enter Your Last Name: ";N$ [RETURN]
E:[SPACE] C = C + 1 [RETURN]
E:[SPACE] FOR C = 1 to 5 [RETURN]
E:[SPACE] PRINT  "This is ";N$;"'s second BASIC09 procedure"
[RETURN]
E:[SPACE] NEXT C [RETURN]
E:[SPACE] END [RETURN]
```

List the current procedure, **secondproc**, in the edit mode:

```
E:L* [RETURN]
```

18. Quit the edit mode and return to BASIC09 mode, then display the contents of the workspace:

```
E:Q [RETURN]
B:DIR [RETURN]
```

Three procedures, **PROGRAM**, **firstproc** and **secondproc**, now resides in the workspace. In this case, the asterisk (*) represents the current workspace procedure. If a command is given that will affect a procedure in the workspace and its name is not specified, the current procedure is assumed (default).

19. List the current workspace procedure:

```
B:LIST [RETURN]
```

List the specific procedure **secondproc**:

```
B:LIST secondproc [RETURN]
```

In both cases, the procedure called **secondproc** was listed or displayed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

20. Run the current workspace procedure:

B:RUN [RETURN]

List the specific procedure **secondproc**:

B:RUN secondproc [RETURN]

In both cases, the procedure called **secondproc** was run or executed.

21. List and run the noncurrent procedure **firstproc**:

B:LIST firstproc [RETURN]

B:DIR [RETURN]

B:RUN [RETURN]

When **firstproc** was listed it became the current procedure so that is was not necessary to specify its name with the RUN command.

22. Moving the edit pointer.

B:E [RETURN]

E:L* [RETURN]

The E command will call the current workspace procedure, if one exits, into the edit mode. The asterisk (*) that appears in the edit mode is known as the edit pointer or pointer. It is used as a reference point when editing a procedure.

Move the edit pointer to the end of the procedure:

E:+* [RETURN]

E:L* [RETURN]

Move the edit pointer to the beginning of the procedure:

E:-* [RETURN]

E:L* [RETURN]

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Move the edit pointer forward one line at a time:

E:+1 [RETURN]

E:L* [RETURN]

E: [RETURN]

E:L* [RETURN]

Notice that in the edit mode the [RETURN] key has the same effect on the edit pointer as **+1**.

Move the edit pointer forward multiple lines at a time:

E:+3 [RETURN]

E:L* [RETURN]

Move the edit pointer backwards one line at a time:

E:-1 [RETURN]

E:L* [RETURN]

E: -1 [RETURN]

E:L* [RETURN]

Move the edit pointer backwards multiple lines at a time:

E:-3 [RETURN]

E:L* [RETURN]

Any number of lines can be specified when moving the edit pointer. However, if the number is greater than the number of lines in either direction, the edit pointer will stop at the beginning or end of the procedure.

Attempt to move the edit pointer by more lines than exist in the procedure from the current pointer position.

E:+50 [RETURN]

E:L* [RETURN]

E:-100 [RETURN]

E:L* [RETURN]

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

23. From the current edit pointer position, renumber the lines to begin with 100 and increase in increments of 10.

E:R 100 10 [RETURN]
E:L* [RETURN]

24. Move the edit pointer to the third line in the procedure (line number 120) and renumber from that point starting with 1500 in increments of 20.

E:R 1500 20 [RETURN]
E:L* [RETURN]

Only numbered lines will be affected by the renumber command (R). Line that do not contain line numbers remain as they were prior to this operation. When a line number has been used with a BASIC09 statement (such as GOTO or GOSUB) it is possible that the renumber command has changed the line number of its destination. BASIC09 will automatically update these line number references also.

25. When a line of source code is assigned a line number, it is possible to move the edit pointer to that line by simply specifying its line number at the E: prompt.

E:1560 [RETURN]

E:L* [RETURN]

26. Change an existing line of source code:

To change an existing line of source code use the CHANGE command (C). In line 1540 of the procedure **firstproc**, you will change the text to include the word **edited**. CHANGE is a formatted command where the current text is specified first, and then the new text is specified second. In line 1540, insert the word **edited** between **first** and BASIC09.

E:1540 [RETURN]

E:C/first/first edited [RETURN]

The word **first** was specified in both the old and new text so that **edited** was correctly placed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

27. Add a line to an existing procedure:

A line or lines of source code can be inserted into a procedure at any point. The new line is always inserted before the line marked by the current edit pointer position.

E:L* [RETURN]

E:[SPACE] PRINT "This is a new line!" [RETURN]

E:L* [RETURN]

Notice that the new line appears immediately prior to the line which was marked as the current line by the edit pointer.

E:-* [RETURN]

E:[SPACE] REM This procedure is part of a practice exercise [RETURN]

E:L* [RETURN]

E:Q* [RETURN]

B: [RETURN]

28. Delete the current line as indicated by the edit pointer using the DELETE command (D).

E:L* [RETURN]

E:D [RETURN]

E:L* [RETURN]

29. Delete all lines from the current edit pointer position to the end of the procedure.

E:L* [RETURN]

E:D*[RETURN]

E:L* [RETURN]

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MODULAR PROGRAMMING AND PARAMETER PASSING

30. Create three new procedures that will be structured into a modular program.

    Enter each of the following three procedures as given:

```
PROCEDURE enter
    PRINT "Please enter two values to be operated upon."
    PRINT
    INPUT "First Number ";A
    PRINT
    INPUT "Second Number ";B
    PRINT
    PRINT
    PRINT "Now calling ADD and passing two parameters!"
    RUN add(A,B)
    PRINT
    PRINT
    PRINT "OPERATION COMPLETE!"

PROCEDURE add
    PARAM A,B:REAL
    PRINT "Now executing ADD"
    PRINT
    PRINT
    PRINT A;" + ";B;" = ";A+B
    PRINT
    PRINT
    PRINT "Now calling SUBTRACT and passing two parameters!"
    RUN subtract(A,B)

PROCEDURE subtract
    PARAM A,B:REAL
    PRINT "Now executing SUBTRACT"
    PRINT
    PRINT
    PRINT A;" - ";B;" = ";A-B
```

    In the first procedure, **enter**, you are asked to provide two values which are assigned as variables A and B. Next **add** calls the procedure named **add** and passes two variables, A and B. The **add** procedure assigns the two passed variables to the same variable names and types them as REAL. A and B are added together and their sum is displayed. Finally, **add** then calls **subtract** and passes two variables, A and B. The third procedure **subtract** assigns the two passed variables to the same variable names and types them as REAL. A and B are subtracted and their difference is displayed. **subtract** then returns to **add** which returns to **enter**, which shows that the operation is complete.

31. Execute the procedure **enter** and observe. Repeat as often as necessary to comprehend modular programming and parameter passing:

```
B:RUN enter [RETURN]
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

32. Rename the workspace procedure **secondproc** as **lastname**:

B:DIR [RETURN]

B:RENAME secondproc lastname [RETURN]

B:DIR [RETURN]

33. Save a source code procedure to COPRAM's current data directory:

B:SAVE firstproc [RETURN]

34. Display the current data directory contents:

B:$DIR [RETURN]

35. Pack a source code procedure. The workspace's source code version of the packed procedure is destroyed. The packed version is saved to COPRAM's current execution directory unless otherwise specified:

B:DIR [RETURN]

B:PACK firstproc [RETURN]

B:DIR [RETURN]

B:$DIR x [RETURN]

The minus sign (-) in the workspace listing indicates that the procedure is packed.

36. Delete a workspace procedure using the **KILL** command.

B:KILL firstproc [RETURN]

B:DIR [RETURN]

37. Delete a packed workspace procedure using the **KILL** command:

B:KILL lastname [RETURN]

B:DIR [RETURN]

To use a BASIC09 procedure that resides in COPRAM, several factors need be considered. Source code procedures must be loaded before they can be run. Packed code procedures cannot be loaded, but can be run from COPRAM.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

38. Load a source code procedure into the workspace from COPRAM.

B:DIR [RETURN]

B:RUN firstproc [RETURN]
ERROR # 043
B:

Error # 043 - Unknown Procedure indicates the procedure named was not found in the BASIC09 workspace:

B:LOAD firstproc [RETURN]

B:DIR [RETURN]

B:RUN [RETURN]

39. RUN a packed procedure from COPRAM:

B:DIR [RETURN]

B:LOAD lastname [RETURN]
ERROR # 216
B:

Error # 216 - Pathname Not Found indicates that the path name **lastname** was not found in the current data directory:

B:RUN lastname [RETURN]

B:DIR [RETURN]

40. Return to OS-9 from BASIC09:

B:BYE [RETURN]

When the **BYE** command is entered, the current workspace is destroyed and any procedures that had not been previously saved to COPRAM or disk at the Terminal Emulator are lost.

CONCLUSION

By completing this tutorial you have gained a hands-on familiarity with the basic operating characteristics and functionality of the BASIC09 Programming Language. Though not all functions and characteristics have been practiced, you should be able to explore these operations on your own.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Section 6. Appendices/Glossary

About This Section

This section presents the following appendices:

Appendix A — Module Model Number "R" Suffix
Appendix B — Cable Construction and Configuration
Appendix C — Augmented Rack Resistor Pack Update
Appendix D — ASCII Code Character Set
Appendix E — RUNB Background
Appendix F — MiniCOP Error Messages
Appendix G — Application Notes
Appendix H — Conversion Tables
Appendix I — BASIC09 and Personality Procedures
Appendix J — Using conf1003 to Load the V4 and K4 Device Descriptors

An itemized glossary is also presented.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX A —
MODULE MODEL NUMBER "R" SUFFIX

All the Honeywell 620 platform controllers, 621 universal I/O modules, and many I/O rack-installed modules are being redesigned to include the capability of insertion and removal within a powered serial I/O rack. The 627-1002 and 627-1003 MiniCOPs are included in this program. Any module having this capability can be identified by its model number including the "R" suffix and is used with the 621-9938RC Serial I/O Module (SIOM).

The 621-9938RC SIOM is equipped with a FREEZE/RUN switch on its frontplate that disables or freezes the backplane. The purpose of the FREEZE/RUN switch is to prevent corrupted data from reaching the CPM or I/O while I/O modules are inserted or removed while the serial I/O rack is under power. It should be noted that the 621-9938RC SIOM itself is **NOT** capable of powered replacement, only modules containing the "R" suffix which are installed in its rack.

OPERATIONAL OVERVIEW

The following three steps provide an operational overview of the 621-9938RC's FREEZE/RUN switch and the theory regarding insertion/removal of I/O modules carrying the "R" suffix.

1. Before removing or installing an "R"-suffixed I/O module under power, place the FREEZE/RUN switch on the SIOM's frontplate to the FREEZE position.

   The 627-9938R SIOM recognizes the disable request and disables the serial rack's backplane communication. Outputs are frozen in their last state. The SIOM will not allow discrete output points or PUSH data to be updated while the FREEZE/RUN switch is in the FREEZE position. Further, the SIOM will not collect the condition of the rack's discrete input points or PULL data. When the serial link module (SLM) polls the affected SIOM during serial scan, the SIOM will respond with the last input and PULL data. This data was automatically stored in the SIOM's internal data tables at the time the backplane was disabled. Only the serial I/O racks that are disabled by the FREEZE/RUN switch will be affected. All other serial racks and the serial scan will continue normal operation.

2. Remove and/or insert any I/O modules carrying the "R" suffix. When installing a replacement module, be sure to set the DIP switches to the same position as those on the original module.

CAUTION
Modules that do NOT carry the "R" suffix should not be removed or inserted while the I/O rack is under power, regardless of the type of SIOM or the position of the FREEZE/RUN switch. Doing so may cause undesirable reactions.

3. Return the 621-9938RC's FREEZE/RUN switch to the RUN position.

   The SIOM recognizes that the normal run mode is requested and the I/O rack's backplane communications are restored. Discrete output point conditions and PUSH data will be updated, and discrete input point conditions and PULL data will be collected during the SLM's normal serial scan.

NOTE
Refer to the *620 Installation and 621 I/O Communications for 620-12/1633/36 Logic Controller Systems User Manual* for additional information regarding the "R" suffix and the operation of the insertion/removal under power technique.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX B —
CABLE CONSTRUCTION AND CONFIGURATION

This appendix provides specific detailed information regarding the construction of various serial data cables used to communicate with target devices via the MiniCOP's ASYNC ports. Included are wiring diagrams for 20mA, RS232, RS422/485, and video cables. Both half- and full-duplex configurations are provided. Also included in this appendix are example wring diagrams for establishing multidrop networks of connected target devices. These diagrams include the location of internal and external biasing and terminating resistors.

When using the RS422/485 wiring diagrams the following items should be noted:

- The prime (') symbol identifies the receiving end of the line.
- The "A" in the pin name refers to negative polarity (+/- is also indicated in the diagram).
- The "B" in the pin name refers to positive polarity (+/- is also indicated in the diagram).
- The location of external terminating resistors is not shown.
- The location of external biasing resistors is shown only in the RS-422/485 half- and full-duplex multidrop communication cable diagrams.

NOTE

Refer to the *Multidrop Communication Network Configuration* heading in this appendix for additional information regarding the location and values of internal and external biasing resistors and external terminating resistors.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ASYNC PORT PIN ASSIGNMENTS

Table 43 lists the signal description, connector designator, and interface type for the each of the 25 pins associated with a MiniCOP's ASYNC port. Use this table as a reference when connecting target devices to one of the MiniCOP's ASYNC ports.

TABLE 43 - MINICOP ASYNC PORT PIN ASSIGNMENTS

Serial Port Pin No.	Signal (I/O Description)	Connector Designator	Interface Type
1	Shield		
2	Transmit Data (output)	TXD	RS232
3	Receive Data (input)	RXD	RS232
4	Request to Send (output)	RTS	RS232
5	Clear to Send (input)	CTS	RS232
6	Data Set Ready (input)	DSR	RS232
7	Signal Ground		
8	Data Carrier Detect (input)	DCD	RS232
9	20mA Transmit Signal	XMT+	20mA
10	–15V through 1K	–12V	RS232
11	20mA Transmit Return	XMT–	20mA
12	Request to Send (output)	RS-A	RS422/485
13	Request to Send, Inverse (output)	RS-B	RS422/485
14	Transmit Data, Inverse (output)	SD-B	RS422/485
15	Transmit Data (output)	SD-A	RS422/485
16	Clear to Send, Inverse (input)	CS-B'	RS422/485
17	Clear to Send (input)	CS-A'	RS422/485
18	20mA Receive Signal	RCV+	20mA
19	+15V through 1K	+12V	RS232
20	Data Terminal Ready (output)	DTR	RS422/485
21	Receive Data, Inverse (input)	RD-B'	RS422/485
22	Receive Data (input)	RD-A'	RS422/485
23	Data Carrier Detect, Inverse (input)	RR-B'	RS422/485
24	Data Carrier Detect (input)	RR-A'	RS422/485
25	20mA Receive Return	RCV–	20mA

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SERIAL COMMUNICATIONS CABLES

20mA COMMUNICATIONS CABLE

NOTE
When the 20ma interface is **ACTIVE** (sourcing, driving), all target devices **MUST** be passive.

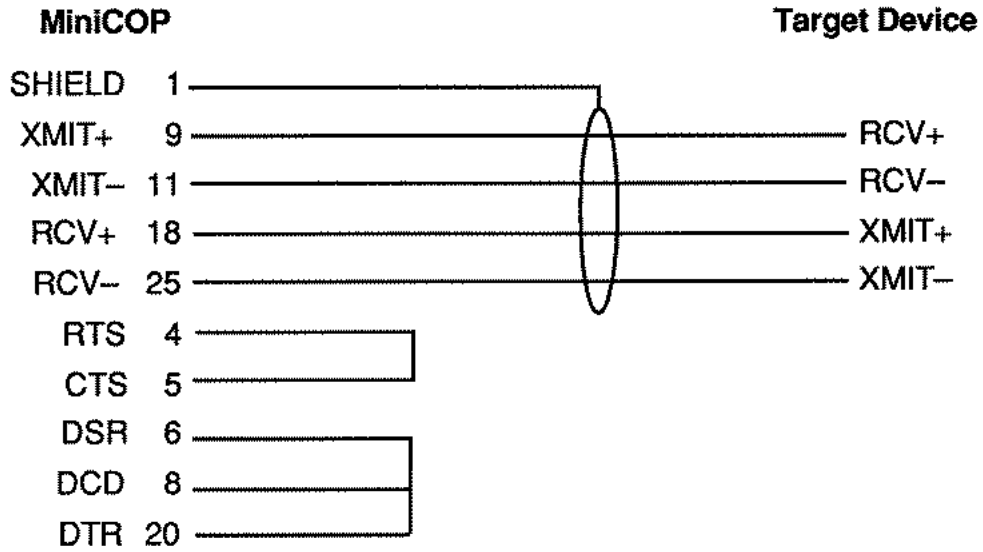


FIGURE 105 - 20MA COMMUNICATION CABLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RS232 HALF-DUPLEX COMMUNICATIONS CABLE

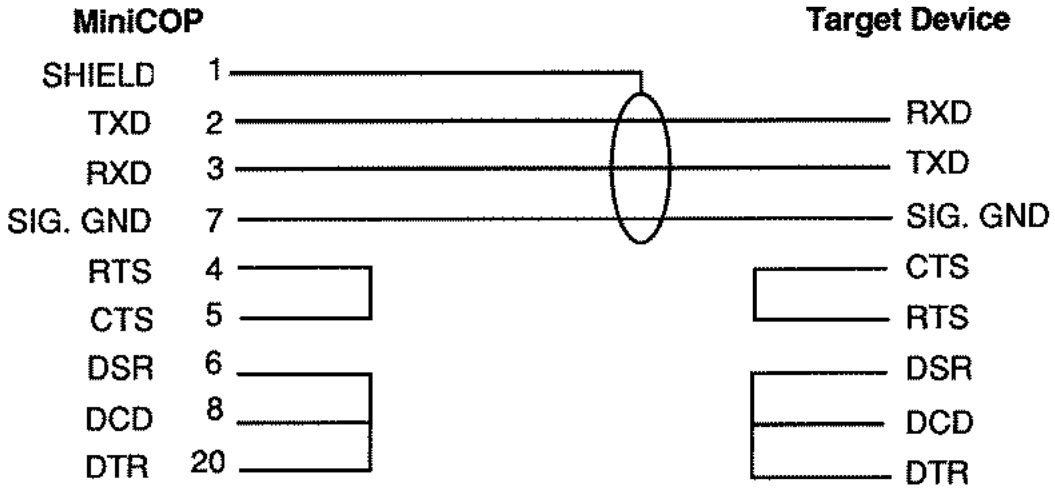


FIGURE 106 - RS232 HALF-DUPLEX COMMUNICATION CABLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RS232 FULL-DUPLEX COMMUNICATIONS CABLE

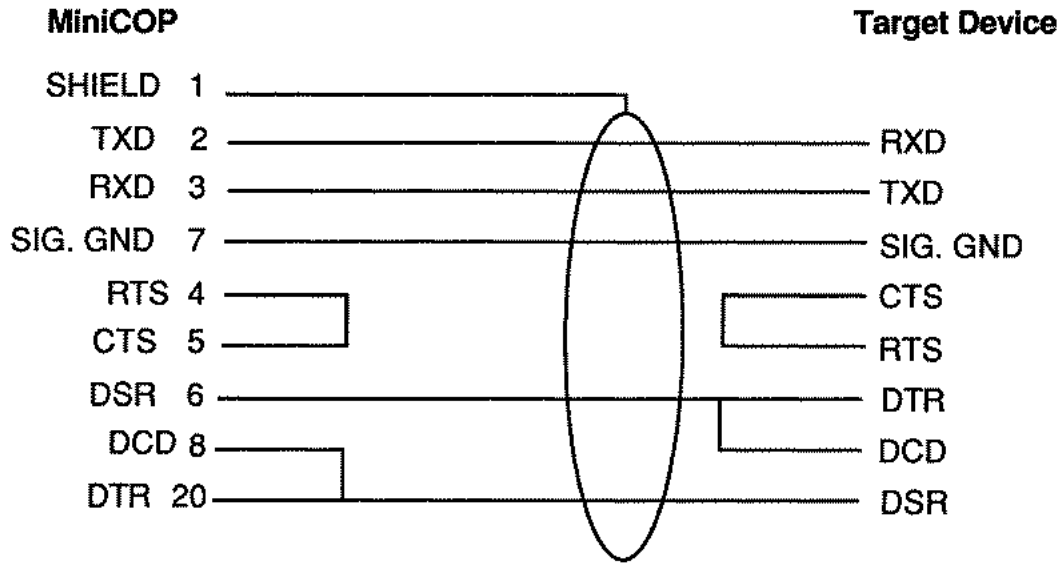


FIGURE 107 - RS232 FULL DUPLEX COMMUNICATION CABLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RS422/485 HALF-DUPLEX POINT-TO-POINT COMMUNICATIONS CABLE

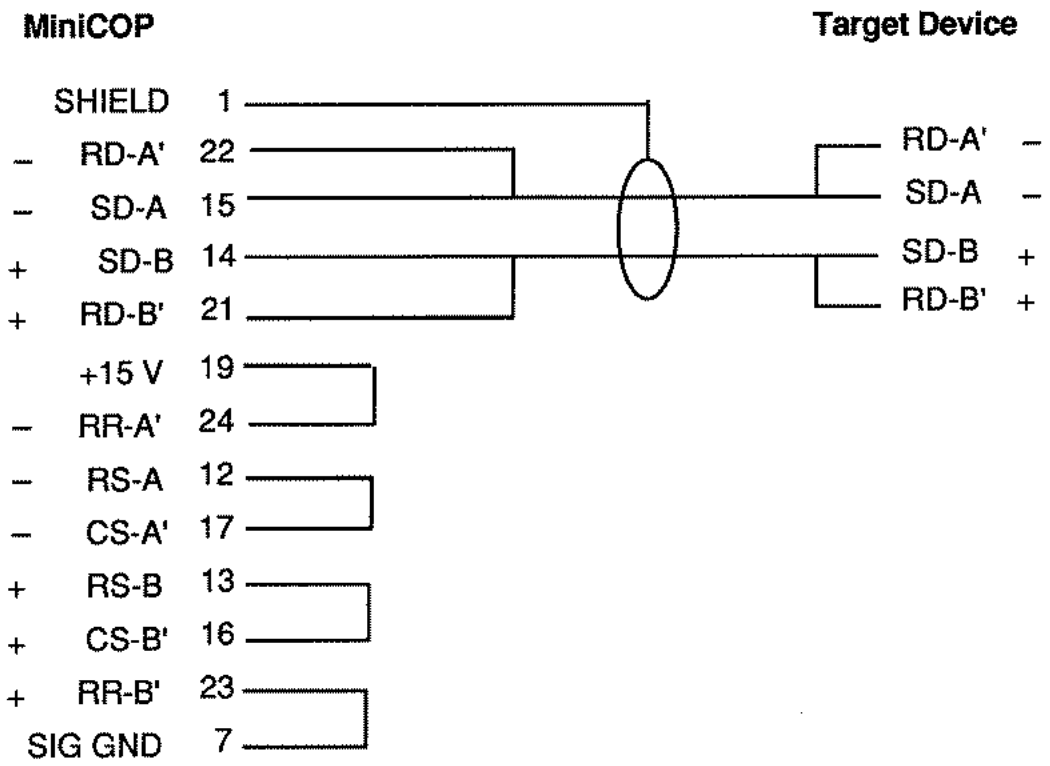


FIGURE 108 - RS422/485 HALF-DUPLEX POINT-TO-POINT COMMUNICATION CABLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RS422/485 FULL-DUPLEX POINT-TO-POINT COMMUNICATIONS CABLE

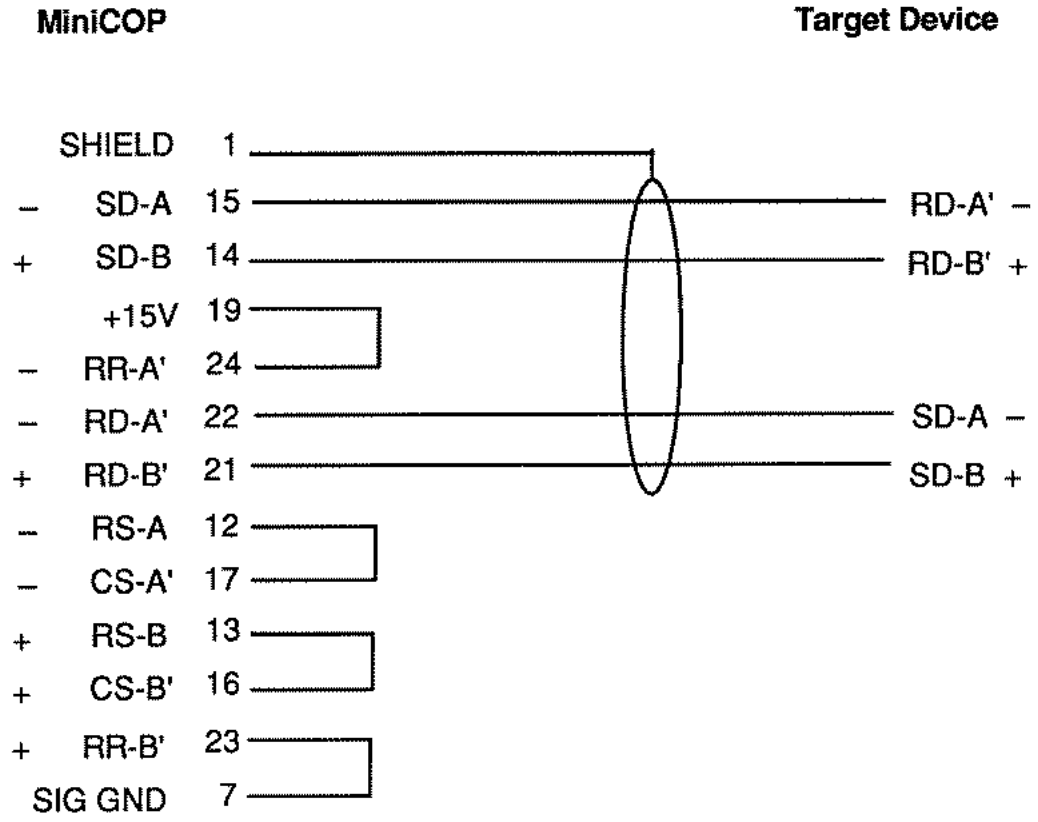


FIGURE 109 - RS422/485 FULL DUPLEX POINT-TO-POINT COMMUNICATION CABLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

RS422/485 HALF-DUPLEX MULTIDROP COMMUNICATIONS CABLE

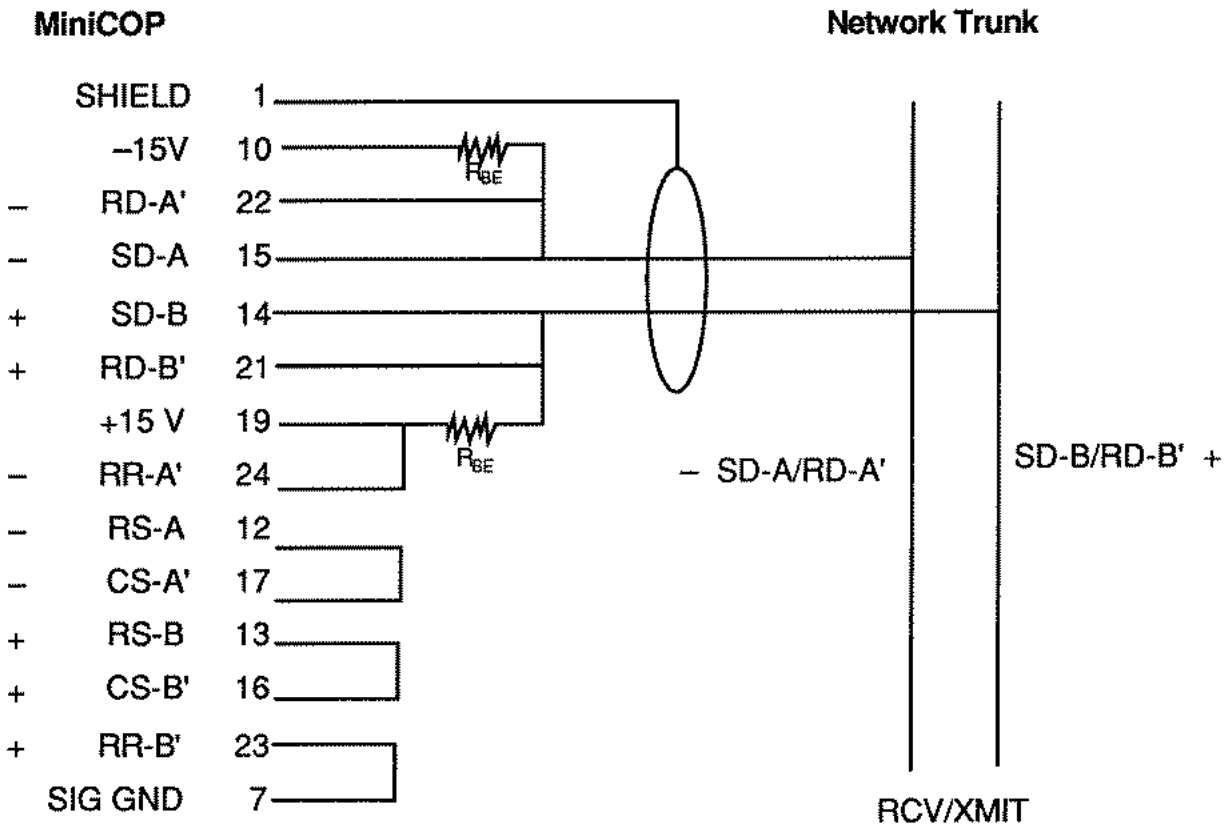


FIGURE 110 - RS422/485 HALF-DUPLEX MULTIDROP COMMUNICATION CABLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

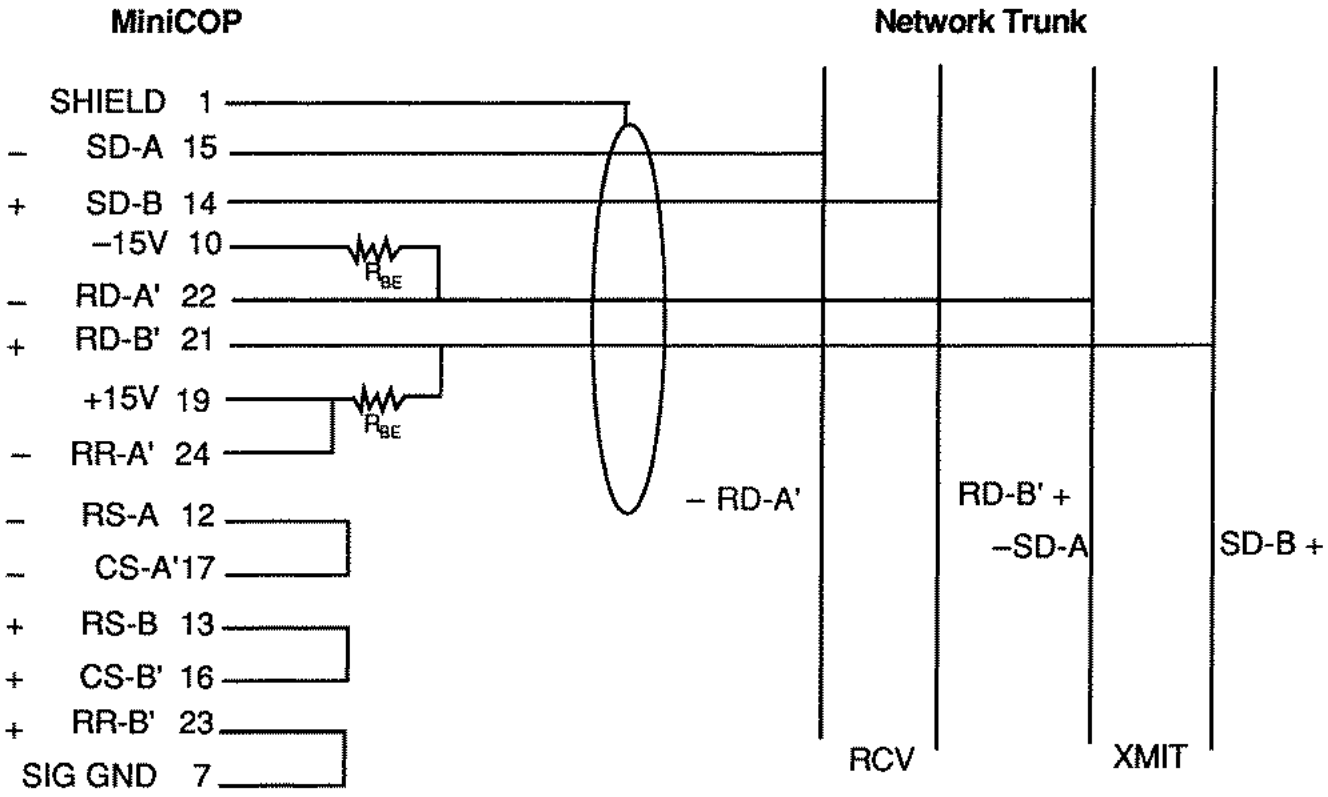


FIGURE 111 - RS422/485 FULL-DUPLEX MULTIDROP COMMUNICATION CABLE

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VIDEO CABLES

627-1003 MiniCOP VIDEO PORT CABLE CONSTRUCTION

Use Figure 112 as an aid to cable construction when connecting an RGB (color or monochrome) video monitor to the BNC type video ports of a 627-1003 MiniCOP.

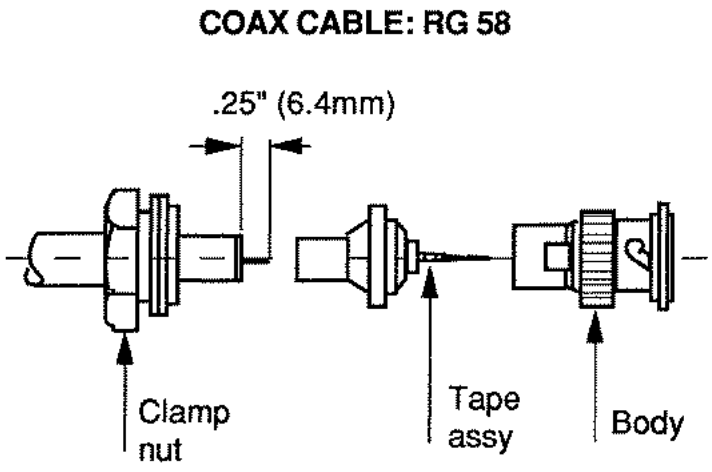


FIGURE 112 - 627-1003 VIDEO PORT CABLE CONSTRUCTION

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MULTIDROP COMMUNICATION NETWORK CONFIGURATION

The following two examples provide a basis for establishing a network of multidropped target devices from a single MiniCOP ASYNC port.

MiniCOP WITHIN THE NETWORK

Follow Figure 113 when placing the MiniCOP at any point within the multidropped network other than at its ends.

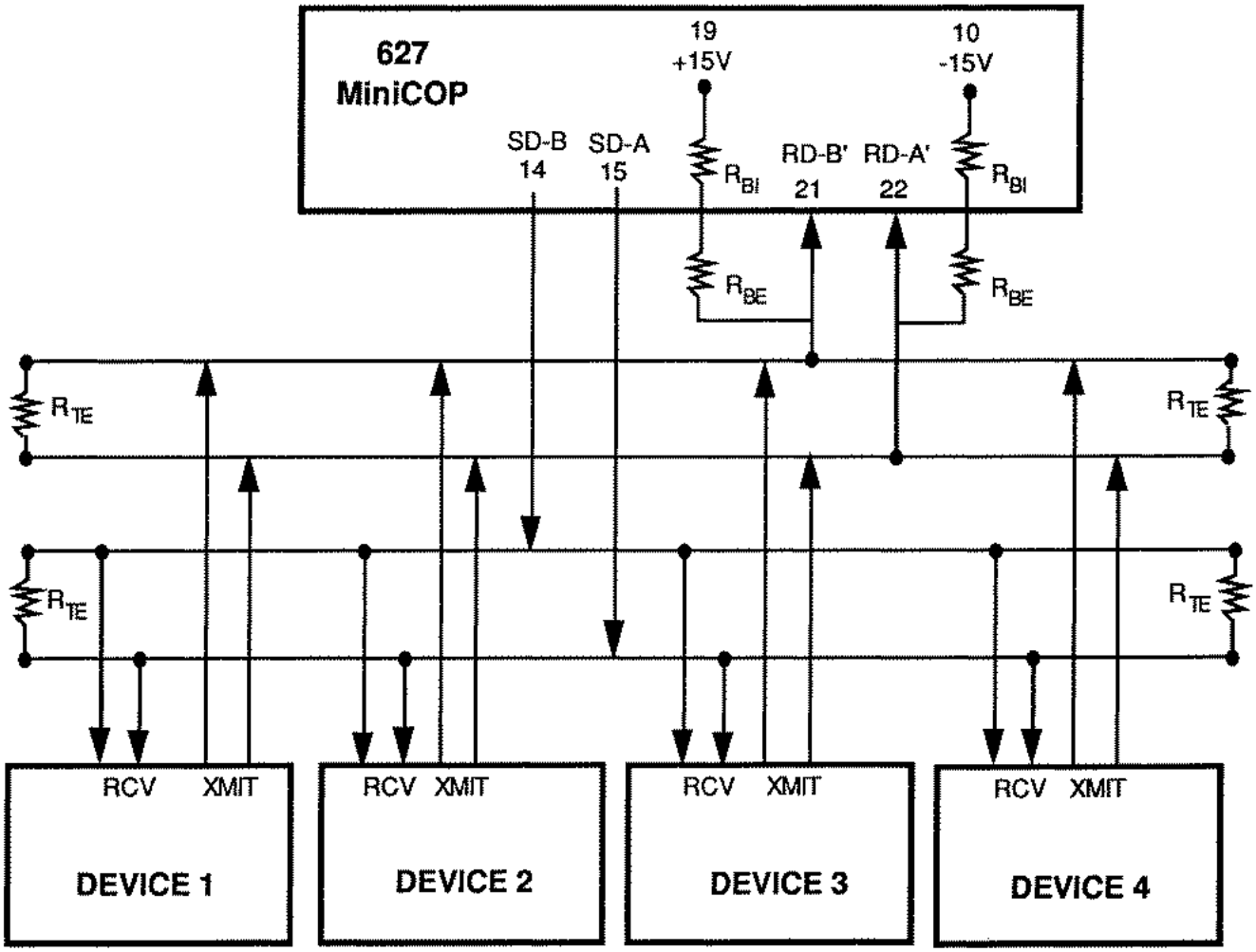


FIGURE 113 - MULTIDROPPING ASYNC PORT DEVICES EXAMPLE (MINICOP WITHIN NETWORK)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MiniCOP AT THE END OF THE NETWORK

Follow Figure 114 when placing the MiniCOP at either end of a multidropped network.

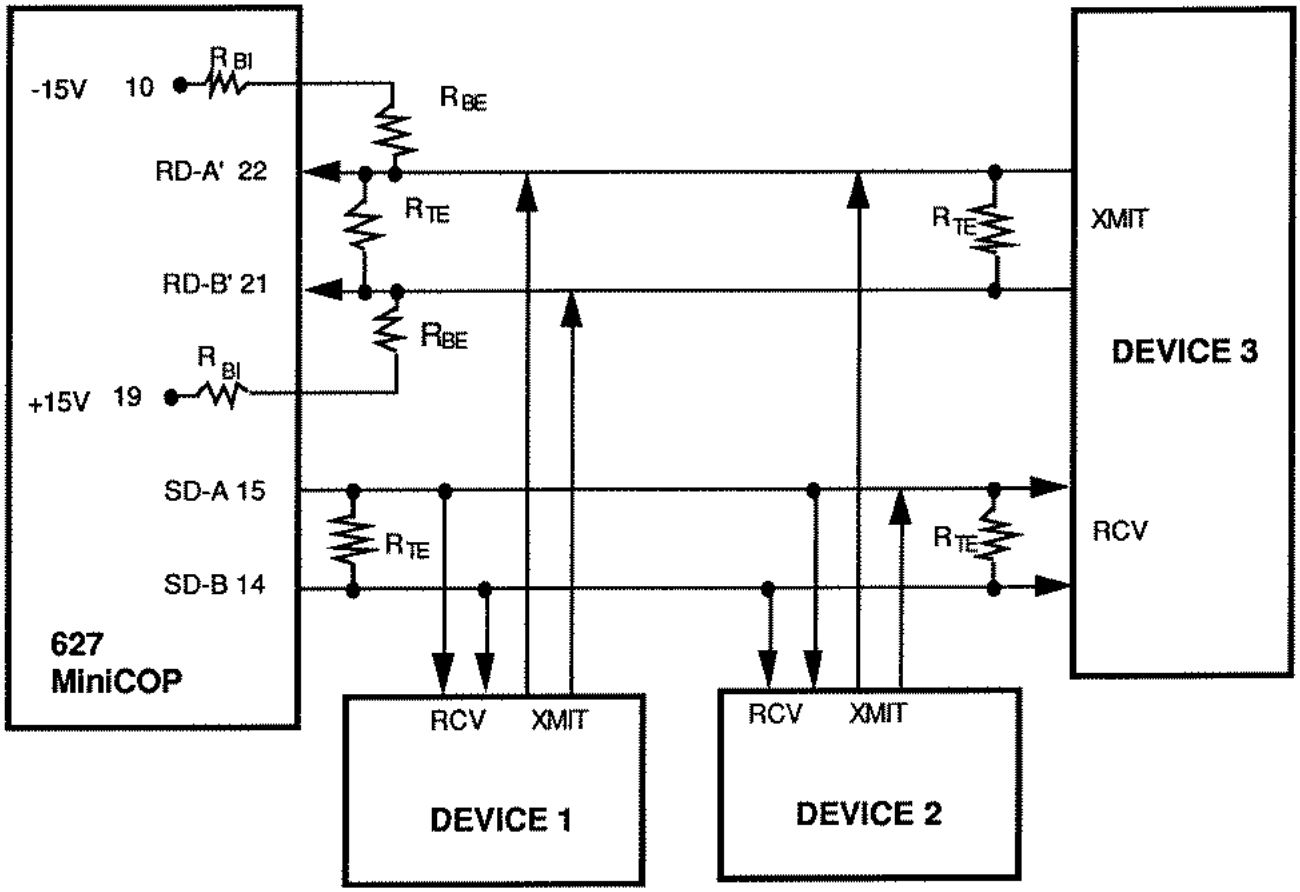


FIGURE 114 - MULTIDROPPING ASYNC PORT DEVICES EXAMPLE (MINICOP AT END OF NETWORK)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BIASING/TERMINATING RESISTORS

Table 44 provides information regarding the internal and external biasing and terminating resistors used in either of the two previous multidrop examples.

NOTE
When using "equivalent cable," all specifications and characteristics must correspond to those given in Table 44.

TABLE 44 - BIASING/TERMINATING RESISTORS

Resistor	Description	Value	Notes
R_{BE}	External biasing resistor	200Ω if R_T = 200Ω 1KΩ if R_T = 300Ω	
R_{BI}	Internal biasing resistor	1KΩ; individually set for each MiniCOP ASYNC Port by DIP switch	
R_{BT}	Total biasing resistor	R_{BI} + R_{BE}	
R_{TE}	External terminating resistor	200Ω with network lengths up to 1100'	using Belden 9729 or equivalent cable
		300Ω with network lengths up to 3000'	using Belden 9182, Alpha 9823, or equivalent cable

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX C —
AUGMENTED RACK RESISTOR PACK UPDATE

I/O modules, such as the MiniCOP, that access the upper/local bus of an augmented rack may experience excessive battery current drain when the rack is powered down. This can drastically shorten the life of the battery. To prevent the excessive current drain the augmented rack's factory-installed resistor packs must be updated.

NOTE

Backup battery power is supplied to the MiniCOP's port/memory board through a ribbon cable connecting it to the CPU board where the battery's electrical connection is made. The ribbon cable is factory-installed on the MiniCOP for installation into a standard rack. This ribbon cable must be removed for the MiniCOP to be installed into an augmented rack. Current flow is then established through the upper/local bus where non-updated resistor packs can cause excessive battery drain.

If the MiniCOP is installed in an augmented rack, battery backup of memory will no longer be available when the module is removed. Always save the MiniCOP's memory to disk via the Terminal Emulator before removal from an augmented rack.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DETERMINE WHICH RACKS MUST BE UPDATED

The following racks have augmented backplanes and must have their resistor packs updated to prevent loss of battery power in the MiniCOP:

620-0093	620-10/15 Half-Size Augmented Processor Rack
620-0091	620-10/15 Full-Size Augmented Processor Rack
621-9992C	621 Full-Size I/O Rack

1. Determine the revision level of the rack's backplane by locating the revision label. The revision label is located on the left side of the backplane in each full-size rack and the right side of the backplane in each half-size rack. See Figure 115.

2. Determine if the rack requires updating by referring to Table 45.

TABLE 45 - RESISTOR PACK UPDATE REQUIREMENT

Rack Model No.	Revision Level	
	Update Needed	No Update Needed
620-0093	Rev. 01, A	Rev. B or greater
620-0091	Rev. 01, A	Rev. B or greater
621-9992C	Rev. A, B	Rev. C or greater

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

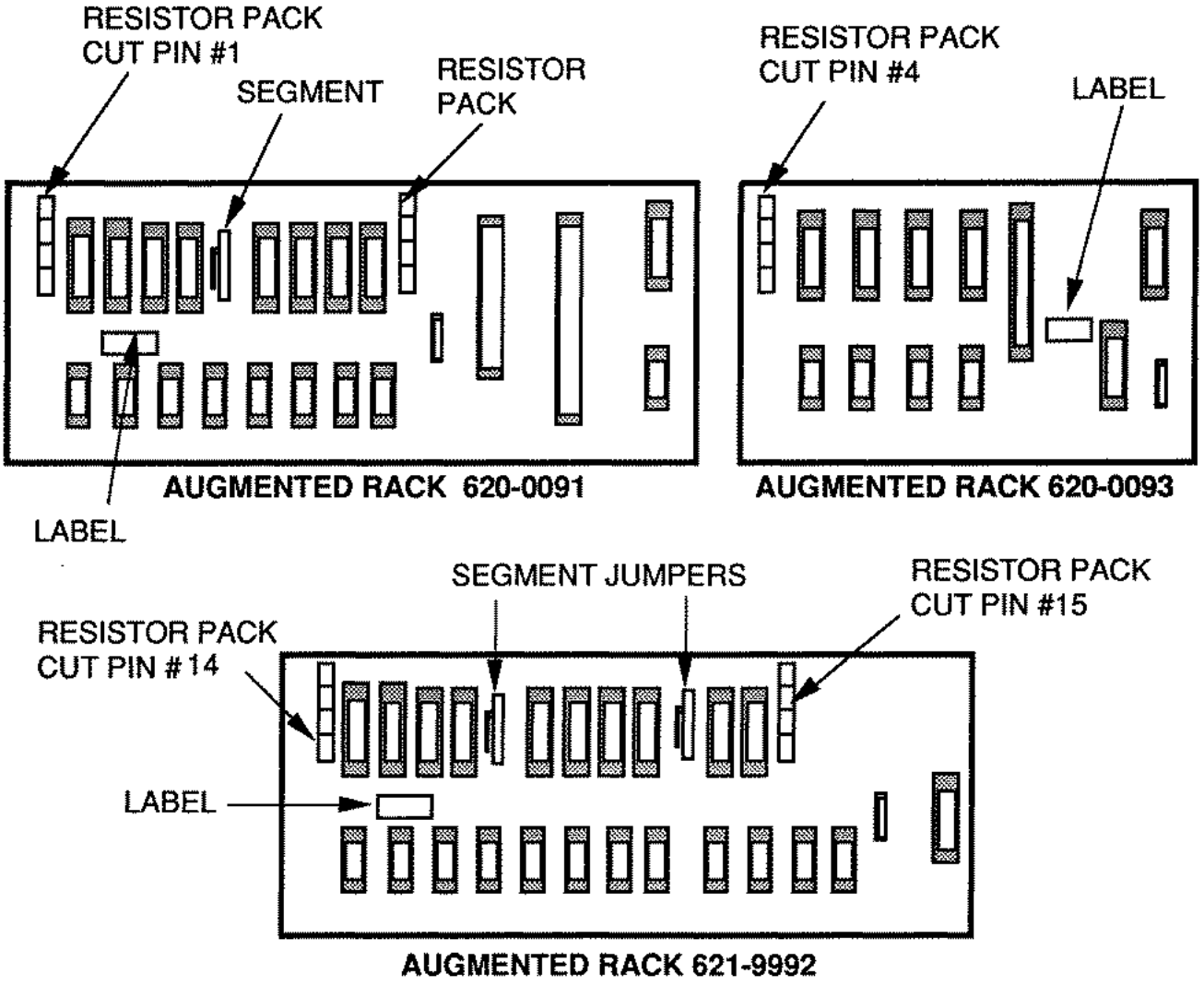


FIGURE 115 - AUGMENTED RACK BACKPLANE LAYOUTS

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

UPDATING AUGMENTED RACKS

There are two methods for updating an augmented rack.

Method 1

Obtain the augmented rack update kit by ordering Model No. 221-0002.

Method 2

Remove the affected resistor packs and cut the appropriate pins as outlined in the following procedure. Figure 116 illustrates the resistor pack pin outs.

NOTE
If the augmented rack update kit is used, follow only the instructions included with the kit.

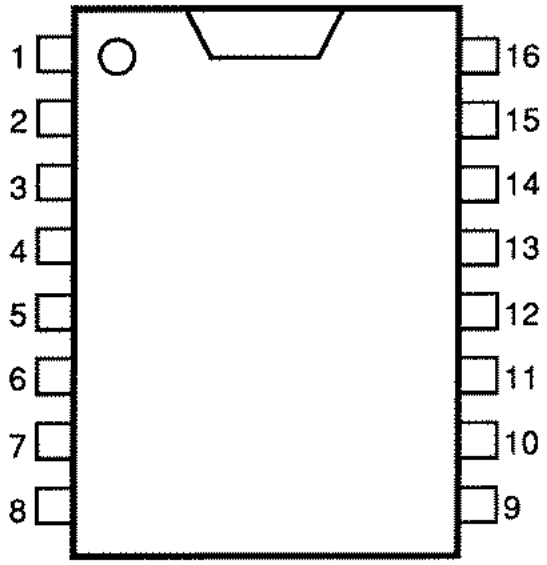


FIGURE 116 - RESISTOR PACK PIN OUTS

1. Locate the resistor packs that must be updated. These resistor pack or packs are not labeled and must be located according to their position on the backplane. See Figure 115.

2. Update one resistor pack at a time to avoid confusion. Remove one resistor pack, cut the appropriate pin (see Figure 116), and reinstall the pack.

   It is recommended that each resistor pack is removed and reinstalled with an IC extraction tool. Cut the proper pin as close to the body of the pack as possible without damaging its body.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TEST THE UPDATED AUGMENTED RACK

1. Pull all the segment jumpers from the racks backplane. See Figure 115.

2. Insert a MiniCOP into the far left-hand I/O slots of the updated rack. Install the rack's power supply module.

3. Remove the battery for the MiniCOP while it is under power.

4. Provide power to the power supply module (observe proper voltage levels and terminal connections).

5. Check the BATT LED on the installed MiniCOP module. If the LED is OFF, this portion of the update procedure was successful. If the LED is ON, this portion of the update procedure was unsuccessful -- refer to the *Troubleshooting* section that follows.

6. If updating a half-size augmented rack, the update procedure is complete. The rack power can be turned off and the battery reinstalled. If updating a full-size rack, continue with step 7.

7. Remove power from the power supply module and move the MiniCOP to the far right-hand I/O slots of the rack.

8. Restore power to the power supply module.

9. Check the BATT LED on the installed MiniCOP module. If the LED is OFF, this portion of the update procedure was successful. If the LED is ON, this portion of the update procedure was unsuccessful -- refer to the *Troubleshooting* section that follows.

TROUBLESHOOTING

10. Remove power from the power supply module.

11. Remove the resistor pack common to the segment that failed the update test. Half-size racks only have one segment. Full-size racks can have two or three segments. In full-size racks, the left-hand segment corresponds to the left-hand resistor pack and the right-hand segment corresponds to the right-hand resistor pack.

12. The update test most commonly will fail because the cut pin is still making contact with the socket. Again, carefully cut the pin off as close to the body of the pack as possible. Reinstall the resistor pack and the MiniCOP module (if removed) and repeat the testing procedure. Both the left- and right-hand segments of the full-size racks must be retested, even if only one side failed.

NOTE
If the update test continually fails, contact your Honeywell IAC service representative or the Technical Assistance Center (TAC, at 1-800-423-9883) for further instructions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX D — ASCII CODE CHARACTER SET

TABLE 46 - ASCII CODE CHARACTER SET

Key	Hex	Decimal	Key	Hex	Decimal	Key	Hex	Decimal	Key	Hex	Decimal
NUL	00	00	%	25	37	J	4A	74	o	6F	111
SOH	01	01	&	26	38	K	4B	75	p	70	112
STX	02	02	'	27	39	L	4C	76	q	71	113
ETX	03	03	(	28	40	M	4D	77	r	72	114
EOT	04	04	)	29	41	N	4E	78	s	73	115
ENO	05	05	*	2A	42	O	4F	79	t	74	116
ACK	06	06	+	2B	43	P	50	80	u	75	117
BEL	07	07	,	2C	44	Q	51	81	v	76	118
BS	08	08	-	2D	45	R	52	82	w	77	119
HT	09	09	.	2E	46	S	53	83	x	78	120
LF	0A	10	/	2F	47	T	54	84	y	79	121
VT	0B	11	0	30	48	U	55	85	z	7A	122
FF	0C	12	1	31	49	V	56	86	{	7B	123
CR	0D	13	2	32	50	W	57	87	:	7C	124
SO	0E	14	3	33	51	X	58	88	}	7D	125
SI	0F	15	4	34	52	Y	59	89	~	7E	126
DLE	10	16	5	35	53	Z	5A	90	DEL	7F	127
DC1	11	17	6	36	54	[	5B	91	F1	80	128
DC2	12	18	7	37	55	\	5C	92	F2	81	129
DC3	13	19	8	38	56	]	5D	93	F3	82	130
DC4	14	20	9	39	57	^	5E	94	F4	83	131
NAK	15	21	:	3A	58	_	5F	95	F5	84	132
SYN	16	22	;	3B	59	`	60	96	F6	85	133
ETB	17	23	<	3C	60	a	61	97	F7	86	134
CAN	18	24	=	3D	61	b	62	98	F8	87	135
EM	19	25	>	3E	62	c	63	99	F9	88	136
SUB	1A	26	?	3F	63	d	64	100	F10	89	137
ESC	1B	27	§	40	64	e	65	101	F11	8A	138
FS	1C	28	A	41	65	f	66	102	F12	8B	139
GS	1D	29	B	42	66	g	67	103	HOME	94	148
RS	1E	30	C	43	67	h	68	104	UP	95	149
US	1F	31	D	44	68	i	69	105	DOWN	96	150
SPACE	20	32	E	45	69	j	6A	106	RIGHT	97	151
!	21	33	F	46	70	k	6B	107	LEFT	98	152
"	22	34	G	47	71	l	6C	108			
#	23	35	H	48	72	m	6D	109			
$	24	36	I	49	73	n	6E	110			

NOTES: BEL is only used for writing;
F8 is not used on 623 keyboard; F9 is LINE UP on 623 keyboard; F10 is RESET on 623 keyboard; F11 is CONF on 623 keyboard; F12 is ON LINE on 623 keyboard.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX E —
RUNB BACKGROUND

INTRODUCTION

RUNB is a "run-only" version of BASIC09. It is similar to BASIC09 but does not incorporate editing or debugging features. RUNB requires only 12Kbytes of workspace memory versus BASIC09's 23 Kbytes (approximately 47.8% less). This allows additional workspace memory for user procedures.

RUNB is loaded to COPRAM's execution directory during MiniCOP initialization.

OPERATIONAL CHARACTERISTICS

Unlike BASIC09, RUNB has only one mode of operation -- the execution mode. With the exception that the RUNB execution mode will only execute packed BASIC09 procedures, it operates identical to the BASIC09 execution mode. Once the execution mode has been entered, the MiniCOP will remain in the mode until one of these events occur:

- An END, STOP, or BYE statement is executed in the procedure.
- You enter [CONTROL - E] to return to OS-9.
- You enter [CONTROL - C] to return to OS-9.
- A run-time error occurs returning the MiniCOP to OS-9.
- A PAUSE statement is executed, which returns the MiniCOP to OS-9.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Like BASIC09, RUNB can utilize the auto-run feature from OS-9. Assume that a packed BASIC09 procedure called **pacproc** resides in the execution directory. Either of the following two SHELL command line entries could be used to execute this procedure using the auto-run feature.

OS9:RUNB pacproc

OS9:pacproc

In the first of the two examples, the auto-run feature is used to execute **pacproc** just as it would in BASIC09. Notice that in the second example only the procedure's name was entered. In this case when **pacproc** was entered on the SHELL command line, OS-9 located this file name in the COPRAM's execution directory. OS-9 recognized **pacproc** as a packed BASIC09 procedure and automatically called the RUNB version of BASIC09 to execute it. In this type of scenario OS-9 will default to RUNB to minimize the amount of workspace memory used. Source code procedures must include BASIC09 as part of the SHELL command line in order to be executed with the auto-run feature. For comparison, assume that a BASIC09 source code procedure called **sorcproc** resides in the MiniCOP's data directory. The following SHELL command line can be used to execute **sorcproc** using the auto-run feature.

OS9:BASIC09 sorcproc

Parameters can also be passed to a packed procedure when using the auto-run feature. The SHELL command line will again be entered as it would to execute source code procedures. For example, assume that the packed procedure **pacproc** requires two parameters to be passed to it when it is called for execution. The SHELL command line might appear as in either of the following two examples:

OS9:RUNB pacproc(10,6)

OS9:pacproc(94,32)

NOTE

For a detailed description of the execution mode or other characteristics shared between RUNB and BASIC09, refer to the appropriate section in this manual. Remember, RUNB is an execution-only version of BASIC09 and operations not involved with procedure execution do not apply.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX F —
MiniCOP ERROR MESSAGES

BASIC09 ERROR CODES

Error codes greater than 80 are those generated by the OS-9 Operating System or external programming languages.

TABLE 47 - BASIC09 ERROR CODES

#	Error	Definition
001	Unconditional Abort	
002	Keyboard Abort	
003	Keyboard Interrupt	
010	Unrecognized Symbol	Unrecognized symbol
011	Excessive Verbage	Too many keyboards or symbols
012	Illegal Statement Construction	
013	I-Code Overflow	More workspace memory required
014	Illegal Channel Reference	Invalid path number given
015	Illegal Mode	Invalid path access mode (READ/WRITE/ UPDATE/DIR)
016	Illegal Number	
017	Illegal Prefix	
018	Illegal Operand	
019	Illegal Operator	
020	Illegal Record Name	
021	Illegal Dimension	
022	Illegal Literal	
023	Illegal Relational	
024	Illegal Type Suffix	
025	Too-Large Dimension	
026	Too-Large Line Number	
027	Missing Assignment Statement	
028	Missing Path Number	
029	Missing Comma	
030	Missing Dimension	
031	Missing DO Statement	
032	Memory Full	Additional workspace memory required
033	Missing GOTO	
034	Missing Left Parenthesis	
035	Missing Line Reference	
036	Missing Operand	
037	Missing Right Parenthesis	
038	Missing THEN Statement	
039	Missing TO	
040	Missing Variable Reference	
041	No Ending Quote	

(Continued)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 47 - BASIC09 ERROR CODES *(Continued)*

#	Error	Definition
042	Too Many Subscripts	
043	Unknown Procedure	
044	Multiply — Define Procedure	
045	Divide by Zero	
046	Operand Type Mismatch	
047	String Stack Overflow	
048	Unimplemented Routine	
049	Undefined Variable	
050	Floating Overflow	
051	Line with a Compiler Error	
052	Value Out of Range	
053	Subroutine Stack Overflow	
054	Subroutine Stack Underflow	
055	Subscript Out of Range	
056	Parameter Error	
057	System Stack Overflow	
058	I/O Type Mismatch	
059	I/O Numeric Input Format Bad	
060	I/O Conversion: Number Out of Range	
061	Illegal Input Format	
062	I/O Format Repeat Error	
063	I/O Format Syntax Error	
064	Illegal Path Number	
065	Wrong Number of Subscripts	
066	Non-Record-Type Operand	
067	Illegal Argument	
068	Illegal Control Structure	
069	Unmatched Control Structure	
070	Illegal FOR Variable	
071	Illegal Expression Type	
072	Illegal Declarative Statement	
073	Array Size Overflow	
074	Undefined Line Number	
075	Multiply — Defined Line Number	
076	Multiply — Defined Variable	
077	Illegal Input Variable	
078	Seek Out of Range	
079	Missing Data Statement	
080	Print Buffer Overflow	

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

OS9 ERROR CODES

Error codes other than those listed are generated by BASIC09, other external programming language or user programs.

TABLE 48 - OS-9 ERROR CODES

#	Error	Definition
200	Path Table Full	File cannot be opened as the system path table is full
201	Illegal Path Number	Given path number is too large and/or non-existent
202	Interrupt Polling Table Full	
203	Illegal Mode	Device is incapable of performing requested I/O function
204	Device Table Full	Cannot add another device, table full
205	Illegal Module Header	Module's sync code, parity, or CRC incorrect, not loaded
206	Module Directory Full	Module directory memory full, cannot add modules
207	Memory Full	Insufficient contiguous COPRAM memory
209	Module Busy	Module in use by another process, non-sharable
210	Boundary Error	Memory allocation or de-allocation not on page boundary
211	End of File	End of file marker encountered during read operation
212	Returning Non-Allocated Memory	De-allocation of unassigned memory attempted
213	Non-Existing Segment	Device has a damaged file structure
214	No Permission	File or device does not permit requested access
215	Bad Path Name	Syntax error in pathlist
216	Path Name Not Found	Cannot find specified pathlist
217	Segment List Full	File cannot be expanded, excessive fragmentation
218	File Already Exists	File name already appears in the specified directory
219	Illegal Block Address	Device's file structure has been damaged
221	Module Not Found	Specified module name and pathlist not found
223	Suicide Attempt	Request to return memory belonging to the stack
224	Illegal Process Number	No such process exists
226	No Children	Cannot wait, process has no children
227	Illegal SWI Code	SWI must be 1 to 3
228	Process Aborted	Process aborted by signal code 2
229	Process Table Full	Cannot fork at this time
230	Illegal Parameter Area	High and low bounds of passed parameters incorrect
231	Known Module	Specified module exists, Internal Use Only
232	Incorrect Module CRC	Bad cyclic redundancy check value
233	Signal Error	Previous unprocessed signal pending in receiving process
234	Non-Existent Module	Unable to locate specified module
235	Bad Name	Illegal name syntax
237	RAM Full	No free system RAM available
238	Unknown Process ID	Incorrect process ID number
239	No Task Number Available	All task numbers in use
240	Unit Error	Specified device unit does not exist
241	Sector Error	Sector number out of range
242	Write Protect	Device is write protected

(Continued)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 48 - OS-9 ERROR CODES *(Continued)*

#	Error	Definition
243	CRC Error	Cyclic redundancy check error during read or write verify
244	Read Error	Data transfer error during read, or SCF buffer overrun
245	Write Error	Hardware error during disk write operation
246	Not Ready	Device is currently unable to perform
247	Seek Error	Physical seek to non-existent sector requested
248	Media Full	Insufficient free space on target media
250	Device Busy	Non-sharable device in use by another process
252	Record is Locked-Out	Another process is accessing the requested record
253	Non-Sharable File Busy	Another process is accessing the requested file

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX G — APPLICATION NOTES

The Honeywell 627 MiniCOP Modules are industrial rated microprocessors providing a mechanism to interface intelligent factory-floor devices to the 620 platform controllers. The MiniCOP is the successor to the 627 COP which was housed in a free-standing enclosure comparable in size to a full-size 19-inch processor or I/O rack. The MiniCOP is essentially a scaled down version of the COP that provides the same power and functionality in drastically less space.

Table 49 provides a listing of application notes written for the COP and MiniCOP. While most of these documents were specifically written for the 627 COP, all are still available and may be useful as a guide in the development of similar applications for the MiniCOP.

For additional information regarding the ordering of these Application Notes contact Honeywell IAC, Automation College—East in Ft. Washington, PA, USA; telephone 215-641-3000.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 49 - APPLICATION NOTE SUMMARY

Title and Form Number	Description
CONVERTING HEX TO DECIMAL AND DECIMAL TO HEX IN THE 627-70 Form 620-610	Provides two BASIC09 procedures that convert values between the decimal and hexadecimal numbering systems. The first of these procedures converts a variable length hexadecimal string into its decimal equivalent. The second converts a positive whole number into its hexadecimal string equivalent. Included in this document is third procedure that will call and execute either of the conversion procedures based on user-provided input.
DETERMINING THE LENGTH OF FILES IN THE 627-70 Form 620-611	Provides a program and background information allowing the user to make the byte count of a specified file available to a BASIC09 procedure for certain data transfer operations in which the amount of data to be passed must be known in advance.
IPC 627-7001 FILE EXAMINER Form 620-612	Includes a general purpose procedure and background information providing the user with the ability to examine the contents of a COPRAM file. The numeric order, ASCII character representation, and decimal and hexadecimal values of each byte are displayed. The procedure can also be used to examine characters coming into a port or already in the port's buffer.
CONVERTING 627 DATE$ TO A STANDARD FORMAT DATE DISPLAY Form 620-613	Provides a BASIC09 procedure and background information allowing the 627 DATE$ (YY/MM/DD) date output listing to be displayed in a standard format (MON DD, YYYY). For example, the DATE$ for 91/08/06 will be displayed as AUG 06, 1991.
INTERFACING THE 627-70 TO THE TECHTRAN 981 MICRO-DISC Form 620-614	Includes the BASIC09 procedures and background information providing a method for interfacing the 627-70 COP to a Techtran 981 Micro-Disc diskette storage device. Data can be transferred from the COP to the Techtran or from the Techtran to the COP.
IPC 627-70 BARCODE READER INTERFACE Form 620-615	Includes the BASIC09 procedure and background information to interface a barcode reader with a 627-70 COP. Additional information describes a method for further interfacing the 627-70 COP to a 620 LCS via a Communications Interface Module (CIM) so that the information provided through the reading of the barcode can be utilized in a ladder logic control scheme.
USING A PROCEDURE FILE TO ALLOCATE PORT VARIABLE STORAGE Form 620-616	Includes an OS-9 procedure file that is used to clear each communications port prior to executing the main BASIC09 procedure.
COMMUNICATION BETWEEN 627 COP AND AB PLC-3 VIA AB DATA HIGHWAY Form 620-617	Outlines and details communication transmissions between the 627 COP and the Allen-Bradley PLC-3 on the AB Data Highway. Included are the hardware requirements to interface the 627 COP to an AB KF or KE Communication Controller Module, which in turn is linked to an AB KA Communication Adapter Module and finally to the AB PLC-3.
INTERFACING THE 623-50 OR 627-70 TO THE DECWRITER LA-120 Form 620-619	Provides the cabling, set-up parameters, and dip switch position requirements to establish serial communications from a Decwriter LA-120 printer and either the 623-50 Loader Terminal or the 627-70 COP.

(Continued)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 49 - APPLICATION NOTE SUMMARY *(Continued)*

Title and Form Number	Description
SETTING TERMINAL ATTRIBUTES Form 620-620	Describes a method of developing a standard BASIC09 procedure to be executed in the 627 COP that will allow the user to easily set the attributes of the 623 Loader Terminal.
UPLOADING/DOWNLOADING 620 PROCESSOR CONTROL PROGRAMS FROM THE 627 COP Form 620-625	Describes a method to create BASIC09 procedures that will allow the user to upload ladder logic programs from a 620 LCS to a 627 COP and to download ladder logic procedures from a 627 COP to a 620 LCS. A sample application using this technique might involve a single 620 LCS that is involved in multiple applications requiring different ladder logic control programs, which are stored in the 627 COP. The 620 LCS can execute on program while having access to several others. Another application may involve the handling of a ladder logic program to large to be loaded in the 620 LCS's memory. Appropriate sections of this larger ladder logic program can be downloaded from the 627 COP as needed.
627-70 COPRAM ORGANIZATION Form 620-626	Contains a complete description of the structure of COPRAM memory organization in the 627-7001 COP. Included is a compilation of factual information concerning the physical COPRAM organization under the OS-9 Operating System.
VOICE SYNTHESIS USING THE 627-7001 COP Form 620-627	Describes the hardware requirements and a demo BASIC09 procedure used to interface the 627-7001 COP to the Votrax Personal Speech System. This technique can be used to add voice synthesis to be used in alarm and operator assistance functions.
DSKPAK: ELIMINATING COPRAM MEMORY AND FILE FRAGMENTATION Form 620-628	Illustrates a method of compressing files and making files contiguous to establish the largest possible blocks of free COPRAM memory in a 627-7001 COP. A BASIC09 procedure that will accomplish this task is also included in this document along with supporting background information.
SINGLE COM PORT OPERATION OF TERMINAL EMULATOR FOR THE 627 COP/MiniCOP Form 620-649	Describes a procedure to allow all the file transfer features of the Terminal Emulator to be used between an MS-DOS compatible personal computer having only one serial port and either the 627 COP or the 627 MiniCOP. Topics included cover: cabling, port configurations, loading the OS-9 system commands from floppy disk, and numerous file operations.
627-1002/1003 MiniCOP TAPE LOADING AND SAVING PROCEDURE USING THE 623-50/51 LOADER/TERMINAL Form 620-653	Provides specific procedures for saving and loading 627 MiniCOP procedures using the 623-50/51 Loader/Terminal. Included are step-by-step operations that require minimal familiarity with the 623-50/51 L/T and the 627 MiniCOP.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX H — CONVERSION TABLES

SELECTED BINARY, DECIMAL, AND HEXADECIMAL CONVERSIONS

TABLE 50 - NUMBER SYSTEM CONVERSIONS

Hex	Decimal	Binary	Hex	Decimal	Binary
1	1	1	100	256	100000000
2	2	10	200	512	1000000000
3	3	11	300	768	1100000000
4	4	100	400	1024	10000000000
5	5	101	500	1280	10100000000
6	6	110	600	1536	11000000000
7	7	111	700	1792	11100000000
8	8	1000	800	2048	100000000000
9	9	1001	900	2304	100100000000
A	10	1010	A00	2560	101000000000
B	11	1011	B00	2816	101100000000
C	12	1100	C00	3072	110000000000
D	13	1101	D00	3328	110100000000
E	14	1110	E00	3584	111000000000
F	15	1111	F00	3840	111100000000
10	16	10000	1000	4096	1000000000000
20	32	100000	2000	8192	10000000000000
30	48	110000	3000	12288	11000000000000
40	64	1000000	4000	16384	10000000000000
50	80	1010000	5000	20480	10100000000000
60	96	1100000	6000	24567	110000000000000
70	112	1110000	7000	28673	111000000000000
80	128	10000000	8000	32768	1000000000000000
90	144	10010000	9000	36864	1001000000000000
A0	160	10100000	A000	40960	1010000000000000
B0	176	10110000	B000	45056	1011000000000000
C0	192	11000000	C000	49152	1100000000000000
D0	208	11010000	D000	53248	1101000000000000
E0	224	11100000	E000	57344	1110000000000000
F0	240	11110000	F000	61440	1111000000000000

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

MATHEMATIC CONVERSIONS

Mathematical functions that are not directly programmable in BASIC09 can be converted using the following formulas.

TABLE 51 - MATHEMATICAL CONVERSIONS

Function	Formula
Log Base B	LOGB(X)=LOG(X)/LOG(B)
Secant	SEC(X)=1/COS(X)
Cosecant	CSC(X)=1/SIN(X)
Cotangent	COT(X)=1/TAN(X)
Inverse Sine	ARCSIN(X)=ATN(X/SQR(1-X*X))
Inverse Cosine	ARCCOS(X)=1.570796-ATN(X/SQR(1-X*X))
Inverse Secant	ARCSEC(X)=ATN(SQR(X*X-1))
Inverse Cosecant	ARCCSC(X)=ATN(1/SQR(X*X-1))=(X0)*3.14159
Inverse Cotangent	ARCCOT(X)=1.570796-ATN(X)
Hyperbolic Sine	SINH(X)=(EXP(X)-EXP(-X))/2
Hyperbolic Cosine	COSH(X)=(EXP(X)-EXP(-X))/2
Hyperbolic Secant	TANH(X)=(EXP(X)-EXP(-X))/EXP(X)=EXP(-X)
Hyperbolic Tangent	SECH(X)=2/(EXP(X)+EXP(X))
Hyperbolic Cosecant	CSCH(X)=2/(EXP(X)+EXP(X))
Hyperbolic Cotangent	COTH(X)=(EXP(X)+EXP(-X))/(EXP(X)-EXP(-X))
Inverse Hyperbolic Sine	ARCSINH(X)=LOG(X+SQR(X*X+1))
Inverse Hyperbolic Cosine	ARCCOSH(X)=LOG(X+SQR(X*X-1))
Inverse Hyperbolic Tangent	ARCTANH(X)=LOG((1+X)/1-X))/2
Inverse Hyperbolic Secant	ARCSECH(X)=LOG((1+SQR(1X*X))/X
Inverse Hyperbolic Cosecant	ARCSCH(X)=LOG((1-SGN(X)*SQR(1+X*X))/X
Inverse Hyperbolic Cotangent	ARCCOTH(X)=LOG((X+1)/(X-1))/2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX I —
BASIC09 AND PERSONALITY PROCEDURES

When BASIC09 initially compiles a procedure, it determines the amount of memory which must be allocated for data and variable storage based strictly from the DIM statements found within that procedure. This memory is allocated in 256-byte pages. It is possible that the last page allocated may be almost entirely full or almost entirely empty depending on the remainder when the memory requirement is divided into these 256-byte pages.

If a Personality Procedure is called from a BASIC09 procedure, memory is not allocated for use by its required communication buffers. The Personality Procedure utilizes the remaining memory on the last 256-byte page, originally assigned to the procedure for data and variable storage, for its own communication buffers. If sufficient free space exists on this last page for use by the Personality Procedures communication buffers, no problem should result. However, if sufficient memory is not available, a portion of the existing program code may be over-written by data buffered by the Personality Procedure. As a result, the BASIC09 procedure may suddenly and mysteriously stop. No errors are reported. The BASIC09 procedure will have simply appeared to die for no apparent reason.

A solution to this problem is to explicitly allocate a sufficient amount of memory when BASIC09 or a specific procedure is invoked. It is recommended that you start with an allocation of 16-Kbytes and decrease that amount only if there is insufficient free memory for such a large allocation.

NOTE

Refer to *Changing the Workspace Size; Section 5, BASIC09 Programming Language* in this manual for additional information regarding the allocation of BASIC09 Workspace memory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX J — USING CONF1003 TO LOAD THE V4 AND K4 DEVICE DESCRIPTORS

The 627-1003 MiniCOP provides three additional, basic features over the 627-1002 MiniCOP; a fourth 25-Pin ASYNC Port, a 623-51 Keyboard compatible Port, and RGB Video Ports. Each of these three ports requires a device descriptor; T4, V4 and K4 respectively.

Due to memory capacities, the upgrades to the MiniCOP's ROMmed firmware (V.R 1.2) necessitated the removal of several low priority files. Because the majority of 627-1003 MiniCOPs are bought to obtain the fourth ASYNC Port, the keyboard and video device descriptors (V4, K4) were removed from firmware. These device descriptors have been added to the OS-9 System Commands disk (V.R 1.1) along with the utility program conf1003. V4, K4 and conf1003 are transferred to COPRAM as part of the CMDS directory during initialization.

The utility program conf1003 is used to facilitate the transfer of V4 and K4 from COPRAM to the module directory when either of these device descriptors are required by a specific MiniCOP application. Since only one of the these three output operations (fourth ASYNC Port, the Keyboard Port, or the Video Port) can be configured in any single MiniCOP, the unneeded files can be removed from the module directory using the UNLINK command.

To use the conf1003 utility program to load the V4 and K4 device descriptors to the module directory, perform the following steps.

1. After the MiniCOP has been initialized, enter conf1003 at the OS9: prompt and press [RETURN].

2. Verify that V4 and K4 have been transferred to the module directory by entering MDIR [RETURN] at the OS9: prompt.

3. If desired, unlink the two device descriptors that are no longer needed by entering UNLINK <filename> <filename> at the OS9: prompt.

4. Verify that the two device descriptors have been removed from the module directory by entering MDIR [RETURN] at the OS9: prompt.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

APPENDIX K —
PRGMB FILE EXAMPLE

The MiniCOP can be configured to power up in the OS9 mode of operation or immediately execute a pre-built user program called, PRGMB. Toggling the unit's toggle switch down repeats PRGMB while up executes another pre-built user program called, PRGMA

The BUILD command (OS9 command) is used to create batch files like PRGMB. Following is an example of a PRGMB batch file. This program includes REM statements describing the purpose of the lines of code.

PRGMB	**-Power up and run this batch file**
RESET F	**-Resets the MiniCOP can clears the module directory providing maximum user memory**
CLEAR /T1	**-Clears and initializes port 1 of MiniCOP**
CLEAR /T2	**-Clears and initializes port 2 of MiniCOP**
CLEAR /T3	**-Clears and initializes port 3 of MiniCOP**
TLOAD IPC621	**-Loads the 621 I/O driver utility that permits the MiniCOP to communicate with the 621 I/O system**
TLOAD ppint	**-Loads the generic program that controls personality procedure execution**
TLOAD packedproc	**-Loads a packed program into memory**
TLOAD RUNB	**-Loads the run-only version of BASIC09 into memory**
RUNB #10K packedproc	**Autoexecutes RUNB, allocates 10K of workspace, and autoexecutes user program packedproc**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

GLOSSARY

This glossary defines the following terms as they pertain to the 627 MiniCOP system and/or related items. These terms may also have other accepted meanings, which are not given in the provided definition.

620

620 LC - 620 Logic Controller . Honeywell's version of the PLC providing logic solutions to industries requiring a combination of process, hybrid, and discrete control solutions. The platform upon which the Series 9000, Series 9000E, and Logic Manager are based. Formally known as the IPC 620.

620 Platform Controller - A term that encompasses any of the Honeywell control systems that are built on the 620 LC platform, including the 620 LC itself. The platform controllers include: all models of the 620 LCs, all models of the Series 9000 and 9000E Loop and Logic Controllers, and the TDC 3000 Logic Manager.

A

ABC Protocol - Asynchronous Byte Count Protocol - a Honeywell communications protocol. Used by the 620-0048 and 620-0052 Data Collections Modules (DCM) to exchange serial data with other devices such as a MiniCOP.

Active Process - A process currently assigned CPU time and engaged in useful execution.

Address - A unique physical location in memory. Typically given as a hexadecimal byte.

Algorithm - A step-by-step procedure for solving a problem. A program consists of a series of algorithms.

ALM - Advanced Logic Manager. A 620 platform controller that is capable of being connected to the TDC 3000 Universal Control Network (UCN) providing peer-to-peer communication with other UCN devices.

Application Program - A program designed by the user to perform a real-world task.

Argument - The number that a function works on to produce a result.

Array - A BASIC09 data structure consisting of an ordered one, two , or three dimensional list of items that share some characteristic.

ASCII - American Standard Code for Information Interchange. The standard code used for exchanging messages between data processing systems. An ASCII file is a text file that represents each character in ASCII code.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A, Continued

Asynchronous Port - ASYNC port - the 25-pin port used by the MiniCOP to communicate with intelligent factory-floor devices and other peripherals. The 627-1002 MiniCOP has three of these ports, while the 627-1003 has four.

Atomic Data Type - One of five data types (REAL, INTEGER, BYTE, STRING, BOOLEAN) used in BASIC09 to declare variables and arrays.

B

Background Processing - Execution of low priority tasks during the idle moments of the primary task's execution.

BASIC09 - The programming language for the MiniCOP. It combines the familiarity of BASIC with the structured approach of PASCAL.

BASIC09 Mode - A mode of operation in the BASIC09 programming language that allows such functions as saving, loading, listing, etc.

Boolean - A purely logical environment where all statements can be evaluated as either being TRUE or FALSE.

Breakpoint - An address at which task execution is suspended. Used as a debugging tool during program development.

Buffer - A portion of memory used as a temporary data storage area. Often used when transmission occurs at different speeds.

C

Call - To direct an execution flow to a subroutine elsewhere within the current procedure; or, to suspend the current procedure and to load and run another, returning to the suspended (calling) procedure after the new procedure's execution terminates.

CIM - Communications Interface Module - a general purpose communications interface device that supports the MODBUS RTU protocol. One of two communications modules that can be used to interface (port method) a MiniCOP to a 620 platform controller. Used with the 620 LCS personality procedure.

Command - A BASIC09 or OS-9 instruction that institutes a direct action. (e.g., SAVE, LOAD, COPY).

Compiler - Reduces BASIC09 procedures into smaller compact forms. An intermediate compiler translates source code procedures into I-code (intermediate code). I-code can be decompiled back into source code. A packing compiler reduces BASIC09 procedures into their most condensed form where only executable code remains. Packed procedures cannot be decompiled.

Concurrent Execution - See Multitasking.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C, Continued

COPRAM - A 256 Kbyte RAMDisk similar in use to a hard disk drive.

CRC - Cyclic Redundancy Check - an error checking technique performed on a program module to ensure its integrity.

Cursor - A moveable marker on the display screen that indicates the current working position.

D

Datacontrol - An executable BASIC09 utility program. Allows the user to write to or read from the data table and is used as a test or debug tool when developing applications

Data Directory - The COPRAM directory that is used to hold text files, procedure files, and BASIC09 source code procedures. One of two possible current COPRAM directories allowed in OS-9 and identified by a directory pointer. Defaults to /D0 (root directory) after MiniCOP initialization.

Data Table - A temporary data storage area separate from COPRAM and main memory. Used as a depository for information being transferred between the MiniCOP and a host 620 platform controller when this communication involves the host's I/O bus.

Debug Mode - A mode of operation in the BASIC09 programming language that is used to test for and locate problems in BASIC09 source code procedures.

Default Value - The value assigned to a parameter or variable at system initialization.

Delimiter - A special character used to group or separate commands, statements, or values.

Device Descriptor - A program module (e.g., T1, T2, TP, or user-defined) that acts as an interface between the MiniCOP's OS-9 operating system and an I/O device.

Device Independent - A property of the BASIC09 programming language that permits procedures to be written without regard for specific end-devices. Simplifies programming and system upgrading.

Directory - A COPRAM file under which other files are organized; identified by the "d" attribute.

Directory Pointer - An internal OS-9 device that assigns a COPRAM directory as the current working directory. Two directory pointers exist in OS-9 -- the data directory pointer and the execution directory pointer. Certain commands and operations will default to either current working directory when a specific path is not identified.

Divided Task Concept - The technique of separating the function of the PLC's sequential logic control and other processing functions, which are transferred to the MiniCOP.

DMCS - Distributed Manufacturing Control System.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

E

Edit Mode - A mode of operation in the BASIC09 programming language that is used to create and edit BASIC09 procedures.

Execution Directory - The COPRAM directory that is used to store program modules and packed BASIC09 procedures. One of two possible current COPRAM directories allowed in OS-9 and identified by a directory pointer. Defaults to /D0/CMDS which is created during MiniCOP initialization.

Execution Mode - A mode of operation in the BASIC09 programming language that is used to execute or run BASIC09 procedures.

F

Field - Within a record, a specific area used for the storage of a particular type of data.

File - A collection of data, stored in system memory, or on floppy or hard disk via the Terminal Emulator.

File Manager - An OS-9 memory module that performs high-level logical I/O operations. There are two file managers (RBFMAN, SCFMAN) both located in level 3 of the OS-9 structure.

Foreground Processing - Execution of a primary task with high priority; used in conjunction with background processing.

H

Header - Contains descriptive information regarding a program module including size, offset, language, etc. Located or begins at the program module's least significant byte address.

Hierarchical - A feature of OS-9 that organizes files in a tree structure; related files are placed into their own directories.

Host 620 Platform Controller - The term given to the 620 platform controller that controls the I/O slots where the MiniCOP is installed.

I

I-Code - Intermediate Code - BASIC09 source code that is compiled as each line is entered or edited.

Interpreter - A programming language, operating system, or communication protocol translator. The SHELL is the OS-9 command interpreter.

I/O Redirection - Channeling data from the standard I/O path to an alternate file or device.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

L

Link Count - A measure of the number of times that a program module has been linked to main memory. When the link count equals zero, the module is unlinked.

LM - Logic Manager - a 620 platform controller that is capable of being connected to the TDC 3000 Universal Control Network (UCN) providing peer-to-peer communication with other UCN devices.

Loop - A set of instructions that are executed continuously under certain circumstances. At a predetermined event, the loop will be exited.

M

Main Memory - The 64 Kbyte area of MiniCOP memory program modules, displayable through the module directory. Main memory also holds the BASIC09 workspace when it is created.

Mantissa - The decimal part of a logarithm; the part of a floating point value that is not the exponent.

Modbus - A communications protocol developed by Modicon and that has been adopted as a "standard" protocol by many small industrial control system developers.

Modifier - A character (# ; > < &) that when processed by the SHELL affects program execution.

Modular Programming - The technique of developing application programs by creating several smaller programs each part of the whole; characterized by their flexibility and ease of debugging and editing.

Multitasking - The ability to concurrently perform several tasks within a single operating system, while sharing data between the tasks.

N

Nesting - To incorporate one program structure entirely with a like program structure. Loops can be nested within loops, and subroutines can be nested within subroutines.

O

Object Code- Any fully compiled code.

Operating System - The program responsible for managing all of a computer's resources, including memory, I/O, command processing, and processor time-sharing.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

O, Continued

Operator - A symbol that represents an operation (e.g. + - / *).

OS-9 - A UNIX-like operating system developed by Microware and used in the MiniCOP.

P

Pack - A command that condenses program source code by removing names, line numbers, and any other nonexecutable code; also protects the program from changes.

Page - A 256-byte block of main memory, defined to facilitate memory management.

Parameter - Data passed to a called procedure from the calling procedure, program module, or SHELL command line.

Parent - A process/procedure responsible for creating or calling a second (child) process/procedure (also known as the calling processor procedure).

Path - A software "channel" used for I/O operations with a specific file or device.

Pathlist - A list of names that identifies the file or device with which the path is communicating.

Personality Procedure - A program loaded into the MiniCOP allowing it to communicate with a 620 platform controller, Allen-Bradley PLC, or Modicon PLC via its ASYNC ports.

PLC - Programmable Logic Controller - a classification of industrial control systems. Originally, a solid-state replacement for hard-wired relay control systems that has advanced into many other areas such as advanced processing, communications, and control.

Position-Independent Code - Program code that can be stored anywhere in memory, allowing two or more programs to be resident at one time; permits modular programming.

Procedure - Another term for program. BASIC09 typically refers to programs as procedures.

Process - See *Task*.

Process ID - The identification number assigned to each process.

Program Module - An OS-9 system utility (e.g., DIR, LIST, SAVE) written in 6809 machine code; stored in the COPRAM execution directory or main memory.

Q

Queue - A list of operations waiting to be executed or serviced.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

R

RAMDisk - See *COPRAM*.

Real-Time Clock - A built-in feature on the system that keeps track of the current time and date.

Recursive - A procedure with the ability to call upon itself.
Reentrant Program - A program that can be loaded once and then used simultaneously by other programs.

Reserved Word - A group of characters (e.g., LIST, BYE, RUN) used by OS-9 or BASIC09 to specify a command or statement. These words cannot be used as names of strings or variables within a BASIC09 procedure.

RGB - Red, Green, Blue - a standard video signal protocol providing for both color and monochrome video output.

RUNB - A run-only system for executing previously-compiled (packed) BASIC09 procedures.

S

Sector - A 256-byte storage unit of COPRAM memory. Sectors are grouped together into one or more contiguous groups called segments that comprise file boundaries.

Segment - A contiguous block of COPRAM memory consisting of one or more sectors.

Sequential Execution - Performing tasks one after another. The next task does not begin until the previous one is completed.

Series 9000 - A 620 platform controller that provides both loop and logic control. Found in three models -- 9000, 9100, and 9200. Also found in an enhanced version in each of these three models, identified by the "E" suffix after the model number (e.g., 9200E).

SHELL - The program that interfaces the user and the OS-9 operating system.

Sleeping Process - A process suspended by a self-generated request for a specified interval of time or until the receipt of a signal.

Source Procedure - Noncompiled BASIC09 code; created in the BASIC09 edit mode of operation.

S-Record File - A text file containing records representing binary data in hexadecimal form.

Standard Error Output - The default error output path for the MiniCOP. The standard error output path defaults the same path as the standard output path, which is the Terminal Emulator video monitor connected to the terminal port, T1.

Standard Input - The default input path for the MiniCOP. The Terminal Emulator keyboard which is connected to the Terminal Port, T1.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

S, Continued

Standard Output - The default output path for the MiniCOP. The Terminal Emulator video monitor which is connected to the Terminal Port, T1.

Statement - A line of code in a BASIC09 procedure.
String - A sequence of characters treated as a single unit.

Syntax - The ordered structure of a statement or command that must be observed for proper execution.

T

Tape Port - The MiniCOP's ASYNC port where files are transferred between the MiniCOP itself and the Terminal Emulator; refers to port 2 or TP.

Task - An independent program that can run at the same time as other programs.

Terminal Emulator - A software package installed in an MS-DOS personal computer that allows the user to communicate with the 627 MiniCOP.

Terminal Port - The MiniCOP's ASYNC port where commands are received from the Terminal Emulator and standard output responses are passed back to the Terminal Emulator; refers to port 1 or T1.

Time-slicing - A method of allocating CPU resources by dividing the available time between process.

V

Variable - A quantity that can assume any given value.

W

Waiting Process - A process that has been suspended until another process terminates.

Workspace - A portion of main memory assigned by the operating system for the storing of BASIC09 procedures.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index

TOPIC	PAGE
$	387
20mA	22, 24-25
20mA Communications Cable	491
24 Line Mode	25
48 Line Mode	25
620 LCS Personality Procedure	289-295
620 Logic Controller System	4, 6
620 Personality Procedure	13
620 Platform Controllers	4, 7, 240
620-0090	39
620-0091	39
620-0093	39
620-15	8
620-1690	39
620-1693	39
620-1695	39
620-25	8
620-2590	39
620-35	8, 11
620-3590	39
620-36	8, 9
620-3691	39
621-9990C	39-40
621-9991C	39
621-9992C	39, 42
623 Loader/Terminal	59
623-6041	13, 61
623-6100	59
623-6150	59
623-8989	13
626-1011	13
626-1012	13, 289
626-1013	13
626-1014	13, 296
626-1016	13, 302
626-1018	3, 13
626-6041	59
626-9500	13
626-9501	13
626-9531	13
627-0005	13
627-0006	13
627-1002	3-4, 13-15
627-1002RC	4, 30, 44, 487
627-1003	3-4, 13-15
627-1003RC	4, 30, 44, 487
627-8991	13
627-8993	13
627-8997	13
628-3006	13
629-1021	13
629-3002	13, 18

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index *(Continued)*

TOPIC	PAGE
ACTIVE	105
Address Assignments	34-38
Allen-Bradley Personality Procedure	13, 296-301
Allen-Bradley PLCs	4, 11-12, 240, 296
Alternate Memory-Size Modifier (#)	120
Anonymous Directory Names	138
Application Notes	517-519
Arithmetic Operators	351
Array Declarations	401
Arrays	347-349
ASCII Code Character Set	509
Assembly Programming Disk	13
Assigned Properties	106-107
ASYNC Port Pin Assignments	490
Asynchronous Serial Communications Ports	4, 14, 19, 26, 95, 101, 103, 240
ATTR	111, 147, 158-159
Attributes	105, 146-147
Augmented Rack	42-43, 45, 503
Automayic Type Converions	346
BASE 0	388
BASE 1	388
BASIC09	102, 240
BASIC09 Commands	328
BASIC09 Error Codes	513-514
BASIC09 Mode	313
BASIC09 Mode Commands	384
BASIC09 Programming Language	3-4, 307
BASIC09 Statements	328
BASIC09 Tutorial	471
BASIC09 Workspace	318-324
BATT LED	18, 19
Battery	13, 16, 18, 22
Battery Backup	4, 14, 103
Baud Rates	103
BINEX	111, 160
BNC port	20
BOOLEAN Constants	331
BREAK	389
BUILD	111, 142, 145, 162
Built-In Commands	109
BYE	390
BYTE Constants	331
Cable and Conduit Routing	47
Cable Configuration	489
Cable Construction	489
Cable Spacing	50
Cabling to External Control/Sensing Devices	47-51
Cabling to the Terminal Emulator	52
CE Conformity	5
CHAIN	391
Characteristics	14

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TOPIC **PAGE**

CHD 111, 115, 128, 137, 163, 392
CHX 111, 115, 129, 137, 163, 393
Circuit Boards 16
CLEAR 111, 164
CLOSE 394
CMDS 89, 129-130
Command Interpreter 313
Command Line 109, 120
Command Line Parameters 113
Command Line Processing 114
Commands 386
Communication Functions 251-252, 254-255
Complex Data Structures 345
Components 13, 15
Concatenation 357
Concurrent Execution of Commands 124
Conduit Shielding 51
Conduit Spacing 49
conf1003 ..... 125, 525
Configurations 5, 21
Constants 330
CONT 395
Control Bits 251, 253
Control Statements 384
COPRAM 86, 89, 109, 125
COPY 111, 165
CPU Board 16-19, 21, 26
CREATE 396
Creating New BASIC09 Procedures 362
Cyclic Redundancy Check (CRC) 89, 98

D0 ..... 89, 96, 101, 118, 130
DATA 397
Data Directory 89, 107, 110, 128
Data Hiway 11
Data Representation 330
Data Table 240-244
Data Table Pointer 249-250
Data Types 338
Data Word 14
DataControl 125
DATE 111, 166
Debug Mode 317
Debug Mode Commands 384
Declaring Variables 333, 400
DEG/RAD 398
DEL 111, 116, 133-134, 167
DELDIR 111, 133-134, 168
DELETE 399
Device Descriptors 96, 101
Device Drivers 95-96
DIM 95, 400
DIP Switch Selectable Features 21
DIR 111, 116, 132, 146, 169, 402
Directories 90, 145

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index *(Continued)*

TOPIC	PAGE
Directory files	131, 139
DISK	95-96, 99, 100-101
DISPLAY	111, 170
DMCS-Connected Devices	10
DO	403
DSKBAK	111, 171
DUMP	111, 172
E/EDIT	404
Edit Mode	314
Editing Existing BASIC09 Procedures	363
Electrical Codes	14
END	405
ENDEXIT	406
ENDIF	406
ENDLOOP	406
ERROR	407
EX	111, 115, 173
EXBIN	111, 160, 161, 174
Executable Program Module Files	139
Execution Directory	89, 107, 110, 129
Execution Mode	315, 316
Execution Modifiers	120
EXITIF	408
Exponent Values Ranges	339
Expressions	350
F10 Exit	79
F3 Send	74
F4 Receive	75
F5 Verify	77
F7 Directory	78
F8 Loader	79
F9 Utility	79
FAULT LED	19
Files	89-90, 125, 139, 141, 145
File Organization	141
Floating Point Arithmetic	339
FOR/NEXT	409
Formatted Output Statement	385
FREE	111, 175
Frontplates	15
Full Duplex	22, 24, 25
Fundamental Operators	350
Fuse	13-14
GET BIT	248
GET REGISTER	248
GET/PUT	410
GOSUB/RETURN	413
GOTO	412
Half Duplex	22, 24-25
Help Screens (ALT-F10)	80-82
Hiway Interface Module	11

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index *(Continued)*

TOPIC	PAGE
Host 620 Platform Controller	240, 249
Hot Insertion/Removal	44
Humidity	14
I-code	311
I/O Device Descriptors	118
I/O Paths	358
I/O Ports	96
I/O Redirection Modifiers (<, >, >>)	121-122
I/O Statements	385
IDENT	176
IF (VERSION 1)	414
IF (VERSION 2)	415
IN/OUT LEDS	18
Inherited Properties	106, 107
INIT	95, 99
INKEY	111, 177
INPUT	416
Input/Output Manager	95
Input/Output Operations	358
Installation	27-28, 31-33
Integer Constants	331
Integer Values	340
Interfacing	240
Interleaved Workspaces	321
Internal Data Statements	385
Interrupts	102, 103
IOMAN	95, 99
IPC 621 Driver Utility	244-246
IPC620	294
IRQ	103
Jumper-Selectable Features	26
K4	96, 101, 103, 118
Kernal	95, 99
KEYBD	19
Keyboard Jack	25
KILL	111, 115, 178, 417
LED Indicators	18
Line Editing Commands	366
Line Numbers	327
LINK	112, 148, 152, 179
LIST	112, 116, 180, 420
Listing BASIC09 Procedures	365
Listing Source Code Procedures to a Printer via the Terminal Emulator	379
LOAD	112, 148-150, 181, 421
Loading Program Modules into Memory	108
Loading Source Code Procedures from COPRAM	377
Loading Source Code Procedures from the Terminal Emulator	378
Logic Manager	4, 7-8, 10
Logical Operators	352
Logical Sector Number	90
LOOP/ENDLOOP/EXITIF/ENDEXIT	422

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TOPIC **PAGE**

Main Memory 86-87, 93, 97
MAKDESC 101, 112, 118, 182-184
MAKDIR 112, 185
Mathematical Conversions 521
MDIR 112, 153, 186, 187
MEM 424
MEMORY 14, 107
Memory Fragmentation 153-155
Memory Module 97-99
Memory Modules 94
Memory Structure 86
MERGE 112, 188
MFREE 112, 153, 189
MINIABDH 300
MiniCOP/OS-9 TUTORIAL 210
MINIMMBUS 304
Modbus Personality Procedure 13, 302-305
Modicon PLCs 4, 12, 240, 302
Modifiers 114, 120
Modular Keyboard Port 19
Modular Programming 325
Module Body 98
Module Directory 87-88, 97
Module Directory Listing 98
Module Header 98
Moving the Edit Pointer 364
Multidrop Communication Network Configuration 22, 24, 25, 499
Multitasking 104, 109
Multitasking Modifier (&) 124

Naming OS-9 Files 140
Nested Workspaces 321
NEXT 425
Noise Test 14
Number System Conversions 521
Numeric constants 330
Numeric variable 332

ON ERROR GOTO 426
ON GOSUB 427
ON GOTO 427
OPEN 428
Operating Temperature 14
Operator Precedence 354
Operators 350
Options 13
OS-9 Assembly Language 3-4, 240
OS-9 Error Codes 515-516
OS-9 Operating System 4, 85, 94, 109
OS-9 System Commands 89, 157
OS-9 Systems Commands Disk 13
OS-9 Systems Commands Tape 13
OS9p1 ..... 95, 99
OS9p2 ..... 95, 99
OS9p3 ..... 95, 99

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index *(Continued)*

TOPIC	PAGE
P (-P)	115
PACK	429
Packed BASIC09 Procedure	109, 141
Packed Code	311
Packing Source Code Procedures to COPRAM	375
Packing Source Code Procedures to the Terminal Emulator	376
Parallel I/O Rack	22
Parallel Programming Block Time Delays	286-288
Parallel Workspaces	323
PARAM	430
Parameter Passing	335
Parameter Variables	335
Passing Parameters by Reference	337
Passing Parameters by Value	337
Pathlist	90, 125, 128, 129, 135, 138
Paths	117
PAUSE	431
PD	112, 128-129, 190
PEEK	356
Personality Procedures	4, 12-13, 240, 289-306, 523
Point-to-Point	22, 24-,25
POKE	432
Port 1	22
Port 2	24
Port 3	24
Port 4	25
PORT/MEMORY board	16-19, 23-24
Power	14
Power Supply Ground	29
PPINT	294, 300, 304
PRGMB	527
Primary module	106
PRINT	433
PRINT TAB	434
PRINT USING	435
PROC	124
PROCEDURE	114
Procedure File	109, 139, 143
Procedure Names	327
Process ID number	105, 107
Process States	105
PROCS	112, 191
Product Registration Label	27
Program	114
Program Modules	141
Program Parameters	120
Programming Plocks	251-252, 254-255
Programming Blocks, Parallel	256, 268
Programming Blocks, Serial	269-285
PULL Data from the MiniCOP	264-265, 280-282
PUSH Data to the MiniCOP	262-263, 278-279
PUT	142, 444
PUT BIT	248
PUT REGISTER	248

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index *(Continued)*

TOPIC	PAGE
Q	445
R Suffix	487
Rack Type Considerations	39
RAD	446
Random Access File Operations	360
Random Access Files	139, 142
Random Block File Manager (RBF)	90, 95, 99
READ (External File Operation)	447
READ (Internal Constant Operation)	448
Read the Real-time Clock	256-257, 270-272
Real Constants	330
Real-Time Clock	4, 14, 95, 99, 102-103, 256, 258, 270, 273
Relational Operators	353
REM	449
RENAME	112, 192, 450
REPEAT/UNTIL	451
Reserved Words	329
RESET	112, 193, 194
Reset the Data Table Pointer	266, 283
Reset the MiniCOP	268, 285
Resistor Pack Update	503-507
RESTORE	452
RETURN	453
RGB Video	4
Root Directory	89, 125
RS232	22, 24-25
RS232 Full-Duplex Communications Cable	493
RS232 Half-Duplex Communications Cable	492
RS422/485	22, 24-25
RS422/485 Full-Duplex Multidrop Communications Cable	497
RS422/485 Full-Duplex Point-to-Point Communications Cable	495
RS422/485 Half-Duplex Multidrop Communications Cable	496
RS422/485 Half-Duplex Point-to-Point Communications Cable	494
RUN	454
RUN Command	454
RUN LED	18
RUN Statement	455
RUNB	102, 507, 508
Running BASIC09 Procedures	369
SAVE	112, 148, 151, 195-196, 456
Saving Source Code Procedures to COPRAM	373
Saving Source Code Procedures to the Terminal Emulator	374
SC6551	95-96, 99-101
SCF	95, 99
Second	102, 103
Sector	141
SEEK	142, 457
Sequential Character File Manager (SCF)	95, 100
Sequential Execution	109
Sequential Execution Modifier (;)	123

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index *(Continued)*

TOPIC	PAGE
Sequential File Operations	359
Serial Communication Cables	491-497
Serial Connectors	4
Serial I/O Rack	22
Series 9000 Loop and Logic Controller (Series 9000e)	4, 7-8
Set Data Table Pointer	261
Set the Data Table Pointer	260, 275-277
Set the Pointer Direction	267, 284
Set the Real-Time Clock	258-259, 273-274
SETIME	112, 197
SETPORT	112, 198
SETPR	112, 115, 199
SHELL	102, 106, 109-110, 112-115, 143, 200, 459
SHELL Command Line	116
Shipping Configuration	21
SLEEP	105, 112, 201
Source Code	311
Special Condition Flag	246, 247
Special Function Operators	355
Stand-Alone	12
Standard Error Output	121, 144
Standard Input and Output Paths	107, 144
Standard Input Path	109, 121
Standard Output Path	121
Standard Rack	40, 41
STATE	460
STEP	461
STOP	462
Storage Temperature	14
String Constant	331
String Editing Commands	367
String Variable	332
Subdirectories	132
Synchronization	20
SYNTAX	56, 157, 386
Syntax Notation	329
SYSGO	95, 99
System Device Directory	125
System Ground	28
System Program Modules	102
T (-T)	115
T1	96, 101, 103, 118
T2	96, 101, 103, 118
T3	96, 101, 103, 118
T4	96, 101, 103, 118
Tape Port	69, 71, 103
TDC 3000	7, 10-11
Terminal Emulator	60, 62-71, 109
Terminal Emulator Disk	13
Terminal Emulator, Changing Menu Parameters	73
Terminal Emulator, Configuration	69, 71-80
Terminal Emulator, Installation	64
Terminal Emulator, Parameter Definitions	72
Terminal Emulator, Start-up	68

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index *(Continued)*

TOPIC	PAGE
Terminal Port	69, 71
Text Files	128, 139, 142
THEN	463
Tick	102-104
Time Delays	286-288
Time-Slicing	104
Tips to Increase Procedure Execution	381
Tips to Reduce Memory Usage	380
TLOAD	112, 150, 202
TMODE	112, 203, 204
Toggle Switch	18, 26
TP	96, 101, 103, 118
Tray Spacing	49, 50
TRON/TROFF	464
TYPE	142, 465
Unified I/O System	109, 117
UNLINK	112, 148, 152, 179, 205
Unprotected I/O Operations	156
UNTIL	467
Upper/Local Bus	44, 46, 503
User ID Number	107
V4	96, 101, 103, 118
Variable Storage	334
Variables	332, 335
VERIFY	112, 206
Video	95-96, 99-101
Video Attribute Register	57
Video Board	17, 25
Video Cables	26, 498
Video Cable Construction	498
Video Color Register	57
Video Command Codes	55
Video Connectors	14
Video Functions	53
Video Ports	20, 53
Video Signal Adjustment	54
Video Signal Standard	14
VideoDemo	125
W	115
WAITING	105
WHILE/DO	468
Working in the BASIC09 Mode	361
Working in the DEBUG Mode	371
Working in the EDIT Mode	363
Working in the EXECUTION Mode	369
Workspace	87-88, 318
WRITE	469
WRITE N BITs	248
WRITE N REGISTERs	248
X (-X)	115
XMODE	112, 207-208

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

READER COMMENTS

Honeywell's IAC Automation College welcomes your comments and suggestions to improve future editions of this and other publications.

You can communicate your thoughts to us either by **mail or fax** using this form, or a **toll-free telephone** call.

BY TELEPHONE: In the U.S.A. use our toll-free number 1*800-822-7673 (available in the 48 contiguous states except Arizona; in Arizona dial 1-602-313-5558).

BY FAX: Use this form; detach and fax this side to us at 1-602-313-4108. We would like to acknowledge your comments; please include your complete name and address.

BY MAIL: Use this form; detach, fold, tape closed, and mail to us. We would like to acknowledge your comments; please include your complete name and address.

Title of Publication: **627 MiniCOP and BASIC09 User Manual**

Publication Number: **627-8991 Rev. F** Issue Date: **12/95**

Writer: **Ray Snyder**

COMMENTS: _____

RECOMMENDATIONS: _____

NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

(If returning by mail, please tape closed; Postal regulations prohibit use of staples.)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Communications concerning technical publications should be directed to:

Automation College
Industrial Automation and Control
Honeywell Inc.
2820 West Kelton Lane
Phoenix, Arizona 85023

FOLD FOLD

From: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE USA

BUSINESS REPLY MAIL

FIRST CLASS **PERMIT NO. 4332** PHOENIX, ARIZONA

POSTAGE WILL BE PAID BY ....

Honeywell

Industrial Automation and Control
2820 West Kelton Lane
Phoenix, Arizona 85023

Attention: Manager, Quality

FOLD FOLD

Cut Along Line

Additional Comments:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Honeywell

Helping You Control Your World

Industrial Automation and Control
Honeywell Inc.
1100 Virginia Drive
Ft. Washington, Pennsylvania 19034

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP