ProSoft TECHNOLOGY

Where Automation Connects.

inRAx®

MVI56-BAS

ControlLogix Platform

BASIC Module (DB/BAS Compatible)

March 2, 2022

USER MANUAL

Your Feedback Please

We always want you to feel that you made the right decision to use our products. If you have suggestions, comments, compliments or complaints about our products, documentation, or support, please write or call us.

How to Contact Us

ProSoft Technology, Inc.

+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com
support@prosoft-technology.com

MVI56-BAS User Manual
For public use.

March 2, 2022

ProSoft Technology® Product Documentation

In an effort to conserve paper, ProSoft Technology no longer includes printed manuals with our product shipments. User Manuals, Datasheets, Sample Ladder Files, and Configuration Files are provided at: www.prosoft-technology.com

For professional users in the European Union

If you wish to discard electrical and electronic equipment (EEE), please contact your dealer or supplier for further information.

Warning – Cancer and Reproductive Harm – www.P65Warnings.ca.gov

Agency Approvals & Certifications

Please visit our website: www.prosoft-technology.com

Important Installation Instructions

Power, Input, and Output (I/O) wiring must be in accordance with Class I, Division 2 wiring methods, Article 501-4 (b) of the National Electrical Code, NFPA 70 for installation in the U.S., or as specified in Section 18-1J2 of the Canadian Electrical Code for installations in Canada, and in accordance with the authority having jurisdiction. The following warnings must be heeded:

WARNING - EXPLOSION HAZARD - SUBSTITUTION OF COMPONENTS MAY IMPAIR SUITABILITY FOR CLASS I, DIV. 2;
WARNING - EXPLOSION HAZARD - WHEN IN HAZARDOUS LOCATIONS, TURN OFF POWER BEFORE REPLACING OR WIRING MODULES
WARNING - EXPLOSION HAZARD - DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NON-HAZARDOUS.
THIS DEVICE SHALL BE POWERED BY CLASS 2 OUTPUTS ONLY.

MVI (Multi Vendor Interface) Modules

WARNING - EXPLOSION HAZARD - DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NON-HAZARDOUS.

AVERTISSEMENT - RISQUE D'EXPLOSION - AVANT DE DÉCONNECTER L'ÉQUIPEMENT, COUPER LE COURANT OU S'ASSURER QUE L'EMPLACEMENT EST DÉSIGNÉ NON DANGEREUX.

Warnings

North America Warnings

Power, Input, and Output (I/O) wiring must be in accordance with Class I, Division 2 wiring methods, Article 501-4 (b) of the National Electrical Code, NFPA 70 for installation in the U.S., or as specified in Section 18-1J2 of the Canadian Electrical Code for installations in Canada, and in accordance with the authority having jurisdiction. The following warnings must be heeded:

A Warning - Explosion Hazard - Substitution of components may impair suitability for Class I, Division 2.
B Warning - Explosion Hazard - When in hazardous locations, turn off power before replacing or rewiring modules.
C Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.

Avertissement - Risque d'explosion - Avant de déconnecter l'équipement, couper le courant ou s'assurer que l'emplacement est désigné non dangereux.

D Suitable for use in Class I, Division 2 Groups A, B, C and D Hazardous Locations or Non-Hazardous Locations.

ATEX Warnings and Conditions of Safe Usage

Power, Input, and Output (I/O) wiring must be in accordance with the authority having jurisdiction.

A Warning - Explosion Hazard - When in hazardous locations, turn off power before replacing or wiring modules.
B Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.
C These products are intended to be mounted in an IP54 enclosure. The devices shall provide external means to prevent the rated voltage being exceeded by transient disturbances of more than 40%. This device must be used only with ATEX certified backplanes.
D DO NOT OPEN WHEN ENERGIZED.

Battery Life Advisory

The MVI46, MVI56, MVI56E, MVI69, and MVI71 modules use a rechargeable Lithium Vanadium Pentoxide battery to backup the real-time clock and CMOS. The battery should last for the life of the module. The module must be powered for approximately twenty hours before the battery becomes fully charged. After it is fully charged, the battery provides backup power for the CMOS setup and the real-time clock for approximately 21 days. When the battery is fully discharged, the module will revert to the default BIOS and clock settings.

Note: The battery is not user replaceable.

Markings

Electrical Ratings

- Backplane Current Load: 800 mA @ 5.1 Vdc; 3 mA @ 24 Vdc
- Operating Temperature: 0°C to 60°C (32°F to 140°F)
- Storage Temperature: -40°C to 85°C (-40°F to 185°F)
- Shock: 30 g, operational; 50 g, non-operational; Vibration: 5 g from 10 Hz to 150 Hz
- Relative Humidity: 5% to 95% with no condensation
- All phase conductor sizes must be at least 1.3 mm(squared) and all earth ground conductors must be at least 4mm(squared).

Label Markings

ATEX

II 3 G

EEx nA IIC T6

0°C <= Ta <= 60°C

cULus

E183151

Class I Div 2 Groups A,B,C,D

T6

-30°C <= Ta <= 60°C

Contents

Guide to the MVI56-BAS User Manual

Function		Section to Read	Details
Introduction (Must Do)	→	Start Here (page 15)	This section introduces the customer to the module. Included are: package contents, system requirements, hardware installation, and basic configuration.
Diagnostic and Troubleshooting	→	Diagnostics and Troubleshooting	This section describes Diagnostic and Troubleshooting procedures.
Reference Product Specifications	→	Reference (page 213) Product Specifications (page 214)	These sections contain general references associated with this product and its Specifications..
Support, Service, and Warranty Index	→	Support, Service and Warranty (page 229) Index	This section contains Support, Service and Warranty information. Index of chapters.

1 Start Here

In This Chapter

To get the most benefit from this User Manual, you should have the following skills:

- **Rockwell Automation® RSLogix™ software:** launch the program, configure ladder logic, and transfer the ladder logic to the processor
- **Microsoft Windows:** install and launch programs, execute menu commands, navigate dialog boxes, and enter data
- **Hardware installation and wiring:** install the module, and safely connect BASIC and ControlLogix devices to a power source and to the MVI56-BAS module's application port(s)

1.1 System Requirements

The MVI56-BAS module requires the following minimum hardware and software components:

- Rockwell Automation ControlLogix™ processor, with compatible power supply and one free slot in the rack, for the MVI56-BAS module. The module requires 800 mA of available power.
- Rockwell Automation RSLogix 5000 programming software version 2.51 or higher
- Rockwell Automation RSLinx communication software
- Pentium® II 450 MHz minimum. Pentium III 733 MHz (or better) recommended
- Supported operating systems:
  - Microsoft Windows XP Professional with Service Pack 1 or 2
  - Microsoft Windows 2000 Professional with Service Pack 1, 2, or 3
  - Microsoft Windows Server 2003
- 128 Mbytes of RAM minimum, 256 Mbytes of RAM recommended
- 100 Mbytes of free hard disk space (or more based on application requirements)
- 256-color VGA graphics adapter, 800 x 600 minimum resolution (True Color 1024 × 768 recommended)
- ProSoft Configuration Builder, HyperTerminal or other terminal emulator program.

Note: You can install the module in a local or remote rack. For remote rack installation, the module requires EtherNet/IP or ControlNet communication with the processor.

1.2 Package Contents

The following components are included with your MVI56-BAS module, and are all required for installation and configuration.

Important: Before beginning the installation, please verify that all of the following items are present.

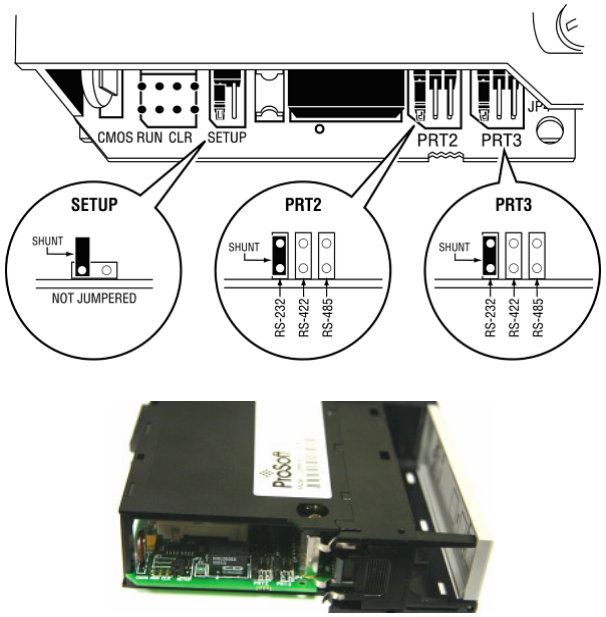
Qty.	Part Name	Part Number	Part Description
1	MVI56-BAS Module	MVI56-BAS	BASIC Module (DB/BAS Compatible)
1	Cable	Cable #15, RS232 Null Modem	For RS232 Connection to the CFG Port
3	Cable	Cable #14, RJ45 to DB9 Male Adapter cable	For DB9 Connection to Module's Port
2	Adapter	1454-9F	Two Adapters, DB9 Female to Screw Terminal. For RS422 or RS485 Connections to Port 1 and 2 of the Module

If any of these components are missing, please contact ProSoft Technology Support for replacement parts.

1.3 Setting Jumpers

If you use an interface other than RS-232 (default), you must change the jumper configuration to select the interface you wish to use. There are three jumpers located at the bottom of the module.

The following illustration shows the MVI56-BAS jumper configuration:



1. Set the PRT 2 (for application port 1) and PRT 3 (for application port 2) jumpers select RS232, RS422, or RS485 to match the wiring needed for your application. The default jumper setting for both application ports is RS-232.
2. The Setup Jumper acts as "write protection" for the module's flash memory. In "write protected" mode, the Setup pins are not connected, and the module's firmware cannot be overwritten. Do not jumper the Setup pins together unless you are directed to do so by ProSoft Technical Support.

1.4 Installing the Module in the Rack

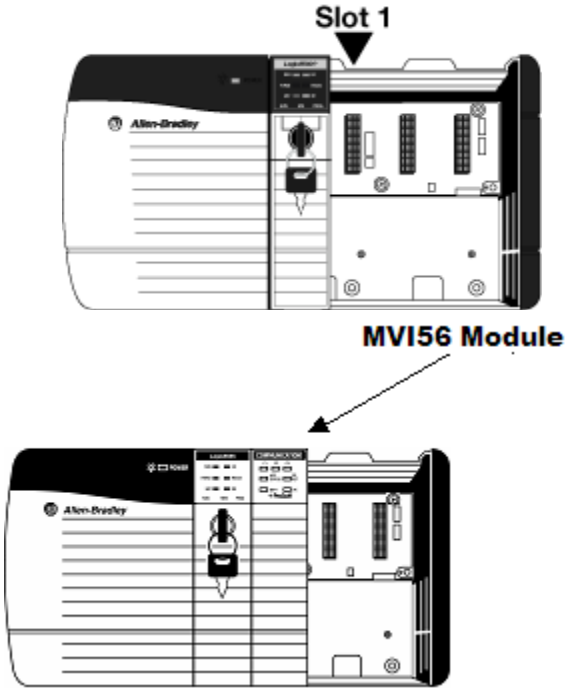
If you have not already installed and configured your ControlLogix processor and power supply, please do so before installing the MVI56-BAS module. Refer to your Rockwell Automation product documentation for installation instructions.

Warning: You must follow all safety instructions when installing this or any other electronic devices. Failure to follow safety procedures could result in damage to hardware or data, or even serious injury or death to personnel. Refer to the documentation for each device you plan to connect to verify that suitable safety procedures are in place before installing or servicing the device.

After you have checked the placement of the jumpers, insert MVI56-BAS into the ControlLogix chassis. Use the same technique recommended by Rockwell Automation to remove and install ControlLogix modules.

Warning: When you insert or remove the module while backplane power is on, an electrical arc can occur. This could cause an explosion in hazardous location installations. Verify that power is removed or the area is non-hazardous before proceeding. Repeated electrical arcing causes excessive wear to contacts on both the module and its mating connector. Worn contacts may create electrical resistance that can affect module operation.

1. Turn power OFF.
2. Align the module with the top and bottom guides, and slide it into the rack until the module is firmly against the backplane connector.



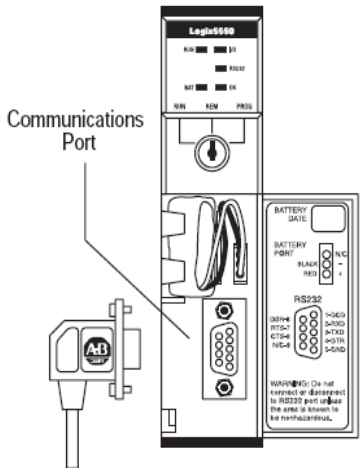
3 With a firm but steady push, snap the module into place.
4 Check that the holding clips on the top and bottom of the module are securely in the locking holes of the rack.
5 Make a note of the slot location. You must identify the slot in which the module is installed in order for the sample program to work correctly. Slot numbers are identified on the green circuit board (backplane) of the ControlLogix rack.
6 Turn power ON.

Note: If you insert the module improperly, the system may stop working, or may behave unpredictably.

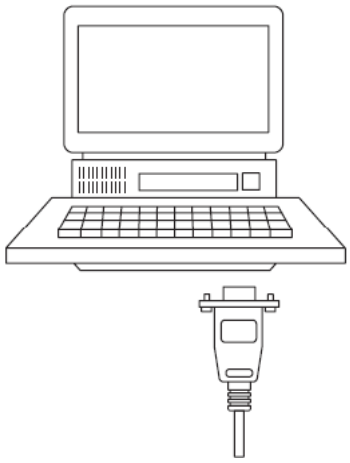
1.5 Connecting Your PC to the ControlLogix Processor

There are several ways to establish communication between your PC and the ControlLogix processor. The following steps show how to establish communication through the serial interface. It is not mandatory that you use the processor's serial interface. You may access the processor through whatever network interface is available on your system. Refer to your Rockwell Automation documentation for information on other connection methods.

1 Connect the right-angle connector end of the cable to your controller at the communications port.



2 Connect the straight connector end of the cable to the serial port on your computer.



1.6 Using the Sample Ladder Logic

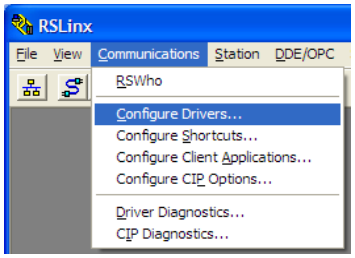
The sample program for your MVI56-BAS module includes custom tags, data types, and ladder logic for data I/O and status monitoring. For most applications, you can run the sample ladder program without modification, or, for advanced applications, you can incorporate the sample program into your existing application.

The version number appended to the file name corresponds with the firmware version number of your ControlLogix processor. The firmware version and sample program version must match.

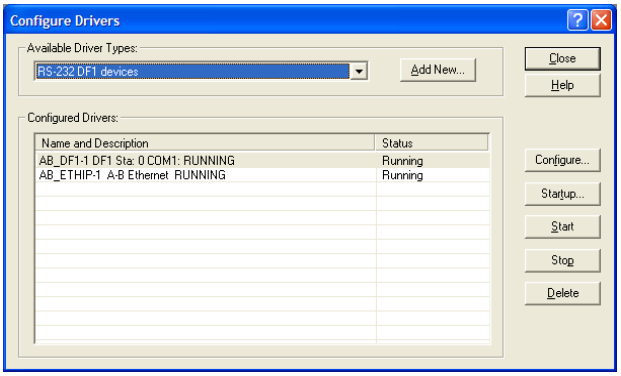
1.6.1 Configuring the RSLinx Driver for the PC COM Port

If RSLogix is unable to establish communication with the processor, follow these steps.

1. Open *RSLinx*.
2. Open the **COMMUNICATIONS** menu, and choose **CONFIGURE DRIVERS**.

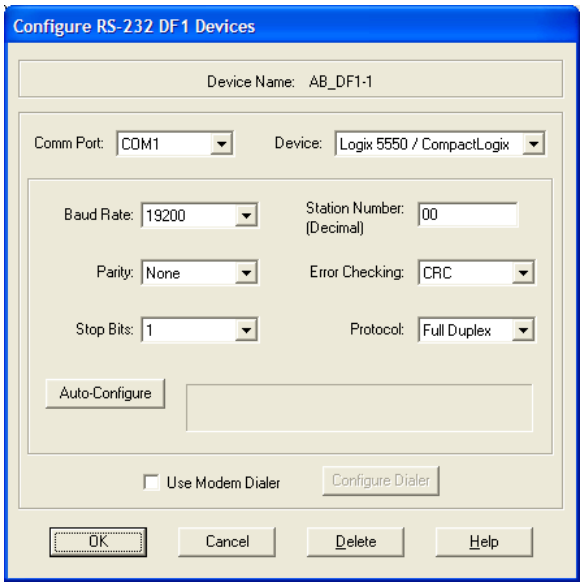


This action opens the *Configure Drivers* dialog box.



Note: If the list of configured drivers is blank, you must first choose and configure a driver from the *Available Driver Types* list. The recommended driver type to choose for serial communication with the processor is *RS-232 DF1 Devices*.

3 Click to select the driver, and then click **CONFIGURE**. This action opens the *Configure RS-232 DF1 Devices* dialog box.



4 Click the **AUTO-CONFIGURE** button. RSLinx will attempt to configure your serial port to work with the selected driver.

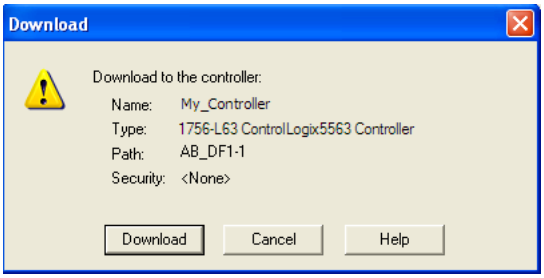
5 When you see the message *Auto Configuration Successful*, click the **OK** button to dismiss the dialog box.

Note: If the auto-configuration procedure fails, verify that the cables are connected correctly between the processor and the serial port on your computer, and then try again. If you are still unable to auto-configure the port, refer to your RSLinx documentation for further troubleshooting steps.

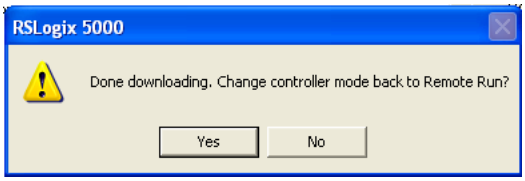
1.7 Downloading the Sample Program to the Processor

Note: The key switch on the front of the ControlLogix processor must be in the REM or PROG position.

1. If you are not already online with the processor, open the *Communications* menu, and then choose **DOWNLOAD**. RSLogix 5000 will establish communication with the processor. You do not have to download through the processor's serial port, as shown here. You may download through any available network connection.
2. When communication is established, RSLogix 5000 will open a confirmation dialog box. Click the **DOWNLOAD** button to transfer the sample program to the processor.



3. RSLogix 5000 will compile the program and transfer it to the processor. This process may take a few minutes.
4. When the download is complete, RSLogix 5000 will open another confirmation dialog box. If the key switch is in the REM position, click **OK** to switch the processor from PROGRAM mode to RUN mode.

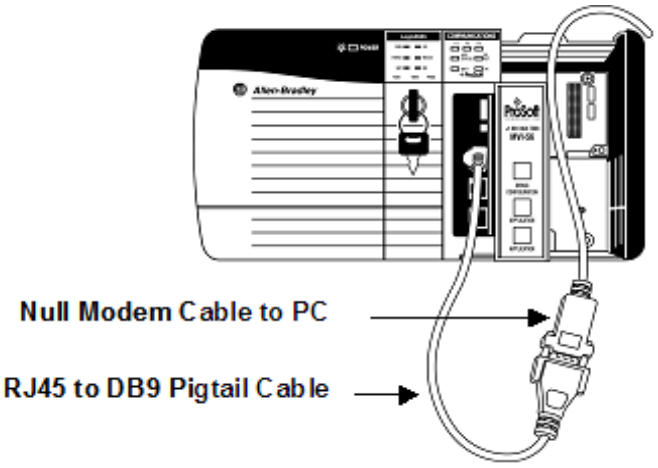


Note: If you receive an error message during these steps, refer to your RSLogix documentation to interpret and correct the error.

1.8 Connecting Your PC to the Module

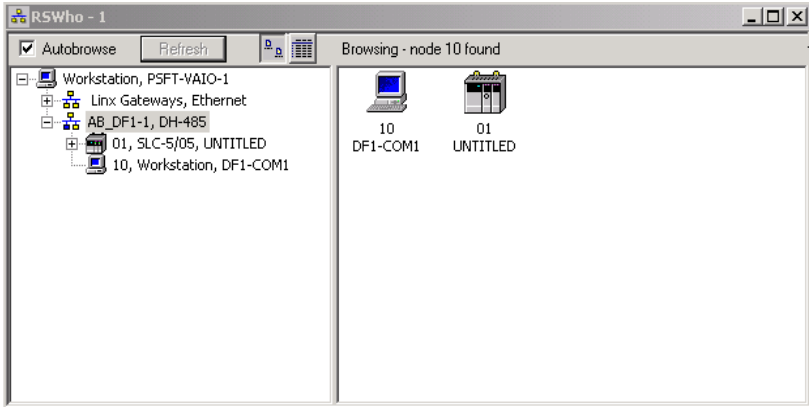
With the module securely mounted, connect your PC to the **Configuration/Debug** port using an RJ45-DB-9 Serial Adapter Cable and a Null Modem Cable.

1. Attach both cables as shown.
2. Insert the RJ45 cable connector into the *Config/Debug* port of the module.
3. Attach the other end to the serial port on your PC.

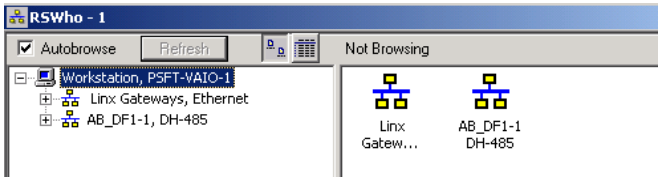


The communication port driver in *RSLinx* can occasionally prevent other applications from using the PC's COM port. If you are not able to connect to the module's configuration/debug port using *ProSoft Configuration Builder (PCB)*, *HyperTerminal* or another terminal emulator, follow these steps to disable the *RSLinx* driver.

1. Open *RSLinx* and go to **COMMUNICATIONS** > **RSWHO**.
2. Make sure that you are not actively browsing using the driver that you wish to stop. The following shows an actively browsed network.



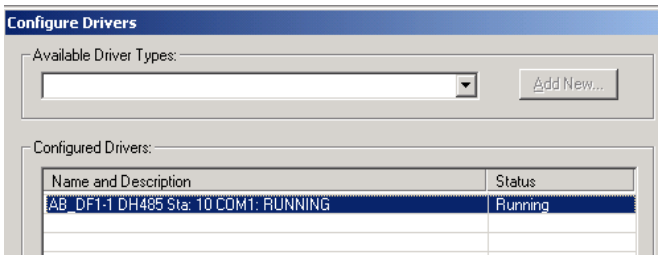
3 Notice how the DF1 driver is opened, and the driver is looking for a processor on Node 1. If the network is being browsed, then you will not be able to stop this driver. To stop the driver your *RSWho* screen should look like this:



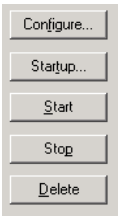
Branches are displayed or hidden by clicking on the  or the  icons.

4 When you have verified that the driver is not being browsed, go to **COMMUNICATIONS** > **CONFIGURE DRIVERS**.

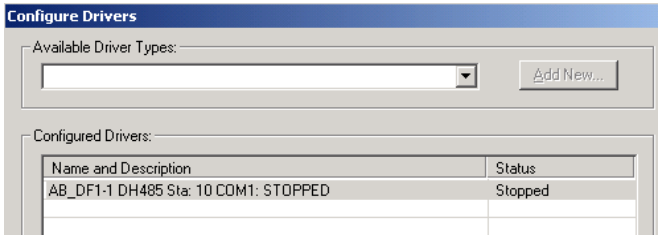
You may see something like this:



5 If you see the status as running, you will not be able to use this COM port for anything other than communication to the processor. To stop the driver press the **STOP** button on the side of the window:



6 After you have stopped the driver you will see the following.



7 You may now use the COM port to connect to the *Config/Debug* port of the module.

Note: You may need to shut down and restart your PC before it will allow you to stop the driver (usually only on *Windows NT* machines). If you have followed all of the above steps, and it will not stop the driver, then make sure you do not have *RSLogix* open. If *RSLogix* is open, you will not be able to stop the DF1 driver. If *RSLogix* is not open, and you still cannot stop the driver, then reboot your PC.

2 Module Configuration

In This Chapter

This section contains the setup procedure, data, and ladder logic for successful application of the MVI56-BAS module. Each step in the setup procedure is defined in order to simplify the use of the module.

2.1 Installing and Configuring the Module

This chapter describes how to install and configure the module to work with your application. The configuration process consists of the following steps.

1 Use RSLogix 5000 to identify the module to the processor and add the module to a project.

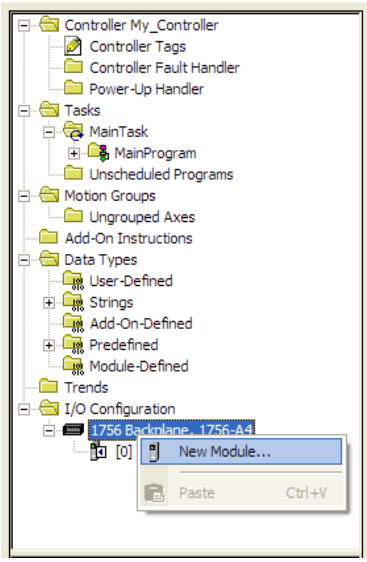
Note: The RSLogix 5000 software should be in "off-line" mode to add the module to a project. Although some newer versions of RSLogix 5000 may allow new modules to be added while in "online" mode, it is always considered safer to add new modules off-line and test the new configuration in a test system before putting the modified program online.

2 Modify the example ladder logic to meet the needs of your application, and download the ladder logic to the processor.

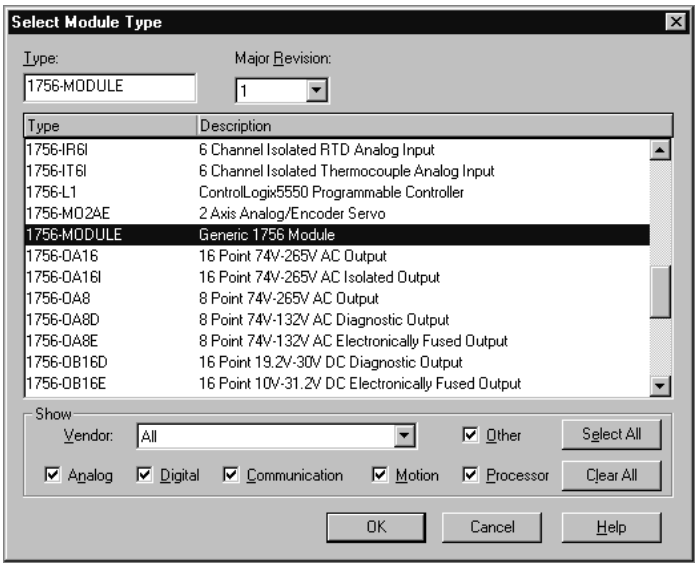
Note: If you are installing this module in an existing application, you can copy the necessary elements from the example ladder logic into your application.

The rest of this chapter describes these steps in more detail.

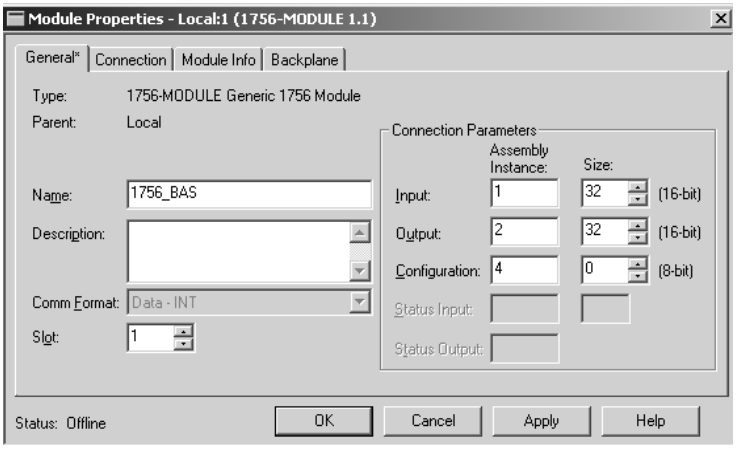
First, define the module to the system. Right-click the mouse button on the I/O Configuration option in the Controller Organization window to display a pop-up menu. Select the New Module option from the I/O Configuration menu.



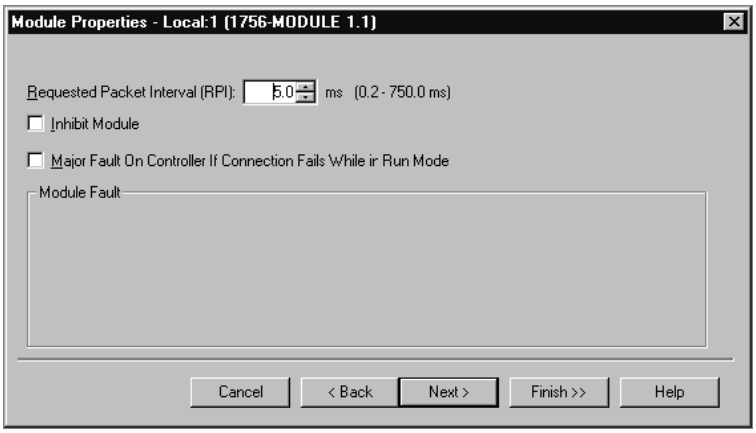
This action opens the following dialog box.



Select the 1756-Module (Generic 1756 Module) from the list and click OK. The following dialog box is displayed.

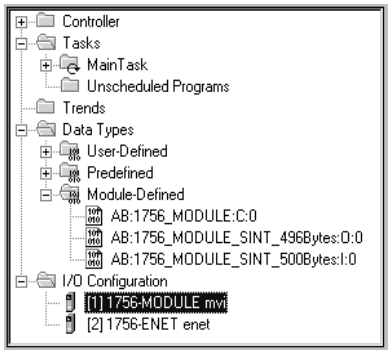


Fill in the dialog box as shown adjusting the Name, Description and Slot options for your application. You must select the **Comm** F**ormat** as **Data - INT** in the dialog box. Failure to set the **Assembly Instance** and **Size** values correctly will result in a module that will not communicate over the backplane of the ControlLogix rack. Click Next to display the next dialog box.



Select the Request Packet Interval value for scanning the I/O on the module. This value represents the minimum frequency that the module will handle scheduled events. This value should not be set to less than 1 millisecond. Values between 1 and 10 milliseconds should work with most applications.

After completing the module setup, the Controller Organization window will display the module's presence. The data required for the module will be defined to the application, and objects will be allocated in the Controller Tags data area. An example of the Controller Organization window is shown below.



Download the new application to the controller and place the processor in run mode. If all the configuration parameters are set, the module's Application LED (APP LED) should remain off and the backplane activity LED (BP ACT) should blink rapidly. Refer to Diagnostics and Troubleshooting if you encounter errors. Attach a computer or terminal to PRT1 on the module and look at the status of the module using the Configuration/Debug Menu in the module.

3 BASIC Programming

In This Chapter

This section presents an overview of the BASIC-52 implementation for the MVI56-BAS. It provides the information necessary to write a BASIC program for the MVI56-BAS.

3.1 The Argument Stack

The results of all BASIC expressions or operations are stored in the argument stack. The argument stack holds up to 20 values.

In order to insert a value in the top of the argument stack, the `PUSH` command must be used.

In order to retrieve the value from the top of the argument stack, a `POP` command must be used. It will remove the current value from the top of the argument stack moving the following value in its place.

For example, entering in the program port command line:

```
>PUSH 1
>PUSH 2
>PUSH 3
>POP A
>POP B
>POP C
>PRINT A
3
>PRINT B
2
>PRINT C
1
```

Use CALL 109 to print the first 9 values in the argument stack to aid in troubleshooting.

For example:

```
>PUSH 1
>PUSH 2
>PUSH 3
>CALL 109
000 1
001 2
002 3
Argument Stack Pointer is 3
```

3.2 Using CALLs

There are several pre-defined functions in BASIC referred to as CALLs. Each CALL can have zero or more input parameters and return zero more output values. The module uses the argument stack to read the input parameters and write the output results. The input parameters and the output result(s) are both located in the argument stack. `PUSH` and `POP` commands are used with CALL functions that require input and output variables.

For example, if a CALL function requires 3 input parameters and returns 2 output values, it should be used in a BASIC program in the following way:

```
PUSH 4
PUSH 2
PUSH 4
CALL [CALLNUMBER]
POP D
POP E
```

The variables D and E will store the 2 output values returned by this CALL. The BASIC CALLs Syntax section provides information about each call supported by the MVI56-BAS.

3.3 Using Strings

Strings are a group of characters that can be accessed in a BASIC program or command line by the following syntax:

```
$(num)
```

Where num varies between 0 and 254.

Before using strings, you must allocate enough memory using the STRING command:

```
STRING(A,B)
```

Where:

A = total number of bytes for the all strings

B = maximum number of bytes for each string

The MVI56-BAS requires an extra byte for each string and an additional overhead byte.

Example:

```
10 STRING 307, 50
20 $(0) = "TEST 1"
30 $(1) = "TEST 2"
40 $(2) = "TEST 3"
50 $(3) = "TEST 4"
60 $(4) = "TEST 5"
70 $(5) = "TEST 6"
```

The STRING command allocated space for 6 strings (50 bytes each) plus one byte for each string and one byte overall.

There are several BASIC CALLs that allow string manipulation. Refer to String CALLs (page 147).

3.4 ControlLogix Processor Interrupt

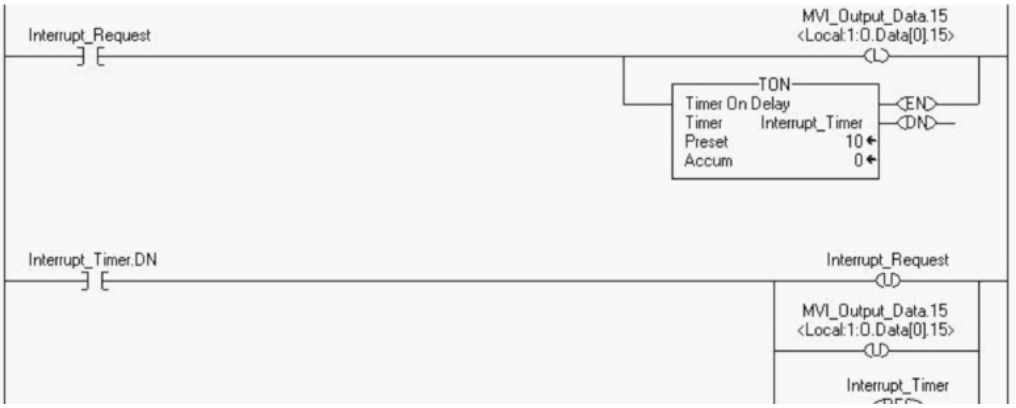
The MVI56-BAS allows the ControlLogix to interrupt the current BASIC program using CALL 20 (Enable Processor Interrupt). When word 0 bit 15 in the output image (Local:x:O:Data[0].15) toggles from OFF to ON, the program jumps to the line number specified by the CALL 20 parameter.

Use the `RETI` command in order to return to the point the program was before being interrupted.

The following BASIC program shows how to use CALL 20:

```
10 PUSH 50: REM JUMPS TO LINE NUMBER 50
20 CALL 20
30 GOTO 30
35 PRINT "LINE 35"
40 PRINT "LINE 40"
50 PRINT "LINE 50"
60 PRINT "LINE 60"
70 RETI
80 END
```

While the program is running, the following ladder logic is executed:



The result is:

```
LINE 50
LINE 60
```

To disable the processor interrupt, use CALL 21 (Disable Processor Interrupt). This routine has no arguments and does not return any value.

Refer to BASIC CALLs Syntax (page 105) for more information about CALLs 20 and 21.

4 Backplane Data Transfer

In This Chapter

This section describes how to transfer data between the MVI56-BAS and the ControlLogix processor.

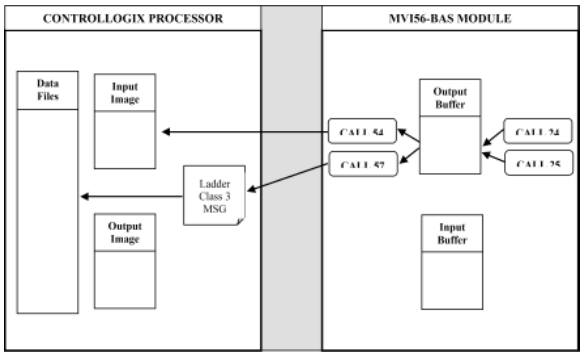
4.1 Data Transfer from Output Buffer to CLX Processor

The MVI56-BAS module output buffer can be used to transfer data to the ControlLogix processor.

The first step is to move data to the output buffer using CALLs 24 and 25. These CALLs convert BASIC floating point data into integer values truncating the fractional part. The result is stored in the output buffer between addresses 0 and 231.

After the data has been moved to the output buffer, there are 2 ways to transfer it to the ControlLogix; writing to the CLX input image file (CALL 54) or using MSG instructions (CALL 57).

The following illustration shows how to transfer data between the MVI56-BAS output buffer and the ControlLogix processor.



The output buffer is divided into areas depending on the destination of the data transferred. The output buffer addressing is shown in the following table.

Address	Definition
0 to 39	DH-485 Common Interface File - data read by other devices
40 to 99	Reserved
100 to 199	Data read by the CLX using MSG instruction
200 to 231	Data transferred to the CLX Input Image file

CALL 54 transfers words 200 to 207 (8 words) from the module output buffer to the CLX input image (Local:x:I.Data[ ]). No ladder logic is required to perform the data transfer.

MVI56-BAS Output Buffer

CLX Input Image	Address	Value
Local:x:I.Data[0]	200	Set by CALL 24 or 25
Local:x:I.Data[1]	201	Set by CALL 24 or 25
Local:x:I.Data[2]	202	Set by CALL 24 or 25
Local:x:I.Data[3]	203	Set by CALL 24 or 25
Local:x:I.Data[4]	204	Set by CALL 24 or 25
...	...	Set by CALL 24 or 25
Local:x:I.Data[30]	230	Set by CALL 24 or 25
Local:x:I.Data[31]	231	Set by CALL 24 or 25

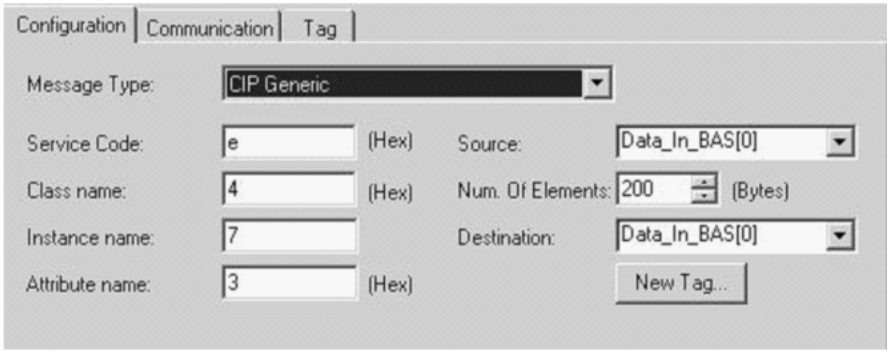
Note: The upper three bits in Word 200 are reserved and cannot be modified. The lower thirteen bits can be used. The upper three bits provide module status information to the ControlLogix, Where

Address	Definition
bit 0	Modifiable
bit 1	Modifiable
bit 2	Modifiable
bit 3	Modifiable
bit 4	Modifiable
bit 5	Modifiable
bit 6	Modifiable
bit 7	Modifiable
bit 8	Modifiable
bit 9	Modifiable
bit 10	Modifiable
bit 11	Modifiable
bit 12	Modifiable
bit 13	Battery status bit (0-battery good 1 -low voltage)
bit 14	Reserved
bit 15	MVI56-BAS program mode (0-run mode , 1-command)

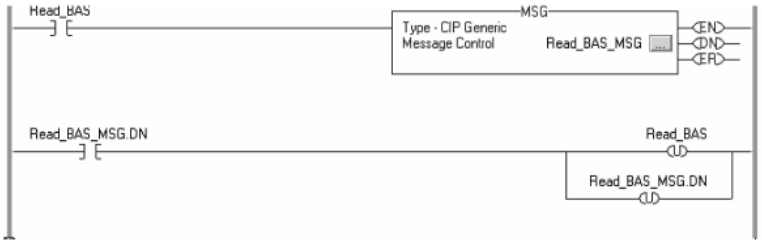
CALL 57 will transfer data from the MVI56-BAS output buffer to a CLX data file using a Class 3 Message. In order to accomplish this, ladder logic is required

```
The following BASIC program transfers data to the output buffer using CALL 24
and executes CALL 57:
1 REM CALL 57
10 FOR I=100 TO 149
30 PUSH 200-I: REM VALUE TO BE TRANSFERRED
40 PUSH I: REM ADDRESS IN OUTPUT BUFFER
50 CALL 24
60 NEXT I
70 PUSH 50: REM NUMBER OF WORDS TO BE TRANSFERRED
80 CALL 57:
90 POP S: REM CALL 57 RESULT
130 END
```

Following CALL 57 execution, the ladder logic should perform a MSG instruction in order to read the data setup by CALL 57. The MSG instruction should be configured as shown in the following example:



The following ladder logic can be used to read the data:

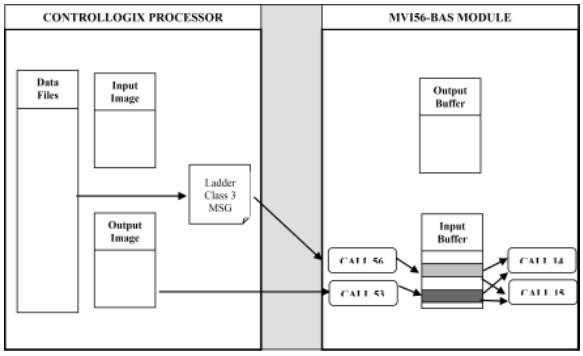


4.2 Data Transfer from CLX Processor to MVI Input Buffer

The following BASIC CALLs 14, 15, 53 and 56 are covered in this section. Refer to BASIC CALLs Syntax (page 105) for detailed syntax information about these CALLs.

The MVI56-BAS input buffer can receive data transferred from the CLX processor through the backplane using CALLs 56 or 53. The input buffer data can be read using CALLs 14 and 15.

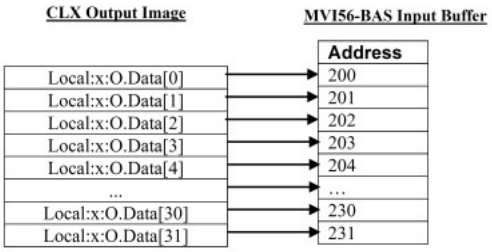
The data transfer from the CLX to the MVI56-BAS input buffer is shown in the following diagram:



The input buffer is divided into areas depending on the source of the data transferred. The input buffer addressing is shown in the following table.

Address	Definition
0 to 39	DH-485 Common Interface File - data written by other devices
40 to 99	Reserved
100 to 199	Data transferred from the CLX using Message instruction
200 to 231	Data transferred from the CLX Output Image file

CALL 53 transfers 8 words from the CLX output image file to the MVI56-BAS input buffer.



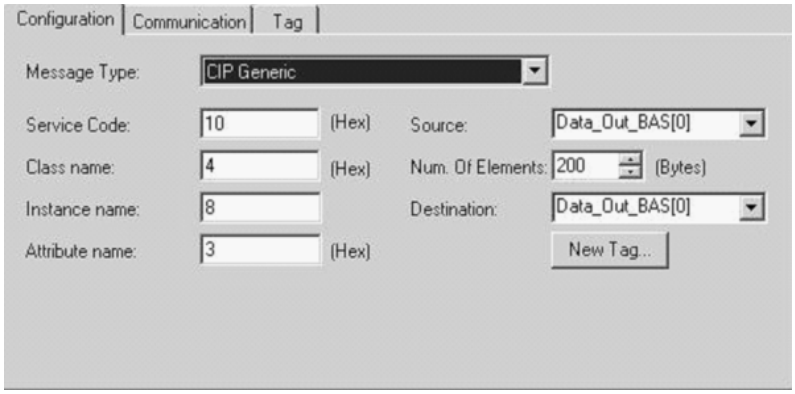
The following example BASIC code uses CALL 53 to transfer data from the CLX output image file to the input buffer. After that, it uses CALL 14 to read the data from the MVI56-BAS input buffer and prints it to PRT1.

```
10 CALL 53
20 POP X: REM CALL 53 status code
30 FOR I = 200 TO 207
```

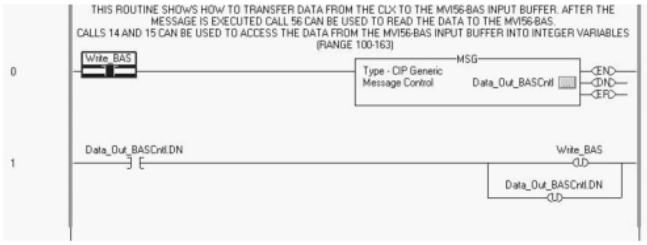
```
40 PUSH I: REM INPUT BUFFER ADDRESS
50 CALL 14
60 POP Y: REM INPUT BUFFER VALUE AT ADDRESS I
65 X = I - 200
70 PRINT "VALUE AT [", X, "] = ", Y
80 NEXT I
90 END
```

CALL 56 transfers up to 100 words from a CLX data file to the MVI56-BAS input buffer using Class 3 message instructions.

Ladder logic must execute a MSG instruction prior to CALL 56 execution in order to move the data to a temporary buffer. The MSG should have the following parameters:



The following ladder logic writes the data that will be read by CALL56.



After the ladder logic execution, a CALL 56 can be performed in order to transfer the data to the MVI56-BAS input buffer.

```
1 REM CALL 56
10 PUSH 10
20 CALL 56
30 POP X
40 PRINT X
```

5 Using the Program Port (PRT1)

In This Chapter

The present section discusses important points while using the MVI56-BAS program port such as creating, editing, and transferring a BASIC program to the module. The program port by default is PRT1 (the middle port).

This section only describes PRT1 as the program port.

5.1 Interfacing the PC and the MVI56-BAS

The software required on your personal computer to interface with the configuration/debugger port is operating system dependent. Tested software includes the following:

DOS	ProComm and several other terminal emulation programs
Windows 3.1	Terminal
Windows 95/98/2000/XP	HyperTerminal
Windows NT	HyperTerminal
Linux	Minicom

Any ASCII terminal emulation software application provided with your operating system should work as long as it can be configured as follows for all MVI56-BAS ports:

Baud Rate	19,200
Parity	None
Data Bits	8
Stop Bits	1
Software Handshaking	XON/XOFF

The following steps are required to interface with the configuration/debugger port:

1. Connect your computer to the module's port using a Null Modem cable.
2. Start the terminal emulation program on your computer and configure the communication parameters.

If there is no response from the module, look at the communication setup and the cable. In addition, make sure you are connected to the correct port on your computer and the module.

5.2 Creating BASIC programs

Creating a BASIC program is simple. First, hit the enter key a couple of times. You should see the following response each time you hit the enter key:

```
Ready
>
```

If you do not get the above response, it is possible that the module is already running a BASIC program. To stop a program that is already running, hit the 'C' while simultaneously holding the control key down. The short symbol for this key stroke combination is ^C. If you still do not get a response, check the module power, check the serial cable, and check port connections.

After you get the Ready prompt, type in the BASIC command:

```
print "Hello World!"
```

followed by and enter key. The BAS module should respond with the following:

```
Ready
>print "Hello World!"
Hello World!
Ready
>
```

You just typed in a PRINT command. Practically any BASIC statement can also be used as a command. For example:

```
Ready
>for i=0 to 5:p. i:next i
 0
 1
 2
 3
 4
 5
Ready
>
```

Note that the "p." is short hand for "PRINT".

The above approach can be used for debugging a program, but there are a few problems writing a program this way. First, there is no permanent copy of the program saved in memory. That is, the program disappears as soon as it is executed. Second, you cannot write a program any longer than one line in length. Third, someone has to be there to write the program each time it is executed.

You can write a program which is permanently saved and executes whenever you want by adding line numbers at the beginning of each line. The program is executed in the order matching the line number. For example:

```
Ready
>10 for i = 0 to 5
Ready
>20 p. i
Ready
>30 next i
Ready
>40 end
Ready
>list
```

```
10 FOR I = 0 TO 5
20 PRINT I
30 NEXT I
40 END
Ready
>
```

The LIST command allows you to see what you have done so far.

You can now execute the program using the RUN command:

```
Ready
>run
 0
 1
 2
 3
 4
 5
Ready
>
```

If you want to add a line in the middle of your program, select an unused line number between the line numbers where you want to insert the line, and type in the new line:

```
Ready
>list
10 FOR I = 0 TO 5
20 PRINT I
30 NEXT I
40 END
Ready
>25 p. "Hello World"
Ready
>list
10 FOR I = 0 TO 5
20 PRINT I
25 PRINT "Hello World"
30 NEXT I
40 END
Ready
>
```

If you want to remove a line from the program, type in that line number:

```
Ready
>list
10 FOR I = 0 TO 5
20 PRINT I
25 PRINT "Hello World"
30 NEXT I
40 END
Ready
>20
Ready
>list
10 FOR I = 0 TO 5
25 PRINT "Hello World"
30 NEXT I
```

```
40 END
Ready
>run
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Ready
>
```

5.3 EDIT Command

Sometimes you may have typed in a long line and made a mistake in the middle. You fear that you may make more mistakes if you attempt to re-type the entire line. To solve this dilemma, use the EDIT command. To edit a line, type EDIT followed by that line number. For example "EDIT 100".

The EDIT command will respond by printing the entire line to the screen with the cursor at the beginning of the line. At this point, EDIT has several sub-commands that must be explained.

EDIT Sub-command	Description
<space>	Hitting the space key will cause the cursor to move to the right.
<backspace>	Hitting the backspace key will cause the cursor to move to the left.
Any character	Hitting any character will replace the character above the cursor with the new character.
^A	^A toggles between insert mode and non-insert mode. When you first enter the EDIT command, you are in non-insert mode. Typing a ^A will erase the line from the cursor to the end. At this point, you can type in any text which you want to insert into the line. When you type ^A again, the part of the line which was erased when you first hit ^A will reappear, and you will exit insert mode.
^D	^D will delete the character above the cursor.
<enter>	The enter key will cause the line to be re-typed as it exists after the edits.
^C	Typing ^C will exit EDIT mode without saving any changes.
^X	Typing ^X will exit EDIT mode and save all changes.

5.4 Permanently Saving BASIC Programs

There are two ways to permanently save a BASIC program. The first method has already been shown. When you type the program into the module, it is permanently saved to a file on the Compact Flash drive called "XRAM.BAS". If you want to save more than one program, you can use the PROG command. Refer to the Using ROM Storage section for more information. The PROG command copies the program from the "XRAM.BAS" file to another file called "EPROM.BAS". The "EPROM.BAS" file differs from the "XRAM.BAS" file in two important points. First, up to 255 programs can be saved in "EPROM.BAS". The programs are stored as ROM 1, ROM 2 , ROM 3, and so on. Second, you cannot modify a program stored in "EPROM.BAS". All you can do is ERASE the last program stored in "EPROM.BAS".

5.4.1 EPROM File Storage

When you want to store a XRAM program in EPROM, you type the PROG command. The programs are stored sequentially starting with ROM 1.

Executing a Program in EPROM

If you want to execute a program in EPROM, use the ROM command. ROM without a number moves the BASIC pointer to point to ROM 1. If you want to select another program stored in EPROM, type ROM followed by the number of the program you wish to execute.

Note that the ROM command only changes the BASIC program pointer. It does not automatically execute the program. You can use the RUN command to execute the program.

If you want to select another EPROM program and execute that program, you can use the RROM (RUN ROM) command followed by the number of EPROM program you wish to execute (that is, RROM 3).

5.4.2 Erasing EPROM Programs

If you want to remove EPROM programs, you can use the ERASE command. The ERASE command removes the last program in EPROM. You cannot remove a program at the beginning or in the middle of the EPROM programs.

5.4.3 Editing EPROM Programs

You cannot modify programs stored in EPROM. If you must modify a program in EPROM, you can use ROM to change the pointer to the program that needs modification. Then you can XFER that program to XRAM. Then you change the BASIC program pointer back to XRAM using the RAM command. Now the program has been moved to XRAM, and the BASIC pointer is pointing to XRAM, so the program can be edited.

5.4.4 Using XRAM and EPROM Programs as Subroutines

The BASIC programs are limited to the size of program that you can squeeze into XRAM. However, you can create larger programs by using CALLs 70, 71, and 72. These three CALLs allow you to use XRAM and EPROM programs as subroutines.

5.4.5 Automatically Executing ROM 1 at Power-up

If the setup jumper is off, then the program in XRAM will automatically be executed at power-up. You can change the module to automatically execute ROM 1 at power-up by using the PROG2 command. The PROG2 will have the same effect at power-up as someone typing RROM 1 (or ROM 1, RUN). If at a later time you want to stop executing ROM 1 at power-up, use PROG0 to clear the effect of PROG2.

5.5 Creating Offline BASIC programs

You can use a regular ASCII editor such as Notepad, Wordpad (Windows) or EDIT.EXE (DOS) in order to edit and save offline BASIC programs.

When saving programs, choose a name starting with a letter and containing no more than 8 (eight) characters. Save your BASIC program with a "BAS" extension (.BAS). For example: TEST.BAS would be a valid BASIC program name.

This section describes how to transfer BASIC programs between the local PC and the MVI56-BAS modules. After the program has been transferred to the MVI56-BAS module it should be loaded to RAM memory using the LOAD command in order to be ready to be executed.

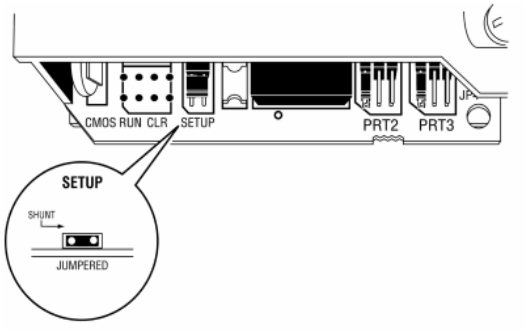
5.5.1 Downloading BASIC Files From a PC to the MVI56-BAS

In order to download BASIC programs to the module, the MVI56-BAS running program will have to be interrupted. You must run the RY.EXE program which uses the Y-Modem protocol. All steps are described in this section:

Step 1

Remove the module from the rack and place the setup jumper ON.

Put the module back into the rack.



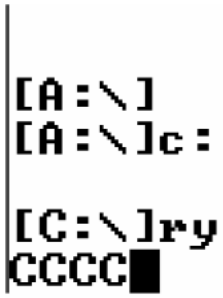
Step 2

Connect the ASCII Terminal software to Port 2 (19200 baud rate) and enter "**exit**" in order to exit the MVI56-BAS program. The program (running at Port 1) should exit to DOS prompt.

```
Ready
>exit
```

Step 3

Connect the ASCII Terminal software to Port 1 (19200 baud rate). Change the current driver from a: to c: by entering "c:" at the DOS prompt. Then, run the RY program by entering "**ry**" at the DOS prompt:



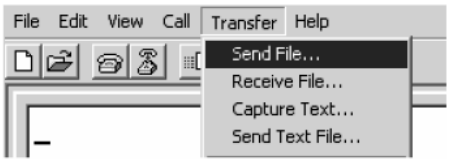
At this point, you should see the characters "C" and "G" being showed on the screen. This means that the module is ready to receive data. The module will wait approximately 1 minute for the user to select and transfer the file.

In a case that you do not send the file within 1 minute, the following message will appear:

"Receive Failed".

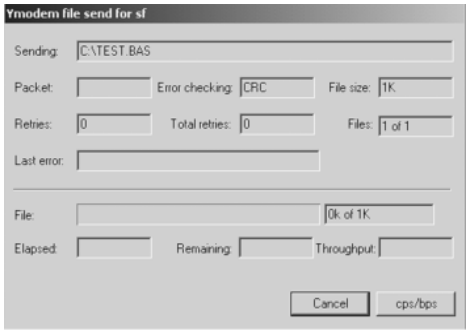
Step 4

Using the menu bar in the ASC Terminal software you are using, select the "Send File…" option.



Select the BASIC program you want to send and select the Ymodem protocol. The MVI56-BAS module only supports the Ymodem protocol, so it is required that the ASCII terminal used in this procedure also supports DH-485.

While transferring the file, a window shows the remaining time:

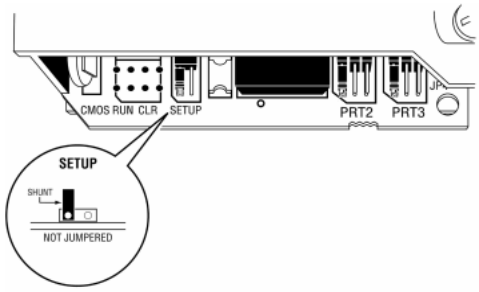


You can type "**dir**" at the DOS prompt in order to verify that the BASIC module is in the module directory.

```
[C:\]ry
CCCCCSSS♦
[C:\]dir

 Volume in drive C has no label
 Directory of C:\

XRAM     BAS          11 27-Jan-1980 09:24a
EPROM    BAS           2 27-Jan-1980 09:24a
BATTERY  BAS          50 27-Jan-1980 09:27a
RY       EXE       41310 05-Sep-2000 03:39p
SY       EXE       42388 05-Sep-2000 03:40p
MVI56BP  EXE      117448 15-May-2000 11:04a
AUTOEXEC BAT         815 16-Apr-2002 01:40p
COMMAND  COM       69028 24-Sep-1999 09:35a
MVI56DD  EXE       19356 15-May-2000 11:42a
CONFIG   SYS         100 22-Apr-2002 04:08p
B52      EXE      594213 30-May-2002 02:42p
TEST     BAS          11 28-Jan-1980 07:40a
        12 file(s)     884732 byte(s)
         0 dir(s)    15052800 byte(s) free

[C:\]
```

Step 5

Now it is time to restart the module. First remove the module from the rack and set the setup jumper in OFF mode:



Put the module back to the rack. The procedure is finished.

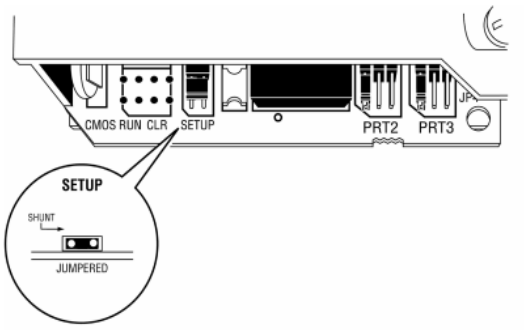
5.5.2 Uploading BASIC files from the MVI56-BAS to a PC

In order to upload BASIC programs from the MVI56-BAS to a PC, the running program will have to be interrupted. Once interrupted, you must run the SY program which uses the Y-Modem protocol. After the upload is complete, the user will have to restart the module. All steps are described in the present section:

Step 1

Remove the module from the rack and place the setup jumper ON.

Put the module back into the rack.



Step 2

Connect the ASCII Terminal software to Port 2 (19200 baud rate) and enter "**exit**" in order to exit the MVI56-BAS program. The program (running at Port 1) should exit to DOS prompt.

```
Ready
>exit
```

Step 3

Connect the ASCII Terminal software to Port 1 (19200 baud rate). Change the current driver from a: to c: by entering "c:" at the DOS prompt. Then, run the SY program by entering **ry "program_name.bas"** at the DOS prompt:

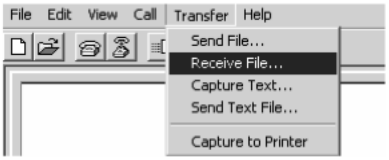
```
[A:\]c:
[C:\]sy "test.bas"
```

At this point you should quickly refer to STEP 4 in order to receive the file before the **sy** program timeout (about 30 seconds).

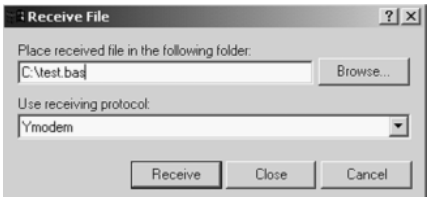
If there are any problems during the file transmission, the message "Send Failed" will be displayed in the screen.

Step 4

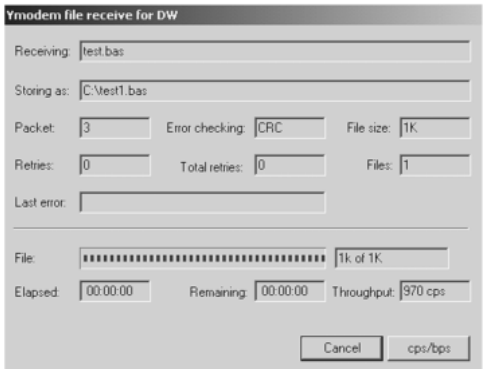
Using the menu bar in the ASCII Terminal software you are using, select the "Send File…" option.



Select the BASIC program you want to send and select the Ymodem protocol. The MVI56-BAS module only supports Ymodem protocol.

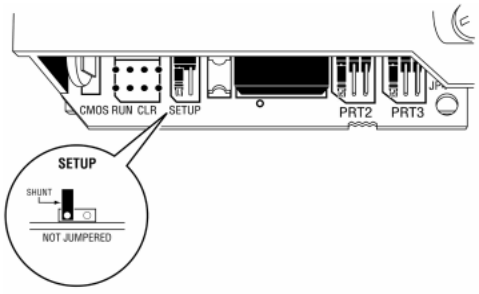


While transferring the file, a window shows the remaining time:



Step 5

Remove the module from the rack and set the setup jumper in OFF mode:



Put the module back to the rack and the procedure is finished.

5.5.3 Loading a BASIC Program

In order to load a BASIC program to RAM memory, use the `LOAD` command as shown in the ASCII terminal connected to the program port:

```
LOAD "NAME.BAS"
```

5.6 Module Backup

There are two simple ways to backup all the program files associated with the MVI56-BAS. The first method assumes that your computer has access to a Compact Flash card reader. The second method assumes that you do not have access to a Compact Flash card reader and is more complicated.

5.6.1 Backup with a Compact Flash Card Reader

Backing up a MVI56-BAS with a Compact Flash card reader is simple. Remove the Compact Flash from the BAS module. Place the Compact Flash in your reader. Using your computer, make copies of three files:

XRAM.BAS

BATTERY.BAS

EPROM.BAS

These three files contain all the information needed to duplicate your application on another BAS module. These three files do not contain the dynamic battery backed data stored during program execution.

5.6.2 Backup without a Compact Flash Card Reader

There are three files on the Compact Flash which contain all the information needed to duplicate your application on another BAS module:

XRAM.BAS

BATTERY.BAS

EPROM.BAS

These files should never be edited. Follow the following steps to make backup copies of the files.

Step 1

Remove the module from the rack and place the setup jumper ON.

Step 2

Connect the ASCII Terminal software to PRT1 (19200,N,8,1) and apply power to the BAS module. When the sign-on message is displayed, enter the EXIT command at the Ready prompt. This will stop the BASIC execution and enter DOS.

Step 3

Change the ASCII cable from PRT1 to the DH-485 port (COM0).

Step 4

Change to the Compact Flash drive by typing "C:" at the DOS prompt. Then run the SY program by entering SY "XRAM.BAS":

```
[A:\]C:
[C:\]sy "xram.bas"
```

At this point you should quickly refer to STEP 4 in order to receive the file before the SY program timeout (about 30 seconds).

In case there are any problems during the file transmission, the message "Send Failed" will be displayed in the screen.

Step 5

Using the menu bar in the ASCII Terminal software you are using select the "Receive File" option. Then select the directory where you want to place the file and select the Ymodem protocol. Then hit the receive button.

Step 6

Repeat STEPs 4 and 5 for "BATTERY.BAS" and "EPROM.BAS".

Step 7

Remove the MVI56-BAS and place the setup jumper OFF. Replace the MVI56-BAS in the rack. Connect the ASCII cable to PRT1.

5.7 Module Restoration

There are two simple ways to restore all the program files associated with a MVI56-BAS application. The first method assumes that your computer has access to a Compact Flash card reader/writer. The second method assumes that you do not have access to a Compact Flash card reader/writer and is more complicated. Both methods assume that you have previously backed up a functioning BAS module application.

5.7.1 Restoration with a Compact Flash Card Reader

Restoration of an MVI56-BAS application with a Compact flash card reader/writer is simple. Remove the Compact flash from the BAS module. Place the Compact flash in your reader/writer. Using your computer, copy the backed up version of the following three files over the files on the Compact flash:

XRAM.BAS

BATTERY.BAS

EPROM.BAS

These three files contain all the information needed to duplicate your application. These three files do not contain the dynamic battery backed data stored during program execution.

5.7.2 Restoration without a Compact flash Card Reader

There are three files on the Compact flash which contain all the information needed to restore your application on another BAS module:

XRAM.BAS

BATTERY.BAS

EPROM.BAS

These files should never be edited. Follow the following steps to restore the files on another BAS module.

Step 1

Remove the module from the rack and place the setup jumper ON.

Step 2

Connect the ASCII Terminal software to PRT1 (19200,N,8,1) and apply power to the BAS module. When the sign-on message is displayed, enter the EXIT command at the Ready prompt. This will stop the BASIC execution and enter DOS.

Step 3

Change the ASCII cable from PRT1 to the DH-485 port (COM0).

Step 4

Change to the Compact flash drive by typing "C:" at the DOS prompt. Then run the RY program by entering RY:

```
[A:\]C:
[C:\]ry
```

At this point you should quickly refer to STEP 4 in order to receive the file before the RY program timeout (about 30 seconds).

In case there are any problems during the file transmission, the message "Receive Failed" will be displayed in the screen.

Step 5

Using the menu bar in the ASCII Terminal software you are using select the "Send File" option. Then select the file and select the Ymodem protocol. Then hit the send button.

Step 6

Repeat STEPs 4 and 5 for all three files.

Step 7

Remove the MVI56-BAS and place the setup jumper OFF. Replace the MVI56-BAS in the rack. Connect the ASCII cable to PRT1.

5.8 Program Copies

The BAS module tokenizes each line that is typed into the module. That is, it searches the line for BASIC commands, statements, and operators. These commands, statements, and operators are replaced with a single character non-printable token. This is the reason that "XRAM.BAS" and "EPROM.BAS" do not look anything like the program that you originally typed in. If you want to save a printable copy of the program, you have two options. First, you can log the data received by your terminal emulation software and LIST the program to the screen. Or, you can EXPORT your program to a file on the Compact Flash. For example,

```
Ready
>export "myprog.txt"
```

Then, exit to DOS and use RY.EXE to copy the program to your host computer. Do not EXPORT to "XRAM.BAS", "BATTERY.BAS", or "EPROM.BAS"!

You can actually recover programs by using the LOAD command. For example,

```
Ready
>load "myprog.txt"
```

At first glance, this approach appears to be a way to backup and recover BASIC programs. However, only the XRAM part of the backup data is stored here. The ASCII port (PRT1 and PRT2) parameters, the DH-485 port parameters, PGMPRT parameters, and other things are stored as a raw data structure in "BATTERY.BAS". Further, the EPROM programs are stored in "EPROM.BAS". For a full backup, "XRAM.BAS", "BATTERY.BAS", and "EPROM.BAS" must all three be backed up and recovered.

5.9 Running a BASIC Program

After the program is in RAM memory, it can be run by entering RUN in the command line:

```
RUN
```

The `RUN` command will also clear all BASIC variables to zero and clear all interrupts.

5.10 Debugging a BASIC Program

This section describes the BASIC commands that help the user debug BASIC code.

The BRKPNT command sets a program break point at the specified line. This means that the BASIC program exits, allowing the user to check all variables. Enter BRKPNT before running a program in order to enable the break point command.

To continue program execution, use the CONT command.

Syntax:

```
BRKPNT [line number]
```

If the line number selected is zero, the BRKPNT command will be disabled.

The SNGLSTP command allows the program to exit at all lines. The program will stop first at the initial line number defined in the SNGLSTP command.

To continue program execution, use the CONT command.

Syntax:

```
SNGLSTP [initial line number]
```

Error Recovery

Use the ONERR command to recover from an error during program execution. When an error occurs the program execution will skip to the line specified by the ONERR command. Each error has an associated code which can be retrieved by using the ERRCODE operator or the XBY function.

Definition of error codes generated with the ONERR command:

1 Missing "=" sign.
2 Bad or missing expression.
3 NOT USED.
4 Variable name too long.
5 Expected a variable, but did not find one.
6 Exceeded maximum number of variables.
7 Missing ")".
8 Missing "(".
9 Bad variable element number.
10 NOT USED.
11 Argument stack error.
12 Bad or missing operator in expression.
13 Too many operators in expression.
14 ELSE token found without prior IF token.
15 Invalid syntax.
16 Missing variable after "LET" token.
17 Missing quote.
18 Statement not terminated correctly.
19 Bad or missing line number
20 No line to return to after GOSUB.
21 Tried executing RUN while already in RUN mode.

22 Bad parameter.
23 DO stack error.
24 Control stack error.
25 GOSUB stack error.
26 Too many strings.
27 Requested STRING array size is bad.
28 Missing constant.
29 Missing comma.
30 Second STRING statement encountered.
31 STRING statement encountered after variable declaration.
32 Bad string number.
33 String encountered before STRING declaration.
34 File name too long.
35 Could not open file.
36 Statement not allowed in RUN mode.
37 Bad variable name.
38 NOT USED.
39 Too many variables.
40 Too much space needed for dimensioned variable.
41 Variable or statement must follow line number.
42 NOT USED.
43 UNTIL without preceding DO.
44 TO missing from FOR statement.
45 NEXT variable does not match FOR variable.
46 STEP size in FOR statement is zero.
47 NEXT statement without preceding FOR.
48 Missing "0" or "1" after CLOCK token.
49 DH-485 driver failure.
50 ONTIME TIME line number is out of range.
51 RETI without corresponding interrupt request.
52 Could not allocate any more memory.
53 Too many constants.
54 Could not find DATA statement.
55 Missing GOTO/GOSUB after ON expression.
56 ON expression is negative.
57 RENumber error (NUM1 must be >= NUM2).
58 Missing string.
59 Bad @ or # value.
60 Unknown CALL number.
61 Bad time parameter.
62 Bad date parameter.
63 String too small.
64 Bad argument.
65 Serial port failure.
66 Serial port failure.
67 Serial port failure.
68 Serial port failure.
69 Setup jumper in wrong position.
70 Backplane interface failure.

71 Backplane interface failure.
72 Backplane interface failure.
73 Backplane interface failure.
74 Backplane interface failure.
75 Backplane interface failure.
76 Not enough space in EPROM.
77 No program in RAM to store.
78 No program in EPROM to erase.
79 ROM program # does not exist.
80 Cannot erase current program.
81 Too many programs stored in PROM.
82 Already executing in RAM.
83 DH485 port failed.
84 DF1 is already enabled.
85 Variable has already been DIMensioned.
86 Serial port failure.
87 DF1 transmit queue is full.
88 DF1 driver failure.
89 DF1 not enabled.
90 DF1 packet is too short.
91 DF1 driver failure.
92 DF1 driver failure.
93 DF1 driver failure.
94 DF1 Status variable not valid.
95 DF1 receive packet is too large.
96 Not legal operation when DF1 is enabled.
97 ONTIME has not been enabled.
98 Cannot IDLE inside ONTIME interrupt routine.
99 Clock has not been enabled.
100 DBY() is no longer needed nor allowed.
101 XBY() is no longer needed nor allowed.
102 No program in XRAM.
103 Domain math error (for example, sqr(-1)).
104 Singularity math error (for example, 0**(-2)).
105 Overflow.
106 Underflow.
107 Total loss of significant digits (for example, sin(10e70)).
108 Partial loss of significant digits.
109 Floating point unit stack overflow.
110 Unknown math error.
111 Divide by zero.
112 DF1 driver failure.
113 DF1 packet transmission already in process.
114 DH-485 is not enabled.
115 DH485 port failed.
116 DH485 port failed.
117 DH485 port failed.
118 Cannot mix foreground and background DF1 CALLs.
119 Cannot mix DH-485 and DF1 functions.

120 Invalid battery backed RAM location.
121 Second STRING value is invalid.
122 Unknown error

Syntax:

```
ONERR [line number]
```

Two new commands have been added to help programmers: `ERRLINE` and `ERRCODE`. `ERRLINE` PUSHes the offending line number onto the argument stack. `ERRCODE` PUSHes the offending error code onto the argument stack.

The following shows an example of how to use the `ONERR` command:

```
10 ONERR 100
20 LET = 5
30 PRINT "THIS IS LINE 1"
40 PRINT "THIS IS LINE 2"
50 PRINT "THIS IS LINE 3"
60 END
100 PRINT "ERROR CODE WAS ", ERRCODE
110 PRINT "ERROR LINE WAS ", ERRLINE
120 END
Ready
>run
ERROR CODE WAS 16
ERROR LINE WAS 20
```

Lines 30 to 60 are skipped and the program execution continues at line 100.

Alternatively, the XBY function can be used:

```
10 ONERR 100
20 LET = 5
30 PRINT "THIS IS LINE 1"
40 PRINT "THIS IS LINE 2"
50 PRINT "THIS IS LINE 3"
60 END
100 PRINT "ERROR CODE WAS ", XBY(257)
110 PRINT "ERROR LINE WAS ", ( 256*XBY(27133) + XBY(27134) )
120 END
Ready
>run
ERROR CODE WAS 16
ERROR LINE WAS 20
```

Note: The values passed as arguments to the `XBY` function in the above example are the only valid variable locations. All other variable locations will return **0**.
`XBY()` can also be used on the left side of the equal sign; this assigns a value to any of the above variable locations, overriding the automatically generated error information. Again, the variable locations used in the above example are the only ones that can be accessed. Any other variable locations will cause an error.
If you wish to store and retrieve user values in battery-backed SRAM, use the `XBYTE` function (see Operators and Statements).

After the error code value has been retrieved, it can be cleared using the CLRERR command; this allows debugging to continue. The following code shows how to use the CLRERR command:

```
10 ONERR 100
20 LET =5
30 PRINT "THIS IS LINE 1"
40 PRINT "THIS IS LINE 2"
50 PRINT "THIS IS LINE 3"
100 E = ERRCODE
200 PRINT "ERROR CODE WAS ", E
300 CLRERR
400 PRINT "ERROR CODE NOW IS ", ERRCODE
Ready
>run
ERROR CODE WAS 16
ERROR CODE NOW IS 0
```

5.11 Commenting a BASIC Program

The REM command allows the user to insert comments in the BASIC program that will be ignored by the BASIC interpreter. Use the REM command after a colon (:) in a program line.

Example:

```
10 REM BASIC PROGRAM
20 LET A=3: REM ASSIGN A VALUE TO A
30 PRINT A: REM PRINTS A
```

5.12 Checking Available and Used RAM Memory

To see the currently available RAM memory that can be used, use the `FREE` operator as shown in the following example:

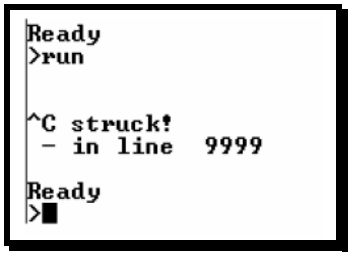
```
PRINT FREE
```

To see the RAM memory currently being used by the selected BASIC program, the `LEN` operator can be used:

```
PRINT LEN
```

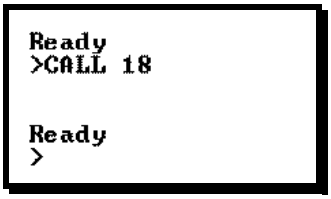
5.13 Exit a BASIC Program (Ctrl+C)

If the module is currently running a BASIC program, you must enter **"CTRL+C"** (^C) in order to quit the program when connected to PRT1 (COM2). This will exit the program and return to the BASIC terminal prompt:



In order to disable the ^C function, CALL 19 (Disable the ^C Function) can be used. It has no parameters and it does not return any output values. Entering CALL 19 will disable the ^C function.

In order to re-enable the ^C function after a CALL 19 execution, CALL 18 must be used (Re-enable the ^C function)



6 Using ASCII Communications

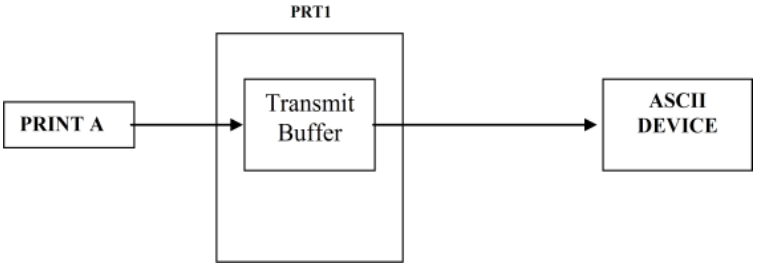
In This Chapter

The MVI56-BAS can also be used to interface a serial communication device with the Rockwell Automation ControlLogix processor. Two ports (PRT1 and PRT2) can be used.

6.1 Port Transmit and Receive Buffers

MVI56-BAS PRT1 and PRT2 have an internal transmit and receive buffer. Each buffer has a capacity for 1024 bytes.

The PRINT command sends characters to the two serial ports. PRINT sends characters to the program port, PRINT@ also sends characters to PRT1, and PRINT# sends characters to PRT2. The MVI56-BAS transmits one character at a time until the buffer is empty. If software handshaking (XON/XOFF) has been selected, and an XOFF character has been received, then the port will not transmit until an XON has been received. If hardware handshaking has been selected, and the CTS signal is OFF, then the port will not transmit until CTS transitions ON. In either case, the transmit buffer will accumulate characters until it is full.



In order to avoid a situation where the BASIC program tries to write characters to a transmit buffer that is already full, the following CALLs can be used to verify the current number of characters in PRT1 and PRT2:

CALL 36 = Get Number of Characters in PRT2 transmit buffer

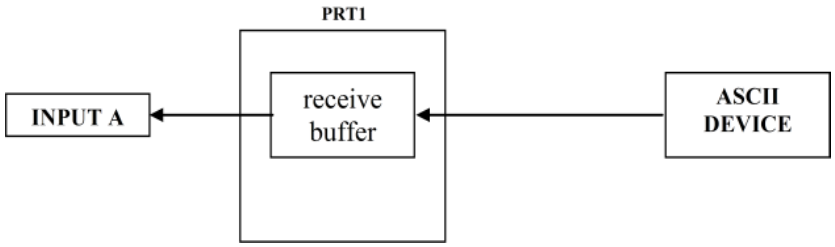
CALL 95 = Get Number of Characters in PRT1 transmit buffer

If necessary, the BASIC program can clear the current transmit and receive buffers using the following CALLs:

CALL 37 = Clear PRT2 Transmit and Receive Buffers

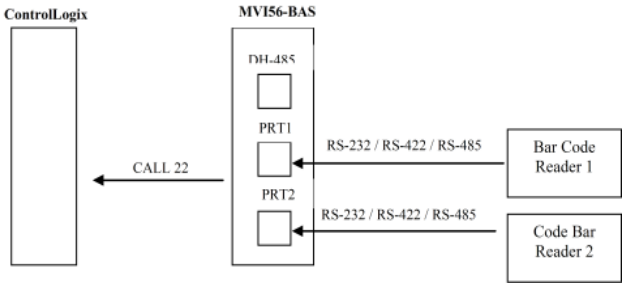
CALL 96 = Clear PRT1 Transmit and Receive Buffers

The same idea applies when an ASCII device writes data to MVI56-BAS PRT1 or PRT2: it will be stored in the receive buffer until is read by a INPUT statement (INPUT@ = PRT1 or INPUT# = PRT2)



6.2 ASCII Data Transfer from MVI56-BAS Serial Port to CLX

The MVI56-BAS can interface serial devices such as printers and code bar readers to the ControlLogix processor. This section shows how to receive data from an ASCII device and transmit it to the CLX processor.



In order to transfer data from the MVI56-BAS serial port to the ControlLogix processor, CALL 22 is required to set up the transfer parameters. The following shows a CALL 22 example:

```
1 REM CALL 22 reads data from the MVIBAS to the CLX
2 REM BASIC Example Program
10 REM MAIN BODY
20 PUSH 1: REM PORT NUMBER
30 PUSH 10: REM RECEIVES 10 BYTES MAXIMUM
40 PUSH 13: REM <CR> IS THERE TERMINATION CHARACTER
50 PUSH 1: REM DESTINATION IS MSG INSTRUCTION
55 PUSH 0: REM DESTINATION FILE OFFSET
60 PUSH 0: REM STRING NUMBER (NOT USED HERE)
65 PUSH 0: REM BYTE SWAP
70 CALL 22
80 POP S:
90 IF ( S <> 0) THEN PRINT "error", S
99 GOTO 99
```

or

```
10 PUSH 1,10,13,1,0,0,0: CALL 22: POP S
20 IF ( S <> 0) THEN PRINT "error", S
30 IF ( S <> 0) THEN STOP
40 GOTO 40
```

After CALL 22 is executed, the serial port (PRT1 or PRT2) continues receiving data until the maximum number of characters is reached, or the character delimiter is received.

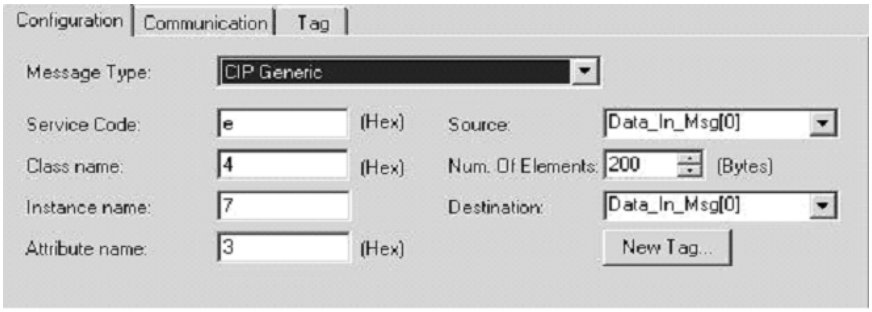
Both parameters are configurable when using CALL 22 (refer to "BASIC CALLs Syntax" section). The maximum number of characters transmitted depends on the destination file used:

- CLX Input Image File => 60 characters maximum
- MSG Instruction => 198 characters maximum
- Internal String => 251 characters maximum

If using CLX input image file the destination file offset parameter in CALL 22 should be no less than 1 otherwise a CALL error will be returned.

In order to use a CLX Message as the transfer destination, the following MSG instruction

has to be configured:



The transfer sequence is described in the following paragraphs:

Step 1 - When the maximum number of characters is reached, or the character delimiter is found, the data is transferred to the destination file. The MVI56-BAS also places the byte count in the lower byte of the first register in the destination file.

Step 2 - Next, the MVI56-BAS turns ON one of the following bits to indicate that new data is available:

PRT1 => Local:x:I.Data[0].8

PRT2 => Local:x:I.Data[0].9

Where x is the MVI56-BAS slot number.

Step 3 - Next, the ladder logic reads the data and turns ON one of the following bits to indicate the data was received:

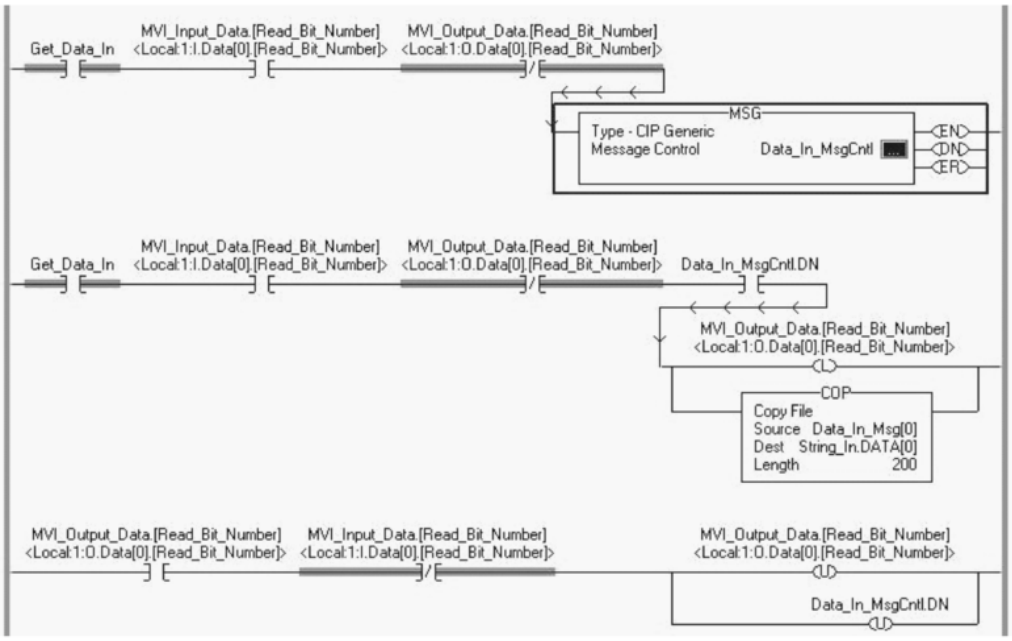
PRT1 => Local:x:O.Data[0].8

PRT2 => Local:x:O.Data[0].9

Step 4 - The MVI56-BAS resets the image file input bit.

Step 5 - The ladder logic resets the image file output bit.

The following shows the sample ladder logic that implements the previously described procedure when the MSG instruction is selected as the destination file:

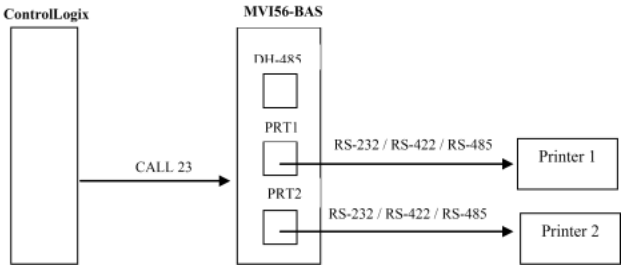


Where Read_Bit_Number variable is either 8 or 9.

Refer to BASIC CALLs Syntax (page 105) for more information about CALL 22.

6.3 ASCII Data Transfer from CLX to MVI56-BAS Serial Port

The MVI56-BAS can interface serial devices such as printers and bar code readers to the ControlLogix processor. This section shows how to send data from the ControlLogix to a remote ASCII device.

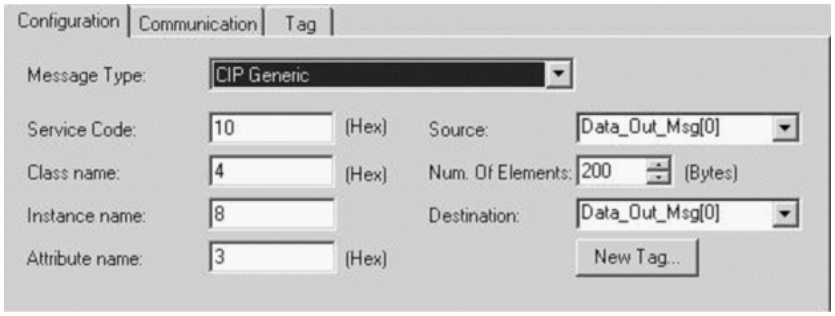


In order to transfer data from the ControlLogix processor to a MVI56-BAS serial port, CALL 23 is required to set up the transfer parameters. The following shows a CALL 23 example:

```
10 REM BASIC CALL 23
20 PUSH 2: REM SERIAL PRT SELECTION
30 PUSH 1: REM SOURCE FILE
40 PUSH 0: REM SOURCE FILE OFFSET
50 PUSH 1: REM INTERNAL STRING NUMBER
60 PUSH 0: REM BYTE SWAP
70 CALL 23
80 POP S: REM CALL STATUS
9999 GOTO 9999
```

After CALL 23 is executes, the MVI56-BAS module gets the data from the selected source file and transfer it to the selected destination port (or internal string). CLX Input and Output image bits are used for handshaking purposes.

If using a MSG instruction as the data source, the following MSG parameters must be configured:



After CALL 23 is successfully executed, the required transfer parameters are configured.

At this point the transfer sequence can begin:

Step 1 - The ladder logic sets the correct output image bit in order to inform the MVI56-BAS there is new data to be transferred:

Destination is PRT1 => Local:x:O.Data[0].6

Destination is PRT2 => Local:x:O.Data[0].7

Where x is the MVI56-BAS slot number

Step 2 - The MVI56-BAS transfers the data from the CLX to one of the serial ports (PRT1 or PRT2).

Step 3 - The MVI56-BAS sets the correct input image bit to inform to the CLX that the data was successfully transferred:

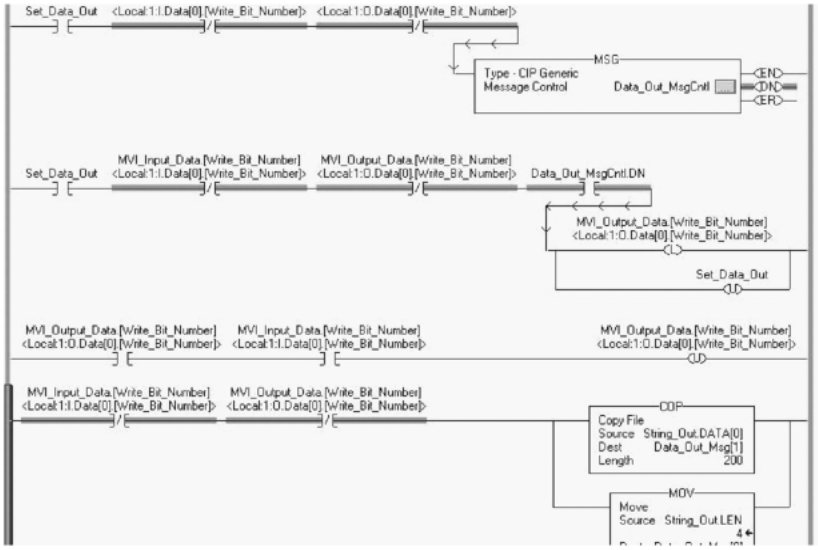
Destination is PRT1 => Local:x:I.Data[0].6

Destination is PRT2 => Local:x:I.Data[0].7

Step 4 - The ladder logic turns OFF the output image bit (Local:x:O.Data[0].6 or Local:x:O.Data[0].7)

Step 5 - The MVI56-BAS turns OFF the input image bit (Local:x:I.Data[0].6 or Local:x:I.Data[0].7)

The following shows the sample ladder required when the selected source of data is the MSG instruction:



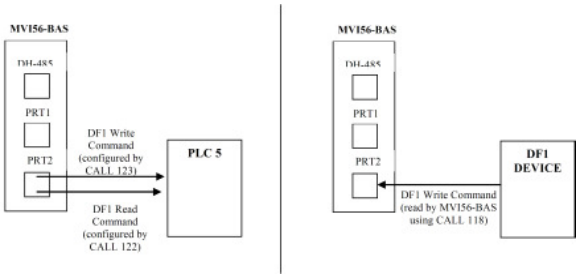
Refer to BASIC CALLs Syntax (page 105) for more information about CALL 23.

7 Using DF1 Protocol Communications

In This Chapter

The MVI56-BAS PRT2 can be configured for DF1 protocol communication. The MVI56-BAS uses DF1 to communicate with external DF1 devices in Full-Duplex or Half-Duplex Slave modes.

The following figure shows possible DF1 applications and the BASIC CALLs for each application. Before running CALLs 118, 122 or 123, it is required that PRT2 is already enabled for DF1 protocol using CALL 108.



The MVI56-BAS supports:

7.1 Operation

Full-Duplex with no handshaking
Full-Duplex Modem
Half-Duplex Slave with no handshaking
Half-Duplex Slave Modem without Continuous Carrier
Half-Duplex Slave Modem with Continuous Carrier

7.2 Communication

Duplicate packet detection
CRC or BCC error checking
Enable or auto-detect embedded responses

7.3 DF1 Commands

The MVI56-BAS can send the following commands:

PLC-2 unprotected READ command

PLC-2 unprotected WRITE command

PLC-3 word range READ command

PLC-3 word range WRITE command

PLC-5 typed READ command

PLC-5 typed WRITE command

The MVI56-BAS accepts the following commands:

PLC (word range writes)

PLC (typed writes)

PLC (unprotected writes)

SLC 5/02 (unprotected writes)

SLC 5/02 (typed writes)

The first step is to enable the DF1 driver using CALL 108, which will configure all required parameters for the DF1 port such as the device node address, communication mode, number of Retries, and so on. Refer to BASIC CALLs Syntax (page 105) for more information about CALL 108.

At any point in the program, the DF1 protocol can be disabled using CALL 113. Refer to BASIC CALLs Syntax (page 105) for more information about CALL 113.

After the DF1 driver is enabled, the module can send read and write commands to other DF1 nodes in the network.

7.4 Sending a DF1 Read Command

In order to read data from a DF1 node, it is required to set up the data transfer parameters using CALL 122 after PRT2 has been enabled for DF1 communications using CALL 108. CALL 108 must configure PRT2 to a full-duplex node in order to send a DF1 read command. Some of the parameters are listed below:

- Type of READ command
- Remote DF1 node address
- File number in remote DF1 node
- Number of elements to be transferred
- Destination file (input image, MSG instruction or internal string)

Refer to BASIC CALLs Syntax (page 105) for more information about CALL 122.

The following shows an example BASIC program that shows how to enable the DF1 driver using CALL 108. It then configures a remote READ command using CALL 122:

```
10 REM Enable DF1 Driver
20 MODE ( PRT2, 19200, N, 8, 1, N, R)
30 CALL 113: REM Disable any Existing DF1 Drivers
40 PUSH 18: REM NHS,ER,Disable DPD,CRC
50 PUSH 200: REM 10 Second Timeout
60 PUSH 2: REM ENQ Retries
70 PUSH 0: REM NAK Retries
80 PUSH 0: REM N/A
90 PUSH 2: REM Module Address
100 CALL 108
110 REM CALL 122 Test
120 REM Reads a DF1 Node
130 PUSH 2: REM PLC/2 File Type
140 PUSH 1: REM Remote Node Address
150 PUSH 7: REM Remote File Number
160 PUSH ASC( N): REM Integer File Type
170 PUSH 0: REM Starting Word Offset
180 PUSH 5: REM Number of elements to read
190 PUSH 10: REM Timeout in 0.1s increments
200 PUSH 1: REM Destination File (1=MSG)
210 PUSH 2: REM Destination file word offset
220 PUSH 0: REM String Number
230 CALL 122
240 POP S: REM Status of CALL
250 PRINT "CALL 122 return code was", S
260 GOTO 260
```

After the READ command is configured, ladder logic is required to actually send the READ command to the remote DF1 node. Input and output image file word 0 bit 10 are used for handshaking purposes. The transfer procedure is listed below:

Step 1: The CLX informs the MVI56-BAS that the READ command configured by CALL 122 should be executed, turning ON output image word 0 bit 10 (Local:x:O.Data[0].10).

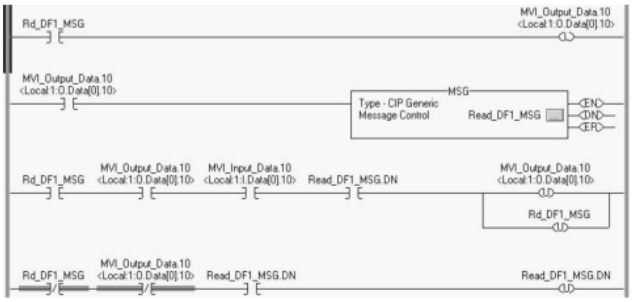
Step 2: The MVI56-BAS module sends the READ command to the remote DF1 node which replies with the data.

Step 3: The MVI56-BAS transfers the data into the CLX destination file and also copies the status code to input image word 1, low byte (Local:x:I.Data[1]). After the transfer is done, the MVI56-BAS sets the input image word 0 bit 10 (Local:x:I.Data[0].10) to inform to the CLX that the data is available.

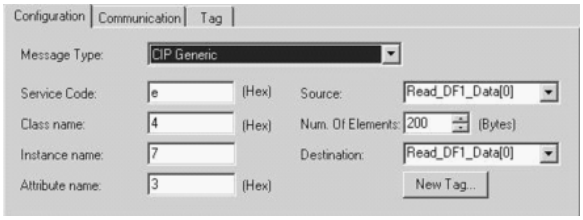
Step 4: The CLX copies the data and resets output image word 0 bit 10 (Local:x:O.Data[0].10) to inform the module that the data was received.

Step 5: The MVI56-BAS resets input image word 0 bit 10 byte (Local:x:I.Data[0].10)

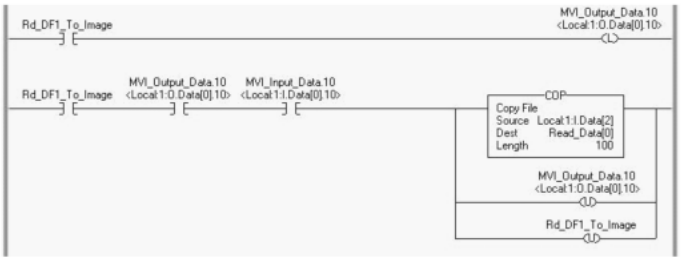
The following ladder file shows an example of how to implement the procedure described above when the destination file is a MSG instruction:



The MSG parameters are described in the following example:



In case the destination file selected by CALL 122 configuration is the CLX image input buffer, the ladder logic for CALL 122 is showed in the following example:



7.5 Sending a DF1 Write Command

In order to write data from the CLX to a DF1 node, you must set up the data transfer parameters using CALL 123 after PRT2 has been enabled for DF1 communication using CALL 108. CALL 108 must configure PRT2 as a full-duplex node in order to send a DF1 write command. Some of the parameters are listed below:

- Type of WRITE command
- Remote DF1 node address
- File number in remote DF1 node
- Number of elements to be transferred
- Source file (input image, MSG instruction or internal string)

Refer to BASIC CALLs Syntax (page 105) for more information about CALL 123.

The following shows an example BASIC program that shows how to enable the DF1 driver using CALL 108. It then configures a remote READ command using CALL 123:

```
10 REM Enable DF1 Driver
20 MODE ( PRT2, 19200, N, 8, 1, N, R)
30 CALL 113: REM Disable any Existing DF1 Drivers
40 PUSH 18: REM NHS,ER,Disable DPD,CRC
50 PUSH 200: REM 10 Second Timeout
60 PUSH 2: REM ENQ Retries
70 PUSH 0: REM NAK Retries
80 PUSH 0: REM N/A
90 PUSH 2: REM Module Address
100 CALL 108
110 REM CALL 123 Test
120 REM Reads a DF1 Node
130 PUSH 2: REM PLC/2 File Type
140 PUSH 1: REM Remote Node Address
150 PUSH 7: REM Remote File Number
160 PUSH ASC( N): REM Integer File Type
170 PUSH 0: REM Starting Word Offset
180 PUSH 5: REM Number of elements to read
190 PUSH 10: REM Timeout in 0.1s increments
200 PUSH 0: REM Destination File (0=Input Image)
210 PUSH 2: REM Destination file word offset
220 PUSH 0: REM String Number
230 CALL 123
240 POP S: REM Status of CALL
250 PRINT "CALL 122 return code was", S
260 GOTO 260
```

After the WRITE command is configured using CALL 123, ladder logic is required to actually send the WRITE command to the remote DF1 node. Input and output image file word 0 bit 11 are used for handshaking purposes. The transfer procedure is listed below:

Step 1: The CLX informs the MVI56-BAS that the WRITE command configured by CALL 123 should be executed turning ON output image word 0 bit 11 (Local:x:O.Data[0].11).

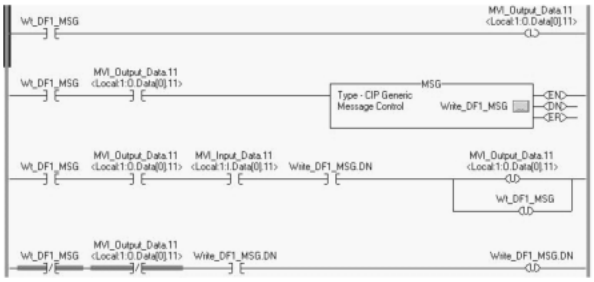
Step 2: The MVI56-BAS module transfers the data to the remote DF1 node.

Step 3: The MVI56-BAS copies the status code to input image word 1, low byte (Local:x:I.Data[1]). After the transfer is done the MVI56-BAS sets the input image word 0 bit 11 (Local:x:I.Data[0].11) to inform the CLX that the data was received .

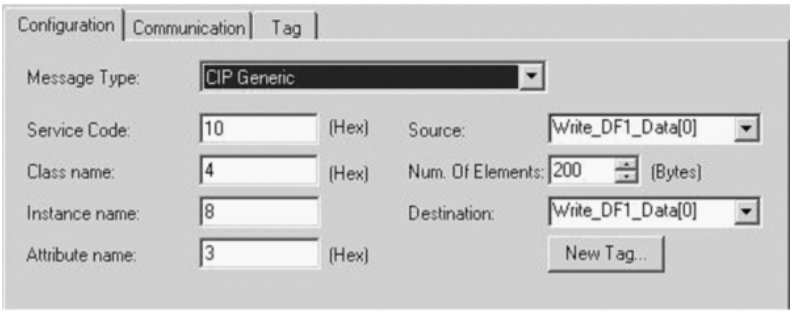
Step 4: The CLX resets output image word 0 bit 11 (Local:x:O.Data[0].11).

Step 5: The MVI56-BAS resets input image word 0 bit 11 byte (Local:x:I.Data[0].11)

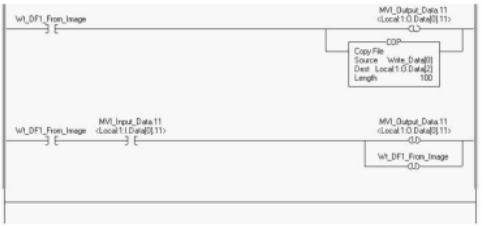
The following ladder file shows an example of how to implement the procedure described above when the destination file is a MSG instruction:



The MSG parameters are described below:



In case the destination file is the CLX input buffer, the ladder logic for CALL 122 is shown in the following example:



7.6 Receiving a DF1 Write Command

In order to receive a write command from a remote DF1 node, you must set up the data transfer parameters using CALL 118. Some of the parameters are listed below:

- CLX destination file
- Word offset in destination file
- Maximum word length

Refer to BASIC CALLs Syntax (page 105) for more information about CALL 118.

The following shows a BASIC program example that shows how to enable the DF1 driver using CALL 108. It then allows PRT2 to receive DF1 WRITE commands using CALL 118:

```
10 REM Enable DF1 Driver
20 MODE ( PRT2, 19200, N, 8, 1, N, R)
30 CALL 113: REM Disable any Existing DF1 Drivers
40 PUSH 2: REM NHS,ER,Disable DPD,CRC
50 PUSH 200: REM 10 Second Timeout
60 PUSH 2: REM ENQ Retries
70 PUSH 0: REM NAK Retries
80 PUSH 0: REM N/A
90 PUSH 2: REM Module Address
100 CALL 108
110 REM CALL 118
120 REM Reads a DF1 Node
130 PUSH 1: REM CALL ENABLE/DISABLE
140 PUSH 1: REM DESTINATION FILE
150 PUSH 0: REM OFFSET IN DESTINATION FILE
160 PUSH 2: REM STRING NUMBER (NOT USED HERE)
170 PUSH 50: REM MAXIMUM WORD LENGTH
180 CALL 118
190 POP M: REM CALL 118 STATUS
200 PRINT "STATUS CALL 118 -> ", M
250 PRINT "CALL 118 return code was", S
260 GOTO 260
```

After CALL 118 is executed, ladder logic is required for handshaking when a new WRITE command is received at PRT2. Input and output image file word 0 bit 12 must be used in ladder logic. The transfer procedure is listed below:

Step 1: A DF1 write command is received at the MVI56-BAS PRT2. The MVI56-BAS module transfers the data into the CLX internal buffer. The BASIC module also places the byte count into the first available word of the destination file.

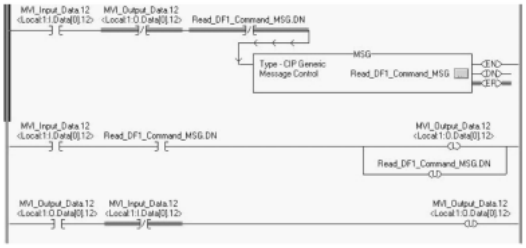
Step 2: The MVI56-BAS sets the input file word 0 bit 12 (Local:x:I.Data[0].12) to inform the CLX that the data is available.

Step 3: The CLX ladder logic retrieves the data from the buffer and sets the output file word 0 bit 12 (Local:x:O.Data[0].12) to inform the MVI56-BAS module that the data was received.

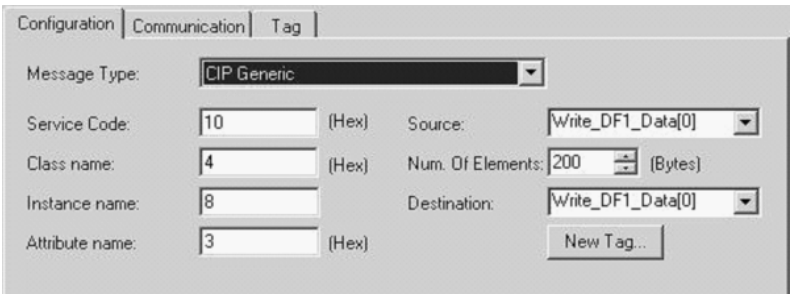
Step 4: The MVI56-BAS module resets input image word 0 bit 12 (Local:x:I.Data[0].12).

Step 5: The CLX resets output image word 0 bit 12 (Local:x:O.Data[0].12).

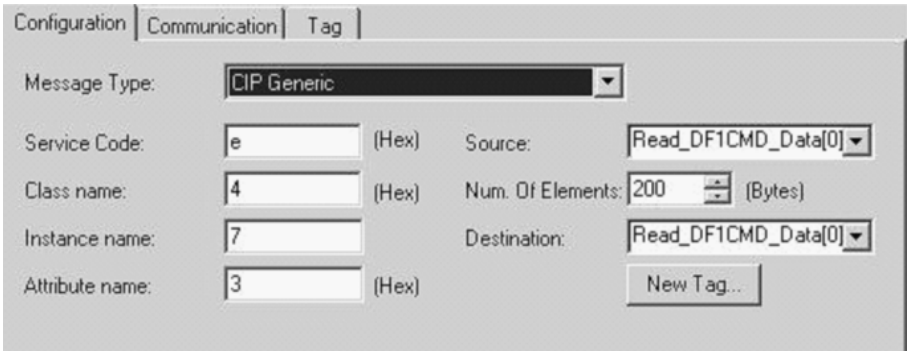
The following ladder file shows an example on how to implement the procedure described above when the destination file is a MSG instruction:



The MSG parameters are described below:



The MSG instruction must have the following parameters configured:



7.7 Transmitting a DF1 Packet

Using CALL 114 (Transmit DF1 Packet) allows the user to transmit a DF1 data packet created by the BASIC program using PRINT#, PH0# or PH1# statements. If PRT2 is configured for full-duplex communication, the packet will be transmitted immediately. If PRT2 is configured as a half-duplex slave, the DF1 packet will be transmitted the next time PRT2 receives an ENQuiry from a DF1 master.

In order to check if the transfer was successful, use CALL 115 (Check DF1 XMIT Status).

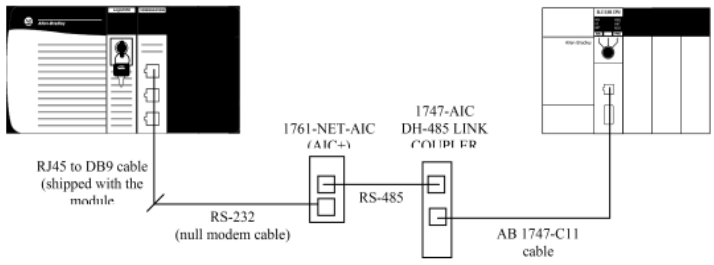
Refer to BASIC CALLs Syntax (page 105) for more information about CALLs 114 and 115.

8 Using DH-485 Communications

In This Chapter

The DH-485 port can also be used by the module to connect the MVI56-BAS to a DH-485 network. The DH-485 port is not directly DH-485 compatible, since it requires an AIC+ module (AB) which is user-supplied.

The following shows an example application where the MVI56-BAS is connected to a DH-485 network. In this example, the MVI56-BAS communicates with the DH-485 port of a SLC 5/03 processor.



Important: The setup jumper (located at the bottom of the MVI56-BAS module) must be OFF in order for DH-485 communications to work properly.

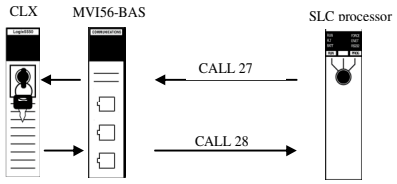
8.1 Data Transfer Between the CLX and a Remote SLC DH-485 Data File

In order to transfer data between the ControlLogix and a remote DH-485 data file, the following CALLs can be used:

CALL 27: Read from a Remote DH-485 SLC Data File

CALL 28: Write to a Remote DH-485 SLC Data File

The following schematic shows the data transfer directions using CALLs 27 and 28.



8.2 Writing to a Remote DH-485 SLC Data File (CALL 28)

Ladder logic is required to transfer data from the local ControlLogix rack to a remote DH-485 SLC data file using the MVI56-BAS through CALL 28.

The source of the data in the local SLC processor is either the CPU output image file, Class 3 MSG instruction, and/or an internal string in the within the MVI56-BAS module.

The data transfer procedure is described below:

Step 1: Run CALL 28 to set up the data transfer parameters.

Step 2: The ladder logic sets output file word 0, bit 11 to inform the MVI56-BAS that the data is ready to be transferred.

Step 3: The MVI56-BAS transfer the data from the ControlLogix processor to the remote DH-485 SLC data file.

Step 4: The MVI56-BAS moves the transfer status into the input file word 1, bits 0 to 7. It then sets input file word 0, bit 11 to inform the ControlLogix that the data had been transferred.

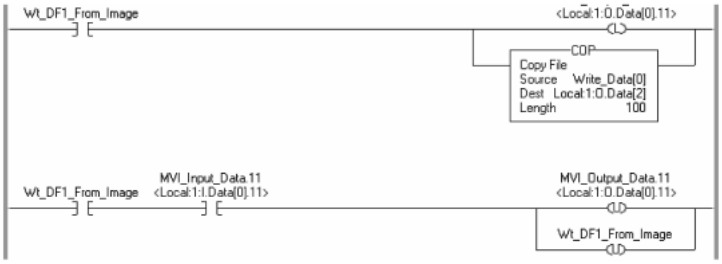
Step 5: The ladder logic resets output file word 0, bit 11.

Step 6: The MVI56-BAS resets the input file word, bit 11.

The following example BASIC code shows the simplest possible example that transfers data from the local SLC (source file = CPU output image file) to a remote SLC DH-485 data file:

```
10 REM TRANSFERS DATA FROM THE CLX TO THE SLC REMOTE DATA FILE
20 PUSH 2, 5, 9, ASC( N), 0, 5, 20, 0, 0, 0: CALL 28: POP S
30 PRINT "S =", S,
130 GOTO 130
```

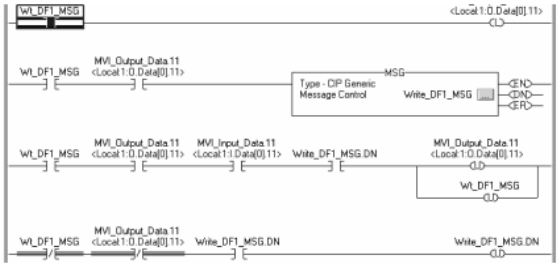
After the program is running, the following ladder example transfers data from the Write_Data array to the remote SLC data file:



The following example BASIC code shows the simplest possible example that transfers data from the local SLC (source file = Class 3 MSG) to a remote SLC DH-485 data file:

```
10 REM TRANSFERS DATA FROM THE CLX TO THE SLC REMOTE DATA FILE
20 PUSH 2, 5, 9, ASC( N), 0, 5, 20, 1, 0, 0: CALL 28: POP S
30 PRINT "S =", S,
```

```
130 GOTO 130
```



Refer to BASIC CALLs Syntax (page 105) for more information about the CALL 28 syntax.

8.3 Reading From a Remote DH-485 SLC Data File (CALL 27)

Ladder logic is required to transfer data from a remote DH-485 SLC data file to the local ControlLogix rack using the MVI56-BAS through CALL 27.

The destination at the local SLC processor is either the CPU output image file, Class 3 MSG instruction, and/or an internal string in the within the MVI56-BAS module.

The data transfer procedure is described below:

Step 1: Run CALL 27 to set up the data transfer parameters.

Step 2: The ladder logic sets output file word 0, bit 10 to inform the MVI56-BAS that the data is ready to be transferred

Step 3: The MVI56-BAS transfers the data from the remote DH-485 SLC data file to the ControlLogix processor.

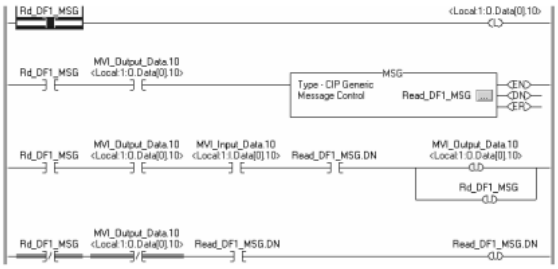
Step 4: The MVI56-BAS moves the transfer status into the input file word 1, bits 0 to 7. After that it sets input file word 0, bit 10 to inform the ControlLogix that the data had been transferred.

Step 5: The ladder logic resets output file word 0, bit 10.

Step 6: The MVI56-BAS resets the input file word, bit 10.

The following example BASIC code shows the simplest possible example that transfers data from the remote SLC DH-485 data file to the local ControlLogix (destination file file = Class 3 MSG instruction):

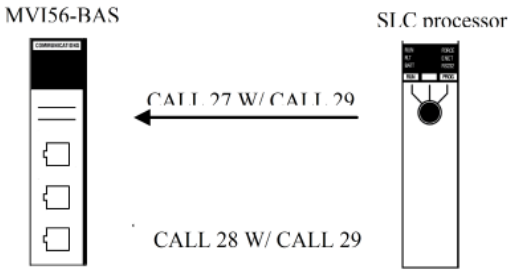
```
10 REM SHOWS HOW TO TRANSFER DATA FROM A SLC REMOTE FILE
80 PUSH 2, 5, 9, ASC( N), 0, 5, 20, 1, 0, 0: CALL 27: POP S
100 GOTO 100
```



Refer to BASIC CALLs Syntax (page 105) for more information about the CALL 27 syntax.

8.4 Data Transfer Between a MVI56-BAS Internal String and a Remote DH-485 SLC Data File

In order to transfer data between a MVI56-BAS internal string and a remote DH-485 SLC data file, use CALL 29 with CALLs 27 or 28. This method does not require ladder logic:



The following is an example BASIC program that uses CALL 29 to read data from a remote DH-485 SLC data file:

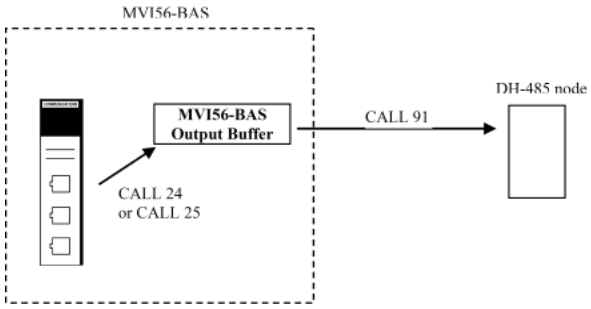
```
10 STRING 100, 90
80 PUSH 2, 5, 9, ASC( N), 0, 5, 20, 2, 0, 0: CALL 27: POP S
90 PUSH 27: CALL 29: POP R
95 PRINT "S =", S, " R =", R
100 GOSUB 1000
130 END
1000 FOR M = 1 TO 20
1010 PH0. ASC( $ ( 0), M),
1020 NEXT M
1030 PRINT
1040 RETURN
```

The following is an example BASIC program that uses CALL 29 to write data to a remote DH-485 SLC data file:

```
10 STRING 100, 90
20 $ ( 0) = "Hello World"
30 PRINT $ ( 0)
80 PUSH 2, 5, 9, ASC( N), 0, 5, 20, 2, 0, 0: CALL 28: POP S
90 PUSH 28: CALL 29: POP R
95 PRINT "S =", S, " R =", R
130 END
```

8.5 Data Transfer Between the MVI56-BAS and Remote DH-485 Data Files

In order to transfer data from the MVI56-BAS output buffer to a Remote DH-485 data file, use CALL 91. It is necessary initially to transfer data from the MVI56-BAS to the internal Output Buffer (starting at address 0) before using CALL 91. Up to 40 words can be transferred using this method.

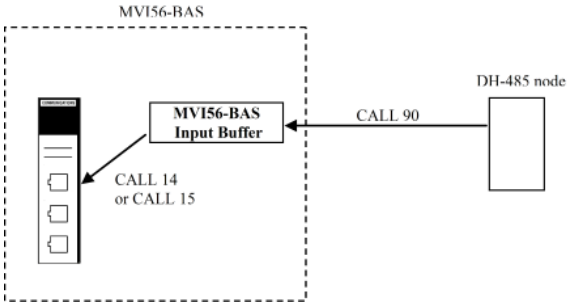


The following BASIC program is an example of how to transfer data using CALL 91. The program initially transfers 20 registers to the module output buffer using CALL 24. After the data has been moved, the program uses CALL 91 to transfer the 20-register block to the remote DH-485 data file.

```
10 FOR I=0 TO 19
20 K=100-I
30 PUSH K: REM VALUE TO BE TRANSFERRED
40 PUSH I: REM OUTPUT BUFFER ADDRESS
50 CALL 24
60 NEXT I
70 REM TEST CALL 91
80 PUSH 5: REM REMOTE NODE ADDRESS
90 PUSH 9: REM REMOTE DEVICE FILE NUMBER
100 PUSH ASC(N): REM REMOTE DEVICE TYPE
110 PUSH 0: REM OFFSET
120 PUSH 20: REM NUMBER OF ELEMENTS
130 PUSH 10: REM TIMEOUT
140 CALL 91
150 POP S: REM STATUS
160 PRINT S
```

Refer to BASIC CALLs Syntax (page 105) for more information about CALLs 24, 25 and 91.

In order to transfer data from a Remote DH-485 data file to the MVI56-BAS input buffer, use CALL 90. After the data has moved into the input buffer, the program must use CALL 14 or 15 to retrieve the data (starting at address 0). Up to 40 words can be transferred using this method.

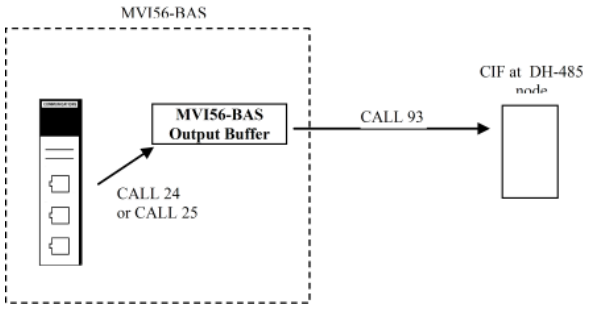


The following BASIC program is an example of how to transfer data using these concepts. CALL 90 transfers data from the DH-485 remote data file to the MVI56-BAS input buffer. Then, use CALL 14 (or CALL 15) to retrieve the data from the input buffer.

```
10 REM EXAMPLE PROGRAM FOR BASIC CALL 90
20 PUSH 5: REM REMOTE NODE ADDRESS
30 PUSH 9: REM REMOTE FILE NUMBER
40 PUSH ASC(N): REM REMOTE FILE TYPE
50 PUSH 0: REM REMOTE OFFSET
60 PUSH 20: REM ELEMENT LENGTH
70 PUSH 5: REM TIMEOUT
80 CALL 90
90 POP E: REM CALL STATUS
100 IF E<>0 THEN GOTO 180 ELSE GOTO 110
110 FOR I=0 TO 19
120 PUSH I: REM THE ADDRESS NUMBER AT THE INPUT BUFFER
130 CALL 14:
140 POP V: REM VALUE AT ADDRESS I
150 PRINT "VALUE AT POSITION ",I," = ",V
160 NEXT I
170 GOTO 200
180 PRINT "CALL 90 ERROR - CODE = ",E
200 END
```

8.6 Transfer Data Between the MVI56-BAS Module and a Remote Common Interface File (CIF)

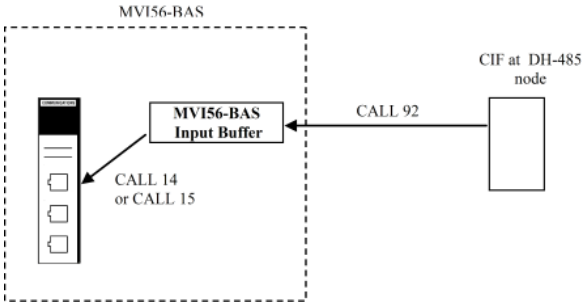
In order to transfer data from the MVI56-BAS output buffer to a Remote Common Interface File, use CALL 93. This CALL transfers up to 40 words of data from the output buffer starting at address 0. The CALL allows the user to configure the remote node address, remote offset number, number of elements, and the timeout. Before using CALL 93, you must use CALLs 24 or 25 to move the data to be transferred from the MVI56-BAS to the output buffer.



The following is a BASIC code example that shows how to transfer data from the MVI56-BAS output buffer to a remote Common Interface File at the DH-485 network:

```
1 REM SAMPLE PROGRAM FOR BASIC CALL 93
10 FOR I=0 TO 19
20 K=200-I
30 PUSH K: REM VALUE TO BE TRANSFERRED
40 PUSH I: REM OUTPUT BUFFER ADDRESS
50 CALL 24
60 NEXT I
80 PUSH 5: REM REMOTE NODE ADDRESS
90 PUSH 0: REM OFFSET
100 PUSH 20: REM NUMBER OF ELEMENTS
110 PUSH 10: REM TIMEOUT
120 CALL 93
130 POP S: REM STATUS
140 PRINT S
150 IF S<>0 PRINT "CALL 93 ERROR: ",S
160 END
```

In order to transfer data from a Remote Common Interface File (CIF) DH-485 data file to the MVI56-BAS input buffer, use CALL 92. After the data has moved into the input buffer, the program must use CALL 14 or 15 to retrieve the data (starting at address 0). Up to 40 words can be transferred using CALL 92.



The following BASIC program is an example of how to transfer data using these concepts. CALL 92 transfers data from the DH-485 remote data file to the MVI56-BAS input buffer. You can then use CALL 14 (or CALL 15) to retrieve the data from the input buffer.

```
10 REM EXAMPLE PROGRAM FOR BASIC CALL 92
20 PUSH 5: REM REMOTE NODE ADDRESS
30 PUSH 0: REM REMOTE OFFSET
40 PUSH 20: REM ELEMENT LENGTH
50 PUSH 20: REM TIMEOUT
60 CALL 92
70 POP E: REM CALL STATUS
80 IF E<>0 THEN GOTO 160 ELSE GOTO 90
90 FOR I=0 TO 19
100 PUSH I: REM THE ADDRESS NUMBER AT THE INPUT BUFFER
110 CALL 14:
120 POP V: REM VALUE AT ADDRESS I
130 PRINT "VALUE AT POSITION ",I," = ",V
140 NEXT I
150 GOTO 170
160 PRINT "CALL 92 ERROR - CODE = ",E
170 END
```

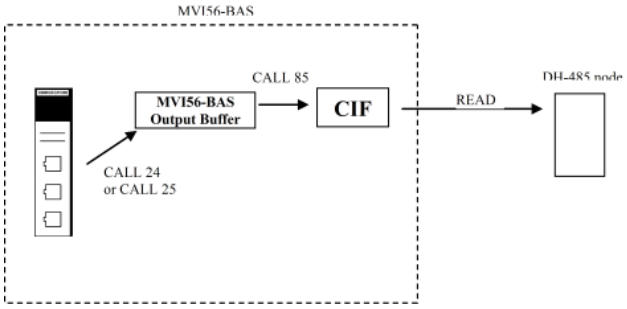
8.6.1 Using the MVI56-BAS Common Interface File (CIF)

The MVI56-BAS has a internal Common Interface File (CIF) that can be used while communicating with other nodes at the DH-485 network. The CIF contains 40 words of data that can be read or written from an external device in the network. In order to transfer data between a MVI56-BAS buffer and the CIF, the following CALLs can be used:

CALL 85: Transfer BASIC Output Buffer to DH-485 CIF

CALL 84: Transfer DH-485 CIF to BASIC Input Buffer

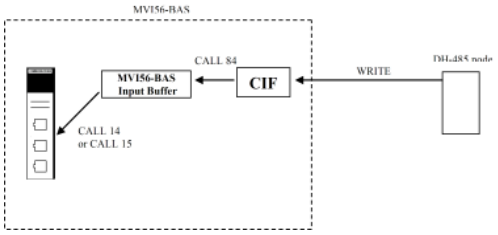
The following is an example application where the data is transferred from the RAM memory to the Output Buffer and from there to the Common Interface File using CALL 85. After the data has been moved to the CIF it can be read by other nodes at the DH-485 network:



The following is example code for BASIC CALL 85:

```
1 REM SAMPLE PROGRAM FOR BASIC CALL 93
10 FOR I=0 TO 19
20 K=200-I
30 PUSH K: REM VALUE TO BE TRANSFERRED
40 PUSH I: REM OUTPUT BUFFER ADDRESS
50 CALL 24
60 NEXT I
70 PUSH 0: REM STARTING OFFSET ADDRESS IN CIF
80 PUSH 10: REM NUMBER OF WORDS TO BE TRANSFERRED
90 CALL 85
100 POP S
110 PRINT "ERROR CODE = ",S
```

The following shows an example application where data is written from a remote DH-485 node to the MVI56-BAS Common Interface File. The BASIC program can transfer the contents of the CIF to the Input Buffer using CALL 84 and then retrieve the data using CALL 14 or 15



The following shows example code for BASIC CALL 84:

```
10 REM EXAMPLE APPLICATION FOR CALL 84
20 PUSH 0: REM OFFSET IN CIF FILE
30 PUSH 0: REM NUMBER OF ELEMENTS TO BE TRANSFERRED
40 CALL 84
50 POP E: REM CALL STATUS
60 IF E<>0 THEN GOTO 140 ELSE GOTO 110
70 FOR I=0 TO 19
80 PUSH I: REM THE ADDRESS NUMBER AT THE INPUT BUFFER
90 CALL 14:
```

```
100 POP V: REM VALUE AT ADDRESS I
110 PRINT "VALUE AT POSITION ",I," = ",V
120 NEXT I
130 GOTO 200
140 PRINT "ERROR CALL 84: ",E
200 END
```

In order to verify if the CIF has been read or written refer to the following BASIC CALLs:

- CALL 86: Checks if any remote device has written to the MVI56-BAS Common Interface File
- CALL 87: Checks if the Common Interface File has been read by the a remote DH-485 device

Refer to BASIC CALLs Syntax (page 105) for more information about CALLs 84, 85, 86 and 87.

9 BASIC CALLs Syntax

In This Chapter

Most BASIC commands, statements, and operators deal with the pure BASIC language itself. However, there are many applications which must interface to real world hardware. The BASIC CALLs perform this hardware interface function. There are CALLs that specifically interface the MVI56 backplane, the serial ports, the on-board wall clock, and so on. In addition, there are CALLs that fulfill BASIC shortcomings such as string manipulation and program flow control. The following topics describe each CALL in detail.

9.1 Data Conversion CALLs

In order for BASIC to communicate with the CLX processor, remote DH-485 modules, and remote DF1 modules, two files have been allocated and defined. These files are called the BASIC Input Buffer and the BASIC Output Buffer. The BASIC Input Buffer contains data that BASIC can read from other devices and is defined as follows:

Address	Definition
0 to 39	DH-485 Common Interface File - data written by other devices
40 to 99	Reserved
100 to 199	Data transferred from the CLX using Message instruction
200 to 231	Data transferred from the CLX Output Image file

The BASIC Output Buffer contains data for other devices which BASIC can write and is defined as follows:

Address	Definition
0 to 39	DH-485 Common Interface File - data read by other devices
40 to 99	Reserved
100 to 199	Data read by the CLX using MSG instruction
200 to 231	Data transferred to the CLX Input Image file

Word 200 in the BASIC Output Buffer is special. This word corresponds to word 0 of the CLX input image. The upper three bits of this word are reserved. Bit 15 indicates whether BASIC is in RUN mode or COMMAND mode (0=RUN, 1=COMMAND). Bits 14 and 13 are reserved. If the user attempts to use word 200 of the BASIC Output Buffer, be aware that the upper three bits will be masked off and used as described above.

9.1.1 Input CALLs

CALL 14: Convert 16-Bit Signed to Float Point

Use CALL 14 to convert a 16-bit signed integer from the MVI56-BAS input buffer to a BASIC floating point value. This CALL can be used to read data transferred from the ControlLogix processor using CALLs 53 and 56.

Syntax:

```
PUSH [A]
CALL 14
POP [B]
```

Where:

A = input buffer word address (0 to 231)

B = converted value

Example:

```
10 REM Example Program
20 PUSH 0: REM Convert 1st word of BASIC Input Buffer
30 CALL 14: REM Do 16-Bit signed to F. P. Conversion
```

```
40 POP W: REM Get converted value
50 PRINT W
```

CALL 15: Convert 16-Bit Unsigned to Float Point

Use CALL 15 to convert a 16-bit unsigned integer from the MVI56-BAS input buffer to a BASIC floating point value. This CALL can be used to read data transferred from the ControlLogix processor using CALLs 53 and 56.

Syntax:

```
PUSH [A]
CALL 14
POP [B]
```

Where:

A = input buffer word address (0 to 231)

B = converted value

Example:

```
10 REM Example Program
20 PUSH 9: REM Convert 10th word of BASIC Input Buffer
30 CALL 15: REM Do 16-Bit unsigned to F. P. Conversion
40 POP W: REM Get converted value
50 PRINT W
```

CALL 89: CLX Floating Point to BASIC Float Point

Use CALL 89 to convert a CLX floating point from the MVI56-BAS input buffer to a BASIC floating point value. This CALL can be used to read data transferred from the ControlLogix processor using CALLs 53 and 56. Because the CLX floating point format requires two words, the offset in the BASIC Input Buffer is limited to even values. Odd offset values will generate an error.

Syntax:

```
PUSH [A]
CALL 89
POP [B]
```

Where:

A = input buffer word address (0 to 230, even values only!)

B = converted value

Example:

```
10 REM Example Program
20 PUSH 0: REM Convert 1st word pair of BASIC Input Buffer
30 CALL 89: REM Do CLX F. P. to BASIC F. P. Conversion
40 POP W: REM Get converted value
50 PRINT W
```

9.1.2 Output CALLs

CALL 24: Convert Floating Point Data to 16-Bit Signed Integer

CALL 24 converts a floating point data value to a 16-bit integer (-32768 to +32767) and places the result into the module output buffer. If the value to be converted is less than -32768, then -32768 will be used. If the value to be converted is greater than 32767, then 32767 will be used. No error will be generated in either case.

Syntax:

```
PUSH [A]
PUSH [B]
CALL 24
```

Where:

A = value to be converted

B = word address of output buffer (between 0 and 231)

Example:

```
10 DIM A(32)
.
.
.
210 GOSUB 720
```

```
.
.
700 REM This routine copies data from variable A()
710 REM to the BASIC Output Buffer Input Image area
720 FOR I = 0 TO 31
730 PUSH A(I),I+200: CALL 24
740 NEXT I
750 RETURN
```

CALL 25: Convert Floating Point Data to 16-Bit Binary

CALL 25 converts a floating point data value to a 16-bit binary representation (0 to 65535) and places the result into the module output buffer. If the value to be converted is less than 0, then 0 will be used. If the value to be converted is greater than 65535, then 65535 will be used. No error will be generated in either case.

Syntax:

```
PUSH [A]
PUSH [B]
CALL 25
```

Where:

A = value to be converted

B = word address of output buffer (between 0 and 231)

Example:

```
10 DIM A(32)
.
.
.
210 GOSUB 720
.
.
700 REM This routine copies data from variable A()
710 REM to the BASIC Output Buffer Input Image area
720 FOR I = 0 TO 31
730 PUSH A(I),I+200: CALL 25
740 NEXT I
750 RETURN
```

CALL 88: Convert BASIC Floating Point Data to CLX Floating Point

CALL 88 converts a BASIC floating point data value to a CLX floating point value and places the result into the module output buffer. If the value to be converted is less than 1.1754944E-38, then 1.1754944E-38 will be used. If the value to be converted is greater than 3.4028237E38, then 3.4028237E38 will be used. No error will be generated in either case. Because the CLX floating point format requires two words, the offset in the BASIC Input Buffer is limited to even values. Odd offset values will generate an error.

Syntax:

```
PUSH [A]
PUSH [B]
CALL 25
```

Where:

A = value to be converted

B = word address of output buffer (between 0 and 230, even values only!)

Example:

```
10 DIM A(16)
.
.
.
210 GOSUB 720
.
.
700 REM This routine copies data from variable A()
710 REM to the BASIC Output Buffer Input Image area
720 FOR I = 0 TO 15
730 PUSH A(I),I*2+200: CALL 88
740 NEXT I
750 RETURN
```

9.2 Backplane CALLs

CALL 51: Check CLX Output Image

Use CALL 51 to determine if the CLX controller has updated the BASIC Input Buffer Output Image since the last time CALL 51 was executed (In this case, update means that the data was written to the Buffer from the CLX, even if the data has not changed values)

Syntax:

```
CALL 51
POP [A]
```

Where:

A =	0 → CLX has not updated the output image
	1 → CLX has updated the output image

Example:

```
.
.
.
125 GOSUB 520
.
.
.
500 REM This routine waits for the output image
510 REM to be updated.
520 CALL 51: POP X
530 CALL 51: POP X
540 IF(X=0) THEN GOT 530
550 RETURN
.
.
.
```

CALL 53: Transfer CLX Output Image to BASIC Input Buffer

Use CALL 53 to transfers 31 words (word 0 to 31) from the CPU output image table to words 200 to 231 of the MVI56-BASIC input buffer.

Syntax:

```
CALL 53
POP [A]
```

Where:

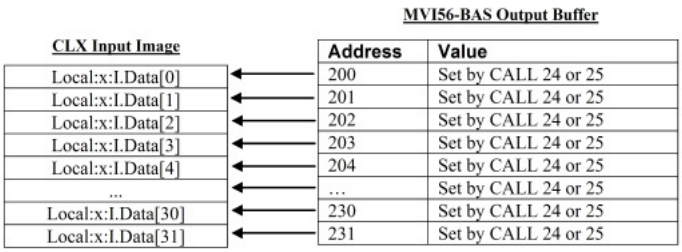
A =	0 → CLX is in RUN mode
	1 → CLX is not in RUN mode

Example:

```
.
.
.
125 GOSUB 520
.
.
.
500 REM This routine transfers the CLX Output Image
510 REM to the BASIC Input Buffer
520 CALL 53: POP X
530 IF(X<>0) THEN PRINT "Processor not in RUN mode."
540 RETURN
.
.
.
```

CALL 54: Transfer Data from BASIC Output Buffer to CLX Input Image

CALL 54 transfer words 200 to 231 (32 words) from the module output buffer to the CLX input image (Local:x:I.Data[ ]). No ladder logic is required to perform the data transfer.



Syntax:

```
CALL 54
POP A
```

Where:

A =	0 → CLX is in RUN mode
	1 → CLX is not in RUN mode

Example:

```
10 DIM A(32)
.
.
210 GOSUB 720
220 GOSUB 820
.
.
700 REM This routine copies data from variable A()
710 REM to the BASIC Output Buffer Input Image area
720 FOR I = 0 TO 31
730 PUSH A(I),I+200: CALL 25
740 NEXT I
750 RETURN
.
.
800 REM This routine copies the BASIC Output Buffer
810 REM Input Image Area to the CLX Input Image
820 CALL 54: POP S
830 IF(S<>0) THEN PRINT "CLX not in RUN mode"
840 RETURN
.
.
.
```

CALL 55: Check CLX Input Image

Use CALL 55 to determine if the CLX processor input image buffer located in the module has been read by the CLX since the last time it was checked.

Syntax:

```
CALL 55
POP A
```

Where:

A =	0 → CLX has not read the input image
	1 → CLX has read the input image

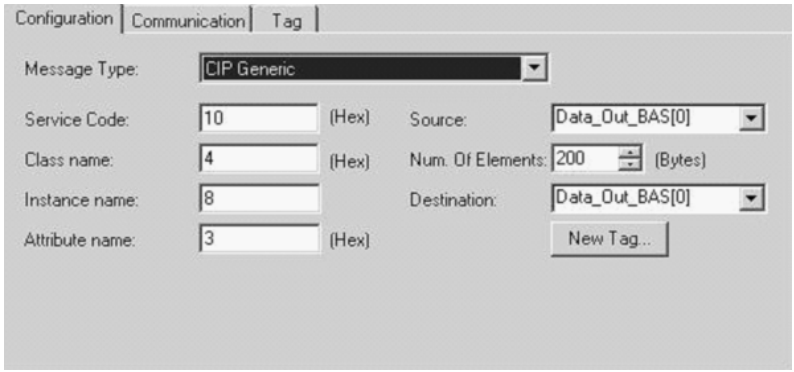
Example:

```
.
.
.
125 GOSUB 520
.
.
.
500 REM This routine waits for the Input Image
510 REM to be read.
520 CALL 55: POP X
530 CALL 55: POP X
540 IF(X=0) THEN GOT 530
550 RETURN
.
.
.
```

CALL 56: Transfer Data from CLX to BASIC Input Buffer using MSG

Use CALL 56 to transfer up to 100 words from a CLX data file to the MVI56-BAS input buffer (starts at address 100) using ladder MSG instruction. The MSG instruction has to be executed before CALL 56 execution.

The ladder MSG instruction has the following parameters.



Syntax:

```
PUSH [A]
CALL 56
POP [B]
```

Where:

A = number of words to be transferred

B = transfer status

The parameter B can assume one of the following values:

0 = Successful Transfer and CLX in Run Mode

1 = Successful Transfer and CLX in Program Mode

2 = Successful Transfer and CLX in Test Mode

10 = Illegal length

Example:

```
.
.
210 GOSUB 720
.
.
700 REM This routine copies the latest CLX
710 REM MSG write data to the BASIC Input Image
720 PUSH 100: CALL 56: POP X
730 IF(X=10) THEN PRINT "Illegal CALL 56 Input Parameter"
740 IF(X=0) THEN RETURN
750 PRINT "CLX is not in RUN mode"
760 RETURN
```

CALL 57: Transfer BASIC Output Buffer to CLX using MSG

CALL 57 can be used to transfer up to 100 words from the BASIC module output buffer to a CLX data file using a MSG instruction. This CALL can be used to transfer word address 100 to 199 in the BASIC output buffer.

Syntax:

```
PUSH A
CALL 57
POP B
```

Where:

A = number of words to be transferred (up to 100)

B = call status (0= successful transfer CLX in Run, 1= successful transfer CLX in Program, 2= successful transfer CLX in Test, 10=invalid length)

Example:

```
10 DIM A(100)
.
.
.
210 GOSUB 720
220 GOSUB 820
.
.
700 REM This routine copies data from variable A()
710 REM to the BASIC Output Buffer CLX MSG area
720 FOR I = 0 TO 99
730 PUSH A(I),I+100: CALL 25
740 NEXT I
750 RETURN
.
.
.
800 REM This routine copies the BASIC Output Buffer
810 REM CLX MSG Area to the CLX
820 PUSH 100: CALL 57: POP S
830 IF(S<>0) THEN PRINT "CLX not in RUN mode"
840 RETURN
.
.
.
```

CALL 58: Check CLX MSG

Use CALL 58 to determine if the CLX processor has written the module data using the MSG instruction since the last time it was checked.

Syntax:

```
CALL 58
POP A
```

Where:

A =	0 → CLX has not written to the module
	1 → CLX has written to the module

Example:

```
.
.
.
125 GOSUB 520
.
.
.
500 REM This routine waits CLX MSG instruction
510 REM to update the module.
520 CALL 58: POP X
530 CALL 58: POP X
540 IF(X=0) THEN GOT 530
550 RETURN
.
.
.
```

CALL 59: Check CLX MSG

Use CALL 59 to determine if the CLX processor has read the module data using the MSG instruction since the last time it was checked.

Syntax:

```
CALL 58
POP A
```

Where:

A =	0 → CLX has not read to the module
	1 → CLX has read to the module

Example:

```
.
.
.
125 GOSUB 520
.
.
.
500 REM This routine waits CLX MSG instruction
510 REM to read the module data.
520 CALL 59: POP X
530 CALL 59: POP X
540 IF(X=0) THEN GOT 530
550 RETURN
.
.
.
```

CALL 75: Check CLX Status

Use CALL 75 to determine if the CLX processor is in Run or Program mode.

Syntax:

```
CALL 75
POP A
```

Where:

A =	0 → CLX is in RUN mode
	1 → CLX is not in RUN mode

Example:

```
.
.
.
120 CALL 75: POP STATUS
130 IF(STATUS=0) THEN GOTO 200
140 REM CLX is not in RUN mode
.
.
.
200 REM CLX is in RUN mode
.
.
.
```

CALL 120: Clear Module Input and Output Buffers

Use CALL 120 to clear the module input and output buffers.

Syntax:

```
PUSH A
CALL 120
```

Where:

A is an 8-bit argument that corresponds to the module input and output buffers, as shown in the following table.

Bit	Decimal Equivalent	Module Input and Output Buffer Area
0	1	Data transferred from the CLX using Message instruction
1	2	Data read by the CLX using MSG instruction
2	4	CLX Output Image
3	8	CLX Input Image
4	16	Common Interface Input File
5	32	Common Interface Output File
6		Not Used
7		Not Used

Example:

```
.
.
.
125 PUSH 63: CALL 120
.
.
.
```

9.3 Serial Port CALLs

CALL 30: Set PRT2 Port Parameters

Use CALL 30 to set the port communication parameters for PRT2.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
CALL 30
```

Where:

A = bits per word (5,6,7,8)

B = parity enable (0 = none, 1= Odd, 2=Even)

C = number of stop bits (1 = 1 stop bits, 2 = 2 stop bits, 3 = 1.5 stop bits)

D = software handshaking (0 = none, 1 = XON-XOF)

E = hardware handshaking (0=Disabled DCD, 1=Enabled DCD)

Example:

```
.
.
.
125 PUSH 8,0,1,1,0: CALL 30
.
.
.
```

CALL 31: Display PRT2 Port Setup

CALL 31 displays the current PRT2 configuration parameters.

Syntax:

```
CALL 31
```

Example:

```
Ready
>call 31
19200 BAUD
Hardware Handshaking OFF
1 Stop Bit
No Parity
8 Bits/Char
Xon/Xoff
Ready
>
```

CALL 35: Get Input Character From PRT2

Use CALL 35 to retrieve the current character in the PRT2 input buffer.

Syntax:

```
CALL 35
POP [S]
```

Where:

S = ASCII value of character

Example:

```
.
.
.
125 CALL 35: POP C
.
.
.
```

CALL 36: Get Number of Characters in PRT2 Buffer

Use CALL 36 to retrieve the number of characters in PRT2 buffer.

Syntax:

```
PUSH A
CALL 36
POP B
```

Where:

A = buffer selection (0 = transmit buffer, 1 = receive buffer)

B = number of characters

Example:

```
.
.
.
210 PUSH 0: CALL 36: POP NUMTXCH
.
.
.
```

CALL 37: Clear PRT2 Buffers

Use CALL 37 to clear the PRT2 receive and transmit buffers.

Syntax:

```
PUSH A
CALL 37
```

Where:

A = buffer selection (0 = transmit buffer, 1 = receive buffer, 2 = both buffers)

Example:

```
.
.
.
210 PUSH 2: CALL 37
.
.
.
```

CALL 78: Set Program Port Baud Rate

Use CALL 78 to set the program port baud rate. By default the program port is PRT1.

Syntax:

```
PUSH [A]
CALL 78
```

Where:

A = baud rate (300, 600, 1200, 2400, 4800, 9600 or 19200)

Example:

```
.
.
.
125 PUSH 19200: CALL 78
.
.
.
```

CALL 94: Display PRT1 Port Setup

CALL 94 displays the current PRT1 configuration parameters.

Syntax:

```
CALL 94
```

Example:

```
Ready
>call 94
19200 BAUD
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
Ready
>
```

CALL 95: Get Number of Characters in PRT1 Buffer

Use CALL 95 to retrieve the number of characters in PRT1 buffer.

Syntax:

```
PUSH A
CALL 95
POP B
```

Where:

A = buffer selection (0 = transmit buffer, 1 = receive buffer)

B = number of characters

Example:

```
.
.
.
210 PUSH 0: CALL 95: POP NUMTXCH
220 PUSH 1: CALL 95: POP NUMRXCH
.
.
.
```

CALL 96: Clear PRT1 Buffers

Use CALL 96 to clear the PRT1 receive or transmit buffers.

Syntax:

```
PUSH A
CALL 96
```

Where:

A = buffer selection (0 = transmit buffer, 1 = receive buffer, 2 = both buffers)

Example:

```
.
.
.
210 PUSH 2: CALL 96
.
.
.
```

CALL 97: Set PRT2 DTR Signal

Use CALL 97 to set (enable) the PRT2 DTR signal.

Syntax:

```
CALL 97
```

Example:

```
.
.
.
210 CALL 97
.
.
.
```

CALL 98: Clear PRT2 DTR Signal

Use CALL 98 to clear (disable) the PRT2 DTR signal.

Syntax:

```
CALL 98
```

Example:

```
.
.
.
210 CALL 98
.
.
.
```

CALL 99: Reset Print Head Pointer

Use CALL 99 to reset the internal print head character counter. In the original BASIC-52, the serial port drivers would automatically insert a CR/LF after 79 characters. MVI56-BAS does not automatically insert the CR/LF after 79 characters. However, the TAB function does keep track of how many characters have been transmitted out each port. These character counts are cleared after a CR has been transmitted. CALL 99 will reset the TAB character count for all ports without having to transmit a CR.

Syntax:

```
CALL 99
```

Example:

```
.
.
.
210 CALL 99
.
.
.
```

CALL 105: Reset PRT1 to Default Parameters

Use CALL 105 to reset the PRT1 communication parameters to its default settings.

Syntax:

```
CALL 105
```

Example:

```
.
.
.
210 CALL 105
.
.
.
```

CALL 119: Reset PRT2 to Default Parameters

Use CALL 119 to reset the PRT2 communication parameters to its default settings.

Syntax:

```
CALL 119
```

Example:

```
.
.
.
210 CALL 119
.
.
.
```

9.4 Wall Clock CALLs

CALL 40: Set Clock Time

Use CALL 40 to set the hours, minutes and seconds.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
CALL 40
```

Where:

A = hours (0 to 23)

B = minutes (0 to 59)

C = seconds (0 to 59)

Example:

```
.
.
.
210 PUSH 3,12,0: CALL 40
.
.
.
```

CALL 41: Set Calendar Date

Use CALL 41 to set the date, month and year.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
CALL 41
```

Where:

A = day (1 to 31)

B = month (1 to 12)

C = year (0 to 99)

Example:

```
.
.
.
210 PUSH 31,8,54: CALL 41
.
.
.
```

CALL 42: Set Day of the Week

In the 1746-BAS and 1771-DB, the internal calendar had no intelligence. CALL 42 would actually set the calendar day of week to any day the programmer suggested. However, the MVI56-BAS uses the DOS calendar. The DOS calendar has enough intelligence to calculate the proper day of week from the date. So, CALL 42 actually does nothing.

Syntax:

```
PUSH [A]
CALL 42
```

Where:

A = day of the week (1 = Sunday, 2 = Monday, 3 = Tuesday, 4 = Wednesday, 5 = Thursday, 6 = Friday, 7 = Saturday)

Example:

```
Ready
>list
10 STRING 100, 30
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Jan. 23, 2003
50 PUSH 23, 1, 3: CALL 41
60 REM Set Day of Week to Monday
70 PUSH 2: CALL 42
80 REM Print Day of Week
90 PUSH 1: CALL 47
100 PRINT $ ( 1)
Ready
>run
THU
Ready
>
```

CALL 43: Retrieve Date and Time String

Use CALL 43 to copy the current date and time to a string format.

Syntax:

```
PUSH [A]
CALL 43
```

Where:

A = the string number where the date and time is stored. The format is:
DD-MMM-YY HH:MM:SS
(at least 18 characters should be allocated)

Example:

```
Ready
>list
10 STRING 100, 30
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Aug. 31, 1954
50 PUSH 31, 8, 54: CALL 41
60 REM Print Time and Date
70 PUSH 1: CALL 43
80 PRINT $ ( 1)
Ready
>run
31-AUG-54 10:30:00
Ready
>
```

CALL 44: Retrieve Date Numeric

Use CALL 44 to return the current date on the argument stack as three numeric values.

Syntax:

```
CALL 44
POP [A]
POP [B]
POP [C]
```

Where:

A = day

B = month

C = year

Example:

```
Ready
>list
10 STRING 100, 30
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Aug. 31, 1954
50 PUSH 31, 8, 54: CALL 41
60 REM Print Time and Date
70 PUSH 1: CALL 43
80 PRINT $ ( 1)
90 CALL 44: POP D, M, Y
100 PRINT D, M, Y
Ready
>run
31-AUG-54 10:30:00
 31 8 54
Ready
>
```

CALL 45: Retrieve Date String

Use CALL 45 to return the current time in string format (HH:MM:SS). A STRING command must be used prior to CALL 45 in order to allocate memory for the string.

Refer to the String Functions section.

Syntax:

```
PUSH [A]
CALL 45
```

Where:

A = the string number where the time is stored .

Example:

```
Ready
>list
10 STRING 100, 30
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Aug. 31, 1954
50 PUSH 31, 8, 54: CALL 41
60 REM Print Time and Date
70 PUSH 1: CALL 45
80 PRINT $ ( 1)
90 CALL 44: POP D, M, Y
100 PRINT D, M, Y
Ready
>run
10:30:00
 31 8 54
Ready
   >
```

CALL 46: Retrieve Time Numeric

Use CALL 46 to return the time of the day in numeric form.

Syntax:

```
CALL 46
POP[A]
POP[B]
POP[C]
```

Where:

A = hour

B = minute

C = second

Example:

```
Ready
>list
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Aug. 31, 1954
50 PUSH 31, 8, 54: CALL 41
60 REM Print Time and Date
70 CALL 46: POP HR, MIN, SEC
80 PRINT HR, MIN, SEC
90 CALL 44: POP D, M, Y
100 PRINT D, M, Y
Ready
>run
 10 30 0
 31 8 54
Ready
>
```

CALL 47: Retrieve Day of Week String

Use CALL 47 to return the current day of the week as a 3 character string.

Syntax:

```
PUSH [A]
CALL 47
```

Where:

A = string number

Example:

```
Ready
>list
10 STRING 100, 30
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Jan. 23, 2003
50 PUSH 23, 1, 3: CALL 41
60 REM Set Day of Week to Monday
70 PUSH 2: CALL 42
80 REM Print Day of Week
90 PUSH 1: CALL 47
100 PRINT $ ( 1)
Ready
>run
THU
Ready
>
```

CALL 48: Retrieve Day of Week Numeric

Use CALL 48 to return the current day of the week on the argument stack as a number.

Syntax:

```
CALL 48
POP[A]
```

Where:

A = day of the week (1 = Sunday, 2 = Monday, 3 = Tuesday, 4 = Wednesday, 5 = Thursday, 6 = Friday, 7 = Saturday)

Example:

```
10 STRING 100, 30
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Jan. 23, 2003
50 PUSH 23, 1, 3: CALL 41
60 REM Set Day of Week to Saturday
70 PUSH 2: CALL 42
80 REM Print Day of Week
90 CALL 48: POP DW
100 PRINT DW
Ready
>run
 5
Ready
>
```

CALL 52: Retrieve Date String

Use CALL 52 to return the current date in a string format.

Syntax:

```
PUSH [A]
CALL 52
```

Where:

A = string number. The string will have the following format: DD-MMM-YY

Example:

```
Ready
>list
10 STRING 100, 30
20 REM Set time to 10:30:00
30 PUSH 10, 30, 0: CALL 40
40 REM Set date to Jan. 23, 2003
50 PUSH 23, 1, 3: CALL 41
60 REM Set Day of Week to Saturday
80 REM Print Date
90 PUSH 1: CALL 52
100 PRINT $ ( 1)
Ready
>run
23-JAN-03
Ready
>
```

9.5 String CALLs

This section describes the string manipulation commands. Remember that before using any command listed in this section you will have to allocate memory space for strings using the STRING command which is defined below.

Before referencing strings or using any string commands you must initially allocate memory space for string usage. The `STRING` command should be used in order to allocate memory space for strings:

```
STRING [A],[B]
```

Where:

A = total number of bytes to be allocated

B = maximum number of bytes to be allocated for each string

The MVI56-BAS requires an extra byte for each string and an additional overhead byte. So, "10 STRING 100, 10" would allocate enough space for 9 strings (11 bytes each plus one more byte is 100 bytes) with no space to spare.

CALL 60: Repeating a Character

In order to repeat a character in a string use CALL 60:

```
PUSH [A]
PUSH [B]
CALL 60
```

Where:

A = number of times

B = string number that contains one character

Example:

```
Ready
>list
10 STRING 200, 40
20 $ ( 1) = "*"
30 PUSH 40, 1: CALL 60
40 PRINT $ ( 1)
50 END
Ready
>run
****************************************
Ready
>
```

CALL 61: Concatenating a String

CALL 61 concatenates a string to another:

```
PUSH [A]
PUSH [B]
CALL 61
```

Where:

A = string number to be appended

B = string number where A will be appended to

Example:

```
Ready
>list
10 STRING 200, 20
20 $ ( 1) = "How are "
30 $ ( 2) = "you?"
40 PRINT "Before:"
50 PRINT $ ( 1)
60 PRINT $ ( 2)
70 PUSH 2, 1: CALL 61
100 PRINT "After:"
110 PRINT $ ( 1)
120 PRINT $ ( 2)
130 END
Ready
>run
Before:
How are
you?
After:
How are you?
you?
Ready
>
```

CALL 62: Converting from Numeric Format to String Format

CALL 62 converts a numeric value to string format:

```
PUSH [A]
PUSH [B]
CALL 62
```

Where:

A = numerical variable to be converted to string format.

B = string number that A will be converted to

Example:

```
Ready
>list
10 STRING 100, 14
20 INPUT "Enter a number to convert to a string ", N
30 PUSH N, 1: CALL 62
40 PRINT $ ( 1)
50 END
Ready
>run
Enter a number to convert to a string 1.23
1.23
Ready
>
```

CALL 63: Converting from String Format to Numeric Format

CALL 63 converts a string to the numeric format:

```
PUSH [A]
CALL 63
POP [B]
POP [C]
```

Where:

A = string number to be converted containing a decimal number [0..255]

B = status (0 = no valid number found , not 0 = valid decimal number found on A)

C = string A converted to numeric value

Example:

```
Ready
>list
10 STRING 100, 20
20 INPUT "Input a string to convert: ", $ ( 1)
30 PUSH 1: CALL 63: POP V, N
40 IF ( V <> 0) THEN PRINT $ ( 1), " ", N: GOTO 60 ""
50 PRINT "Invalid or no value found"
60 GOTO 20
Ready
>run
Input a string to convert: 1.23Hello
1.23Hello 1.23
Input a string to convert: 456World
456World 456
Input a string to convert: ^
^C struck!
 - in line 30
Ready
>
```

CALL 64: Finding a String within Another String

CALL 64 finds a string in another string returning the first occurrence location in the string:

```
PUSH [A]
PUSH [B]
CALL 64
POP [C]
```

Where:

A = string number to be found

B = string number searched for string A

C = position number in string B

Example:

```
Ready
>list
10 STRING 100, 20
20 $ ( 1) = "456"
30 $ ( 2) = "12345678"
40 PUSH 1, 2: CALL 64: POP L
50 IF ( L = 0) THEN PRINT "Not found!"
60 IF ( L > 0) THEN PRINT "Found at location", L
70 END
Ready
>run
Found at location 4
Ready
>
```

CALL 65: Replacing a String in Another String

CALL 65 replaces a string in another string:

```
PUSH [A]
PUSH [B]
PUSH [C]
CALL 65
```

Where:

A = string number that will replace string B in string C

B = string number that will be replaced by string A

C = string number that will have substring B replaced by string A

Example:

```
Ready
>list
10 STRING 100, 20
20 $ ( 0) = "Red-lines"
40 $ ( 1) = "Red"
50 $ ( 2) = "Blue"
60 PRINT "Before:"
70 PRINT "$(0) = ", $ ( 0)
80 PUSH 2, 1, 0: CALL 65
90 PRINT "After:"
100 PRINT "$(0) = ", $ ( 0)
110 END
Ready
>run
Before:
$(0) = Red-lines
After:
$(0) = Blue-lines
Ready
>
```

CALL 66: Inserting a String in Another String

CALL 66 inserts a string within another string:

```
PUSH [A]
PUSH [B]
PUSH [C]
CALL 66
```

Where:

A = position number in string C

B = string number that will be inserted in string C

C = string number where string B will be inserted

Example:

```
Ready
>list
10 STRING 100, 15
20 $ ( 0) = "1234590"
30 $ ( 1) = "67890"
40 PRINT "Before:"
50 PRINT "$(0) = ", $ ( 0)
60 PUSH 6, 1, 0: CALL 66
70 PRINT "After:"
80 PRINT "$(0) = ", $ ( 0)
Ready
>run
Before:
$(0) = 1234590
After:
$(0) = 123456789090
Ready
>
```

CALL 67: Deleting a String in Another String

CALL 67 deletes a string within another string:

```
PUSH [A]
PUSH [B]
CALL 67
```

Where:

A = string number which will have substring B deleted from

B = string number that will be deleted

Example:

```
Ready
>list
10 STRING 100, 14
20 $ ( 1) = "123456789012"
30 $ ( 2) = "12"
40 PRINT "Before:"
50 PRINT "$(1) = ", $ ( 1)
60 PUSH 1, 2: CALL 67
70 PRINT "After:"
80 PRINT "$(1) = ", $ ( 1)
90 END
Ready
>run
Before:
$(1) = 123456789012
After:
$(1) = 3456789012
Ready
>
```

CALL 68: Determining the Length of a String

CALL 68 returns the length of a string:

```
PUSH [A]
PUSH [B]
CALL 68
```

Where:

A = string number

B = string A length (number of characters)

Example:

```
Ready
>list
10 STRING 100, 10
20 $ ( 1) = "1234567"
30 PUSH 1: CALL 68: POP L
40 PRINT "The string length of $(1) is", L
50 END
Ready
>run
The string length of $(1) is 7
Ready
>
```

9.6 DH-485 CALLs

CALL 27: Read Remote DH-485 SLC Data File

CALL 27 reads up to 99 words from a remote DH-485 node data file to the CLX or a string within the MVI56-BAS module. The DH-485 section in this User Manual explains how to implement this BASIC CALL in detail.

Note: If using an internal string as the destination file, you may use CALL 29 to initiate the data transfer without ladder logic.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
PUSH [G]
PUSH [H]
PUSH [I]
PUSH [J]
CALL 27
POP [S]
```

Where:

A = Type of DH-485 SLC Read Command to be issued. The following types are supported:

Code	Command
0	Disable the previously executed CALL 27
1	Common Interface File Read
2	SLC Type Read

B = Node address of the SLC remote device (0 through 31)

C = File Number on the SLC remote device (0 through 255)

D = File type of the to be read from the remote device. The following types are supported:

File Type Code	File Type Name	Words/Elements	Valid Range
ASC(N)	Integer File	1 word/element	1 to 99
ASC(C)	Counter File	3 words/element	1 to 33
ASC(T)	Timer File	3 words/element	1 to 33
ASC(B)	Bit File	1 word/element	1 to 99
ASC(R)	Control File	3 words/element	1 to 33

E = The word offset on the remote device file (0 through 32767)

F = Number of word to be transferred. Refer to the previous table for possible values. If using the Common Interface File, use a value between 1 to 64.

G = Message Timeout (1 through 255. For example: 1 = 100ms, 25 = 250 ms, and so on)

H = Data Destination. Use one of the following codes:

Destination File Code	Description
0	CPU Input Image File
1	CPU M1 File
2	Internal String
3	CPU Input Image File and Internal String
4	CPU M1 file and Internal String

I = The word offset in the destination file.

Note: If the parameter H chosen as 0, the words 0 and 1 are reserved.

J = The string number.

S = The CALL result code.

Code	Description
0	Successful
1	Disabled
2	Bad Input Parameter
3	Port DH-485 not enabled (DF1 enabled)
4	String is too small
5	String is not dimensioned

Example:

```
.
.
.
90 PUSH 2,2,7,ASC(N),100,20,5,1,0,0: CALL 27: POP STATUS
100 IF(STATUS<>0) THEN PRINT "Unsuccessful CALL 27 Setup"
110 REM CALL 27 remains active as long as BASIC
120 REM continues to RUN.
.
.
.
```

CALL 28: Write to Remote DH-485 SLC Data File

CALL 28 writes up to 99 words of data from the ControlLogix processor (or MVI56-BAS string) to a remote DH-485 node data file. The DH-485 section in this User Manual explains how to implement this BASIC CALL in detail.

Note: If using an internal string as the destination file, you may use CALL 29 to initiate the data transfer without ladder logic.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
PUSH [G]
PUSH [H]
PUSH [I]
PUSH [J]
CALL 27
POP [S]
```

Where:

A = Type of DH-485 SLC Write Command to be issued. The following types are supported:

Code	Command
0	Disable the previously executed CALL 28
1	Common Interface File Write
2	SLC Type Write

B = Node address of the SLC remote device (0 through 31)

C = File Number on the SLC remote device (0 through 255)

D = File type to be written to the remote device. The following types are supported.

File Type Code	File Type Name	Words/Elements	Valid Range
ASC(N)	Integer File	1 word/element	1 to 99
ASC(C)	Counter File	3 words/element	1 to 33
ASC(T)	Timer File	3 words/element	1 to 33
ASC(B)	Bit File	1 word/element	1 to 99
ASC(R)	Control File	3 words/element	1 to 33

E = The word offset on the remote device file (0 through 32767)

F = Number of word to be transferred. Refer to the previous table for possible values. If using the Common Interface File, use a value between 1 and 64.

G = Message Time out (1 through 255 For example, 1 = 100ms, 25 = 250 ms, and so on.)

H = Data Source. Use one of the following codes:

Destination File Code	Description
0	CPU Output Image File
1	CPU M0 File
2	Internal String

I = The word offset in the destination file.

Note: If the parameter H is chosen as 0, the words 0 and 1 are reserved.

J = The string number.

S = The CALL result code.

Code	Description
0	Successful
1	Disabled
2	Bad Input Parameter
3	Port DH-485 not enabled (DF1 enabled)
4	String is too small
5	String is not dimensioned
6	Data packet is too long

Example:

```
.
.
.
90 PUSH 2,1,7,ASC(N),0,20,10,1,0,0: CALL 28: POP STATUS
100 IF(STATUS<>0) THEN PRINT "Unsuccessful CALL 28 Setup"
110 REM CALL 28 remains active as long as BASIC
120 REM continues to RUN.
.
.
.
```

CALL 83: Display DH-485 Port Parameters

Use CALL 83 to display the DH-485 port setup parameters.

Syntax:

```
CALL 83
```

Example:

```
Ready
>call 83
19200 Baud
Host Node Address 0
Module Node Address 1
Maximum Node Address 31
Ready
>
```

CALL 84: Transfer DH-485 Interface File to MVI56-BAS Input Buffer

CALL 84 transfers up to 40 words of data from the DH-485 CIF to the BASIC input buffer.

Syntax:

```
PUSH [A]
PUSH [B]
CALL 84
POP [S]
```

Where:

A = word offset in DH-485 interface file

B = number of words to be transferred

S = CALL status

Code	Description
0	Successful
1	Illegal offset
2	Illegal number of words

Example:

```
.
.
.
100 GOSUB 1020
.
.
.
1000 REM This routine copies the CIF File to the
1010 REM BASIC Input Buffer
1020 PUSH 0,32: CALL 84: POP STATUS
1030 IF(STATUS<>0) THEN PRINT "Transfer error code =",STATUS
1040 RETURN
.
.
.
```

CALL 85: Transfer MVI56-BAS Output Buffer to DH-485 Interface File

CALL 85 transfers up to 40 words of data from the BASIC output buffer to the DH-485 CIF file.

Syntax:

```
PUSH [A]
PUSH [B]
CALL 85
POP [S]
```

Where:

A = word offset in DH-485 interface file

B = number of words to be transferred

S = CALL status

Code	Description
0	Successful
1	Illegal offset
2	Illegal number of words

Example:

```
.
.
.
100 GOSUB 1020
.
.
.
1000 REM This routine copies the BASIC Output Buffer to the
1010 REM CIF
1020 PUSH 0,32: CALL 85: POP STATUS
1030 IF(STATUS<>0) THEN PRINT "Transfer error code =",STATUS
1040 RETURN
.
.
.
```

CALL 86: Check DH-485 Interface File Remote Write Status

CALL 86 determines if the DH-485 CIF file was written by a DH-485 node device.

Syntax:

```
CALL 86
POP [S]
```

Where:

S = CALL Status

Status	Description
0	No device has written to the CIF file since last CALL or power up.
1	A device has written to the CIF file since last CALL or power up.

Example:

```
.
.
.
100 GOSUB 1040
.
.
.
1000 REM This routine waits for the CIF to be written to.
1010 REM CALL 86 simply tells us that the CIF has been
1020 REM written to some time in the past. This routine
1030 REM waits for the CIF to be updated again.
1040 CALL 86: POP STATUS
1050 CALL 86: POP STATUS
1060 IF(STATUS=0) THEN GOTO 1050
1070 RETURN
.
.
.
```

CALL 87: Check DH-485 Interface File Remote Read Status

CALL 87 determines if the DH-485 CIF file was read by a DH-485 node device.

Syntax:

```
CALL 87
POP [S]
```

Where:

S = CALL Status

Status	Description
0	No device has read the CIF file since last CALL or power up.
1	A device has read the CIF file since last CALL or power up.

Example:

```
.
.
.
100 GOSUB 1040
.
.
.
1000 REM This routine waits for the CIF to be read.
1010 REM CALL 87 simply tells us that the CIF has been
1020 REM read some time in the past. This routine
1030 REM waits for the CIF to be read again.
1040 CALL 85: POP STATUS
1050 CALL 85: POP STATUS
1060 IF(STATUS=0) THEN GOTO 1050
1070 RETURN
.
.
.
```

CALL 90: Read Remote DH-485 Data File to MVI56-BAS Input Buffer

CALL 90 reads up to 40 words from a remote DH-485 node address data file to the MVI56-BAS input buffer.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
CALL 90
POP [S]
```

Where:

A = remote device node address (0 through 31)

B = remote device file number (0 through 255)

C = remote device file type (ASC(N), ASC(S), ASC(C), ASC(T), ASC(B) or ASC(R))

File Type Code	File Type	Words/Element
ASC(N)	Integer	1 word/element
ASC(S)	Status	1 word/element
ASC(C)	Counter	3 words/element
ASC(T)	Timer	3 words/element
ASC(B)	Bit	1 word/element
ASC(R)	Control	3 words/element

D = device file offset (0 through 32767)

E = number of elements to be transferred

File Type Code	Valid Length Range
ASC(N)	1 to 40
ASC(S)	1 to 40
ASC(C)	1 to 13
ASC(T)	1 to 13
ASC(B)	1 to 40
ASC(R)	1 to 13

F = message time-out (1 through 50)

This value is expressed as hundreds of milliseconds.

S = CALL status (0 = success)

Example:

```
.
.
.
100 GOSUB 1040
```

```
.
.
.
1000 REM This routine waits for the CIF to be read.
1010 REM CALL 87 simply tells us that the CIF has been
1020 REM read some time in the past. This routine
1030 REM waits for the CIF to be read again.
1040 CALL 85: POP STATUS
1050 CALL 85: POP STATUS
1060 IF(STATUS=0) THEN GOTO 1050
1070 RETURN
.
.
.
```

CALL 91: Write MVI56-BAS Output Buffer to Remote DH-485 Data File

CALL 91 writes up to 40 words from the MVI56-BAS output buffer to a remote DH-485 node address data file.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
CALL 90
POP [S]
```

Where:

A = remote device node address (0 through 31)

B = remote device file number (0 through 255)

C = remote device file type (ASC(N), ASC(S), ASC(C), ASC(T), ASC(B) or ASC(R))

File Type Code	File Type	Words/Element
ASC(N)	Integer	1 word/element
ASC(S)	Status	1 word/element
ASC(C)	Counter	3 words/element
ASC(T)	Timer	3 words/element
ASC(B)	Bit	1 word/element
ASC(R)	Control	3 words/element

D = device file offset (0 through 32767)

E = number of elements to be transferred

File Type Code	Valid Length Range
ASC(N)	1 to 40
ASC(S)	1 to 40
ASC(C)	1 to 13
ASC(T)	1 to 13
ASC(B)	1 to 40
ASC(R)	1 to 13

F = message time-out (1 through 50)

This value is expressed as hundreds of milliseconds.

S = CALL status (0 = success)

Example:

```
.
.
.
100 GOSUB 1040
```

```
.
.
.
1000 REM This routine uses a DH-485 write command to
1010 REM write BASIC Output Buffer CIF data to a remote
1020 REM DH-485 module data file.
1030 REM
1040 PUSH 1,7,ASC(N),0,10,5: CALL 91: POP STATUS
1050 IF(STATUS<>0) THEN PRINT "CALL 91 Error Code =",STATUS
1060 RETURN
.
.
.
```

CALL 92: Read Remote DH-485 Common Interface File to MVI56-BAS Input Buffer

CALL 92 reads up to 40 words from the remote DH-485 Common Interface File to the MVI56-BAS Input Buffer.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
CALL 92
POP [S]
```

Where:

A = remote device node address (0 through 31)

B = starting element offset of a remote device file (0 to 32767)

C = number of words to be transferred (1 to 40)

D = time-out as hundreds of milliseconds (1 to 50)

S = CALL status (0 = success)

CALL 93: Write From BAS Output Buffer to Remote DH-485 CIF

CALL 93 writes up to 40 words from the MVI56-BAS Output Buffer to the remote DH-485 Common Interface File.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
CALL 93
POP [S]
```

Where:

A = remote device node address (0 through 31)

B = starting element offset of a remote device file (0 to 32767)

C = number of words to be transferred (1 to 40)

D = time-out as hundreds of milliseconds (1 to 50)

S = CALL status (0 = success)

Example:

```
.
.
.
100 GOSUB 1040
.
.
.
1000 REM This routine uses a DH-485 write command to
1010 REM write BASIC Output Buffer CIF data to a remote
1020 REM DH-485 module CIF.
1030 REM
1040 PUSH 1,0,10,5: CALL 93: POP STATUS
1050 IF(STATUS<>0) THEN PRINT "CALL 93 Error Code =",STATUS
1060 RETURN
.
.
.
```

9.7 DF1 CALLs

CALL 108: Enable DF1 Driver to PRT2

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
CALL 108
```

Where:

A = operational code

The operational code parameter can assume one of the following values:

Code	Mode	Communication Parameters
0	Half-Duplex Slave	NHS, Disable DPD, BCC Error Checking
1	Half-Duplex Slave	NHS, Enable DPD, BCC Error Checking
2	Half-Duplex Slave	NHS, Disable DPD, CRC Error Checking
3	Half-Duplex Slave	NHS, Enable DPD, CRC Error Checking
4	Half-Duplex Slave	HDMwoCC, Disable DPD, BCC Error Checking
5	Half-Duplex Slave	HDMwoCC, Enable DPD, BCC Error Checking
6	Half-Duplex Slave	HDMwoCC, Disable DPD, CRC Error Checking
7	Half-Duplex Slave	HDMwoCC, Enable DPD, CRC Error Checking
8	Half-Duplex Slave	HDMwCC, Disable DPD, BCC Error Checking
9	Half-Duplex Slave	HDMwCC, Enable DPD, BCC Error Checking
10	Half-Duplex Slave	HDMwCC, Disable DPD, CRC Error Checking
11	Half-Duplex Slave	HDMwCC, Enable DPD, CRC Error Checking
16	Full-Duplex	NHS, ER, Disable DPD, BCC Error Checking
17	Full-Duplex	NHS, ER, Enable DPD, BCC Error Checking
18	Full-Duplex	NHS, ER, Disable DPD, CRC Error Checking
19	Full-Duplex	NHS, ER, Enable DPD, CRC Error Checking
20	Full-Duplex	NHS, ADER, Disable DPD, BCC Error Checking
21	Full-Duplex	NHS, ADER, Enable DPD, BCC Error Checking
22	Full-Duplex	NHS, ADER, Disable DPD, CRC Error Checking
23	Full-Duplex	NHS, ADER, Enable DPD, CRC Error Checking
24	Full-Duplex	FDM, ER, Disable DPD, BCC Error Checking
25	Full-Duplex	FDM, ER, Enable DPD, BCC Error Checking
26	Full-Duplex	FDM, ER, Disable DPD, CRC Error Checking
27	Full-Duplex	FDM, ER, Enable DPD, CRC Error Checking
28	Full-Duplex	FDM, ADER, Disable DPD, BCC Error Checking
29	Full-Duplex	FDM, ADER, Enable DPD, BCC Error Checking

Code	Mode	Communication Parameters
30	Full-Duplex	FDM, ADER, Disable DPD, CRC Error Checking
31	Full-Duplex	FDM, ADER, Enable DPD, CRC Error Checking

Where:

NHS	No Handshaking
HDMwoCC	Half-Duplex Modem without Continuous Carrier
HDMwCC	Half-Duplex Modem with Continuous Carrier
DPD	Duplicate Packet Detection
ER	Enable Embedded Responses
ADER	Auto-Detect Embedded Response

In order to suppress the end of transmission (EOT) packets for Half-Duplex Slave, add 32 to each Code (0 to 11) listed above. For example, to configure PRT2 as Half-Duplex Slave, NHS, Enable DPD, CRC error checking and without EOT use code 35 instead of 3.

B =	Poll Time out (Half-Duplex Slave) or Acknowledgement Time-Out (Full-Duplex): This parameter is expressed as 5 ms increments and can assume values in the range 2 to 65535.
C =	Message Retries (Half-Duplex Slave) or ENQuiry Retries (Full-Duplex): The value entered should be between 0 and 254.
D =	RTS On Delay Time period (Half-Duplex Slave) or Number of NAK Retries (Full-Duplex Mode): RTS On Delay is expressed as 5ms increments and values in the range 0 to 65535 are accepted. If using NAK Retries values between 0 to 254 are accepted.
E =	RTS Off Delay (only used for Half-Duplex Mode): If used for Full-Duplex the value will be ignored but still must be PUSHed. Values are accepted within 0 to 65499 range.
F =	DF1 address: This parameter specifies the BASIC module address that the DF1 driver responds to when receiving enquires from a remote DF1 device.

Example:

```
.
.
.
90 PUSH 5,200,2,4,4,10: CALL 108
.
.
.
```

CALL 113: Disable PRT2 DF1 Driver

Use CALL 113 to disable the DF1 driver previously enabled by CALL 108. This CALL clears PRT2 buffers.

Syntax:

```
CALL 113
```

Example:

```
.
.
.
90 CALL 113
.
.
.
```

CALL 114: Transmit DF1 Packet

Using CALL 114 (Transmit DF1 Packet) allows the user to transmit a DF1 data packet created by the BASIC program using PRINT#, PH0# or PH1# statements. If PRT2 is configured for full-duplex, the DF1 packet will be transmitted the next time PRT2 receives an ENQuiry from a DF1 master.

In order to check if the transfer was successful use CALL 115 (Check DF1 XMIT Status).

This CALL does not use any parameters:

Syntax:

```
CALL 114
```

Example:

```
.
.
.
90 CALL 114
.
.
.
```

CALL 115: Check DF1 XMIT Status

Use CALL 115 to check the DF1 transmit status. This CALL should be used after CALL 114 in order to check if the last DF1 packet was successfully sent and if a new data packet can be sent.

Syntax:

```
CALL 115
POP [A]
```

Where:

A = DF1 transmit status. The possible values are:

0 =	no transmit result pending
1 =	transmit result pending
2 =	transmission successful
3 =	transmission failed
4 =	enquiry time out
5 =	if modem handshaking is selected, either loss of CTS signal while transmitting or a fatal transmitter error has occurred.
6 =	if modem handshaking with constant carrier is selected, this error means transmission failure due to modem disconnection.
7 =	DF1 driver is not enabled

Example:

```
.
.
.
CALL 115
100 POP STATUS
.
.
.
```

CALL 117: Get DF1 Packet Length

Use CALL 117 to get the length of the data packet. After the length of the DF1 packet has been read, it should be used with the GET command in order to retrieve the data in the received packet.

Syntax:

```
CALL 117
POP [A]
```

Where:

A = length of the oldest DF1 packet in the DF1 receive buffer.

Example:

```
.
.
.
90 CALL 117
100 POP CHRCNT
.
.
.
```

9.8 BASIC Program Flow Control CALLs

CALL 0: Reset BASIC

There are times during BASIC program development when a reset seems like the only option. CALL 0 performs a soft BASIC reset.

Syntax:

```
CALL 0
```

Example:

```
Ready
>CALL 0
MVI56-BAS
Firmware Release: 1.02
Module S/N: 0000192E
Online Development Inc.
Copyright 1995-2002
!!!!!!!!!!! Warning !!!!!!!!!!!!
DH-485 will not function properly
if the setup jumper is installed.
!!!!!!!!!!! Warning !!!!!!!!!!!!
Ready
>
```

CALL 16: Enable DF1 Packet Interrupt

Use CALL 16 to enable DF1 packet interrupt capability. This means that if a DF1 packet arrives at PRT2 due to CALL 122 or CALL 123, the BASIC control will jump to another BASIC program line. Use RETI command to exit the interrupt.

Syntax:

```
PUSH [A]
CALL 16
```

Where:

A = BASIC line number

Example:

```
10 REM Enable DF1 Packet Interrupt
20 PUSH 800: CALL 16
.
.
.
800 REM DF1 Packet Interrupt Routine
.
.
.
850 RETI
.
.
.
```

CALL 17: Disable DF1 Packet Interrupt

Use CALL 17 to disable the DF1 packet interrupt capability previously enabled by CALL 16.

Syntax:

```
CALL 17
```

Example:

```
.
.
.
800 CALL 17: REM Disable DF1 Packet Interrupt
.
.
.
```

CALL 20: Enable Processor Interrupt

Use CALL 20 to allow the ControlLogix to interrupt the current BASIC program. When word 0 bit 15 in the output image (Local:x:O:Data[0].15) toggles from OFF to ON, the program jumps to the line number specified by the CALL 20 parameter.

```
Use the RETI command in order to return to the point the program was before
being interrupted.
```

Syntax:

```
PUSH [A]
CALL 20
```

Where:

A = BASIC line number

Example:

```
10 PUSH 50: REM JUMPS TO LINE NUMBER 50
20 CALL 20
30 GOTO 30
35 PRINT "LINE 35"
40 PRINT "LINE 40"
50 PRINT "LINE 50"
60 PRINT "LINE 60"
70 RETI
80 END
```

CALL 21: Disable Processor Interrupt

Use CALL 21 to disable the processor interrupt after using CALL 20.

Syntax:

```
CALL 21
```

Example:

```
.
.
.
210 CALL 21
.
.
.
```

CALL 29: Read/Write To/From Internal String (DF1 or DH-485)

Use CALL 29 with CALL 27, CALL 28 (DH-485) or CALL 122, CALL 123 (DF1) to use an internal string as a source (or destination) data. The advantage is that using CALL 29 does not require ladder to transfer data if one of these CALLs was executed before CALL 29.

Syntax:

```
PUSH [A]
CALL 29
POP [B]
```

Where:

A = BASIC CALL to be activated. Valid values are 27, 28 (DH-485) or 122 or 123 (DF1).

B = CALL 29 status code. It returns one of the following values:

0 = successful

1 = selected CALL is disabled (27, 28, 122 or 123)

255 = the internal string is not selected in CALL (27, 28, 122 or 123)

Example:

```
10 REM Execute DF1 PLC remote read from internal
20 REM string with no CLX intervention
30 PUSH 5,3,10,ASC(N),0,10,10,1,1,1: CALL 122: POP STATUS
40 PUSH 122: CALL 29: POP S
50 IF(S=1) THEN PRINT "CALL 122 Not Active!"
60 IF(S=255) THEN PRINT "CLX File Chosen for CALL 122"
70 IF(S=0) THEN PRINT "Successful Transfer"
80 IF (S<>0) THEN PRINT "Unsuccessful Transfer!"
90 END
```

CALL 70: ROM to RAM Program Transfer

Use CALL 70 to move the program execution from a running ROM program to the beginning of the RAM program.

Syntax:

```
CALL 70
```

Example:

```
10 REM This program is stored in ROM 1
20 PRINT "Running ROM 1"
30 CALL 70
40 PRINT "Back in ROM 1"
50 END
10 REM This program is stored in RAM
20 PRINT "Running in RAM"
30 END
Ready
>rom 1
Ready
>run
Running ROM 1
Running in RAM
Ready
>
```

CALL 71: ROM/RAM to ROM Program Transfer

Use CALL 71 to move the program execution from a running ROM or RAM program to the beginning of a ROM program.

Syntax:

```
PUSH [ROM program number]
CALL 71
```

Example:

```
10 REM This program is stored in ROM 1
20 PRINT "ROM 1 Program!"
30 CALL 72
40 END
10 REM This program is stored in ROM 2
20 PRINT "ROM 2 Program!"
30 CALL 72
40 END
10 REM This program CALLs ROM 1 and ROM 2
20 PRINT "Running in RAM!"
30 PUSH 1: CALL 71: REM Call ROM 1
40 PUSH 2: CALL 71: REM Call ROM 2
50 PRINT "Back in RAM!"
60 END
Ready
>ram
Ready
>run
Running in RAM!
ROM 1 Program!
ROM 2 Program!
Back in RAM!
Ready
>
```

CALL 72: ROM/RAM Return

Use CALL 72 to return to the ROM or RAM program that called the running program. CALL 72 should be used after CALL 70 or 71 in order to return to the calling program.

Syntax:

```
CALL 72
```

Example:

```
10 REM This program is stored in ROM 1
20 PRINT "ROM 1 Program!"
30 CALL 72
40 END
10 REM This program is stored in ROM 2
20 PRINT "ROM 2 Program!"
30 CALL 72
40 END
10 REM This program CALLs ROM 1 and ROM 2
20 PRINT "Running in RAM!"
30 PUSH 1: CALL 71: REM Call ROM 1
40 PUSH 2: CALL 71: REM Call ROM 2
50 PRINT "Back in RAM!"
60 END
Ready
>ram
Ready
>run
Running in RAM!
ROM 1 Program!
ROM 2 Program!
Back in RAM!
Ready
>
```

9.9 Background CALLs

9.9.1 ASCII Background CALLs

CALL 22: Transfer Data from a Serial Port to CLX

Use CALL 22 to transfer data from PRT1 or PRT2 serial ports to the CLX processor. After this CALL is executed, the selected serial port will continue receiving data until the maximum number of characters is reached or the character delimiter is received. At this point all data is transferred to the CLX processor.

For more information about CALL 22 usage, refer to ASCII Data Transfer from MVI56-BAS Serial Port to CLX (page 75)

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
PUSH [G]
CALL 53
POP [H]
```

Where:

A = serial port number (0 = CALL 22 disabled, 1 = PRT1, 2 = PRT2)

B = maximum number of characters

The valid range depends on which destination file is selected:

1 to 60 for CLX input image file

1 to 198 for MSG instruction

1 to 251 for internal string

C = decimal value of character delimiter

D = destination file

0 = CLX input image file

1 = MSG instruction

2 = CLX input image file and internal string

3 = MSG instruction and internal string

4 = Internal string

E = word offset within the destination file

If using the CLX input image as the destination file the word offset should be 2 Minimum, because words 0 and 1 are reserved.

This parameter has no effect when internal string is selected (it will always start on the third character)

F = string number (if destination file is a string)

G = byte swap (0 do not swap bytes , 1 swap bytes)

H = CALL status
0 = successful setup
1 = disabled
2 = invalid parameter
3 = PRT2 already enabled for DF1 protocol
4 = string too small
5 = string not dimensioned

Example:

```
10 PUSH 1: REM PRT1 active for CALL 22
20 PUSH 10: REM Receiving 10 bytes of data maximum
30 PUSH 13: REM <CR> used as termination character
40 PUSH 1: REM send data to MSG instruction
50 PUSH 0: REM Offset into MSG instruction data
60 PUSH 0: REM String number (not used)
70 PUSH 1: REM Byte swapping enabled
80 CALL 22
90 POP S: REM Status of CALL 22 setup
100 IF(S<>0) THEN PRINT "Unsuccessful CALL 22 setup"
110 IF(S<>0) THEN END
120 GOTO 120
or
10 PUSH 1,10,13,1,0,0,1: CALL 22: POP S
20 IF(S<>0) THEN PRINT "Unsuccessful CALL 22 setup"
30 IF(S<>0) THEN END
40 GOTO 40
```

CALL 23: Transfer Data from CLX to a Serial Port

Use CALL 23 to set up the data transfer configuration from the CLX directly to a MVI56-BAS serial port (PRT1 or PRT2). The low byte of the first word of the source data file determines the byte count to be transferred. The high byte of the first word is not used.

For more information about CALL 23 usage (including ladder logic) refer to the section "**Data Transfer from CLX to MVI56-BAS Serial Port**"

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
CALL 23
POP [F]
```

Where:

A = destination serial port

0 = disable CALL 23

1 = internal string

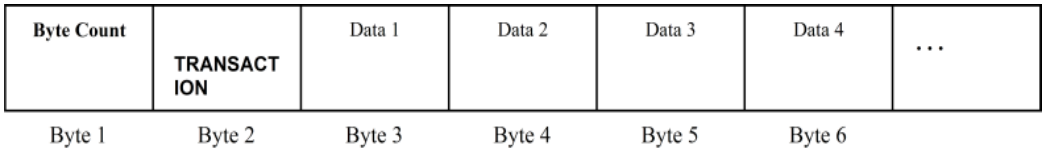
2 = serial PRT1

3 = serial PRT1 and internal string

4 = serial PRT2

5 = serial PRT2 and internal string

Note: A string is structured in the following way:



B = source file

0 = CLX output image file

1 = CLX MSG instruction

The first word of the source file must contain the byte count.

C = word offset within the source file

If the CLX output image file is chosen as the source file it must have the offset as 1 or more, because word 0 is reserved for handshaking.

D = internal string number (if the destination file is not a string this value is ignored)

E = byte swap (0 = do not swap bytes , 1 = swap bytes)

F = CALL 23 status

0 = successful

1 = disabled

2 = invalid parameter

3 = PRT2 already enabled for DF1

4 = string too small

5 = string not dimensioned

Example:

```
10 REM Enable CALL 23
20 PUSH 2,1,0,0,1: CALL 23: POP S
30 IF(S<>0) THEN PRINT "CALL 23 Unsuccessful"
40 GOTO 40
```

9.9.2 DH-485 Background CALLs

CALL 118: Receive DF1 or DH-485 Unsolicited Write

Use CALL 118 to allow the MVI56-BAS to receive write command from remote DF1 devices such as PLC-2, PLC-3, PLC-5 or from the DH-485 network. The following WRITE commands are accepted:

- PLC (word range writes)
- PLC (typed writes)
- PLC (unprotected writes)
- SLC 5/02 (unprotected writes)
- SLC 5/02 (typed writes)

For more information about CALL 118 usage (including ladder logic) refer to Using DF1 Protocol Communications.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
CALL 118
POP [F]
```

Where:

A = CALL enable / disable (0 = enable CALL 118 , 1 = disable CALL 118)

B = selection of destination file. The following values are valid:

- 0 = CLX input image file
- 1 = CLX MSG instruction
- 2 = internal string
- 3 = CLX input image file and internal string
- 4 = CLX MSG instruction and internal string

If using an internal string the BASIC program must monitor the second character which contains the transaction number which is incremented at every successful transaction (from 0 to 255)

C = word offset in destination file. The offset for an internal string is always 2, because each string has the following structure:

If using the input image file as the destination file the offset should be a minimum of 2, because the first 2 words are reserved.

D = string number. This parameter is only used if the destination file selected is an internal string

E = maximum word length. Any packets received which contains more than this parameter is rejected.

F = CALL 118 status. It can assume one of the following values:

- 0 = successful
- 1 = disabled
- 2 = invalid parameter
- 3 = DF1 port is not enabled
- 4 = selected DF1 port is not enabled
- 5 = string is not dimensioned

Example:

```
.
.
.
90 PUSH 1,1,0,0,20: CALL 118: POP STATUS
100 IF(STATUS<>0) THEN PRINT "Unsuccessful CALL 118 Setup"
110 REM CALL 118 remains active as long as BASIC
120 REM continues to RUN.
.
.
.
```

9.9.3 DF1 Background CALLs

CALL 118: Receive DF1 or DH-485 Unsolicited Write

Use CALL 118 to allow the MVI56-BAS to receive write command from remote DF1 devices such as PLC-2, PLC-3, PLC-5 or from the DH-485 network. The following WRITE commands are accepted:

PLC (word range writes)

PLC (typed writes)

PLC (unprotected writes)

SLC 5/02 (unprotected writes)

SLC 5/02 (typed writes)

For more information about CALL 118 usage (including ladder logic) refer to Using DF1 Protocol Communications.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
CALL 118
POP [F]
```

Where:

A = CALL enable / disable (0 = enable CALL 118 , 1 = disable CALL 118)

B = selection of destination file. The following values are valid:

- 0 = CLX input image file
- 1 = CLX MSG instruction
- 2 = internal string
- 3 = CLX input image file and internal string
- 4 = CLX MSG instruction and internal string

If using an internal string the BASIC program must monitor the second character which contains the transaction number which is incremented at every successful transaction (from 0 to 255)

C = word offset in destination file. The offset for an internal string is always 2, because each string has the following structure:

Byte Count	TRANSACTION	Data 1	Data 2	Data 3	Data 4	...
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	

If using the input image file as the destination file the offset should be 2 minimum, because the first 2 words are reserved.

D = string number . This parameter is only used if the destination file selected is an internal string

E = maximum word length . Any packet received which contains more than this parameter is rejected.

F = CALL 118 status. It can assume one of the following values:

- 0 = successful
- 1 = disabled
- 2 = invalid parameter
- 3 = DF1 port is not enabled
- 4 = selected DF1 port is not enabled
- 5 = string is not dimensioned

Example:

```
.
.
.
90 PUSH 1,1,0,0,20: CALL 118: POP STATUS
100 IF(STATUS<>0) THEN PRINT "Unsuccessful CALL 118 Setup"
110 REM CALL 118 remains active as long as BASIC
120 REM continues to RUN.
.
.
.
```

CALL 122: Read Remote DF1 PLC Data File

Use CALL 122 to read up to 99 words of data from a remote DF1 node to the CLX or a MVI56-BAS internal string. For more information about CALL 122 usage (including ladder logic) refer to Using DF1 Protocol Communications.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
PUSH [G]
PUSH [H]
PUSH [I]
PUSH [J]
CALL 122
POP [K]
```

Where:

A = type of PLC READ command, can have one of the following values:

- 0 = disable the previously executed CALL 122
- 2 = PLC-2 unprotected READ command (common interface file)
- 3 = PLC-3 file: word range READ command
- 5 = PLC-5 file: typed READ command

B = remote DF1 node address

C = file number to be read (between 0 and 255). If sending a PLC-2 unprotected READ command to a SLC DF1 channel use 9 as the file number.

D = file type. It should have one of the following values:

File Type Code	Description	Words/Elements	Valid Range
ASC(N)	Integer File	1 word/element	1 to 99
ASC(S)	Status File	1 word/element	1 to 99
ASC(C)	Counter File	3 words/element	1 to 33
ASC(T)	Timer File	3 words/element	1 to 33
ASC(B)	Bit File	1 word/element	1 to 99
ASC(R)	Control FIle	3 words/element	1 to 33
ASC(I)	Input File	1 word/element	1 to 99
ASC(O)	Output File	1 word/element	1 to 99

E = remote file offset

F = number of elements to be transferred. Refer to the table above for the valid ranges.

G = message time-out. This value is expressed as 0.1s and valid range is between 1 to 255

H = destination file. The valid values are listed below:

- 0 = CLX input image file
- 1 = CLX MSG instruction
- 2 = internal string
- 3 = CLX input image file and internal string
- 4 = CLX MSG instruction and internal string

If number 2 is selected (internal string) the best technique is to use CALL 29 after CALL 122 in order to transfer the string. Using CALL 29 does not require ladder logic to transfer data.

I = word offset within destination file. If the CLX input image file is the selected destination file, the offset cannot be 0 or 1 because these are reserved words.

J = string number. If the destination file is not an internal string this value is ignored.

K = CALL 122 status code:

- 0 = successful setup
- 1 = disabled
- 2 = bad input parameter
- 3 = DF1 not enabled
- 4 = string too small
- 5 = string not dimensioned

CALL 122 copies the transfer status code to the CLX input image file 1. A value 0 indicates that the transaction was successful.

Example:

```
.
.
.
50 PUSH 5,200,2,4,4,10: CALL 108
.
.
.
90 PUSH 5,0,7,ASC(N),0,20,10,1,0,0: CALL 122: POP STATUS
100 IF(STATUS<>0) THEN PRINT "Unsuccessful CALL 122 Setup"
110 REM CALL 122 remains active as long as BASIC
120 REM continues to RUN.
.
.
.
```

CALL 123: Write Remote DF1 PLC Data File

Use CALL 123 to write up to 99 words of data from the CLX to a remote DF1 node. For more information about CALL 123 usage (including ladder logic) refer to Using DF1 Protocol Communications.

Syntax:

```
PUSH [A]
PUSH [B]
PUSH [C]
PUSH [D]
PUSH [E]
PUSH [F]
PUSH [G]
PUSH [H]
PUSH [I]
PUSH [J]
CALL 122
POP [K]
```

Where:

A = type of PLC WRITE command, can have one of the following values:

- 0 = disable the previously executed CALL 123
- 2 = PLC-2 unprotected WRITE command (common interface file)
- 3 = PLC-3 file: word range WRITE command
- 5 = PLC-5 file: typed WRITE command

B = remote DF1 node address

C = file number to be read (between 0 and 255) . If sending a PLC-2 unprotected READ command to a SLC DF1 channel use 9 as the file number.

D = file type. It should have one of the following values:

File Type Code	Description	Words/Elements	Valid Range
ASC(N)	Integer File	1 word/element	1 to 99
ASC(S)	Status File	1 word/element	1 to 99
ASC(C)	Counter File	3 words/element	1 to 33
ASC(T)	Timer File	3 words/element	1 to 33
ASC(B)	Bit File	1 word/element	1 to 99
ASC(R)	Control FIle	3 words/element	1 to 33
ASC(I)	Input File	1 word/element	1 to 99
ASC(O)	Output File	1 word/element	1 to 99

E = remote file offset

F = number of elements to be transferred. Refer to the table above for the valid ranges.

G = message time-out. This value is expressed as 0.1s and valid range is between 1 to 255

H = source file. The valid values are listed below:

- 0 = CLX input image file
- 1 = CLX MSG instruction
- 2 = internal string

If number 2 is selected (internal string) the best technique is to use CALL 29 after CALL 123 in order to transfer the string. Using CALL 29 does not require ladder logic to transfer data.

I = word offset within the source file. If the CLX input image file is the selected as the source file, the offset cannot be 0 or 1 because these are reserved words.

J = string number. If the destination file is not an internal string this value is ignored.

K = CALL 123 status code:

- 0 = successful setup
- 1 = disabled
- 2 = bad input parameter
- 3 = DF1 not enabled
- 4 = string too small
- 5 = string not dimensioned

CALL 123 copies the transfer status code to the CLX input image file 1. A value 0 indicates that the transaction was successful.

Example:

```
.
.
.
50 PUSH 5,200,2,4,4,10: CALL 108
.
.
.
90 PUSH 5,0,7,ASC(N),0,20,10,1,0,0: CALL 123: POP STATUS
100 IF(STATUS<>0) THEN PRINT "Unsuccessful CALL 123 Setup"
110 REM CALL 123 remains active as long as BASIC
120 REM continues to RUN.
.
.
.
```

9.10 Miscellaneous CALLs

CALL 18: Enable ^C Check

Use CALL 18 to re-enable ^C checking after CALL 19 has been used to disable ^C checking.

Syntax:

CALL 18

Example:

```
.
.
.
100 CALL 18
.
.
.
```

CALL 19: Disable ^C Check

There are times when PRT1 must be used for data collection or some protocol and the BASIC program should not be stopped when a ^C (03H) is received. CALL 19 disables ^C checking.

If ^C checking has been disabled, and the program must be stopped then the only way to stop the program is to power the unit off, set the setup jumper ON, and power the unit back on. So, CALL 19 should be used with caution.

Syntax:

CALL 19

Example:

```
.
.
.
100 CALL 19
.
.
.
```

CALL 80: Check Battery Condition

Use CALL 80 to check the battery condition.

Syntax:

```
CALL 72
POP A
```

Where;

A =	0 → Battery is okay
	1 → Battery is low

Example:

```
.
.
.
120 CALL 80: POP ST
130 IF(ST=0) THEN PRINT "Battery okay." ELSE PRINT "Battery Low"
.
.
.
```

CALL 81: EPROM Check

Use CALL 81 to see if the program in XRAM will fit into EPROM.

Syntax:

```
CALL 81
```

Example:

```
Ready
>call 81
Number of BASIC programs in EPROM..........1
Available bytes to end of user EPROM.......31906
Length of BASIC program in RAM.............24
Program will fit in EPROM.
Ready
>
```

CALL 109: Print Argument Stack

There are times when developing and debugging a BASIC program that it would be nice to see exactly what has been pushed onto the argument stack. Use CALL 109 for this purpose.

Syntax:

```
CALL 109
```

Example:

```
Ready
>push 1,2,3,4
Ready
>call 109
000 1
001 2
002 3
003 4
Argument stack pointer is: 4
Ready
>
```

This section provides information on diagnostics and troubleshooting in the following forms:

- Status CALLs
- LED status indicators on the front of the module provide information on the modules status.

9.11 LED Status Indicators

The LEDs indicate the module's operating status as follows:

LED	Color	Status	Indication
CFG (BASIC DH-485)	Green	On	Data is being transferred between the module and a remote terminal using the Configuration/Debug port. **Note:** This LED is normally illuminated solid green whenever a user program is running, whether the module is performing DH-485 communication.
		Off	No data is being transferred on the Configuration/Debug port.
P1 (BASIC PRT1)	Green	On	Data is being transferred between the second port (BASIC PRT1) and a remote terminal or an ASCII device.
		Off	No data is being transferred
P2 (BASIC PRT2)	Green	On	Data is being transferred between the third port (BASIC PRT2) and a remote terminal, an ASCII device or a DF1 device
		Off	No data is being transferred
OK	Red/ Green	Off	The card is not receiving any power and is not securely plugged into the rack.
		Green	The module is operating normally.
		Red	The program has detected an error or is being configured. If the LED remains red for over 10 seconds, the program has probably halted. Remove the card from the rack and re-insert the card to restart the module's program.
BAT	Red	Off	The battery voltage is OK and functioning.
		On	The battery voltage is low or battery is not present. Allow battery to charge by keeping module plugged into rack for 24 hours. If BAT LED still does not go off, contact ProSoft Technology, as this is not a user serviceable item.

Note: The user can set in the BASIC program the LEDS APP STATUS and BP ACT using CALL 112 setting any meaning for these LEDS:

CALL 112 Syntax:

```
PUSH [APP STATUS LED]
PUSH [BP ACT LED]
CALL 112
```

Where both arguments turn ON the respective LED when equal to 1 and turn OFF when equal to 0.

9.11.1 Clearing a Fault Condition

Typically, if the OK LED on the front of the module turns RED for more than ten seconds, a hardware problem has been detected in the module or the program has exited.

To clear the condition, follow these steps:

1. Turn off power to the rack.
2. Remove the card from the rack.
3. Verify that all jumpers are set correctly.
4. If the module requires a Compact Flash card, verify that the card is installed correctly.
5. Re-insert the card in the rack and turn the power back on.
6. Verify correct configuration data is being transferred to the module from the ControlLogix controller.

If the module's OK LED does not turn GREEN, verify that the module is inserted completely into the rack. If this does not cure the problem, contact ProSoft Technology Technical Support.

9.11.2 Troubleshooting

Use the following troubleshooting steps if you encounter problems when the module is powered up. If these steps do not resolve your problem, please contact ProSoft Technology Technical Support.

Processor Errors

Problem Description	Steps to take
Processor Fault	Verify that the module is plugged into the slot that has been configured for the module. Verify that the slot in the rack configuration has been set up correctly in the ladder logic.
Processor I/O LED flashes	This indicates a problem with backplane communications. Verify that all modules in the rack are configured in the ladder logic.

Module Errors

Problem Description	Steps to take
OK LED remains red	The program has halted or a critical error has occurred. Connect to the Configuration/Debug port to see if the module is running. If the program has halted, remove the card from the rack, then re-insert.

10 BASIC-52 Implementation

In This Chapter

This section describes BASIC-52 implementation items.

10.1 Operators and Statements

Operator / Statement	Description	Example
+	Add	PRINT 1+2
/	Divide	PRINT 4/2
**	Exponentiation	PRINT 2**3
*	Multiply	PRINT 2*3
-	Subtract	PRINT 5-3
-	Negation	PRINT -(5*4)
AND	Bit AND Operator	PRINT 2.AND.3
OR	Bit OR Operator	PRINT 2.OR.3
XOR	Bit XOR Operator	PRINT 2.XOR.3
=	Equal comparison	IF X=3 THEN GOTO 60
>	Greater than	IF X<3 THEN GOTO 60
<	Less than	IF X>3 THEN GOTO 60
<>	Different from	IF X<>3 THEN GOTO 60
SIN()	Returns the sine of argument	PRINT SIN(0)
COS()	Returns the cosine of argument	PRINT COS(0)
TAN()	Returns the tangent of argument	PRINT TAN(0)
ATN()	Returns the arctangent of argument	PRINT ATAN(1)
ABS()	Returns the absolute value of argument	PRINT ABS(-5)
NOT()	Returns 16 bit complement of argument	PRINT NOT(3.OR.2)
INT()	Returns the integer portion of argument	PRINT INT(3.14)
PI	Returns the PI number	PRINT PI
SGN()	Returns 1 if arg >0, -1 if arg<0 and 0 if arg=0	PRINT SGN(-5)
SQR()	Returns the square root of argument	PRINT SQR(9)
RND	Generates a random number between 0 and 1	PRINT RND
LOG()	Returns the logarithm	PRINT LOG(10)
EXP()	Calculates e^{arg}	PRINT EXP(1)
ASC()	Returns the integer value of the ASCII	PRINT ASC(A)
CHR()	Converts the expression to an ASCII character	PRINT CHR(32)
CR		
LF		
GET	Reads the console input device (use also GET# or GET@ to read from the ports)	A = GET
LEN	Returns the number of bytes the current program occupies	PRINT LEN

Operator / Statement	Description	Example
FREE	Returns the number of RAM bytes available to the user	PRINT FREE
TAB()		
#	It directs the operator to PRT2	PRINT# A GET# A INPUT# "Weight?",W
@	It directs the operator to PRT1	PRINT# B GET# B INPUT@ "Length?",L
BRKPNT		
CALL	Allows the user to reference functions that are not supported directly in BASIC.	CALL 37
CLEAR	Resets all variables, interrupts and stacks	CLEAR
CLEARI	Resets all interrupts	CLEARI
CLEARS	Resets all stacks	CLEARS
CLOCK0	Disables the internal BASIC real time clock. CLOCK0 is mainly used in conjunction with the TIME operator and the ONTIME statement.	CLOCK0
CLOCK1	Enables the internal BASIC real time clock. CLOCK1 is mainly used in conjunction with the TIME operator and the ONTIME statement.	CLOCK1
CLRERR	Clears the previous error stored after a ONERR statement	CLRERR
CONT	Resumes a program execution after a BRKPNT [Ctrl-C], or a STOP statement	CONT
DATA	Stores information that will not change for the READ statement.	DATA 5,6,13,24,-17
DIM	Reserves memory for matrices	DIM A(25)
DO/UNTIL	Set up a loop control within a module program. The loop is executed until the condition is TRUE	10 DO 20 A=A+1 30 UNTIL A=4
DO/WHILE	Set up a loop control within a module program. The loop is executed while the condition is TRUE	10 DO 20 A=A+1 30 WHILE A<4
EDIT	Edits the program line number currently in RAM	EDIT 10
END	Terminates program execution	END
EOF	Checks if a PRT input buffer is empty	IF(EOF) THEN …
ERASE	Erase the current program stored in ROM mode	ERASE

Operator / Statement	Description	Example
ERRCODE	Returns the last error code since the beginning of the program or the last CLRERR call	10 ONERR 5000 … 5000 E = ERRCODE
ERRLINE	Returns the line number where the last error occurred since the beginning of the program or the last CLRERR call	10 ONERR 5000 … 5000 L = ERRLINE
EXIT	EXIT BASIC to the DOS command prompt.	EXIT
EXPORT	Saves the current RAM program to the Compact Flash	EXPORT "test.bas"
FOR/TO/STEP	Control program loops	E=0:B=10: G=1 FOR A=E to C STEP G
GOTO	GOTO forces the next BASIC line to be executed to change to the line specified.	GOTO 500
GOSUB	GOSUB forces the next BASIC line to be executed to change to the line specified. The program returns to the most recently executed GOSUB statement once it finds a RETURN statement	GOSUB 100
IDLE	Halts the program execution until an ONTIME condition is met	10 TIME = 0 20 CLOCK1 30 ONTIME 2, 70 40 IDLE 50 PRINT "END TEST" 60 END 70 PRINT "TIMER INTERRUPT AT -" 80 RETI
IF/THEN/ELSE	Set up a conditional test	10 IF A<>8 GOTO 20 ELSE GOTO 30 20 PRINT "NOT 8" 30 PRINT "IS 8"
INPL	Reads an entire line from the program port buffer	INPL $(0) INPL# $(1) INPL@ $(0), 1
INPS	Reads an entire string of characters	INPS $(0),1
INPUT	Enter data from the console device during program execution	INPUT A
LD_AT	Retrieves a floating-point value that was stored using a ST_AT()	10 LD_AT(A) 20 POP X 30 PRINT X
LET	Assign a value to a variable. The LET statement is optional in BASIC.	LET A = 1 A = 25 * A
LIST		

Operator / Statement	Description	Example
LOAD	LOAD loads a program into RAM from a DOS ASCII file.	LOAD "PROGAB.TXT"
MODE	Sets the port parameters of ports PRT1, PRT2, and DH-485	MODE(PRT1,19200,N,8
NEXT	NEXT returns the FOR-TO-(STEP)-NEXT loop to the beginning of the loop and add the value of the increment	10 FOR A=0 TO 10 20 PRINT A 30 NEXT A 40 END
NEW	Deletes the program currently stored in RAM memory, sets all variables to zero, and clears all strings and BASIC interrupts	NEW
NULL		
ONDF1		
ONERR	Handles arithmetic errors during program execution. The program execution jumps to the specified line number	ONERR 500
ON/GOTO	Allows the program to select which line to jump to next.	ON A GOTO 100,200,500
ON/GOSUB	Allows the program to select which subroutine to execute next.	ON B GOSUB 100,300,900
ONTIME	Compensates for the incompatibility between timer/counters on the microprocessor and the MVI56-BAS module	ONTIME 2, 100
PH0.	Print data in hexadecimal.	PH0. A,Z
PH1.	Print data in hexadecimal. Four hex digits will always be printed.	PH1. A,B
POP	POP arguments off the argument stack.	POP A,B,Z
PRERR	Prints the last error description and error line number to the program port.	PRERR
PRINT	PRINT transmits data out the serial port. The # and @ operators can be used with the PRINT.	PRINT A,CHR(N) PRINT@ Z PRINT# LB
PROG	Writes the current program in RAM memory to the Compact Flash (ROM)	PROG
PROG0	Returns the module to normal operation after a PROG2 command	PROG0
PROG1	Loads the port configuration during power up	PROG1
PROG2	After a PROG2 command, the module will load the first program at ROM (using the PROG command) at power up	PROG2

Operator / Statement	Description	Example
PUSH	PUSH arguments onto the argument stack.	PUSH A,B,ASC(N),SIN(2)
RAM	Selects the program in RAM memory. It can be used after a ROM command	RAM
READ		
REM	Specifies a comment line in a BASIC program	REM
REN	Renumbers program lines REN (new line, old line, increment)	REN 1000, 800, 100
RESTORE	Resets the internal DATA READ pointer	RESTORE
RETI	Exits from an interrupt	RETI
RETURN	Returns control to the statement following the most recently executed GOSUB statement	10 GOSUB 100 20 PRINT "LINE 20" 25 GOTO 200 100 PRINT "LINE 100" 110 RETURN 200 END
ROM	Selects a program from ROM memory (Compact Flash), based on a given program location number	ROM 3
RROM	Selects and runs a program from ROM memory (Compact Flash), based on a given program location number	RROM 3
RUN	Runs the currently selected program	RUN
SNGLSTP	Initiates a single step program execution. Use this statement before a RUN command	SNGLSTP
SPC()		
ST_AT	Writes a floating-point number to memory	PUSH A ST_AT (B)
STOP	Breaks the program execution at specific points. The CONT command will resume program execution	10 FOR I = 1 TO 20 20 PRINT I 30 STOP 40 NEXT I
STRING	STRING allocates memory for BASIC strings.	STRING 100,10
TIME	Retrieve or assign a value to the internal free running clock	PRINT TIME TIME = 0
USING		
VER	Prints the current version of the firmware	VER
XBY()	Accesses error information.	See Debugging a BASIC Program

Operator / Statement	Description	Example
XBYTE()	Stores and retrieves user values in battery-backed SRAM. Valid argument range (variable locations) is **0** to **4095** (**0FFF**h).	10 FOR I = 1 TO 10 20 XBYTE(I) = I 30 PRINT XBYTE(I) 40 NEXT I
XFER	Transfers the currently selected program from ROM mode to RAM and selects the RAM mode.	XFER

10.2 Control Expressions

10.2.1 IF-THEN-ELSE

Sets up a conditional test.

Syntax:

IF condition THEN action1 ELSE action2

Example:

```
IF A>0 THEN B=0 ELSE B=1
```

10.2.2 DO-UNTIL

Sets up a loop control until the condition specified in the UNTIL operator is not satisfied.

Syntax:

DO action UNTIL condition

Example:

```
10 A=1
20 DO
30 PRINT A
40 A=A+2
50 UNTIL A>20
```

10.2.3 DO-WHILE

Sets up a loop control while the condition specified in the WHILE operator is satisfied.

Syntax:

DO action WHILE condition

Example:

```
10 A=1
20 DO
30 PRINT A
40 A=A+2
50 WHILE A<20
```

10.2.4 FOR-TO- (STEP)-NEXT

Sets up a control loop.

Syntax:

FOR [expr1] TO [expr2] STEP [expr3]

..

..

..

NEXT [expr5]

Example:

```
10 FOR A=0 TO 10 STEP 2
20 PRINT A
30 NEXT A
40 PRINT "END OF LOOP"
```

Result:

```
0
2
4
6
8
10
END OF LOOP
```

10.2.5 GOTO

Use the GOTO statement to force the BASIC program to jump to a specific line number.

Syntax:

GOTO [line number]

Example:

```
10 PRINT "ENTER A NUMBER"
20 INPUT A
30 IF A>0 THEN GOTO 40 ELSE GOTO 60
40 PRINT "NUMBER IS POSITIVE"
50 GOTO 70
60 PRINT "NUMBER IS NEGATIVE"
70 END
```

10.2.6 END

Use END to terminate the program execution.

Syntax:

```
END
```

10.2.7 GOSUB

Use the GOSUB statement to transfer the module control to a specific subroutine. This statement allows the program to be divided in different areas.

Syntax:

```
GOSUB [line number]
10 PRINT "ENTER A NUMBER"
20 INPUT A
30 IF A=1 GOSUB 50 ELSE GOSUB 100
50 PRINT "THIS IS ONE ROUTINE"
60 RETURN
100 PRINT "THIS IS ANOTHER ROUTINE"
110 RETURN
```

11 Reference

In This Chapter

11.1 Product Specifications

This single-slot module is a powerful solution that allows the large installed base of legacy BASIC programs for the MVI56-DB and 1747-BAS modules to be migrated to the ControlLogix platform with minimal effort.

The MVI56-BAS module allows you to

- Create BASIC programs to transfer data between the ControlLogix and the MVI56-BAS module
- Create foreign device interface for specific applications and easily integrate the device with the ControlLogix processor
- Interface to many devices that use ASCII interface to a ControlLogix backplane such as modems, printers, bar code readers, and scales
- Save ControlLogix processor memory by running math algorithms
- The module acts as DF1 full-duplex device or a DF1 half-duplex slave

11.1.1 General Specifications

- Single Slot - 1756 backplane-compatible
- The module is recognized as an Input/Output module and has access to processor memory for data transfer between processor and module.
- Ladder Logic is used for data transfer between module and processor. Sample ladder file included.
- Configuration data obtained from configuration text file downloaded to module. Sample configuration file included
- Local or remote rack

11.1.2 Hardware Specifications

Specification	Description
Backplane Current Load	800 mA @ 5 Vdc 3 mA @ 24 Vdc
Operating Temperature	0°C to 60°C (32°F to 140°F)
Storage Temperature	-40°C to 85°C (-40°F to 185°F)
Shock	30 g operational 50 g non-operational Vibration: 5 g from 10 Hz to 150 Hz
Relative Humidity	5% to 95% (without condensation)
LED Indicators	Module Status Backplane Transfer Status Application Status Serial Activity
Debug/Configuration port (CFG)	
CFG Port (CFG)	RJ45 (DB-9M with supplied cable) RS-232 only
Application ports (PRT1 & PRT2) Full hardware handshaking control, providing radio, modem and multi-drop support	
Software configurable communication parameters	Baud rate: 110 to 19200, depending on protocol RS-232, 485 and 422 Parity: none, odd or even Data bits: 5, 6, 7, or 8 Stop bits: 1 or 2 RTS on/off delay: 0 to 65535 milliseconds
App Ports (P1, P2) (Serial modules)	RJ45 (DB-9M with supplied cable) RS-232 handshaking configurable 500V Optical isolation from backplane
Shipped with Unit	RJ45 to DB-9M cables for each port 6-foot RS-232 configuration cable

11.1.3 Functional Specifications

The BASIC programs are stored in the MVI56-BAS Compact Flash disk allowing the user to easily backup the BASIC programs to another Compact Flash disk or to a local PC. Up to 24065 Bytes of RAM are available.

The module contains three ports:

- DH-485: DH-485 communication or BASIC program transfer
- PRT1: ASCII communication or program port
- PRT2: ASCII communication or DF1 communication

The user can either create the BASIC programs using the command line, then save it to Compact Flash, or create the program on a PC and download it to the MVI56-BAS module. The user can also configure the module to execute a BASIC program automatically at power up.

- Supports BASIC-52 Programming language
- Stores BASIC programs on Compact Flash disk
- Supports 24064 bytes of RAM
- Easy BASIC program backup

DF1 Communications

- PRT2 can be enabled and configured for DF1 Communications through BASIC calls
- Uses ladder logic to transfer data between the DF1 device and the ControlLogix processor
- PRT2 port can be configured to act as a full-duplex device or half-duplex slave

The MVI56-BAS can send the following DF1 commands:

- PLC-2 unprotected READ command
- PLC-2 unprotected WRITE command
- PLC-3 word range READ command
- PLC-3 word range WRITE command
- PLC-5 typed READ command
- PLC-5 typed WRITE command

The MVI56-BAS accepts the following commands:

- PLC (word range writes)
- PLC (typed writes)
- PLC (unprotected write)
- SLC 5/02 (unprotected write)
- SLC 5/02 (typed writes)

DH-485 Communications

- The MVI56-BAS DH-485 port can be used to interface the ControlLogix processor with DH-485 Devices
- Rockwell Automation AIC+ device is required (user-supplied)
- Ladder logic is required to transfer data between the ControlLogix processor and remote data file or common interface file

11.2 Functional Overview

11.2.1 General Concepts

The following discussion explains several concepts that are important for understanding module operation.

Module Power Up

On power up the module begins performing the following logical functions:

- If the setup jumper is ON, the module powers up in command mode with PRT1 designated as the program port at 19200 baud, no parity, eight bits per character, one stop bit, and XON/XOFF flow control.

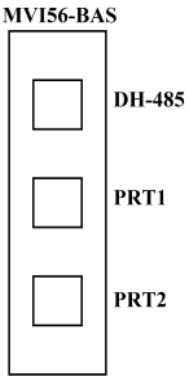
  Else:
- The program port is defined to be what ever port was designated via the last PGMPRT command.
- If there is a BASIC program stored in EPROM and PROG2 has previously been executed, then start executing ROM1 program.

  Else:
- If there is a BASIC program in XRAM, then start executing the XRAM program.

  Else:
- Go to the BASIC command prompt (command mode).

MVI56-BAS Ports Overview

The MVI56-BAS module uses three communication ports. These ports can execute different tasks such as a program port, ASCII communication port, DH-485 port, or DF1 port and can be used to allow access to the MVI56-BAS BASIC programs.



DH-485

The DH-485 port has the following main functions:

- DH-485 communications
- BASIC program transfer

DH-485 Communications

The DH-485 port allows the MVI56-BAS to interface with DH-485 networks. This port is not directly DH-485 compatible as it requires the Rockwell Automation AIC+ module (user-supplied). Further, the setup jumper must be OFF for the DH-485 driver to operate correctly. The default parameters are:

Baud Rate	19200
Node	1
Maximum Node Address	31

These parameters can be changed using the MODE statement.

Refer to Using DH-485 Communications (page 93) for more information about using the DH-485 protocol.

BASIC Program Transfer

If the user quits the BASIC application entering "EXIT" while connected to the program port (PRT1), the DH-485 port allows the user to access the MVI56-BAS C: drive (Compact Flash) where the programs are stored in DOS. The user can move BASIC programs from the PC to the MVI56-BAS or vice versa using this port.

PRT1

- Program Port
- ASCII Communication

PRT1 is the default BASIC program port. This means that all command line statements should be entered using an ASCII terminal software connected to PRT1. The default parameters are 19200 baud rate, 8 bits per character, no parity and one stop bit. If the setup jumper is installed, then the PRT1 parameters cannot be changed. If the setup jumper is removed, the MODE command can change the communication parameters.

An RJ45 to DB-9 adapter cable is required (supplied with the MVI56-BAS module) to use PRT1.

The PRT1 port also can be used to send and receive serial data, interfacing the ControlLogix processor with ASCII devices such as printers and bar readers as will be described later in this manual.

PRT2

- DF1 Communication
- ASCII Communication

The PRT2 port can be used to send and receive serial data, interfacing the ControlLogix processor with ASCII devices such as printers and bar code readers as will be described later in this manual. The default parameters are 19200 baud, 8 bits per character, no parity and one stop bit. The setup jumper has no effect on this port.

An RJ45 to DB-9 adapter cable is required (supplied with the MVI56-BAS module) to use PRT2.

PRT2 can also be used as a DF1 port if the DF1 driver is enabled (using CALL 108). Refer to Using DF1 Protocol Communications for more information on using the DF1 protocol.

Changing the Port Communication Parameters

The port communication parameters can be changed using the MODE command:

The MVI56-BAS ports PRT1 (COM2) and PRT2 (COM3) have the following default communication parameters:

Description	Value
Baud Rate	19200
Parity	N
Number of Data Bits	8
Number of Stop Bits	1
Handshaking	N

In order to change a port communication parameter, you should use the MODE function.

MODE Syntax for PRT1 (COM2) and PRT2 (COM3)

MODE(port,baud rate, parity, data bits, stop bits, handshake, storage type)

Where:

Port:	PRT1, PRT2
Communication rate:	300, 600, 1200, 2400, 4800, 9600, 19200.
Parity:	N (none), E (even), O (odd).
Number of data bits:	7 or 8 .
Number of Stop Bits:	1 or 2 .
Handshaking:	N (no handshaking), S (software handshaking),
	H (hardware handshaking), B (hardware and software handshaking)
Storage type:	E (store information in user ROM and RAM) ,
	R (store information in battery backed RAM)

For example, in order to change the PRT 1 communication baud rate to 9600, the user would enter:

```
Ready
>MODE(PRT1,9600,N,8,1,N,R)
```

MODE Syntax for DH-485 (COM1) port

MODE(port, baud rate, host node address, module node address, maximum node address, , storage type)

Where:

Port:	DH485
Communication rate:	300, 600, 1200, 2400, 4800, 9600, 19200.
Host Node Address:	0 to 31

Port:	DH485
Module Node Address:	1 to 31
Maximum Node Address:	1 to 31
Storage type:	E (store information in user ROM and RAM) ,
	R (store information in battery backed RAM)

The following CALLs have the same effect as using the MODE command for PRT1 and PRT2:

CALL 30: Set PRT2 parameters

CALL 78: Set Program Baud Rate

The user can also display the current port configuration for PRT1 and PRT2 using the following CALLs:

CALL 31: Display Current PRT2 Port Setup

CALL 94: Display Current PRT1 Port Setup

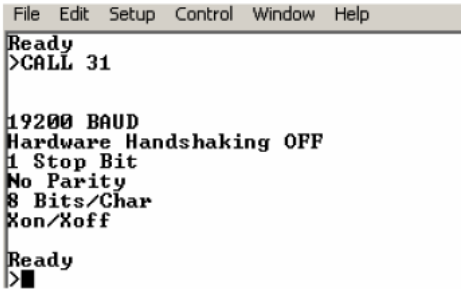
For example, in order to change the DH-485 port parameters to 9600 baud, Host Address = 5, Module Address = 1, Maximum Address = 10; the user would enter:

```
Ready
>MODE(DH485,9600,5,1,10,,R)
```

Further, parameters can be omitted if the user does not want to change them. For example, if the user only wanted to change the baud rate to 1200 baud for PRT1:

```
Ready
>MODE(PRT1,1200,,,,,)
```

For example, running CALL 31 displays:



After changing the PORT communication parameters using the MODE command, the user can reset the communication parameters to their default values using the following CALLs:

CALL 105: Reset PRT1 to Default Settings

CALL 119: Reset PRT2 to Default Settings

Refer to BASIC CALLs Syntax (page 105) for more information about CALL 30, 78, 105 and 119.

MVI56-BAS Internal Real Time Clock

The MVI56-BAS has an internal real-time clock that allows date and time information to be used in the BASIC programs.

In order to enable the free running clock, the CLOCK1 statement must be used in the BASIC program or in the Command Line. The real time clock starts running until it is disabled by the CLOCK0 statement. The clock increments from 0 to 65535.995 seconds. When it reaches its maximum value, it will rollover to 0.

The current time value can be retrieved by the TIME statement which shows the current time value. It is incremented once every 5 milliseconds. The TIME statement can also be used to change the current time value.

For example (using the command line):

```
>CLOCK1
> PRINT TIME
45.655
> TIME = 0
>PRINT TIME
.115
>CLOCK0
```

There are also BASIC CALLs that allow the use of date and time information in the BASIC programs. Refer to Wall Clock CALLs (page 137).

In order to compensate for the incompatibility between the timer in the microprocessor and the MVI56-BAS, the ONTIME statement can be used. The MVI56-BAS can operate a line in an order of milliseconds and the microprocessor works in the order of microseconds. The ONTIME statement generates an interrupt every time the TIME operator is equal or greater than the expression following the ONTIME statement.

Example:

```
Ready
>list
10 TIME = 0
20 CLOCK1
30 ONTIME 2, 80
40 DO
50 WHILE TIME < 10
60 CLOCK0
70 END
80 PRINT "Timer interrupt at -", TIME, " seconds"
90 ONTIME TIME + 2, 80
100 RETI
Ready
>run
Timer interrupt at - 2.004 seconds
Timer interrupt at - 4.011 seconds
Timer interrupt at - 6.016 seconds
Timer interrupt at - 8.022 seconds
Ready
>
```

BASIC Program Usage

BASIC-52 originally stored BASIC programs in two memory areas: battery backed external RAM (XRAM), and EEPROM (ROM). In the MVI56-BAS BASIC, two files on the Compact Flash emulate XRAM and EEPROM. These two files are named "XRAM.BAS" and "EPROM.BAS". When a program is typed directly into the module, the program is tokenized and stored in "XRAM.BAS". The PROG command copies the XRAM program into "EPROM.BAS". The XFER command copies one of the programs back from "EPROM.BAS" into "XRAM.BAS". The following topics explain how the module uses the RAM and ROM storage areas.

Using RAM Memory

To actually run a BASIC program, you must get the BASIC program into RAM. There are three ways to get a program into RAM:

1. Type the program in a line at a time using an ASCII terminal.
2. Use the EXPORT and LOAD commands.
3. Use the XFER command.

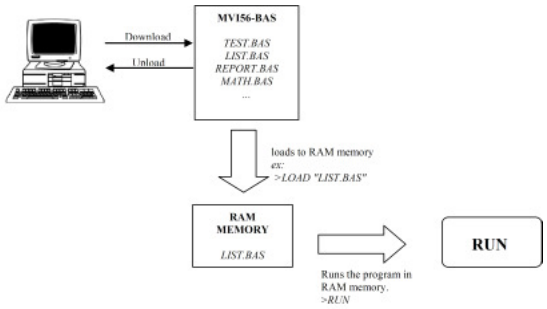
Typing the Program

When typing the program in directly, type the program in a line at a time. If a mistake is made in the middle of a line, either type the line over or use the EDIT command. If an existing line must be replaced with a new one, type the line in again. Lines can be inserted between two existing lines by typing in a line with a line number between the lines. If the line numbers are consecutive integers, and another line cannot be inserted, use the REN command. If a line must be deleted, type the line number. Refer to Creating BASIC programs (page 47) for more information.

Using the EXPORT and LOAD Commands

As stated above, "XRAM.BAS" is tokenized before it is saved, so the file is not directly readable. One may want to save a version that is readable by the human eye. After the program has been entered and debugged, a permanent readable copy of the program can be saved to the Compact flash using the EXPORT command. Then EXIT to DOS where the SY.EXE program can be used to upload the program to a host computer.

Later, the readable version can be downloaded from the host computer using RY.EXE. Then use the LOAD command to copy the readable program back into RAM. Refer to the following illustration and Creating BASIC programs (page 47) for more information:



Use the NEW command to initialize the RAM memory. This command causes the RAM program to be deleted, and clears all variables, strings, and interrupts.

Example:

```
>NEW
```

The program in RAM mode is actually saved in a file in the Compact flash called "XRAM.BAS". It allows the MVI56-BAS module to run the program at RAM during start up.

Important: Do not try to edit or delete "XRAM.BAS", "BATTERY.BAS", or "EPROM.BAS". These files are created and edited by the MVI56-BAS.

Using ROM Storage

Optionally, the BASIC programs can also be stored in the ROM area. Using ROM storage allows the user to assign each program to a specific numeric location in the module. It can be easily accessed later using ROM and RROM commands. The user may also set up the module to run the BASIC program located at ROM storage location 1 during power up (PROG2 command). The ROM storage area is located in the Compact Flash file. The total storage capacity is 32 Kbytes.

In order to place a BASIC program in ROM storage, there must be a program in RAM.

Example:

```
>LOAD "LIST.TXT"
```

After the program is in RAM memory, the user can move it to ROM storage using the `PROG` command. The `PROG` command moves the BASIC program currently in RAM memory to the first available location in the ROM area.

Example:

```
>PROG
```

In order to select a program currently stored in the ROM area, the `ROM [program location]` command is used. A simple `RUN` command will run the selected BASIC program.

The following example shows how to select and run the BASIC program at the third location from the MVI56-BAS ROM storage area.

Example:

```
>ROM 3
>RUN
```

Using the `RROM [program location]` command allows the user to select and run the BASIC program from ROM area. In this case, it is not necessary to use the `RUN` command.

Example:

```
>RROM 3
```

The `ROM` (or RROM) command selects the BASIC program from the ROM area even if there is a different BASIC program in the RAM memory. For example, commands such as `LIST` and `RUN` will refer to the program selected from ROM. In order to select the BASIC program out of RAM memory, the `RAM` command is used:

```
>RAM
```

The PROG1 command loads the port configuration during the power up.

Example:

```
>PROG1
```

The MVI56-BAS module allows the user to load the first BASIC program (location n° 1) in the ROM storage area after the module power up. In order to accomplish this, the `PROG2` command is used.

Example:

```
>PROG2
```

After the `PROG2` command is used, the `PROG0` command can be used in order to return the module to normal operation.

Example:

```
>PROG0
```

To delete the latest BASIC program from the ROM area, the `ERASE` command can be used:

Example:

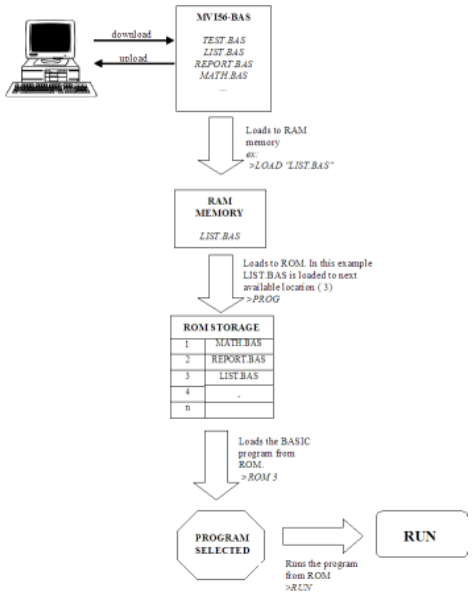
```
>ERASE
```

In order to check if the current program in RAM will fit in the ROM storage area CALL 81 can be used.

```
>CALL 81

Number of BASIC programs in EPROM..........3
Available bytes to end of user EPROM.......31553
Length of BASIC program in RAM.............277

Program will fit in EPROM.
```

For more information about transferring a program between the PC and the MVI56-BAS refer to the "Using the Program Port" section.

The following schematic provides a better understanding on how to store and access a BASIC program in the ROM storage area:



11.3 Cable Connections

The MVI56-BAS module has the following communication connections on the module:

- BASIC DH-485
- BASIC PRT1
- BASIC PRT2

Each of the above module connectors are RJ45 style. Short cable adapters are provided to convert the RJ45 connections to DB-9. The following table shows the pinout for the DH-485 connector:

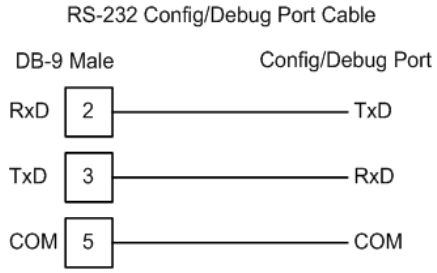
RJ45 Pin #	DB-9 Pin #	Description
1	1	DCD
2	2	RXD
3	3	TXD
4	4	DTR
5	5	Logic Ground
6	6	DSR
7	7	RTS
8	8	CTS
	9	Open
Shell	Shell	Earth Ground

The following table shows the pinout for both PRT1 and PRT2:

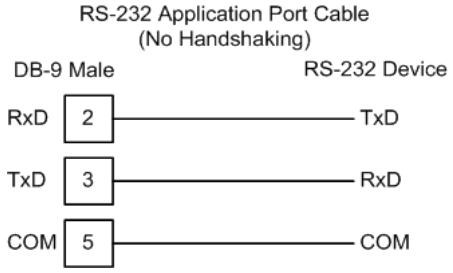
RJ45 Pin #	DB-9 Pin #	RS-232	RS-422	RS-485
1	1	DCD	TXD+	TXD/RXD+
2	2	RXD	RXD+	
3	3	TXD		
4	4	DTR		
5	5	Common	Common	Common
6	6	DSR	RXD-	
7	7	RTS		
8	8	CTS	TXD-	TXD/RXD-
	9	Open	Open	Open
Shell	Shell	Earth Ground	Earth Ground	Earth Ground

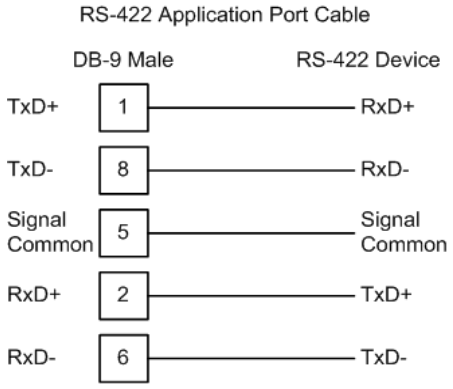
11.3.1 BASIC DH-485 Port

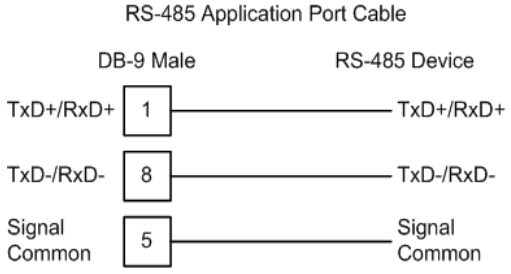
This port is physically an RJ45 connection. An RJ45 to DB-9 adapter cable is shipped with the module. This port interfaces to the Rockwell Automation 1761-NET-AIC (AIC+) which interfaces directly to a DH-485 network. It is important that the setup jumper be OFF when interfacing to a DH-485 network. The cable for communications on this port is shown in the following diagram:



11.3.2 BASIC PRT1 and PRT2







11.4 1746-BAS Comparison

11.4.1 CALLs Not Supported

CALL 26 not supported

CALL 55 not supported

CALL 77 does not return any practical result because MTOP is not used in the MVI56- BAS

CALL 82 not supported

CALL 103 not supported

CALL 104 not supported

CALL 110 not supported

CALL 111 not supported

CALL 121 not supported

DBY command not supported

LD@ ST@ replaced by LD_AT and ST_AT

11.4.2 New Commands

- PGMPRT: Allows the user to select DH-485 or PRT1 as the program port.
- EXPORT: Allows an edited program to be saved in the Compact Flash.
- LOAD: Allows the user to load a DOS ASCII Basic file into emulated external RAM memory.
- EXIT: Allows the user to exit the BASIC interpreter to the DOS Command Prompt (DH-485 port). The setup jumper must be installed.
- ERRCODE: A regular operator that returns the last error encountered since the beginning of the program or since the last CLRERR.
- ERRLINE: A regular operator that returns the line where the last error was encountered since the beginning of the program or since the last CLRERR.
- PRERR: Prints the description and line number of the last encountered error.
- PROG0: Causes the previous PROG2 command to be cancelled. If a bug is found that needs correcting and the user does not want to power up the MVI56-BAS (causing the ROM program to start automatically), PROG0 solves this problem.
- XBYTE: Functions like the 1746-BAS XBY. However, the argument range is 0000h to 0FFFh (0 to 4095)

11.4.3 Differences

- The MVI56-BAS variables are defined by all the characters in the variable name.
- MVI56-BAS uses 64-bit IEE floating point math routines. The difference can involve numerical round-offs issues.
- The MVI56-BAS has no read RAM or ROM memory. Files in the Compact Flash simulate RAM and ROM storage.

12 Support, Service & Warranty

12.1 Contacting Technical Support

ProSoft Technology, Inc. is committed to providing the most efficient and effective support possible. Before calling, please gather the following information to assist in expediting this process:

1. Product Version Number
2. System architecture
3. Network details

If the issue is hardware related, we will also need information regarding:

1. Module configuration and associated ladder files, if any
2. Module operation and any unusual behavior
3. Configuration/Debug status information
4. LED patterns
5. Details about the serial, Ethernet or Fieldbus devices interfaced to the module, if any.

Note: For technical support calls within the United States, ProSoft's 24/7 after-hours phone support is available for urgent plant-down issues.

North America (Corporate Location)	**Europe / Middle East / Africa Regional Office**
Phone: +1.661.716.5100 info@prosoft-technology.com Languages spoken: English, Spanish REGIONAL TECH SUPPORT support@prosoft-technology.com	Phone: +33.(0)5.34.36.87.20 france@prosoft-technology.com Languages spoken: French, English REGIONAL TECH SUPPORT support.emea@prosoft-technology.com
Latin America Regional Office	**Asia Pacific Regional Office**
Phone: +52.222.264.1814 latinam@prosoft-technology.com Languages spoken: Spanish, English REGIONAL TECH SUPPORT support.la@prosoft-technology.com	Phone: +60.3.2247.1898 asiapc@prosoft-technology.com Languages spoken: Bahasa, Chinese, English, Japanese, Korean REGIONAL TECH SUPPORT support.ap@prosoft-technology.com

For additional ProSoft Technology contacts in your area, please visit:
https://www.prosoft-technology.com/About-Us/Contact-Us.

12.2 Warranty Information

For complete details regarding ProSoft Technology's TERMS & CONDITIONS OF SALE, WARRANTY, SUPPORT, SERVICE AND RETURN MATERIAL AUTHORIZATION INSTRUCTIONS please see the documents at:
www.prosoft-technology/legal