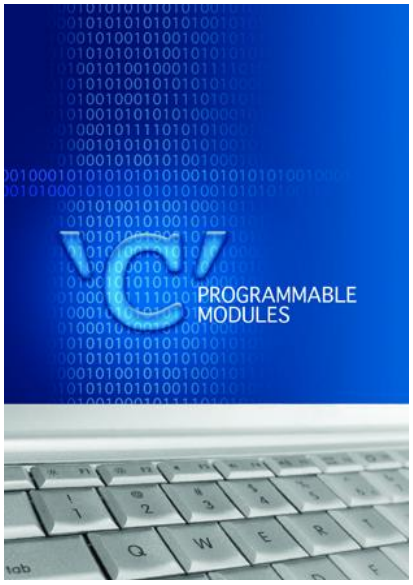
Where Automation Connects.



PTQ-ADM

'C' Programmable

'C' Programmable Network Interface Module for Quantum

February 20, 2013

DEVELOPER GUIDE

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.

ProSoft Technology
5201 Truxtun Ave., 3rd Floor
Bakersfield, CA 93309
+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com
support@prosoft-technology.com

PTQ-ADM Developer Guide

February 20, 2013

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 on the enclosed CD-ROM, and are available at no charge from our web site: www.prosoft-technology.com.

Content Disclaimer

Printed documentation is available for purchase. Contact ProSoft Technology for pricing and availability.

North America: +1.661.716.5100

Asia Pacific: +603.7724.2080

Europe, Middle East, Africa: +33 (0) 5.3436.87.20

Information for ProTalk® Product Users

The statement "power, input and output (I/O) wiring must be in accordance with Class I, Division 2 wiring methods Article 501-10(b) of the National Electrical Code, NFPA 70 for installations in the U.S., or as specified in section 18-1J2 of the Canadian Electrical Code for installations within Canada and in accordance with the authority having jurisdiction".

The following or equivalent warnings shall be included:

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 Wiring Modules, and
C Warning - Explosion Hazard - Do not Disconnect Equipment unless Power has been switched Off or the Area is known to be Nonhazardous.
D Caution: The Cell used in this Device may Present a Fire or Chemical Burn Hazard if Mistreated. Do not Disassemble, Heat above 100°C (212°F) or Incinerate.

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.

Class I, Division 2 GPs A, B, C, D

II 3 G

Ex nA IIC X

0° C <= Ta <= 60° C

II - Equipment intended for above ground use (not for use in mines).

3 - Category 3 equipment, investigated for normal operation only.

G - Equipment protected against explosive gasses.

Warnings

North America Warnings

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.
Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.
C 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.

Electrical Ratings

- Backplane Current Load: 1100 mA maximum @ 5 Vdc ± 5%
- 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 to 150 Hz
- Relative Humidity: 5% to 95% (without condensation)
- All phase conductor sizes must be at least 1.3 mm(squared) and all earth ground conductors must be at least 4mm(squared).

Markings:

CSA/cUL	C22.2 No. 213-1987
CSA CB Certified	IEC61010
ATEX	EN60079-0 Category 3, Zone 2 EN60079-15

243333

Important Notice:

⚠	CAUTION: THE CELL USED IN THIS DEVICE MAY PRESENT A FIRE OR CHEMICAL BURN HAZARD IF MISTREATED. DO NOT DISASSEMBLE, HEAT ABOVE 100°C (212°F) OR INCINERATE. Maximum battery load = 200 μA. Maximum battery charge voltage = 3.4 VDC. Maximum battery charge current = 500 μA. Maximum battery discharge current = 30 μA.

Contents

1 Start Here

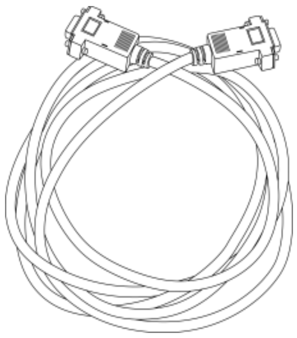
In This Chapter

This guide is intended to guide you through the ProTalk module setup process, from removing the module from the box to exchanging data with the processor. In doing this, you will learn how to:

- Set up the processor environment for the PTQ module
- View how the PTQ module exchanges data with the processor
- Edit and download configuration files from your PC to the PTQ module
- Monitor the operation of the PTQ module

1.1 Hardware and Software Requirements

1.1.1 Package Contents

	
ProTalk Module	Null Modem Serial Cable
1454-9F DB-9 Female to 9 Pos Screw Terminal adapter (Serial protocol modules only)	ProSoft Solutions CD

Note: The DB-9 Female to 5 Pos Screw Terminal adapter is not required on Ethernet modules and is therefore not included in the carton with these types of modules.

Quantum Hardware

This guide assumes that you are familiar with the installation and setup of the Quantum hardware. The following should be installed, configured, and powered up before proceeding:

- Quantum Processor
- Quantum rack
- Quantum power supply
- Quantum Modbus Plus Network Option Module (NOM Module) (optional)
- Quantum to PC programming hardware
- NOM Ethernet or Serial connection to PC

PC and PC Software

- Windows-based PC with at least one COM port
- Quantum programming software installed on machine
  or
- Concept™ PLC Programming Software version 2.6
  or
  ProWORX PLC Programming Software
  or
  Unity™ Pro PLC Programming Software

Note: ProTalk modules are compatible with common Quantum programming applications, including Concept and Unity Pro. For all other programming applications, please contact technical support.

1.1.2 Recommended Compact Flash (CF) Cards

What Compact Flash card does ProSoft recommend using?

Some ProSoft products contain a "Personality Module", or Compact Flash card. ProSoft recommends using an industrial grade Compact Flash card for best performance and durability. The following cards have been tested with ProSoft's modules, and are the only cards recommended for use. These cards can be ordered through ProSoft, or can be purchased by the customer.

Approved ST-Micro cards:

- 32M = SMC032AFC6E
- 64M = SMC064AFF6E
- 128M = SMC128AFF6E

Approved Silicon Systems cards:

- 256M = SSD-C25MI-3012
- 512M = SSD-C51MI-3012
- 2G = SSD-C02GI-3012
- 4G = SSD-C04GI-3012

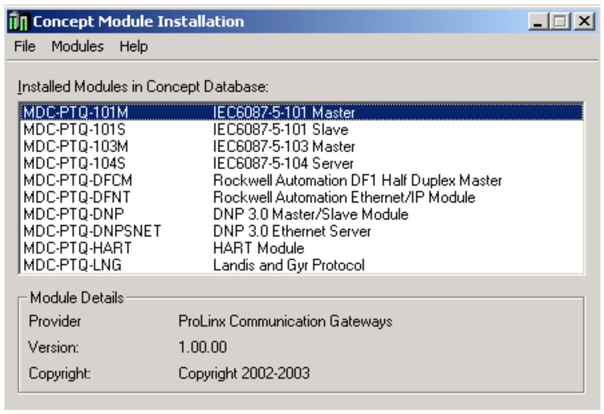
1.2 Information for Concept Version 2.6 Users

This guide uses Concept PLC Programming Software version 2.6 to configure the Quantum PLC. The ProTalk installation CD includes MDC module configuration files that help document the PTQ installation. Although not required, these files should be installed before proceeding to the next section.

1.2.1 Installing MDC Configuration Files

1 From a PC with Concept 2.6 installed, choose **START / PROGRAMS / CONCEPT / MODCONNECT TOOL**.

This action opens the *Concept Module Installation* dialog box.



Installed Modules in Concept Database:

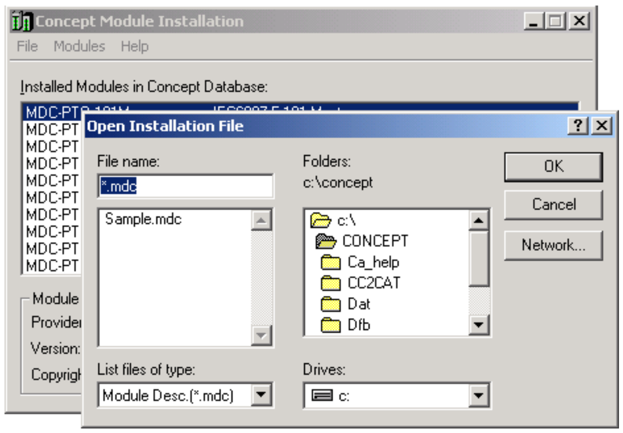
MDC-PTQ-101M	IEC6087-5-101 Master
MDC-PTQ-101S	IEC6087-5-101 Slave
MDC-PTQ-103M	IEC6087-5-103 Master
MDC-PTQ-104S	IEC6087-5-104 Server
MDC-PTQ-DFCM	Rockwell Automation DF1 Half Duplex Master
MDC-PTQ-DFNT	Rockwell Automation Ethernet/IP Module
MDC-PTQ-DNP	DNP 3.0 Master/Slave Module
MDC-PTQ-DNPSNET	DNP 3.0 Ethernet Server
MDC-PTQ-HART	HART Module
MDC-PTQ-LNG	Landis and Gyr Protocol

Module Details

Provider	ProLinx Communication Gateways
Version:	1.00.00
Copyright:	Copyright 2002-2003

2 Choose **FILE / OPEN INSTALLATION FILE.**

This action opens the *Open Installation File* dialog box:



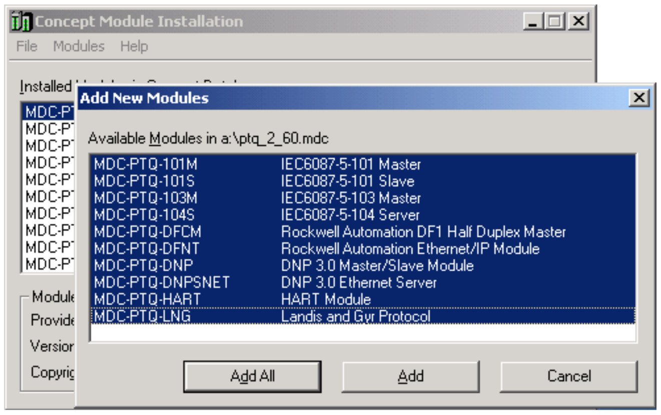
File name: *.mdc
Folders: c:\concept
Sample.mdc
List files of type: Module Desc.(*.mdc)
Drives: c:

3 If you are using a Quantum processor, you will need the MDC files. In the *Open Installation File* dialog box, navigate to the *MDC Files* directory on the ProTalk CD.

4 Choose the MDC file and help file for your version of Concept:

- Concept 2.6 users: select PTQ_2_60.mdc and PTQMDC.hlp
- Concept 2.5 users: select PTQ_2_50.mdc and PTQMDC.hlp.

Select the files that go with the Concept version you are using, and then click **OK**. This action opens the *Add New Modules* dialog box.



Available Modules in a:\ptq_2_60.mdc

MDC-PTQ-101M	IEC6087-5-101 Master
MDC-PTQ-101S	IEC6087-5-101 Slave
MDC-PTQ-103M	IEC6087-5-103 Master
MDC-PTQ-104S	IEC6087-5-104 Server
MDC-PTQ-DFCM	Rockwell Automation DF1 Half Duplex Master
MDC-PTQ-DFNT	Rockwell Automation Ethernet/IP Module
MDC-PTQ-DNP	DNP 3.0 Master/Slave Module
MDC-PTQ-DNPSNET	DNP 3.0 Ethernet Server
MDC-PTQ-HART	HART Module
MDC-PTQ-LNG	Landis and Gyr Protocol

5 Click the **ADD ALL** button. A series of message boxes may appear during this process. Click **YES** or **OK** for each message that appears.

6 When the process is complete, open the **FILE** menu and choose **EXIT** to save your changes.

2 Configuring the Processor with Concept

In This Chapter

The following steps are designed to ensure that the processor is able to transfer data successfully with the PTQ module. As part of this procedure, you will use Concept configuration software from Schneider Electric to create a project, add the PTQ module to the project, set up data memory for the project, and then download the project to the processor.

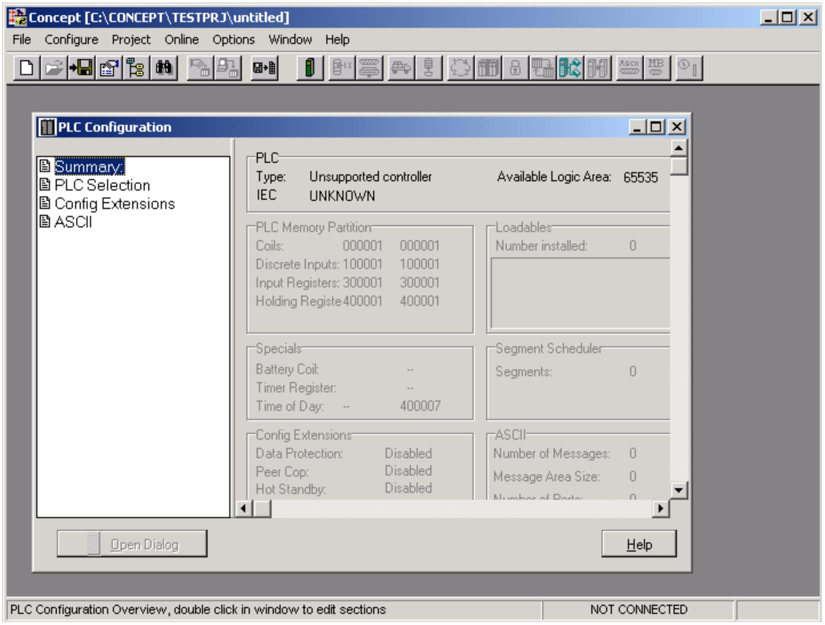
Important Note: Concept software does not report whether the PTQ module is present in the rack, and therefore is not able to report the health status of the module when the module is online with the Quantum processor. Please consider this when monitoring the status of the PTQ module.

2.1 Create a New Project

This phase of the setup procedure must be performed on a computer that has the Concept configuration software installed.

1 From your computer, choose **START / PROGRAMS / CONCEPT V2.6 XL.EN / CONCEPT**. This action opens the *Concept* window.

2 Open the File menu, and then choose **NEW PROJECT**. This action opens the *PLC Configuration* dialog box.


Concept [C:\CONCEPT\TESTPRJ\untitled]
File Configure Project Online Options Window Help
PLC Configuration
Summary
PLC Selection
Config Extensions
ASCII
Open Dialog
Help

PLC
Type: Unsupported controller Available Logic Area: 65535
IEC UNKNOWN

PLC Memory Partition
Coils: 000001 000001
Discrete Inputs: 100001 100001
Input Registers: 300001 300001
Holding Registe 400001 400001

Loadables
Number installed: 0

Specials
Battery Coil: --
Timer Register: --
Time of Day: -- 400007

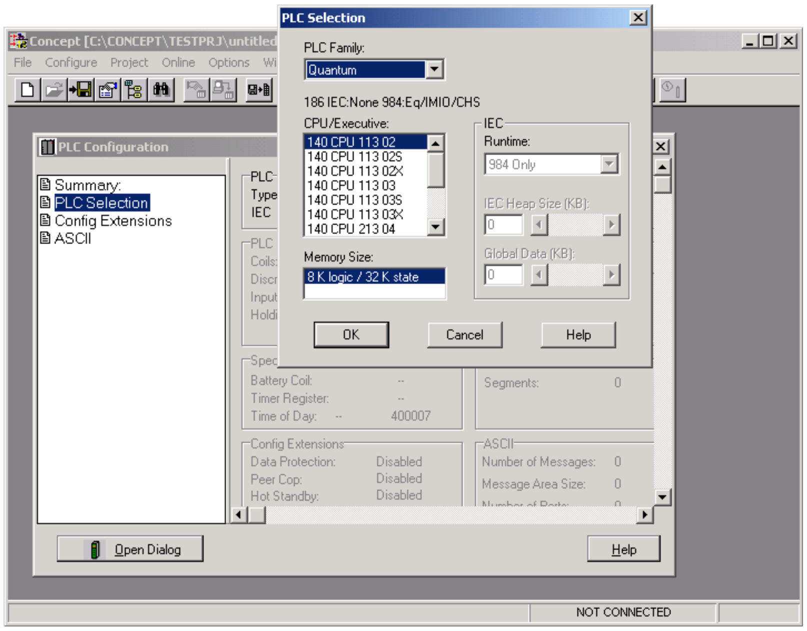
Segment Scheduler
Segments: 0

Config Extensions
Data Protection: Disabled
Peer Cop: Disabled
Hot Standby: Disabled

ASCII
Number of Messages: 0
Message Area Size: 0

PLC Configuration Overview, double click in window to edit sections NOT CONNECTED

3 In the list of options on the left side of this dialog box, double-click the **PLC SELECTION** folder. This action opens the *PLC Selection* dialog box.



PLC Family: Quantum

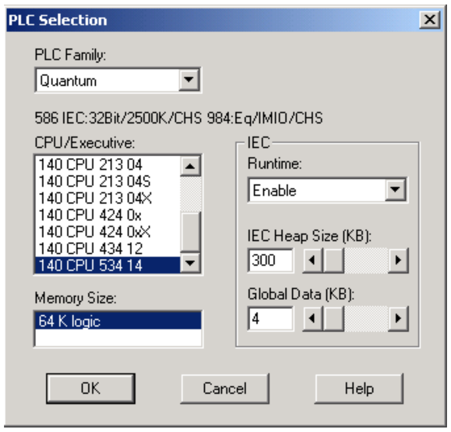
186 IEC:None 984:Eq/IMIO/CHS

CPU/Executive:
140 CPU 113 02
140 CPU 113 02S
140 CPU 113 02X
140 CPU 113 03
140 CPU 113 03S
140 CPU 113 03X
140 CPU 213 04

IEC
Runtime: 984 Only
IEC Heap Size (KB): 0
Global Data (KB): 0

Memory Size: 8 K logic / 32 K state

4 In the *CPU/Executive* pane, use the scroll bar to locate and select the **PLC** to configure.


PLC Selection

PLC Family: Quantum

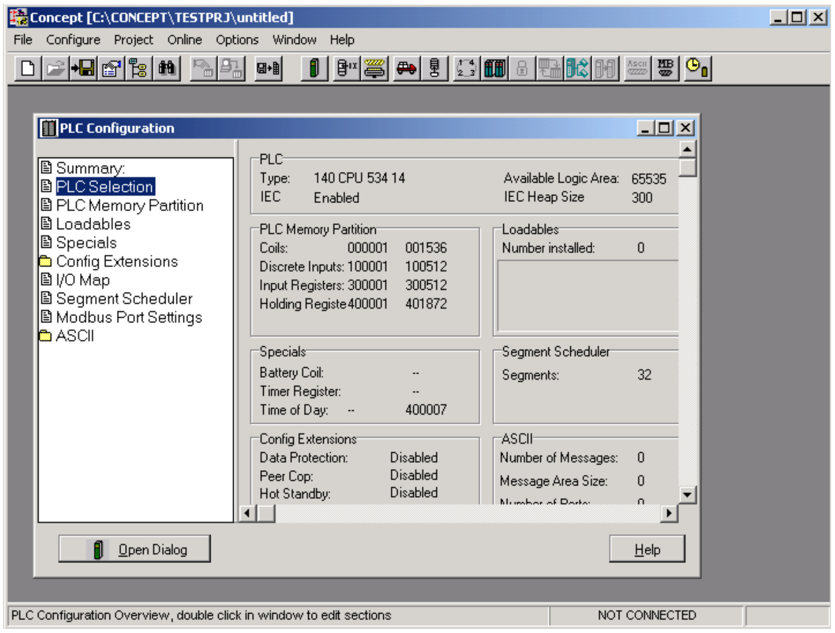
586 IEC:32Bit/2500K/CHS 984:Eq/IMIO/CHS

CPU/Executive:
140 CPU 213 04
140 CPU 213 04S
140 CPU 213 04X
140 CPU 424 0x
140 CPU 424 0xX
140 CPU 434 12
140 CPU 534 14

Memory Size: 64 K logic

IEC
Runtime: Enable
IEC Heap Size (KB): 300
Global Data (KB): 4

5 Click **OK.** This action opens the *PLC Configuration* dialog box, populated with the correct values for the PLC you selected.


Concept [C:\CONCEPT\TESTPRJ\untitled]

PLC Configuration

Summary
PLC Selection
PLC Memory Partition
Loadables
Specials
Config Extensions
I/O Map
Segment Scheduler
Modbus Port Settings
ASCII

PLC
Type: 140 CPU 534 14 Available Logic Area: 65535
IEC Enabled IEC Heap Size 300

PLC Memory Partition
Coils: 000001 001536
Discrete Inputs: 100001 100512
Input Registers: 300001 300512
Holding Registe 400001 401872

Loadables
Number installed: 0

Specials
Battery Coil: --
Timer Register: --
Time of Day: -- 400007

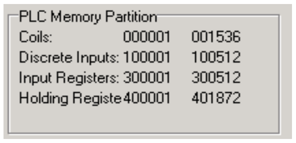
Segment Scheduler
Segments: 32

Config Extensions
Data Protection: Disabled
Peer Cop: Disabled
Hot Standby: Disabled

ASCII
Number of Messages: 0
Message Area Size: 0
Number of Ports: 0

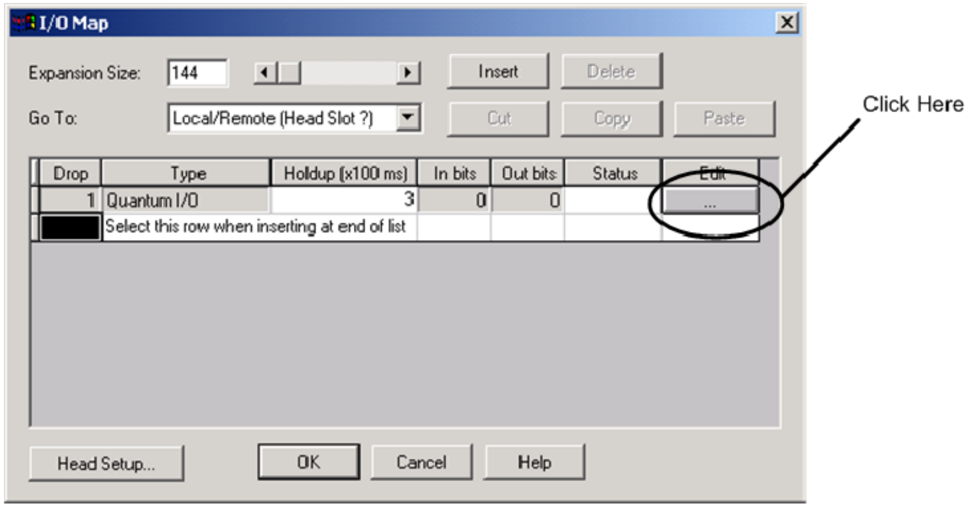
PLC Configuration Overview, double click in window to edit sections NOT CONNECTED

6 Make a note of the holding registers for the module. You will need this information when you modify your application. The Holding Registers are displayed in the *PLC Memory Partition* pane of the *PLC Configuration* dialog box.


PLC Memory Partition
Coils: 000001 001536
Discrete Inputs: 100001 100512
Input Registers: 300001 300512
Holding Registe 400001 401872

2.2 Add the PTQ Module to the Project

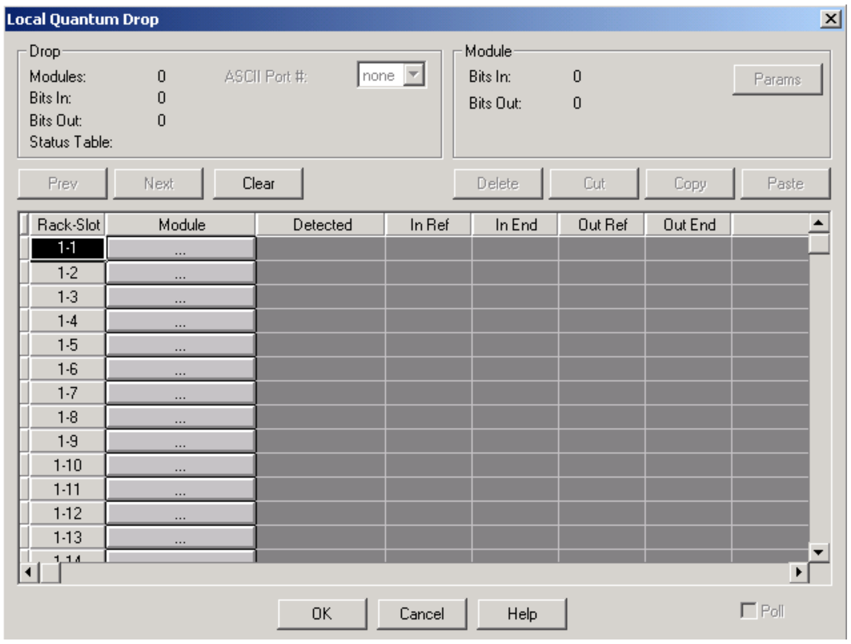
1 In the list of options on the left side of the *PLC Configuration* dialog box, double-click **I/O MAP**. This action opens the *I/O Map* dialog box.



Expansion Size: 144
Go To: Local/Remote (Head Slot ?)

Drop	Type	Holdup (x100 ms)	In bits	Out bits	Status	Edit
1	Quantum I/O	3	0	0		...
	Select this row when inserting at end of list					

2 Click the **EDIT** button to open the *Local Quantum Drop* dialog box. This dialog box is where you identify rack and slot locations.

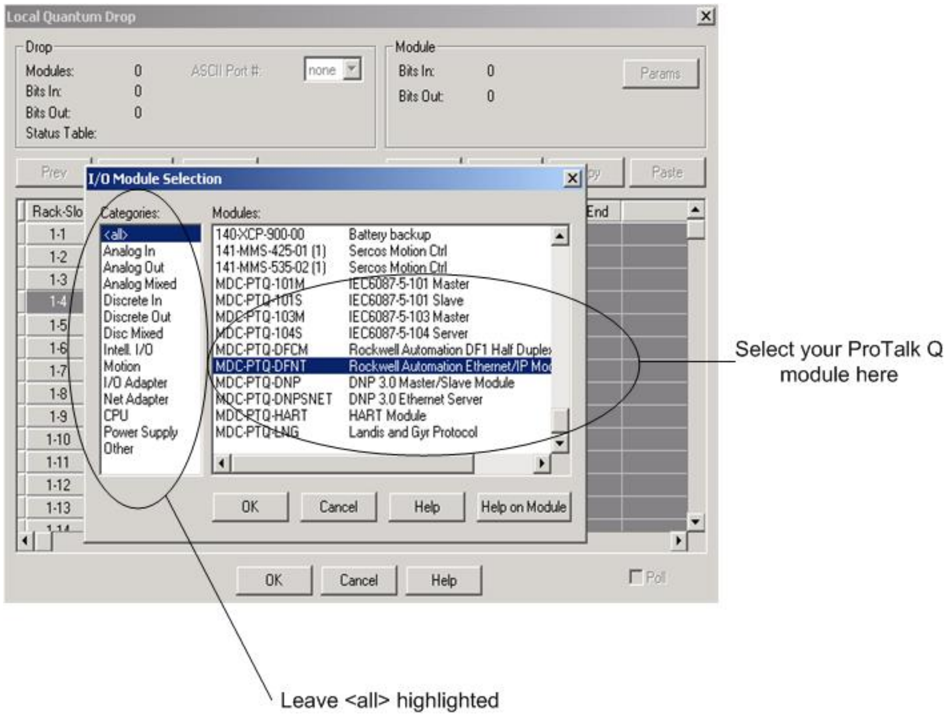


Drop
Modules: 0
Bits In: 0
Bits Out: 0
Status Table:
ASCII Port #: none

Module
Bits In: 0
Bits Out: 0

Rack-Slot	Module	Detected	In Ref	In End	Out Ref	Out End
1-1	...					
1-2	...					
1-3	...					
1-4	...					
1-5	...					
1-6	...					
1-7	...					
1-8	...					
1-9	...					
1-10	...					
1-11	...					
1-12	...					
1-13	...					
1-14						

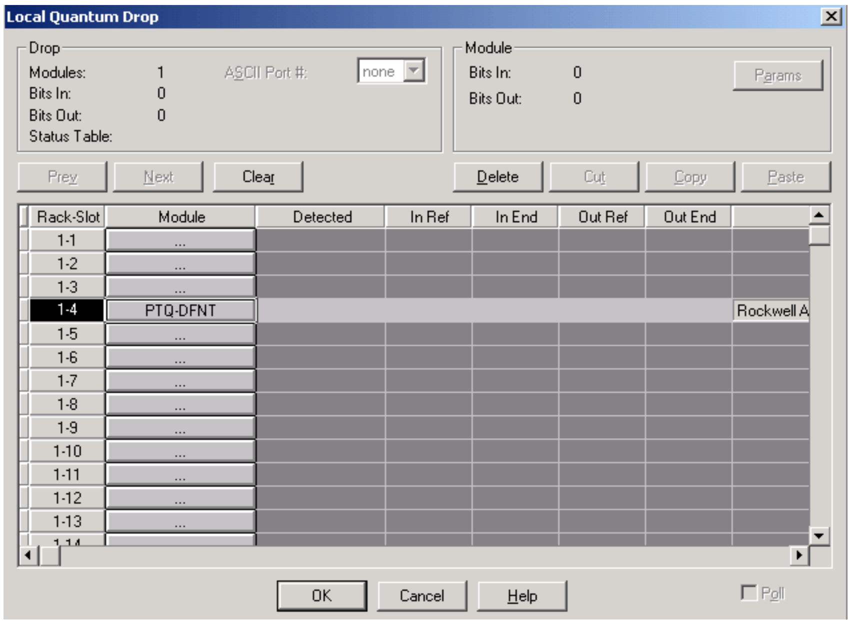
3 Click the **MODULE** button next to the rack/slot position where the ProTalk module will be installed. This action opens the *I/O Module Selection* dialog box.



Categories: <all>, Analog In, Analog Out, Analog Mixed, Discrete In, Discrete Out, Disc Mixed, Intell. I/O, Motion, I/O Adapter, Net Adapter, CPU, Power Supply, Other

Modules:	
140-XCP-900-00	Battery backup
141-MMS-425-01 (1)	Sercos Motion Ctrl
141-MMS-535-02 (1)	Sercos Motion Ctrl
MDC-PTQ-101M	IEC6087-5-101 Master
MDC-PTQ-101S	IEC6087-5-101 Slave
MDC-PTQ-103M	IEC6087-5-103 Master
MDC-PTQ-104S	IEC6087-5-104 Server
MDC-PTQ-DFCM	Rockwell Automation DF1 Half Duplex
MDC-PTQ-DFNT	Rockwell Automation Ethernet/IP Mod
MDC-PTQ-DNP	DNP 3.0 Master/Slave Module
MDC-PTQ-DNPSNET	DNP 3.0 Ethernet Server
MDC-PTQ-HART	HART Module
MDC-PTQ-LNG	Landis and Gyr Protocol

4 In the *Modules* pane, use the scroll bar to locate and select the ProTalk module, and then click **OK.** This action copies the description of the ProTalk module next to the assigned rack and slot number of the *Local Quantum Drop* dialog box.

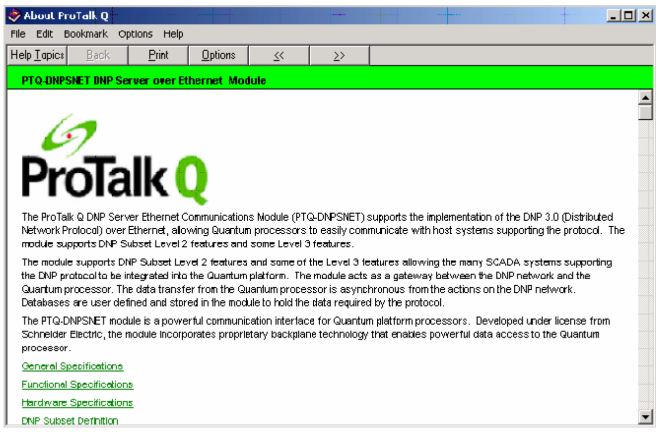


Drop — Modules: 1; Bits In: 0; Bits Out: 0; Status Table:; ASCII Port #: none
Module — Bits In: 0; Bits Out: 0

Rack-Slot	Module	Detected	In Ref	In End	Out Ref	Out End	
1-1	...						
1-2	...						
1-3	...						
1-4	PTQ-DFNT						Rockwell A
1-5	...						
1-6	...						
1-7	...						
1-8	...						
1-9	...						
1-10	...						
1-11	...						
1-12	...						
1-13	...						

5 Repeat steps 3 through 5 for each ProTalk module you plan to install. When you have finished installing your ProTalk modules, click **OK** to save your settings. Click **YES** to confirm your settings.

Tip: Select a module, and then click the Help on Module button for help pages.



PTQ-DNPSNET DNP Server over Ethernet Module

The ProTalk Q DNP Server Ethernet Communications Module (PTQ-DNPSNET) supports the implementation of the DNP 3.0 (Distributed Network Protocol) over Ethernet, allowing Quantum processors to easily communicate with host systems supporting the protocol. The module supports DNP Subset Level 2 features and some Level 3 features.

The module supports DNP Subset Level 2 features and some of the Level 3 features allowing the many SCADA systems supporting the DNP protocol to be integrated into the Quantum platform. The module acts as a gateway between the DNP network and the Quantum processor. The data transfer from the Quantum processor is asynchronous from the actions on the DNP network. Databases are user defined and stored in the module to hold the data required by the protocol.

The PTQ-DNPSNET module is a powerful communication interface for Quantum platform processors. Developed under license from Schneider Electric, the module incorporates proprietary backplane technology that enables powerful data access to the Quantum processor.

General Specifications

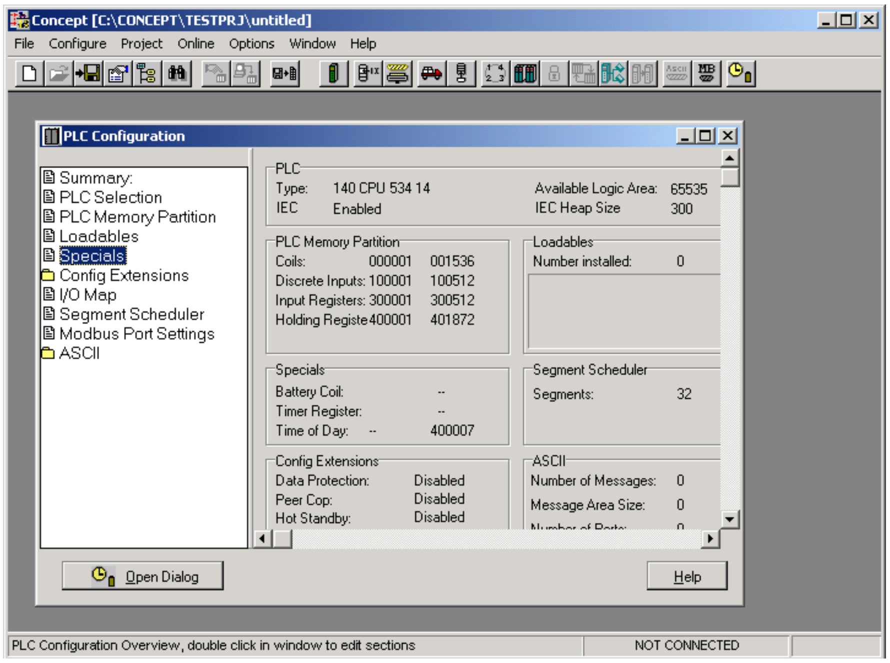
Functional Specifications

Hardware Specifications

DNP Subset Definition

2.3 Set up Data Memory in Project

1 In the list of options on the left side of the *PLC Configuration* dialog box, double-click **SPECIALS.**



PLC
Type: 140 CPU 534 14 Available Logic Area: 65535
IEC Enabled IEC Heap Size 300

PLC Memory Partition
Coils: 000001 001536
Discrete Inputs: 100001 100512
Input Registers: 300001 300512
Holding Registe 400001 401872

Loadables
Number installed: 0

Specials
Battery Coil: --
Timer Register: --
Time of Day: -- 400007

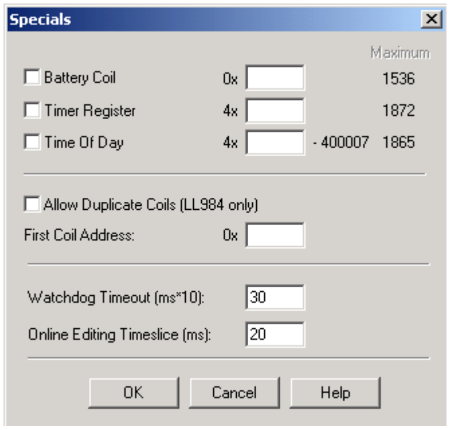
Segment Scheduler
Segments: 32

Config Extensions
Data Protection: Disabled
Peer Cop: Disabled
Hot Standby: Disabled

ASCII
Number of Messages: 0
Message Area Size: 0
Number of Ports: 0

PLC Configuration Overview, double click in window to edit sections NOT CONNECTED

2 This action opens the *SPECIALS* dialog box.



Specials

				Maximum
☐ Battery Coil	0x			1536
☐ Timer Register	4x			1872
☐ Time Of Day	4x		- 400007	1865

☐ Allow Duplicate Coils (LL984 only)

First Coil Address: 0x

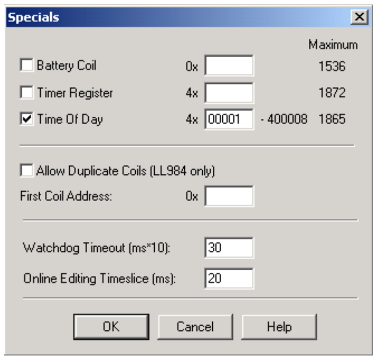
Watchdog Timeout (ms*10): 30

Online Editing Timeslice (ms): 20

OK Cancel Help

Selecting the Time of Day

1 Select (check) the *Time of Day* box, and then enter the value 00001 as shown in the following illustration. This value sets the first time of day register to 400001.



Specials

				Maximum
☐ Battery Coil	0x			1536
☐ Timer Register	4x			1872
☑ Time Of Day	4x	00001	- 400008	1865

☐ Allow Duplicate Coils (LL984 only)

First Coil Address: 0x

Watchdog Timeout (ms*10): 30

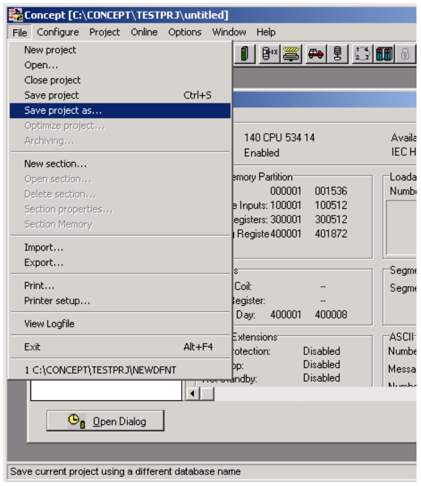
Online Editing Timeslice (ms): 20

OK Cancel Help

2 Click **OK** to save your settings and close the *Specials* dialog box.

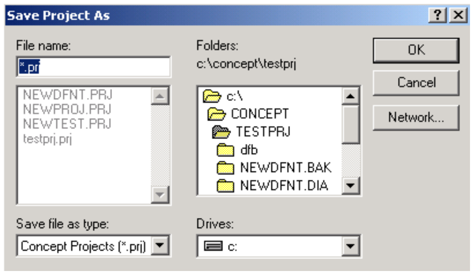
Saving your project

1 In the *PLC Configuration* dialog box, choose **FILE / SAVE PROJECT AS.**



Save current project using a different database name

2 This action opens the *Save Project As* dialog box.



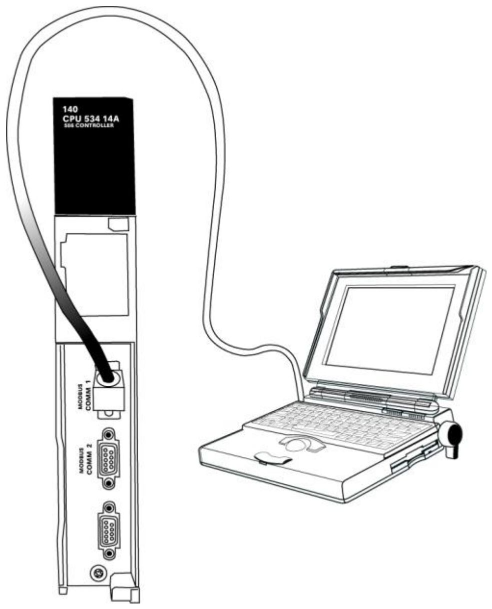
File name: *.prj
Folders: c:\concept\testprj
Save file as type: Concept Projects (*.prj)
Drives: c:

3 Name the project, and then click **OK** to save the project to a file.

2.4 Download the Project to the Processor

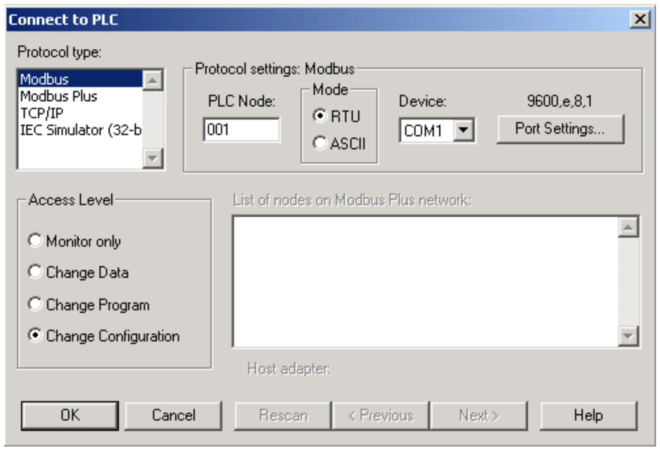
Next, download (copy) the project file to the Quantum Processor.

1 Use the null modem cable to connect your PC's serial port to the Quantum processor, as shown in the following illustration.



Note: You can use a Modbus Plus Network Option Module (NOM Module) module in place of the serial port if necessary.

2 Open the **PLC** menu, and then choose **CONNECT.**
3 In the *PLC Configuration* dialog box, open the **ONLINE** menu, and then choose **CONNECT.** This action opens the *Connect to PLC* dialog box.


Connect to PLC

Protocol type:
Modbus
Modbus Plus
TCP/IP
IEC Simulator (32-b

Protocol settings: Modbus
PLC Node: 001
Mode: [✓ RTU] ASCII
Device: COM1
9600,e,8,1

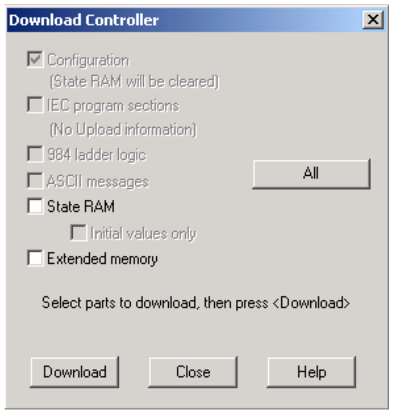
Access Level
Monitor only
Change Data
Change Program
[✓ Change Configuration]

List of nodes on Modbus Plus network:
Host adapter:

4 Leave the default settings as shown and click **OK.**

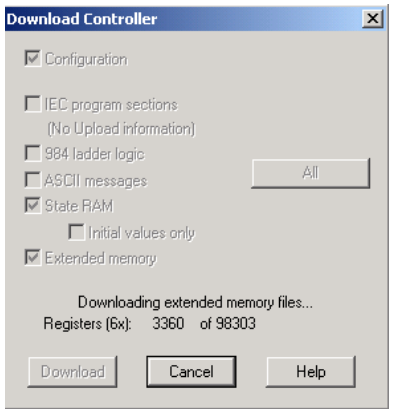
Note: Click **OK** to dismiss any message boxes that appear during the connection process.

5 In the *PLC Configuration* window, open the **ONLINE** menu, and then choose **DOWNLOAD.** This action opens the *Download Controller* dialog box.



Select parts to download, then press <Download>

6 Click **ALL,** and then click **DOWNLOAD.** If a message box appears indicating that the controller is running, click **YES** to shut down the controller. The *Download Controller* dialog box displays the status of the download as shown in the following illustration.



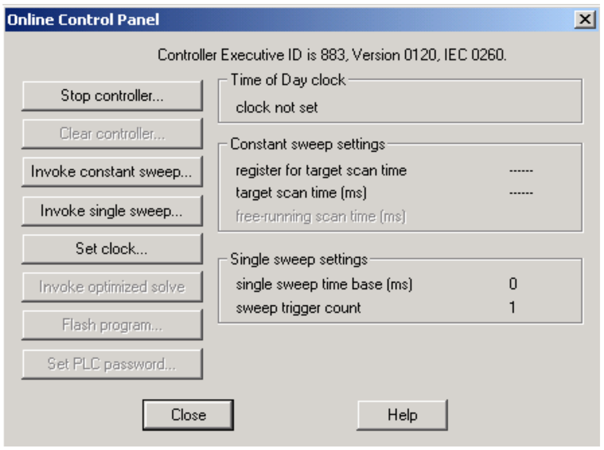
Downloading extended memory files...
Registers (6x): 3360 of 98303

7 When the download is complete, you will be prompted to restart the controller. Click **YES** to restart the controller.

2.5 Verify Successful Download

The final step is to verify that the configuration changes you made were received successfully by the module, and to make some adjustments to your settings.

1 In the *PLC Configuration* window, open the **ONLINE** menu, and then choose **ONLINE CONTROL PANEL**. This action opens the *Online Control Panel* dialog box.


Online Control Panel

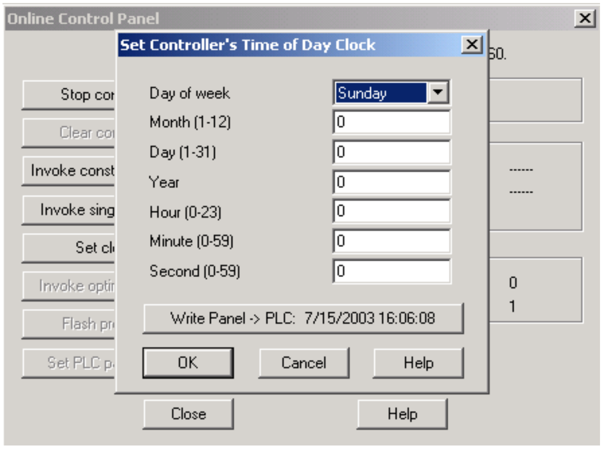
Controller Executive ID is 883, Version 0120, IEC 0260.

Time of Day clock: clock not set

Constant sweep settings:
register for target scan time ------
target scan time (ms) ------
free-running scan time (ms)

Single sweep settings:
single sweep time base (ms) 0
sweep trigger count 1

2 Click the **SET CLOCK** button to open the *Set Controller's Time of Day Clock* dialog box.


Set Controller's Time of Day Clock

Day of week: Sunday
Month (1-12): 0
Day (1-31): 0
Year: 0
Hour (0-23): 0
Minute (0-59): 0
Second (0-59): 0

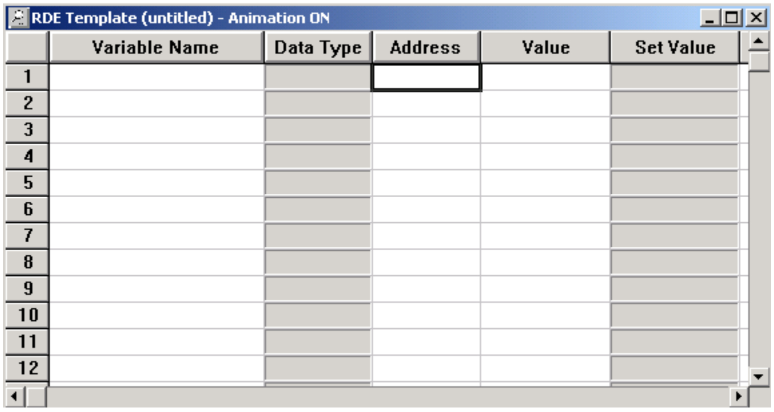
Write Panel -> PLC: 7/15/2003 16:06:08

3 Click the **WRITE PANEL** button. This action updates the date and time fields in this dialog box. Click **OK** to close this dialog box and return to the previous window.

4 Click **CLOSE** to close the *Online Control Panel* dialog box.

5 In the *PLC Configuration* window, open the **ONLINE** menu, and then choose **REFERENCE DATA EDITOR.** This action opens the *Reference Data Editor* dialog box. On this dialog box, you will add preset values to data registers that will later be monitored in the ProTalk module.

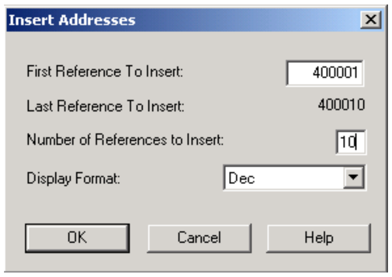
6 Place the cursor over the first address field, as shown in the following illustration.



	Variable Name	Data Type	Address	Value	Set Value
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					

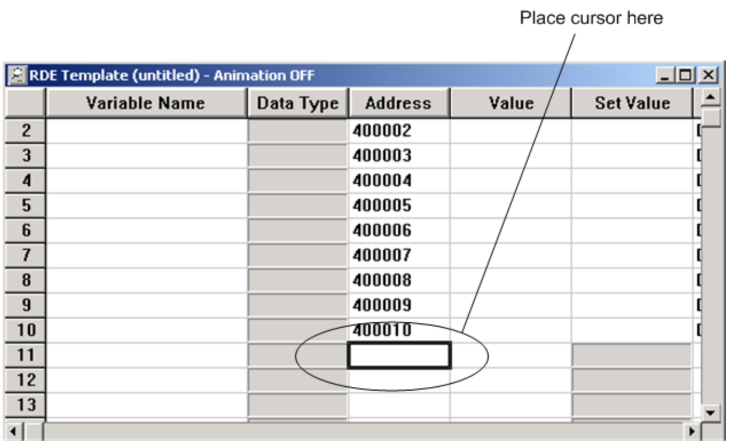
7 In the *PLC Configuration* window, open the **TEMPLATES** menu, and then choose **INSERT ADDRESSES.** This action opens the Insert addresses dialog box.

8 On the *Insert Addresses* dialog box, enter the values shown in the following illustration, and then click **OK.**



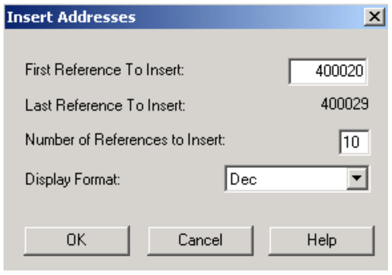
First Reference To Insert: 400001
Last Reference To Insert: 400010
Number of References to Insert: 10
Display Format: Dec

9 Notice that the template populates the address range, as shown in the following illustration. Place your cursor as shown in the first blank address field below the addresses you just entered.



	Variable Name	Data Type	Address	Value	Set Value
2			400002		
3			400003		
4			400004		
5			400005		
6			400006		
7			400007		
8			400008		
9			400009		
10			400010		
11					
12					
13					

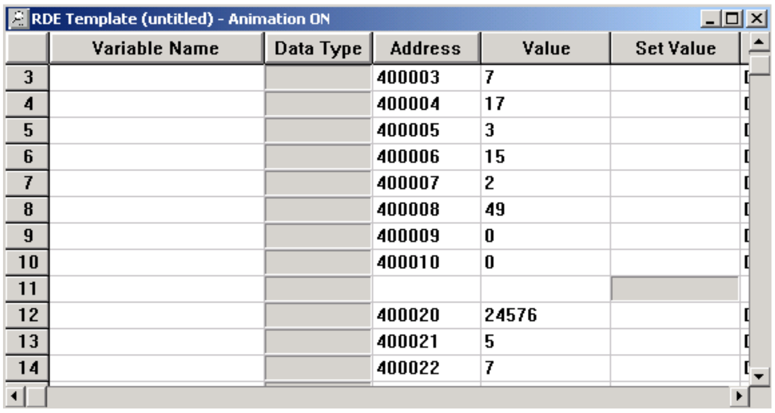
10 Repeat steps 6 through 9, using the values in the following illustration:



Insert Addresses

First Reference To Insert:	400020
Last Reference To Insert:	400029
Number of References to Insert:	10
Display Format:	Dec

11 In the *PLC Configuration* window, open the **ONLINE** menu, and then choose **ANIMATE.** This action opens the *RDE Template* dialog box, with animated values in the *Value* field.



RDE Template (untitled) - Animation ON

	Variable Name	Data Type	Address	Value	Set Value
3			400003	7	
4			400004	17	
5			400005	3	
6			400006	15	
7			400007	2	
8			400008	49	
9			400009	0	
10			400010	0	
11					
12			400020	24576	
13			400021	5	
14			400022	7	

12 Verify that values shown are cycling, starting from address 400065 and up.
13 In the *PLC Configuration* window, open the **TEMPLATES** menu, and then choose **SAVE TEMPLATE AS**. Name the template *ptqclock***,** and then click **OK** to save the template.
14 In the *PLC Configuration* window, open the **ONLINE** menu, and then choose **DISCONNECT.** At the disconnect message, click **YES** to confirm your choice.

At this point, you have successfully

- Created and downloaded a Quantum project to the PLC
- Preset values in data registers that will later be monitored in the ProTalk module.

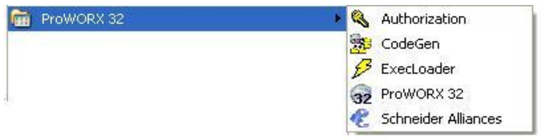
You are now ready to complete the installation and setup of the ProTalk module.

3 Configuring the Processor with ProWORX

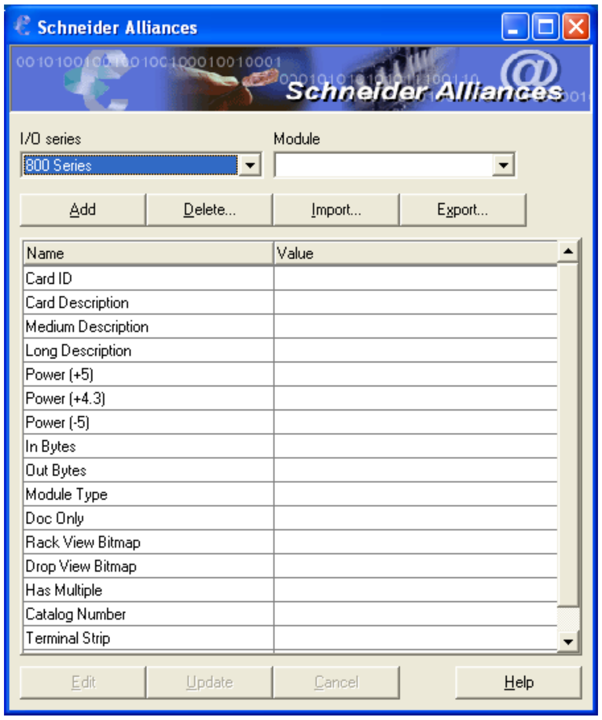
When you use ProWORX 32 software to configure the processor, use the example SAF file provided on the ProTalk Solutions CD-ROM.

Important Note: ProWORX software does not report whether the PTQ module is present in the rack, and therefore is not able to report the health status of the module when the module is online with the Quantum processor. Please consider this when monitoring the status of the PTQ module.

1 Run the **SCHNEIDER_ALLIANCES.EXE** application that is installed with the ProWORX 32 software:



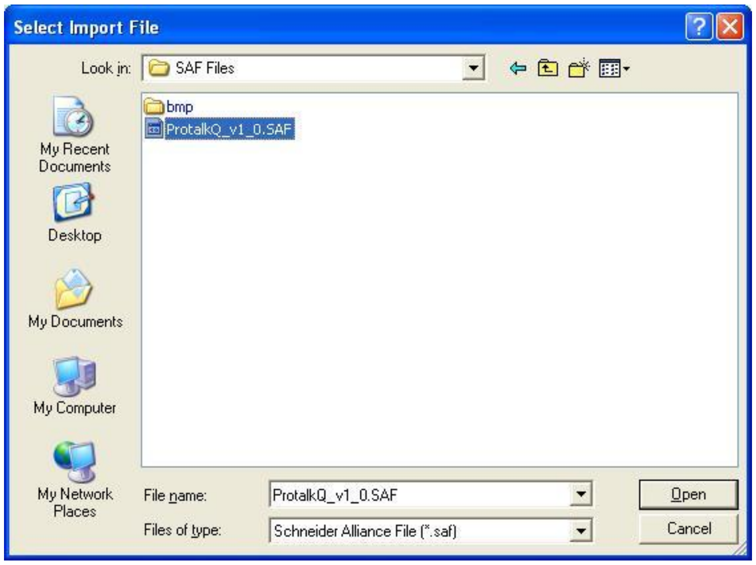
2 Click on **IMPORT…**



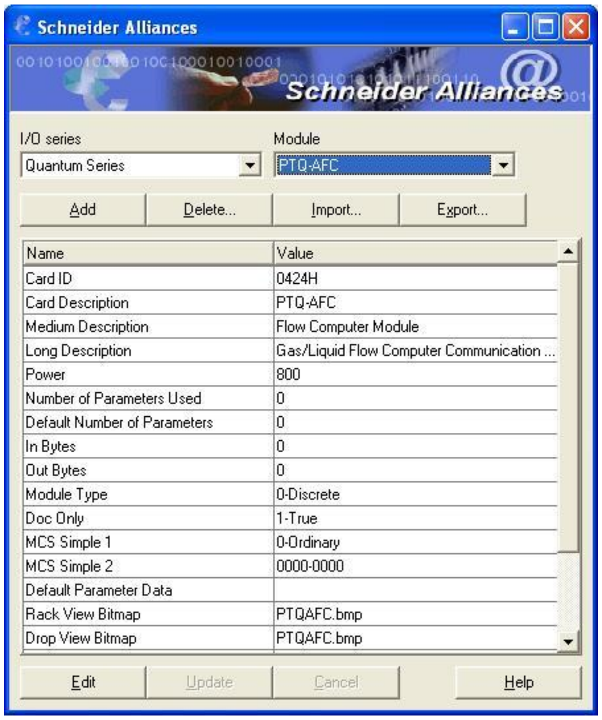
I/O series: 800 Series
Module:

Name	Value
Card ID	
Card Description	
Medium Description	
Long Description	
Power (+5)	
Power (+4.3)	
Power (-5)	
In Bytes	
Out Bytes	
Module Type	
Doc Only	
Rack View Bitmap	
Drop View Bitmap	
Has Multiple	
Catalog Number	
Terminal Strip	

3 Select the *.SAF* File that is located on the CD-ROM shipped with the PTQ module.



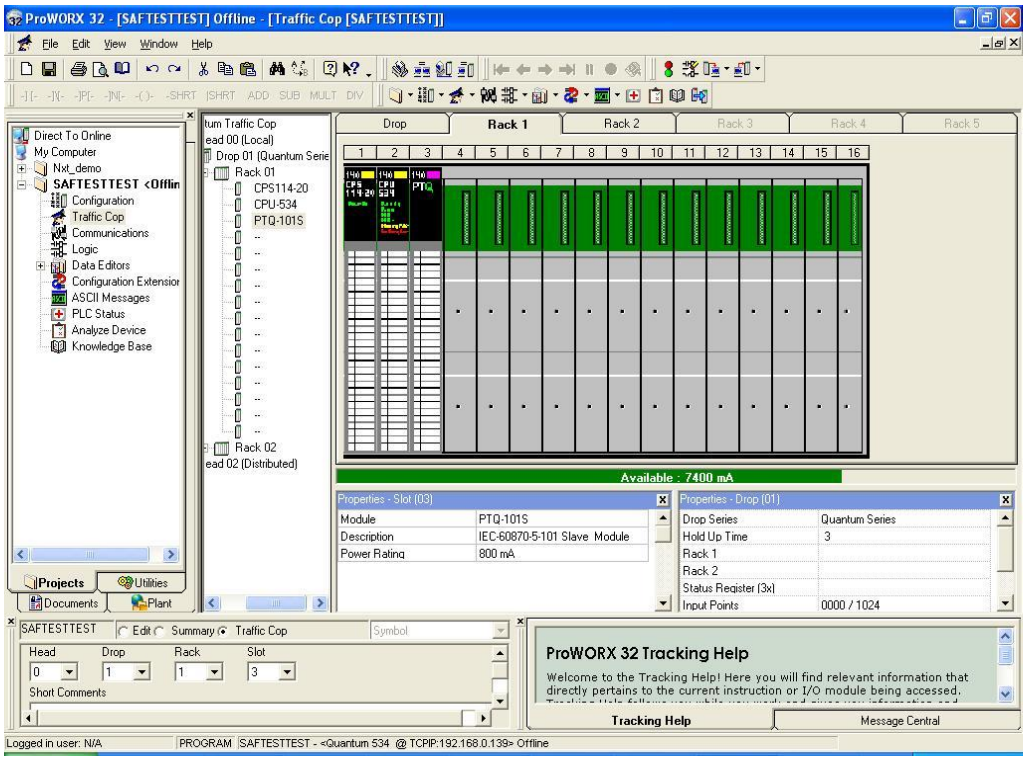
4 After you click on **OPEN** you should see the PTQ modules imported (select **I/O SERIES** as **QUANTUM**):



I/O series: Quantum Series
Module: PTQ-AFC

Name	Value
Card ID	0424H
Card Description	PTQ-AFC
Medium Description	Flow Computer Module
Long Description	Gas/Liquid Flow Computer Communication ...
Power	800
Number of Parameters Used	0
Default Number of Parameters	0
In Bytes	0
Out Bytes	0
Module Type	0-Discrete
Doc Only	1-True
MCS Simple 1	0-Ordinary
MCS Simple 2	0000-0000
Default Parameter Data	
Rack View Bitmap	PTQAFC.bmp
Drop View Bitmap	PTQAFC.bmp

Now you can close the Schneider alliances application and run the ProWORX 32 software. At the *Traffic Cop* section, select the PTQ module to be inserted at the slot:



Properties - Slot (03)	
Module	PTQ-101S
Description	IEC-60870-5-101 Slave Module
Power Rating	800 mA

Properties - Drop (01)	
Drop Series	Quantum Series
Hold Up Time	3
Rack 1	
Rack 2	
Status Register (3x)	
Input Points	0000 / 1024

SAFTESTTEST ( ) Edit ( ) Summary (•) Traffic Cop

Head	Drop	Rack	Slot
0	1	1	3

Short Comments

ProWORX 32 Tracking Help

Welcome to the Tracking Help! Here you will find relevant information that directly pertains to the current instruction or I/O module being accessed.

Logged in user: N/A | PROGRAM | SAFTESTTEST - <Quantum 534 @ TCPIP:192.168.0.139> Offline

4 Configuring the Processor with Unity Pro

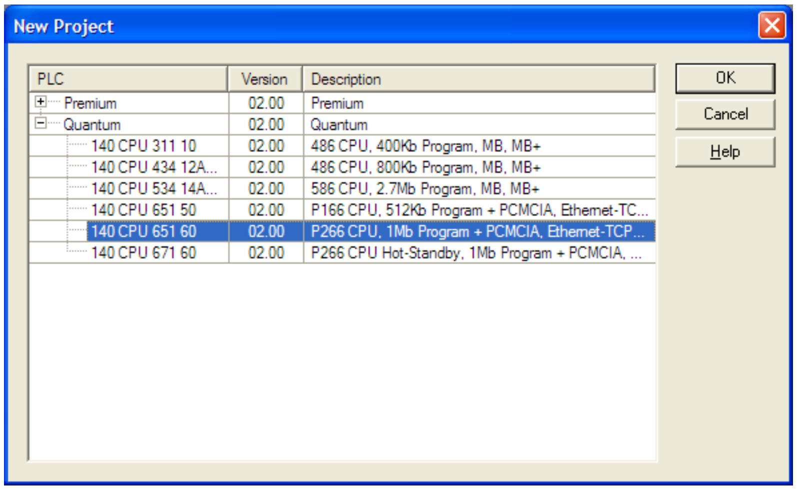
In This Chapter

The following steps are designed to ensure that the processor (Quantum or Unity) is able to transfer data successfully with the PTQ module. As part of this procedure, you will use Unity Pro to create a project, add the PTQ module to the project, set up data memory for the project, and then download the project to the processor.

4.1 Create a New Project

The first step is to open Unity Pro and create a new project.

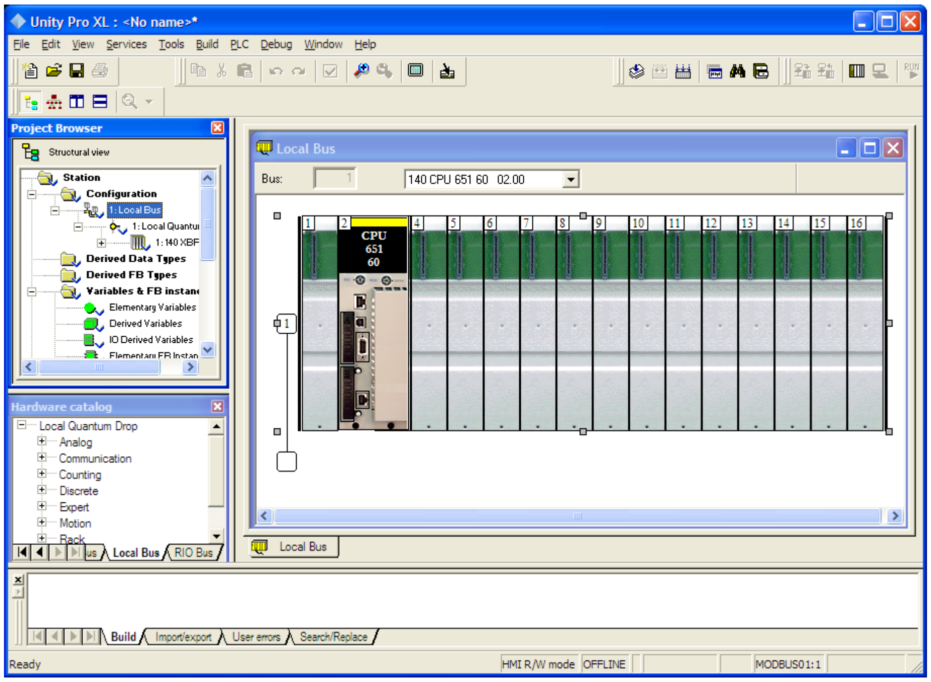
1 In the *New Project* dialog box, choose the CPU type. In the following illustration, the CPU is 140 CPU 651 60. Choose the processor type that matches your own hardware configuration, if it differs from the example. Click **OK** to continue.



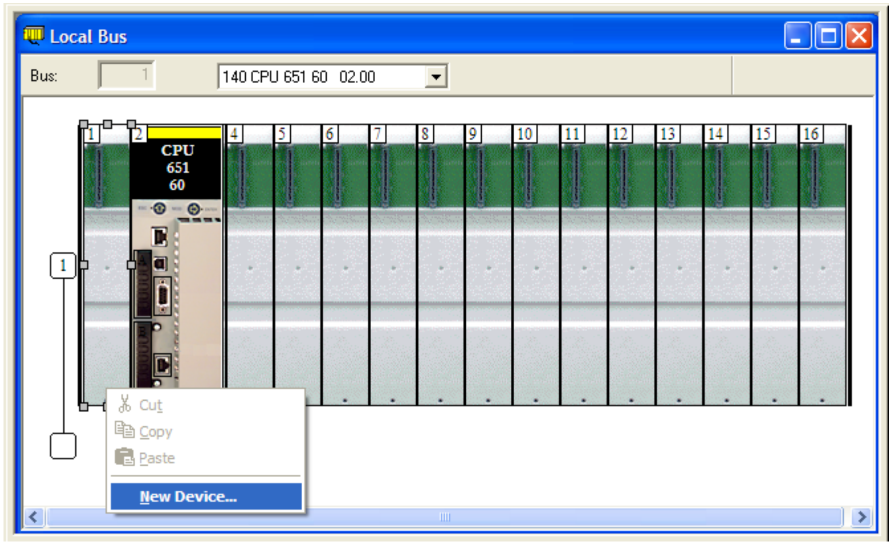
New Project

PLC	Version	Description
⊞ Premium	02.00	Premium
⊟ Quantum	02.00	Quantum
140 CPU 311 10	02.00	486 CPU, 400Kb Program, MB, MB+
140 CPU 434 12A...	02.00	486 CPU, 800Kb Program, MB, MB+
140 CPU 534 14A...	02.00	586 CPU, 2.7Mb Program, MB, MB+
140 CPU 651 50	02.00	P166 CPU, 512Kb Program + PCMCIA, Ethernet-TC...
140 CPU 651 60	02.00	P266 CPU, 1Mb Program + PCMCIA, Ethernet-TCP...
140 CPU 671 60	02.00	P266 CPU Hot-Standby, 1Mb Program + PCMCIA, ...

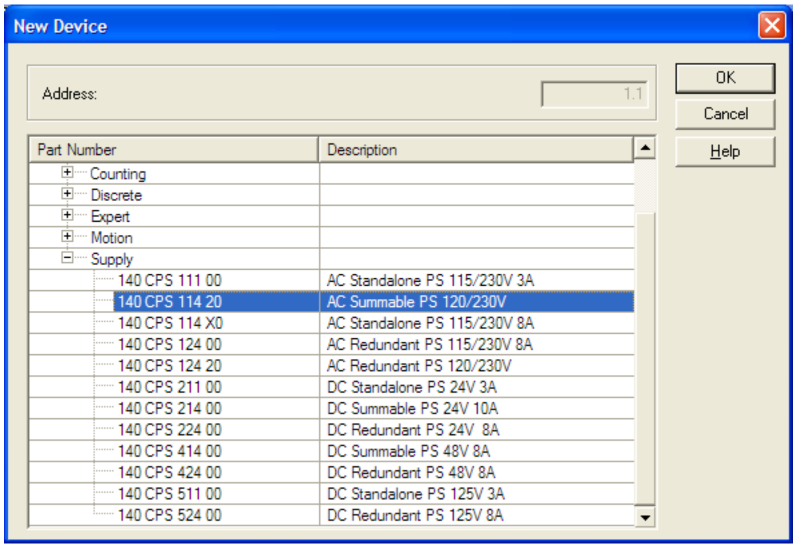
2 Next, add a power supply to the project. In the *Project Browser*, expand the *Configuration* folder, and then double-click the **1:LOCALBUS** icon. This action opens a graphical window showing the arrangement of devices in your Quantum rack.



3 Select the rack position for the power supply, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW DEVICE**.



4 Expand the *Supply* folder, and then select your power supply from the list. Click **OK** to continue.



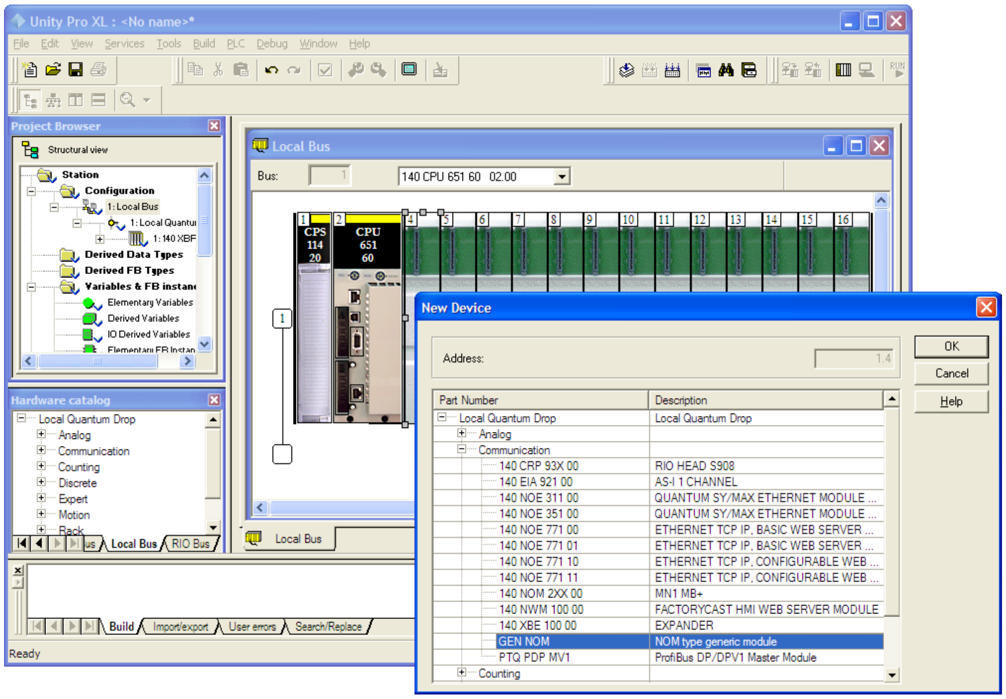
Address: 1.1

Part Number	Description
Counting	
Discrete	
Expert	
Motion	
Supply	
140 CPS 111 00	AC Standalone PS 115/230V 3A
140 CPS 114 20	AC Summable PS 120/230V
140 CPS 114 X0	AC Standalone PS 115/230V 8A
140 CPS 124 00	AC Redundant PS 115/230V 8A
140 CPS 124 20	AC Redundant PS 120/230V
140 CPS 211 00	DC Standalone PS 24V 3A
140 CPS 214 00	DC Summable PS 24V 10A
140 CPS 224 00	DC Redundant PS 24V 8A
140 CPS 414 00	DC Summable PS 48V 8A
140 CPS 424 00	DC Redundant PS 48V 8A
140 CPS 511 00	DC Standalone PS 125V 3A
140 CPS 524 00	DC Redundant PS 125V 8A

5 Repeat these steps to add any additional devices to your Quantum Rack.

4.2 Add the PTQ Module to the Project

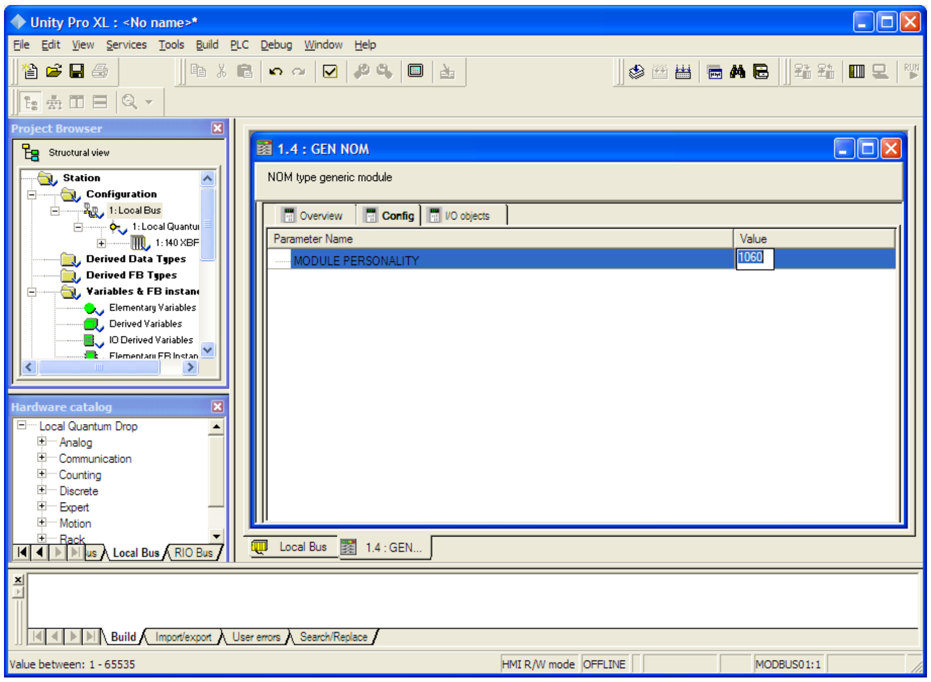
1 Expand the *Communication* tree, and select **GEN NOM**. This module type provides extended communication capabilities for the Quantum system, and allows communication between the PLC and the PTQ module without requiring additional programming.



Address: 1.4

Part Number	Description
Local Quantum Drop	Local Quantum Drop
Analog	
Communication	
140 CRP 93X 00	RIO HEAD S908
140 EIA 921 00	AS-I 1 CHANNEL
140 NOE 311 00	QUANTUM SY/MAX ETHERNET MODULE ...
140 NOE 351 00	QUANTUM SY/MAX ETHERNET MODULE ...
140 NOE 771 00	ETHERNET TCP IP, BASIC WEB SERVER ...
140 NOE 771 01	ETHERNET TCP IP, BASIC WEB SERVER ...
140 NOE 771 10	ETHERNET TCP IP, CONFIGURABLE WEB ...
140 NOE 771 11	ETHERNET TCP IP, CONFIGURABLE WEB ...
140 NOM 2XX 00	MN1 MB+
140 NWM 100 00	FACTORYCAST HMI WEB SERVER MODULE
140 XBE 100 00	EXPANDER
GEN NOM	NOM type generic module
PTQ PDP MV1	ProfiBus DP/DPV1 Master Module
Counting	

2 Next, enter the module personality value. The correct value for ProTalk modules is 1060 decimal (0424 hex).



NOM type generic module

Parameter Name	Value
MODULE PERSONALITY	1060

Value between: 1 - 65535   HMI R/W mode   OFFLINE   MODBUS01:1

3 Before you can save the project in Unity Pro, you must validate the modifications. Open the **EDIT** menu, and then choose **VALIDATE.** If no errors are reported, you can save the project.

4 **SAVE** the project.

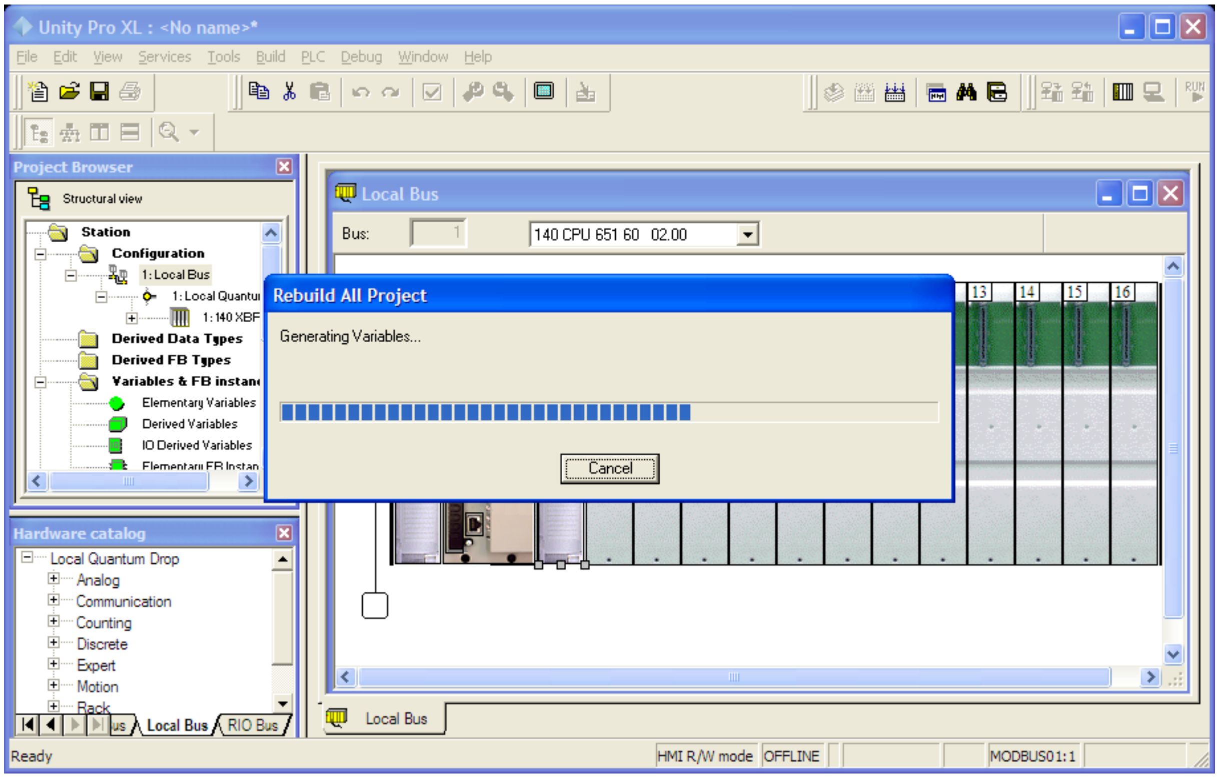
4.3 Build the Project

Whenever you update the configuration of your PTQ module or the processor, you must import the changed configuration from the module, and then build (compile) the project before downloading it to the processor.

Note: The following steps show you how to build the project in Unity Pro. This is not intended to provide detailed information on using Unity Pro, or debugging your programs. Refer to the documentation for your processor and for Unity Pro for specialized information.

To build (compile) the project:

1 Review the elements of the project in the *Project Browser*.
2 When you are satisfied that you are ready to download the project, open the **BUILD** menu, and then choose **REBUILD ALL PROJECT**. This action builds (compiles) the project into a form that the processor can use to execute the instructions in the project file. This task may take several minutes, depending on the complexity of the project and the resources available on your PC.
3 As the project is built, Unity Pro reports its process in a *Progress* dialog box, with details appearing in a pane at the bottom of the window. The following illustration shows the build process under way.



After the build process is completed successfully, the next step is to download the compiled project to the processor.

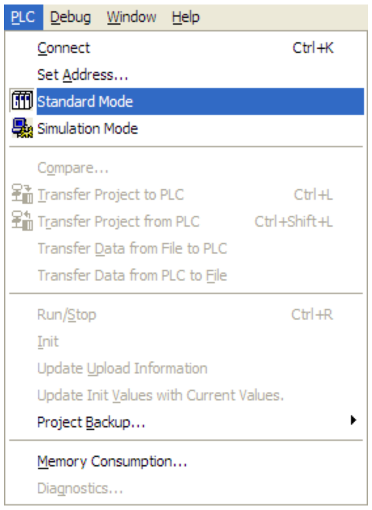
4.4 Connect Your PC to the Processor

The next step is to connect to the processor so that you can download the project file. The processor uses this project file to communicate over the backplane to modules identified in the project file.

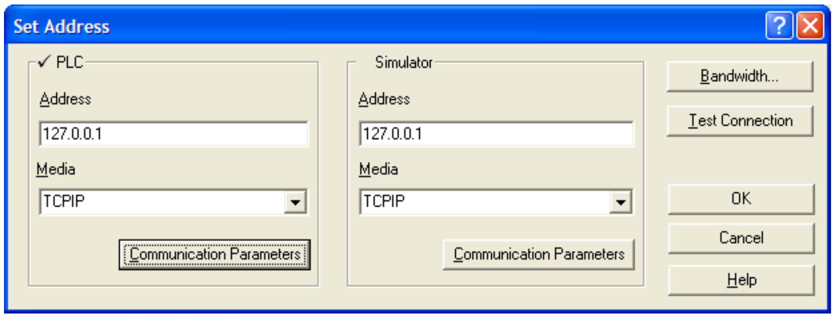
Note: If you have never connected from the PC to your processor before, you must verify that the necessary port drivers are installed and available to Unity Pro.

To verify address and driver settings in Unity Pro:

1 Open the **PLC** menu, and choose **STANDARD MODE**. This action turns off the PLC Simulator, and allows you to communicate directly with the Quantum or Unity hardware.


PLC
Debug
Window
Help
Connect Ctrl+K
Set Address...
Standard Mode
Simulation Mode
Compare...
Transfer Project to PLC Ctrl+L
Transfer Project from PLC Ctrl+Shift+L
Transfer Data from File to PLC
Transfer Data from PLC to File
Run/Stop Ctrl+R
Init
Update Upload Information
Update Init Values with Current Values.
Project Backup...
Memory Consumption...
Diagnostics...

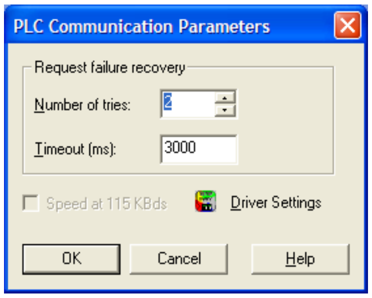
2 Open the **PLC** menu, and choose **SET ADDRESS...** This action opens the *Set Address* dialog box. Open the **MEDIA** dropdown list and choose the connection type to use (TCPIP or USB).


Set Address

✓ PLC
Address: 127.0.0.1
Media: TCPIP

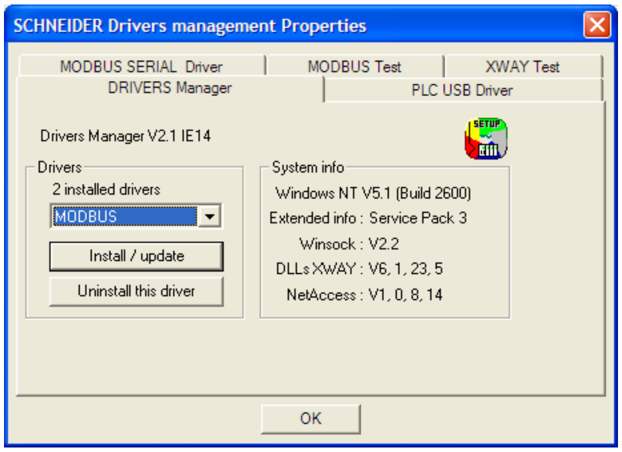
Simulator
Address: 127.0.0.1
Media: TCPIP

3 If the **MEDIA** dropdown list does not contain the connection method you wish to use, click the **COMMUNICATION PARAMETERS** button in the PLC area of the dialog box. This action opens the *PLC Communication Parameters* dialog box.


PLC Communication Parameters

Request failure recovery
Number of tries: 2
Timeout (ms): 3000
Speed at 115 KBds

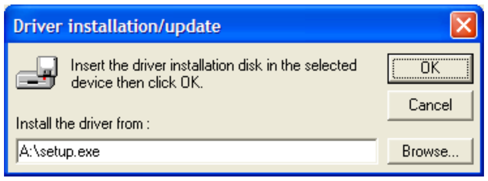
4 Click the **DRIVER SETTINGS** button to open the *SCHNEIDER Drivers management Properties* dialog box.


SCHNEIDER Drivers management Properties

Drivers Manager V2.1 IE14
Drivers
2 installed drivers
MODBUS

System info
Windows NT V5.1 (Build 2600)
Extended info : Service Pack 3
Winsock : V2.2
DLLs XWAY : V6, 1, 23, 5
NetAccess : V1, 0, 8, 14

5 Click the **INSTALL/UPDATE** button to specify the location of the Setup.exe file containing the drivers to use. You will need your Unity Pro installation disks for this step.


Driver installation/update

Insert the driver installation disk in the selected device then click OK.
Install the driver from :
A:\setup.exe

6 Click the **BROWSE** button to locate the *Setup.exe* file to execute, and then execute the setup program. After the installation, restart your PC if you are prompted to do so. Refer to your Schneider Electric documentation for more information on installing drivers for Unity Pro.

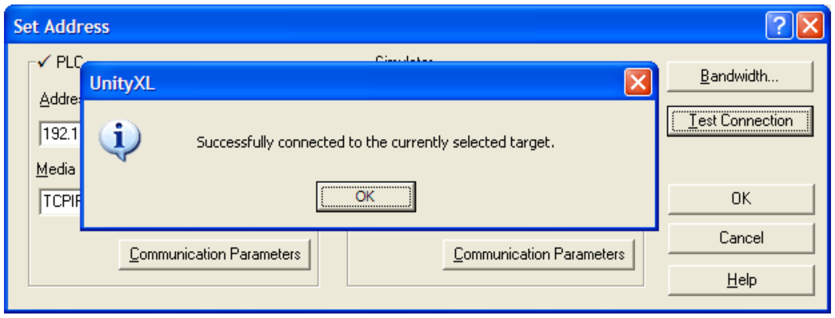
4.4.1 Connecting to the Processor with TCPIP

The next step is to download (copy) the project file to the processor. The following steps demonstrate how to use an Ethernet cable connected from the Processor to your PC through an Ethernet hub or switch. Other connection methods may also be available, depending on the hardware configuration of your processor, and the communication drivers installed in Unity Pro.

1 If you have not already done so, connect your PC and the processor to an Ethernet hub.
2 Open the **PLC** menu, and then choose **SET ADDRESS**.

- **Important:** Notice that the *Set Address* dialog box is divided into two areas. Enter the address and media type in the **PLC** area of the dialog box, not the **SIMULATOR** area.

3 Enter the IP address in the address field. In the **MEDIA** dropdown list, choose TCPIP.
4 Click the **TEST CONNECTION** button to verify that your settings are correct.

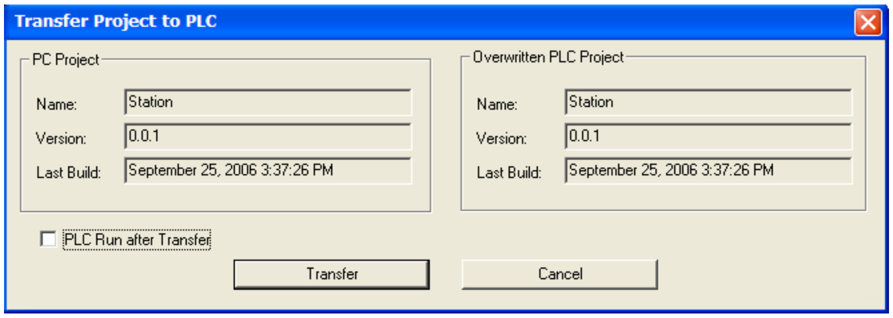


Successfully connected to the currently selected target.

4.5 Download the Project to the Quantum Processor

1 Open the **PLC** menu and then choose **CONNECT.** This action opens a connection between the Unity Pro software and the processor, using the address and media type settings you configured in the previous step.

2 On the **PLC** menu, choose **TRANSFER PROJECT TO PLC**. This action opens the **TRANSFER PROJECT TO PLC** dialog box. If you would like the PLC to go to "Run" mode immediately after the transfer is complete, select (check) the **PLC RUN AFTER TRANSFER** check box.



PC Project
Name: Station
Version: 0.0.1
Last Build: September 25, 2006 3:37:26 PM

Overwritten PLC Project
Name: Station
Version: 0.0.1
Last Build: September 25, 2006 3:37:26 PM

☐ PLC Run after Transfer

3 Click the **TRANSFER** button to download the project to the processor. As the project is transferred, Unity Pro reports its process in a **PROGRESS** dialog box, with details appearing in a pane at the bottom of the window.

When the transfer is complete, place the processor in Run mode. The processor will start scanning your process logic application.

5 Setting Up the ProTalk Module

In This Chapter

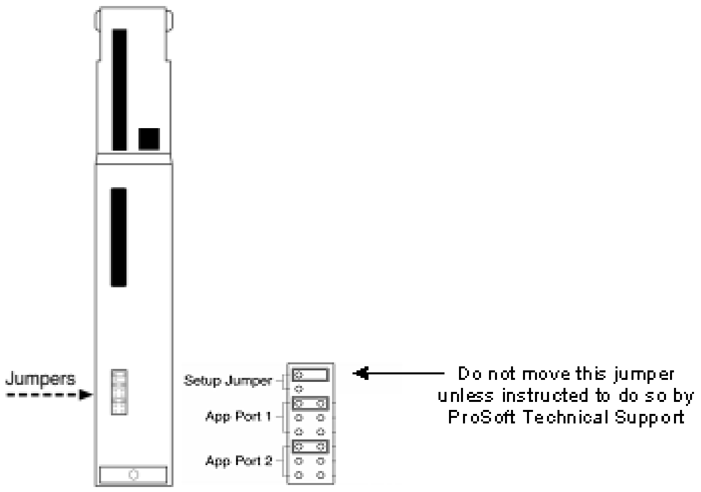
After you complete the following procedures, the ProTalk module will actively be transferring data bi-directionally with the processor.

5.1 Install the ProTalk Module in the Quantum Rack

5.1.1 Verify Jumper Settings

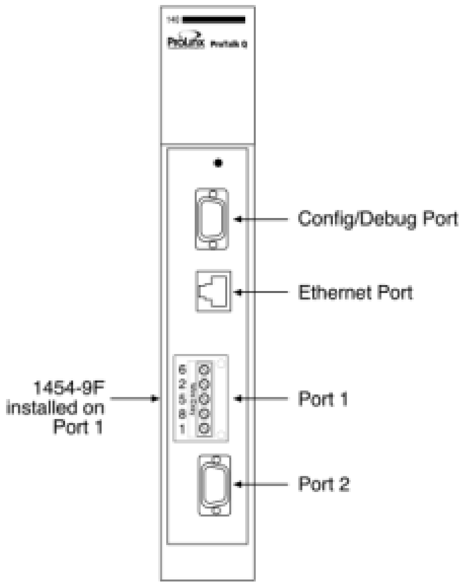
ProTalk modules are configured for RS-232 serial communications by default. To use RS-422 or RS-485, you must change the jumpers.

The jumpers are located on the back of the module as shown in the following illustration:


Jumpers
Setup Jumper
App Port 1
App Port 2
Do not move this jumper unless instructed to do so by ProSoft Technical Support

5.1.2 Inserting the 1454-9F connector

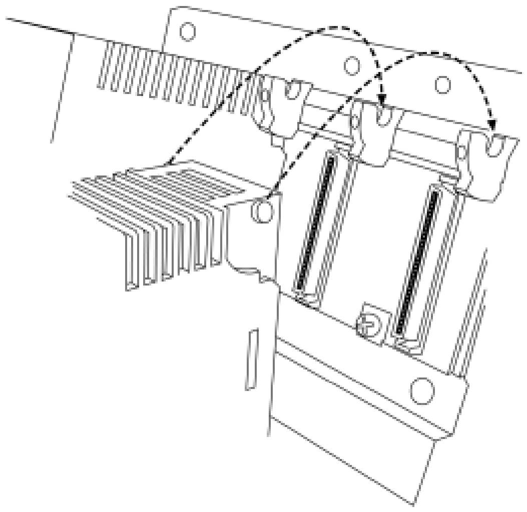
Insert the 1454-9F connector as shown. Wiring locations are shown in the table:



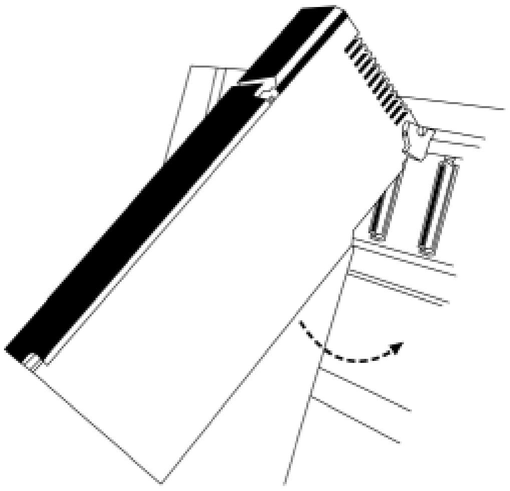
5.1.3 Install the ProTalk Module in the Quantum Rack

1 Place the Module in the Quantum Rack. The ProTalk module must be placed in the same rack as the processor.

2 Tilt the module at a 45° angle and align the pegs at the top of the module with slots on the backplane.



3 Push the module into place until it seats firmly in the backplane.



Caution: The PTQ module is hot-swappable, meaning that you can install and remove it while the rack is powered up. You should not assume that this is the case for all types of modules unless the user manual for the product explicitly states that the module is hot-swappable. Failure to observe this precaution could result in damage to the module and any equipment connected to it.

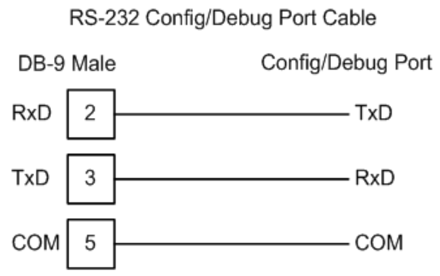
5.1.4 Cable Connections

The application ports on the PTQ-ADM module support RS-232, RS-422, and RS-485 interfaces. Please inspect the module to ensure that the jumpers are set correctly to correspond with the type of interface you are using.

Note: When using RS-232 with radio modem applications, some radios or modems require hardware handshaking (control and monitoring of modem signal lines). Enable this in the configuration of the module by setting the UseCTS parameter to 1.

RS-232 Configuration/Debug Port

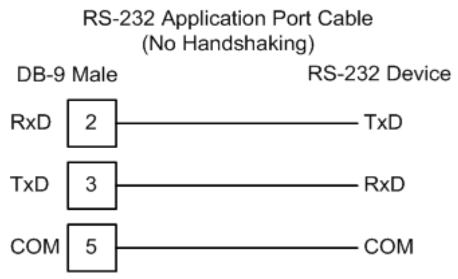
This port is physically a DB-9 connection. This port permits a PC based terminal emulation program to view configuration and status data in the module and to control the module. The cable for communications on this port is shown in the following diagram:



The Ethernet port on this module (if present) is inactive.

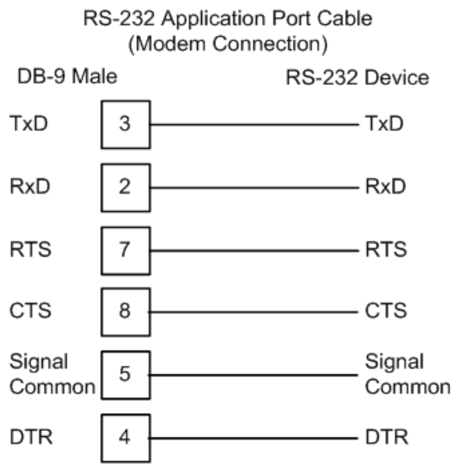
RS-232 Application Port(s)

When the RS-232 interface is selected, the use of hardware handshaking (control and monitoring of modem signal lines) is user definable. If no hardware handshaking will be used, the cable to connect to the port is as shown below:



RS-232: Modem Connection

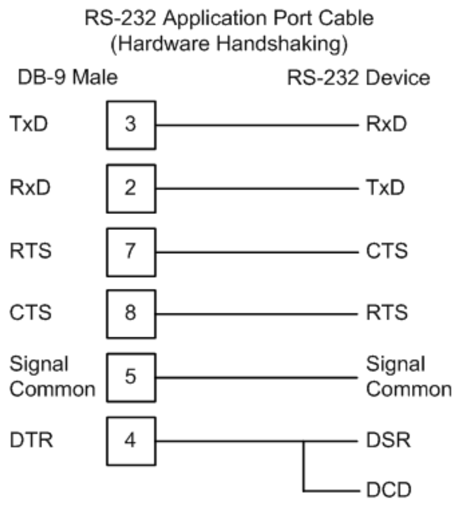
This type of connection is required between the module and a modem or other communication device.



The "Use CTS Line" parameter for the port configuration should be set to 'Y' for most modem applications.

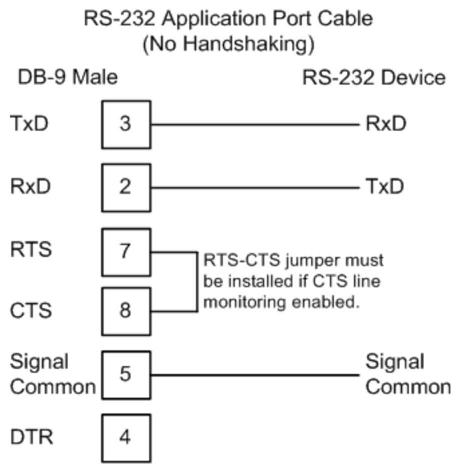
RS-232: Null Modem Connection (Hardware Handshaking)

This type of connection is used when the device connected to the module requires hardware handshaking (control and monitoring of modem signal lines).



RS-232: Null Modem Connection (No Hardware Handshaking)

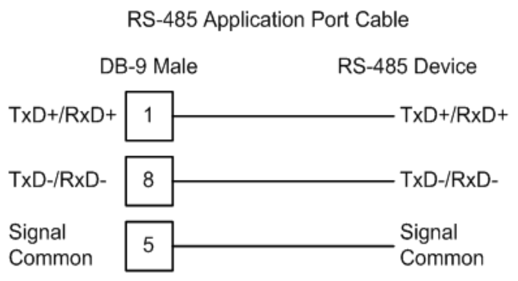
This type of connection can be used to connect the module to a computer or field device communication port.


RS-232 Application Port Cable
(No Handshaking)
DB-9 Male
RS-232 Device
TxD
3
RxD
RxD
2
TxD
RTS
7
CTS
8
RTS-CTS jumper must be installed if CTS line monitoring enabled.
Signal Common
5
Signal Common
DTR
4

Note: If the port is configured with the "Use CTS Line" set to 'Y', then a jumper is required between the RTS and the CTS line on the module connection.

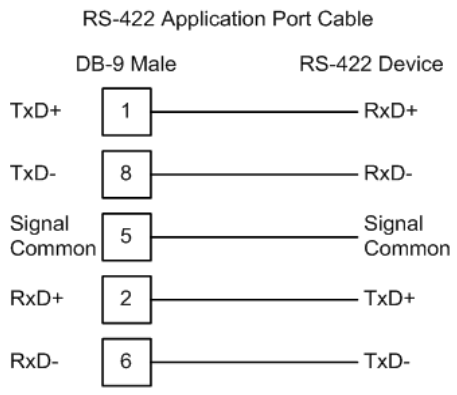
RS-485 Application Port(s)

The RS-485 interface requires a single two or three wire cable. The Common connection is optional and dependent on the RS-485 network. The cable required for this interface is shown below:



Note: Terminating resistors are generally not required on the RS-485 network, unless you are experiencing communication problems that can be attributed to signal echoes or reflections. In these cases, installing a 120-ohm terminating resistor between pins 1 and 8 on the module connector end of the RS-485 line may improve communication quality.

RS-422



RS-485 and RS-422 Tip

If communication in the RS-422/RS-485 mode does not work at first, despite all attempts, try switching termination polarities. Some manufacturers interpret +/- and A/B polarities differently.

6 Introduction to PTQ-ADM

In This Chapter

This document provides information needed for the development of application programs for the PTQ-ADM Serial Communication Module. The PTQ suite of modules is designed to allow devices with a serial port to be accessed by a Quantum PLC. The ProTalk module is the platform used.

ProTalk modules are programmable to accommodate devices with unique serial protocols. Included in this document is information about the available software API libraries and tools, module configuration and programming information, and example code for the module. For the Quantum PLC, refer to ProTalk Setup Guide Phase 1 and 2 for more information. This document assumes the reader is familiar with software development in the 16-bit DOS environment using the 'C' programming language. This document also assumes that the reader is familiar with Quantum PLC platform.

6.1 Operating System

The PTQ module includes General Software Embedded DOS 6-XL. This operating system provides DOS compatibility along with real-time multi-tasking functionality. The operating system is stored in Flash ROM and is loaded by the BIOS when the module boots.

DOS compatibility allows user applications to be developed using standard DOS tools, such as Digital Mars C++ and Borland compilers. User programs may be executed automatically by loading them from either the CONFIG.SYS file or an AUTOEXEC.BAT file.

Note: DOS programs that try to access the video or keyboard hardware directly will not function correctly on the PTQ module. Only programs that use the standard DOS and BIOS functions to perform console I/O are compatible.

Refer to the **General Software Embedded DOS 6-XL Developer’s Guide (page 227)** on the PTQ-ADM CD-ROM for more information.

7 Understanding the PTQ-ADM API

In This Chapter

The PTQ-ADM API Suite allows software developers to access the PLC backplane and serial ports without needing detailed knowledge of the module’s hardware design. The PTQ-ADM API Suite consists of three distinct components: the Serial Port API, the Backplane/CIP API and the ADM API. The Backplane API provides access to the PLC, the Serial Port API provides access to the serial ports and the ADM API provides functions designed to ease development.

Applications for the PTQ-ADM module may be developed using industry-standard DOS programming tools and the appropriate API components.

This section provides general information pertaining to application development for the PTQ-ADM module.

7.1 API Libraries

Each API provides a library of function calls. The library supports any programming language that is compatible with the Pascal calling convention.

Each API library is a static object code library that must be linked with the application to create the executable program. It is distributed as a 16-bit large model OMF library, compatible with Digital Mars or Borland development tools.

Note: The following compiler versions are intended to be compatible with the PTQ module API:
Digital Mars C++ 8.49
Borland C++ V5.02
More compilers will be added to the list as the API is tested for compatibility with them.

7.1.1 Calling Convention

The API library functions are specified using the 'C' programming language syntax. To allow applications to be developed in other industry-standard programming languages, the standard Pascal calling convention is used for all application interface functions.

7.1.2 Header File

A header file is provided along with each library. This header file contains API function declarations, data structure definitions, and miscellaneous constant definitions. The header file is in standard 'C' format.

7.1.3 Multithreading Considerations

The DOS 6-XL operating system supports the development of multi-threaded applications.

Note: The multi-threading library *kernel.lib* in the DOS folder on the distribution CD-ROM is compiler-specific to Borland C++ 5.02. It is *not* compatible with Digital Mars C++ 8.49. ProSoft Technology, Inc. does not support multi-threading with Digital Mars C++ 8.49.

Note: The ADM DOS 6-XL operating system has a system tick of 5 milliseconds. Therefore, thread scheduling and timer servicing occur at 5ms intervals. Refer to the *DOS 6-XL Developer's Guide* on the distribution CD-ROM for more information.

Multi-threading is also supported by the API.

- *DOS* and *cipapi* libraries have been tested and are thread-safe for use in multi-threaded applications.
- *MVIbp* and *MVIsp* libraries are safe to use in multi-threaded applications with the following precautions: If you call the same *MVIbp* or *MVIsp* function from multiple threads, you will need to protect it, to prevent task switches during the function's execution. The same is true for different *MVIbp* or *MVIsp* functions that share the same resources (for example, two different functions that access the same read or write buffer).

WARNING: *ADM* and *ADMNET* libraries are *not* thread-safe. ProSoft Technology, Inc. does not support the use of *ADM* and *ADMNET* libraries in multi-threaded applications.

7.2 Development Tools

An application that is developed for the PTQ-ADM module must be stored on the module’s Flash ROM disk to be executed. Tools are provided with the API to build the disk image and download it to the module via the programming port PRT1.

7.3 Theory of Operation

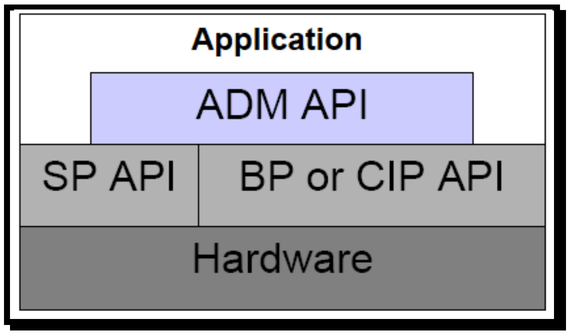
7.3.1 ADM API

The ADM API is one component of the PTQ-ADM API Suite. The ADM API provides a simple module level interface for the ProLinx, MVI and PTQ Families. This is useful when developing an application that implements a serial protocol for a particular device, such as a scale or bar code reader. After the application has been developed, it can be used on any of the PTQ family modules.

7.3.2 ADM API Architecture

The ADM API is composed of a statically-linked library (called the ADM library). Applications using the ADM API must be linked with the ADM library. The ADM API encapsulates the hardware, making it possible to design PTQ applications that can be run on any of the PTQ family of modules.

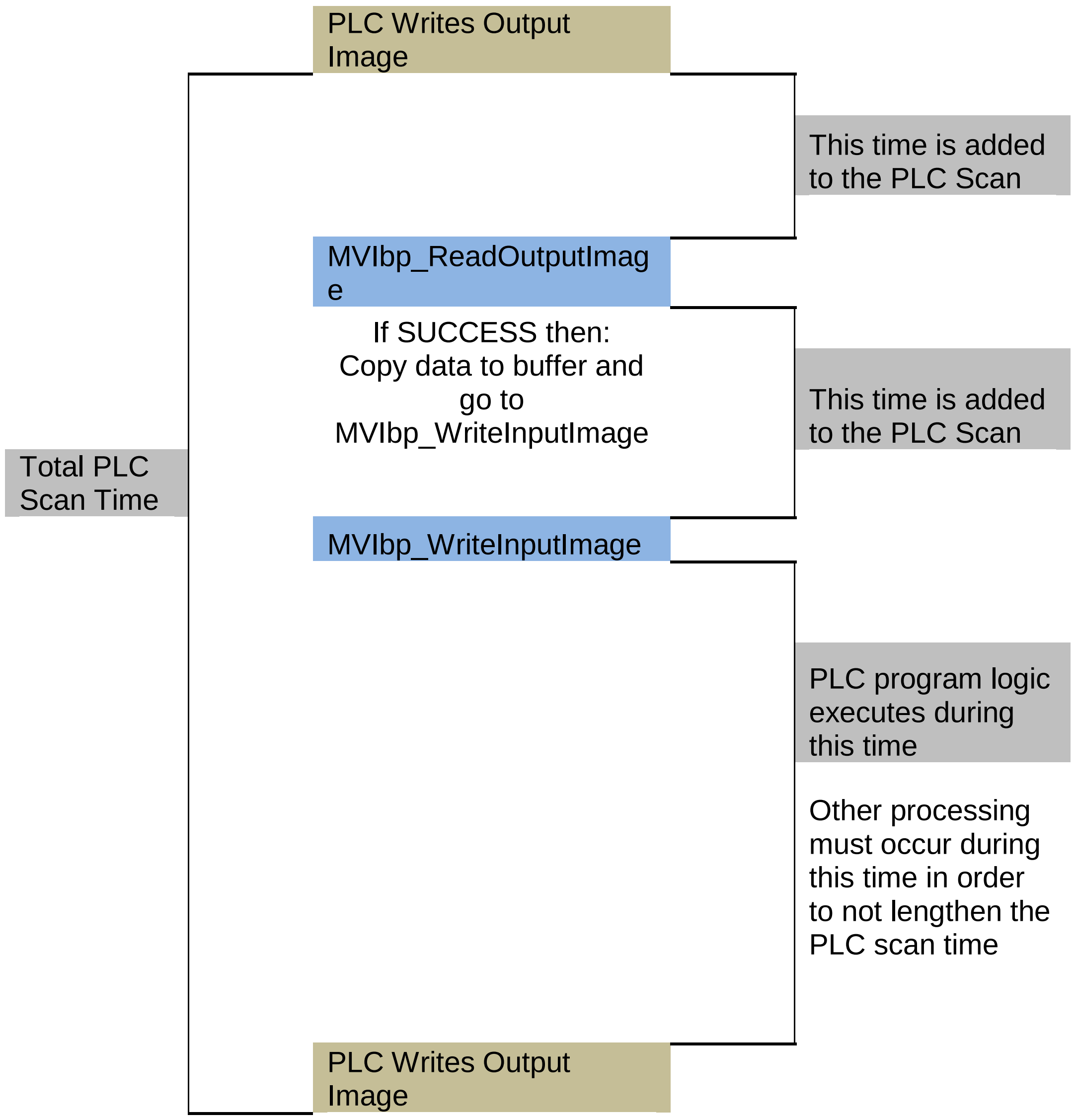
The following illustration shows the ADM API architecture:



7.3.3 PTQ Big I/O Backplane Model Theory of Operation

When the PLC has data to write to the PTQ module it will write to the backplane and pass the lock to the PTQ module. The module program must call MVIbp_ReadOutputImage to see if data is available for reading. If data is available the function will return MVI_SUCCESS. If not, it will return MVI_TIMEOUT. The call to MVIbp_ReadOutputImage should be called often until MVI_SUCCESS is returned. As soon as MVI_SUCCESS is returned, action should be taken on the data. Once this is completed, a call to MVIbp_WriteInputImage should be made.

The lock is not returned to the PLC until the call to MVIbp_WriteInputImage is made. The program time between a successful MVIbp_ReadOutputImage and the call to MVIbp_WriteInputImage is added to the PLC scan time. It is recommended to keep this time to a minimum to avoid unduly lengthening the PLC scan time.


PLC Writes Output Image
This time is added to the PLC Scan
MVIbp_ReadOutputImage
If SUCCESS then: Copy data to buffer and go to MVIbp_WriteInputImage
This time is added to the PLC Scan
Total PLC Scan Time
MVIbp_WriteInputImage
PLC program logic executes during this time
Other processing must occur during this time in order to not lengthen the PLC scan time
PLC Writes Output Image

7.4 Database

The database functions of the ADM API allow the creation of a database in memory to store data to be accessed via the backplane interface and the application ports. The database consists of word registers that can be accessed as bits, bytes, words, longs, floats or doubles. Functions are provided for reading and writing the data in the various data types. The database serves as a holding area for exchanging data with the processor on the backplane, and with a foreign device attached to the application port. Data transferred into the module from the processor can be requested via the serial port. Conversely data written into the module database by the foreign device can be transferred to the processor over the backplane.

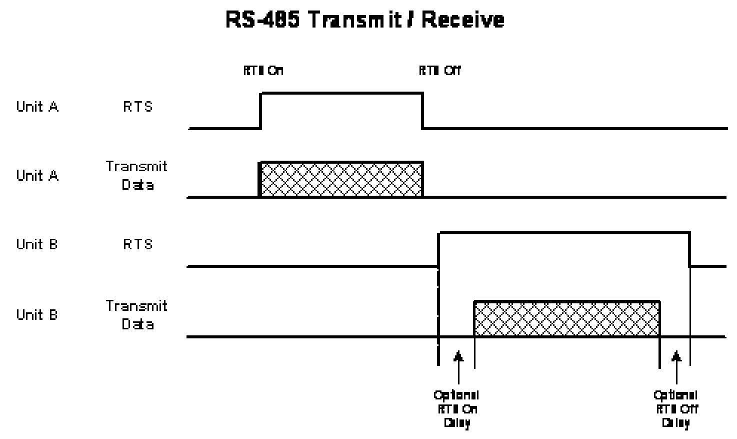
7.5 RS-485 Programming Note

7.5.1 Hardware

The serial port has two driver chips, one for RS-232 and one for RS-422/485. The Request To Send (RTS) line is used for hardware handshaking in RS-232 and to control the transmitter in RS-422/485.

In RS-485, only one node can transmit at a time. All nodes should default to listening (RTS off) unless transmitting. If a node has its RTS line asserted, then all other communication is blocked. An analogy for this is a 2-way radio system where only one person can speak at a time. If someone holds the talk button, then they cannot hear others transmitting.

In order to have orderly communication, a node must make sure no other nodes are transmitting before beginning a transmission. The node needing to transmit will assert the RTS line then transmit the message. The RTS line must be de-asserted as soon as the last character is transmitted. Turning RTS on late or off early will cause the beginning or end of the message to be clipped resulting in a communication error. In some applications it may be necessary to delay between RTS transitions and the message. In this case RTS would be asserted, wait for delay time, transmit message, wait for delay time, and de-assert RTS.



7.5.2 Software

The following is a code sample designed to illustrate the steps required to transmit in RS-485. Depending on the application, it may be necessary to handle other processes during this transmit sequence and to not block. This is simplified to demonstrate the steps required.

```
int length = 10;    // send 10 characters
int CharsLeft;
BYTE buffer[10];
// Set RTS on
MVIsp_SetRTS(COM2, ON);
// Optional delay here (depends on application)
// Transmit message
MVIsp_PutData(COM2, buffer, &length, TIMEOUT_ASAP);
// Check to see that message is done
MVIsp_GetCountUnsent(COM2, &CharsLeft);
// Keep checking until all characters sent
while(CharsLeft)
{
MVIsp_GetCountUnsent(COM2, &CharsLeft);
}
// Optional delay here (depends on application)
// Set RTS off
MVIsp_SetRTS(COM2, OFF);
```

8 Setting Up Your Development Environment

In This Chapter

8.1 Setting Up Your Compiler

There are some important compiler settings that must be set in order to successfully compile an application for the PTQ platforms. The following topics describe the setup procedures for each of the supported compilers.

8.1.1 Configuring Digital Mars C++ 8.49

The following procedure allows you to successfully build the sample ADM code supplied by ProSoft Technology using Digital Mars C++ 8.49. After verifying that the sample code can be successfully compiled and built, you can modify the sample code to work with your application.

Note: This procedure assumes that you have successfully installed Digital Mars C++ 8.49 on your workstation.

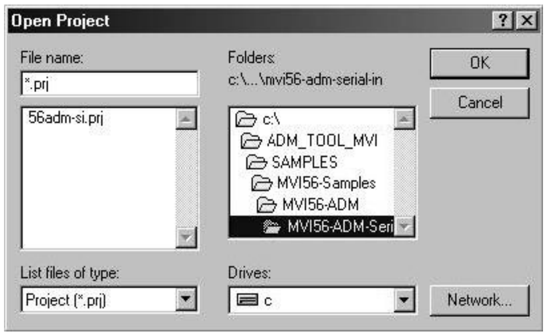
Downloading the Sample Program

The sample code files are located in the ADM_TOOL_PTQ.ZIP file. This zip file is available from the CD-ROM shipped with your system or from the www.prosoft-technology.com web site. When you unzip the file, you will find the sample code files in \ADM_TOOL_PTQ\SAMPLES\.

Important: The sample code and libraries in the 1756-MVI-Samples folder are not compatible with, and are not supported for, the Digital Mars compiler.

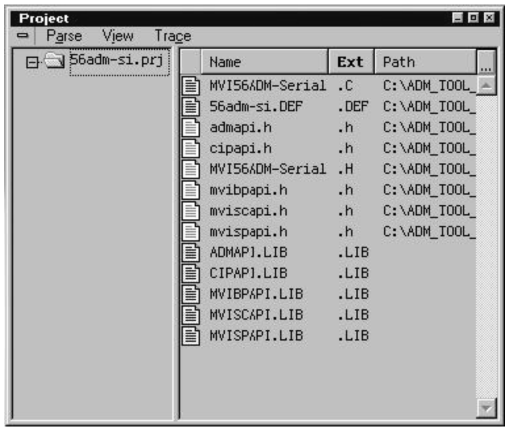
Building an Existing Digital Mars C++ 8.49 ADM Project

1 Start Digital Mars C++ 8.49, and then click **Project** → **Open** from the *Main Menu*.



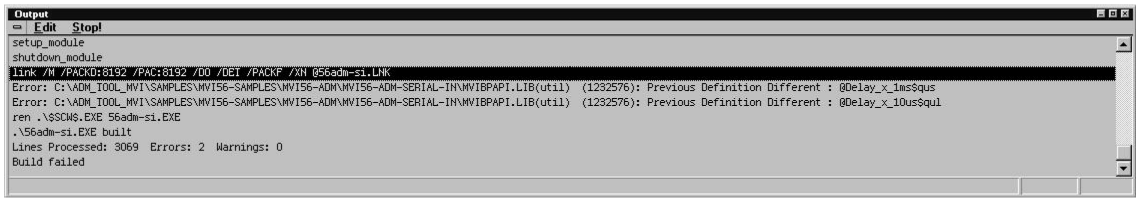
2 From the *Folders* field, navigate to the folder that contains the project (C:\ADM_TOOL_PTQ\SAMPLES\…).
3 In the *File Name* field, click on the project name (56adm-si.prj).

4 Click **OK**. The *Project* window appears:



Name	Ext	Path
MVI56ADM-Serial	.C	C:\ADM_TOOL_
56adm-si.DEF	.DEF	C:\ADM_TOOL_
admapi.h	.h	C:\ADM_TOOL_
cipapi.h	.h	C:\ADM_TOOL_
MVI56ADM-Serial	.H	C:\ADM_TOOL_
mvibpapi.h	.h	C:\ADM_TOOL_
mviscapi.h	.h	C:\ADM_TOOL_
mvispapi.h	.h	C:\ADM_TOOL_
ADMAPI.LIB	.LIB	
CIPAPI.LIB	.LIB	
MVIBPAPI.LIB	.LIB	
MVISCAPI.LIB	.LIB	
MVISPAPI.LIB	.LIB	

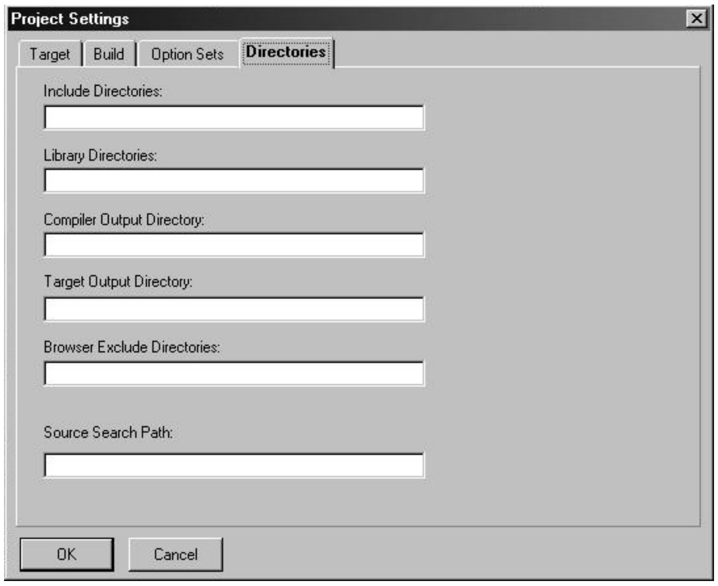
5 Click **Project** → **Rebuild All** from the *Main Menu* to create the .exe file. The status of the build will appear in the Output window:



```
setup_module
shutdown_module
link /M /PACKD:8192 /PAC:8192 /DO /DEI /PACKF /XN @56adm-si.LNK
Error: C:\ADM_TOOL_MVI\SAMPLES\MVI56-SAMPLES\MVI56-ADM\MVI56-ADM-SERIAL-IN\MVIBPAPI.LIB(util)  (1232576): Previous Definition Different : @Delay_x_1ms$qus
Error: C:\ADM_TOOL_MVI\SAMPLES\MVI56-SAMPLES\MVI56-ADM\MVI56-ADM-SERIAL-IN\MVIBPAPI.LIB(util)  (1232576): Previous Definition Different : @Delay_x_10us$qul
ren .\$SCW$.EXE 56adm-si.EXE
.\56adm-si.EXE built
Lines Processed: 3069  Errors: 2  Warnings: 0
Build failed
```

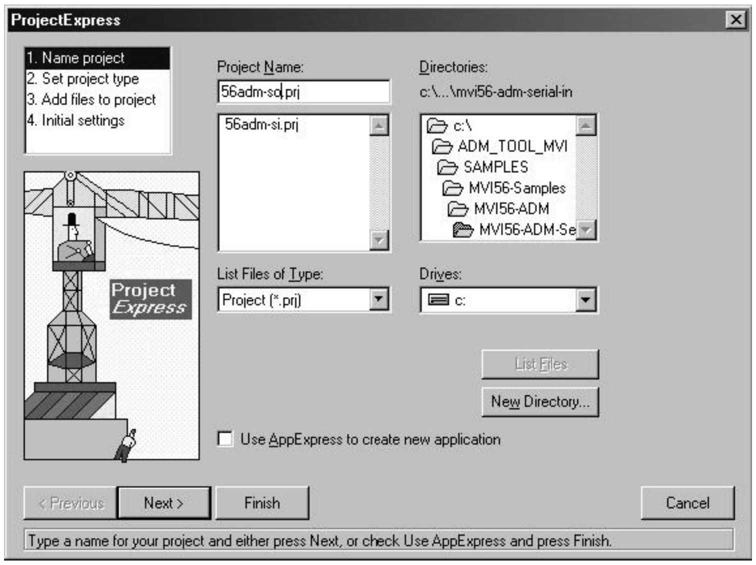
Porting Notes: *The Digital Mars compiler classifies duplicate library names as Level 1 Errors rather than warnings. These errors will manifest themselves as "Previous Definition Different: function name". Level 1 errors are non-fatal and the executable will build and run. The architecture of the ADM libraries will cause two or more of these errors to appear when the executable is built. This is a normal occurrence. If you are building existing code written for a different compiler you may have to replace calls to run-time functions with the Digital Mars equivalent. Refer to the Digital Mars documentation on the Run-time Library for the functions available.*

6 The executable file will be located in the directory listed in the Compiler Output Directory field. If it is blank then the executable file will be located in the same folder as the project file. The *Project Settings* window can be accessed by clicking **Project** → **Settings** from the *Main Menu*.



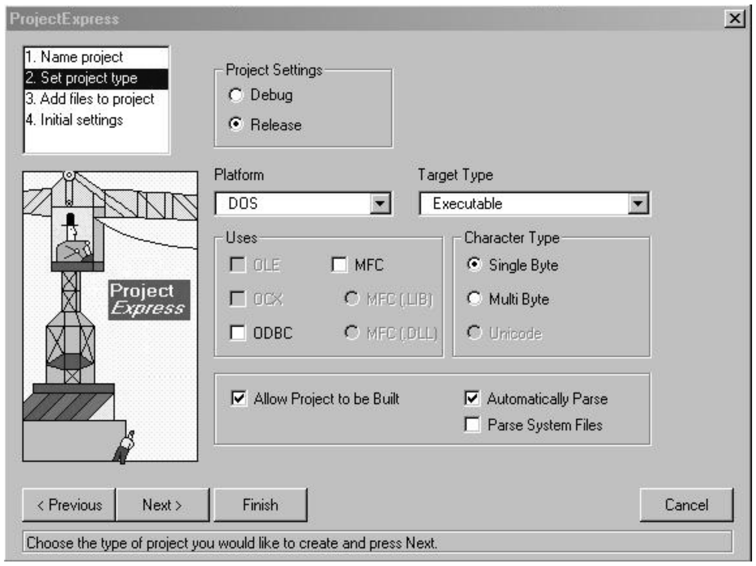
Creating a New Digital Mars C++ 8.49 ADM Project

1 Start Digital Mars C++ 8.49, and then click **Project** → **New** from the *Main Menu*.



Project Name: 56adm-sd.prj
56adm-si.prj
Directories: c:\...\mvi56-adm-serial-in
c:\
ADM_TOOL_MVI
SAMPLES
MVI56-Samples
MVI56-ADM
MVI56-ADM-Se
List Files of Type: Project (*.prj)
Drives: c:
Use AppExpress to create new application
Type a name for your project and either press Next, or check Use AppExpress and press Finish.

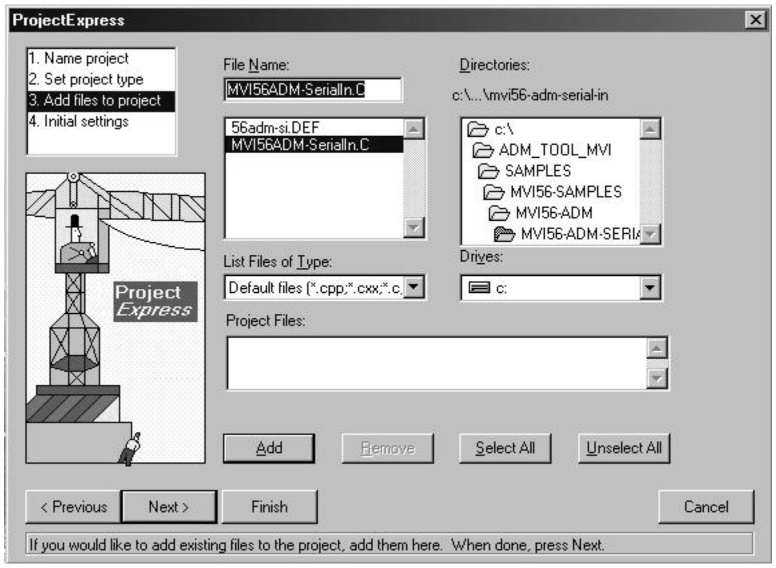
2 Select the path and type in the **Project Name**.
3 Click Next.



Project Settings: Debug [✓ Release]
Platform: DOS
Target Type: Executable
Uses: OLE, OCX, ODBC, MFC, MFC (LIB), MFC (DLL)
Character Type: [✓ Single Byte] Multi Byte Unicode
[✓] Allow Project to be Built
[✓] Automatically Parse
Parse System Files
Choose the type of project you would like to create and press Next.

4 In the *Platform* field, choose **DOS**.
5 In the Project Settings choose Release if you do not want debug information included in your build.

6 Click Next.



File Name: MVI56ADM-SerialIn.C

56adm-si.DEF
MVI56ADM-SerialIn.C

Directories: c:\...\mvi56-adm-serial-in

c:\
ADM_TOOL_MVI
SAMPLES
MVI56-SAMPLES
MVI56-ADM
MVI56-ADM-SERIA

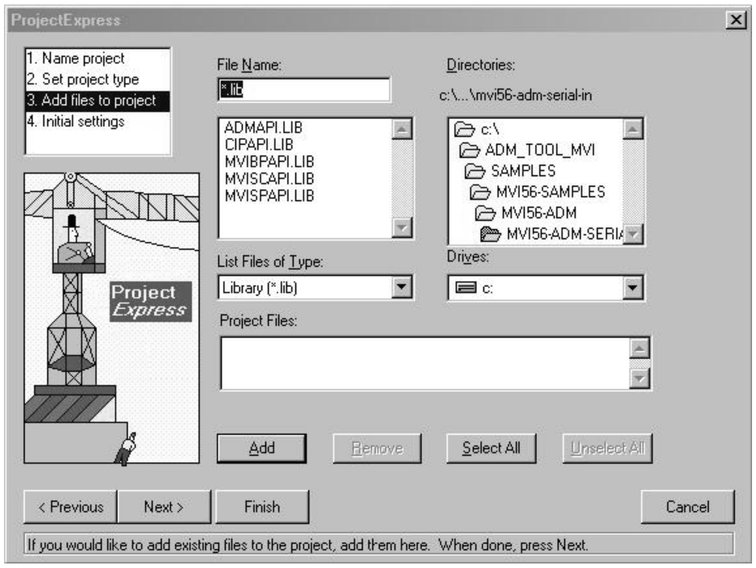
List Files of Type: Default files (*.cpp;*.cxx;*.c;

Drives: c:

Project Files:

If you would like to add existing files to the project, add them here. When done, press Next.

7 Select the first source file necessary for the project.
8 Click Add.
9 Repeat this step for all source files needed for the project.
10 Repeat the same procedure for all library files (.lib) needed for the project.
11 Choose Libraries (*.lib) from the *List Files of Type* field to view all library files:



File Name: *.lib

ADMAPI.LIB
CIPAPI.LIB
MVIBPAPI.LIB
MVISCAPI.LIB
MVISPAPI.LIB

Directories: c:\...\mvi56-adm-serial-in

c:\
ADM_TOOL_MVI
SAMPLES
MVI56-SAMPLES
MVI56-ADM
MVI56-ADM-SERIA

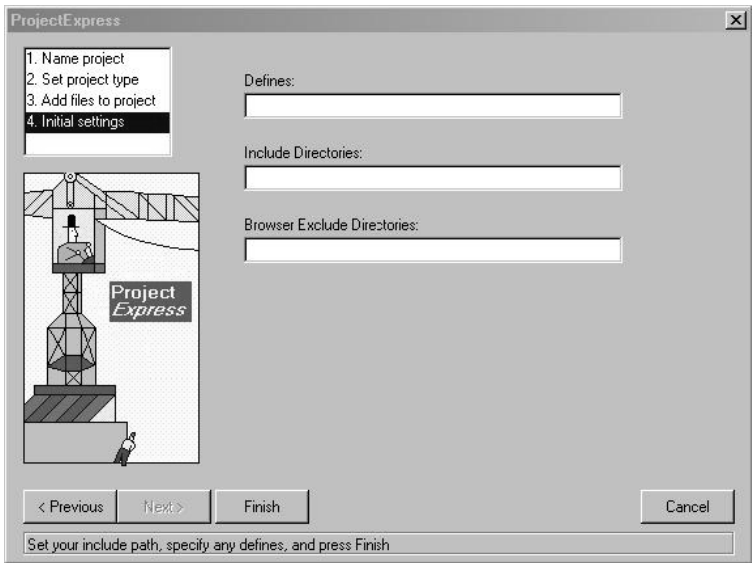
List Files of Type: Library (*.lib)

Drives: c:

Project Files:

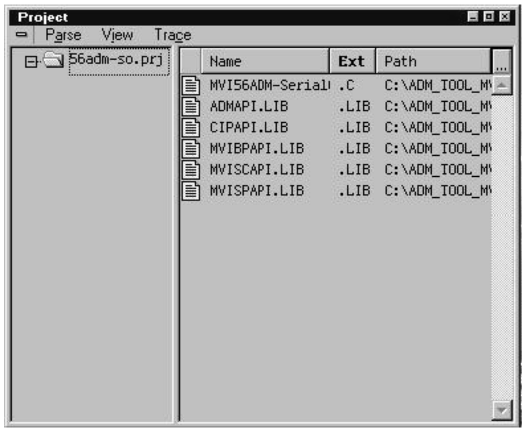
If you would like to add existing files to the project, add them here. When done, press Next.

12 Click Next.



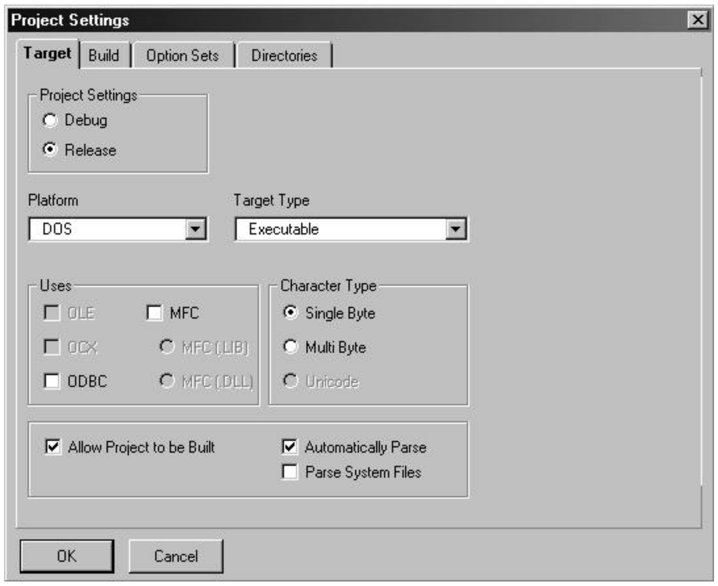
Set your include path, specify any defines, and press Finish

13 Add any defines or include directories desired.
14 Click **Finish**.
15 The *Project* window should now contain all the necessary source and library files as shown in the following window:



Name	Ext	Path
MVI56ADM-Serial	.C	C:\ADM_TOOL_M
ADMAPI.LIB	.LIB	C:\ADM_TOOL_M
CIPAPI.LIB	.LIB	C:\ADM_TOOL_M
MVIBPAPI.LIB	.LIB	C:\ADM_TOOL_M
MVISCAPI.LIB	.LIB	C:\ADM_TOOL_M
MVISPAPI.LIB	.LIB	C:\ADM_TOOL_M

16 Click **Project** → **Settings** from the *Main Menu*.



Project Settings: Debug, [✓ Release]

Platform: DOS

Target Type: Executable

Uses: ☐ OLE, ☐ MFC, ☐ OCX, ○ MFC (.LIB), ☐ ODBC, ○ MFC (.DLL)

Character Type: [✓ Single Byte], Multi Byte, Unicode

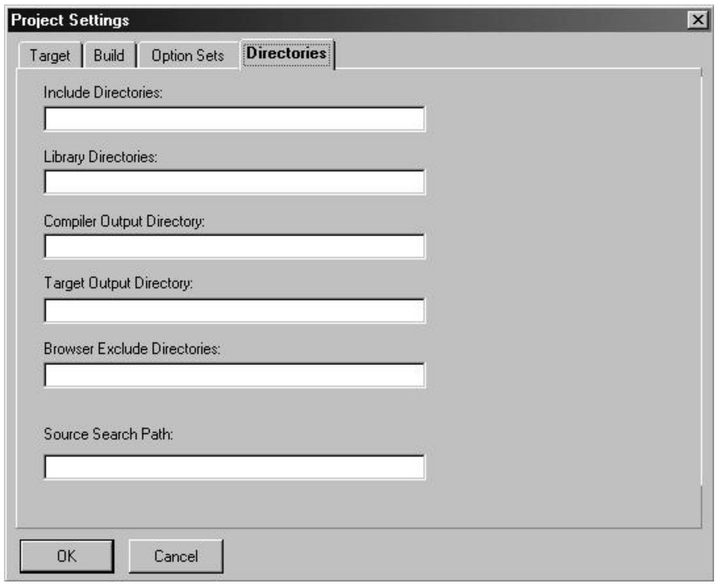
☑ Allow Project to be Built

☑ Automatically Parse

☐ Parse System Files

17 These settings were set when the project was created. No changes are required. The executable must be built as a DOS executable in order to run on the PTQ platform.

18 Click the **Directories** tab and fill in directory information as required by your project's directory structure.



Include Directories:

Library Directories:

Compiler Output Directory:

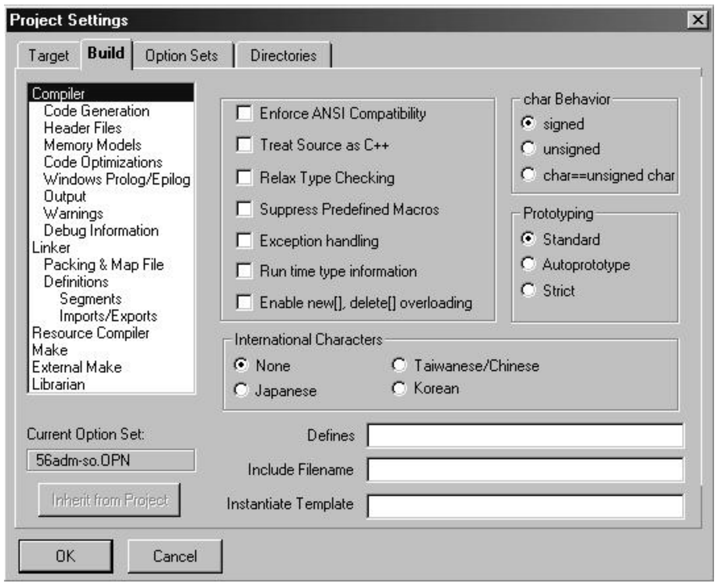
Target Output Directory:

Browser Exclude Directories:

Source Search Path:

19 If the fields are left blank then it is assumed that all of the files are in the same directory as the project file. The output files will be placed in this directory as well.

20 Click on the **Build** tab, and choose the **Compiler** selection. Confirm that the settings match those shown in the following screen:



Enforce ANSI Compatibility ☐
Treat Source as C++ ☐
Relax Type Checking ☐
Suppress Predefined Macros ☐
Exception handling ☐
Run time type information ☐
Enable new[], delete[] overloading ☐

char Behavior: [✓ signed] unsigned char==unsigned char

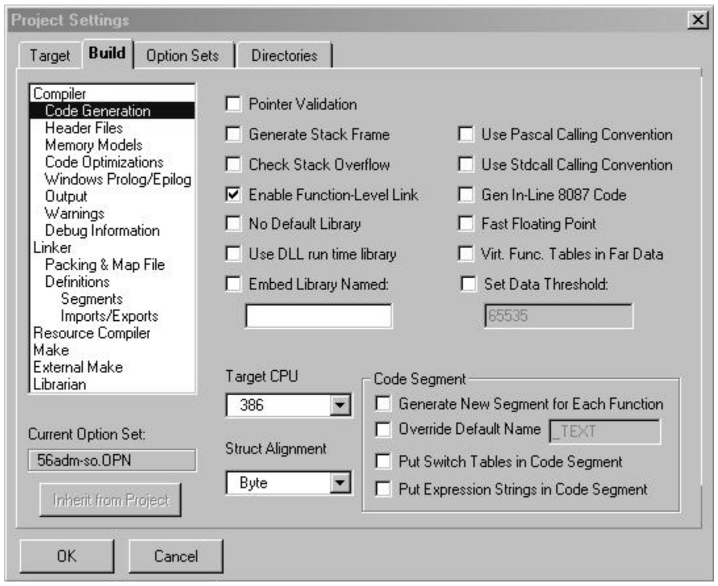
Prototyping: [✓ Standard] Autoprototype Strict

International Characters: [✓ None] Japanese Taiwanese/Chinese Korean

Current Option Set: 56adm-so.OPN

Defines:
Include Filename:
Instantiate Template:

21 Click **Code Generation from** the *Topics* field and ensure that the options match those shown in the following screen:



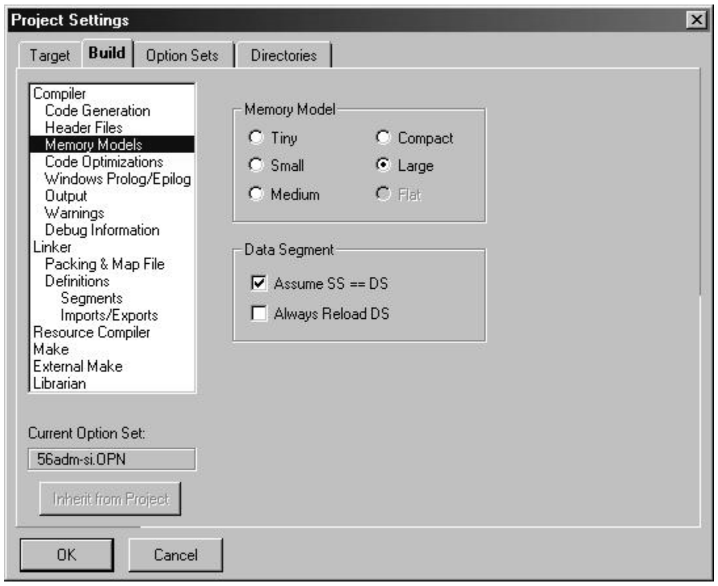
Pointer Validation ☐
Generate Stack Frame ☐
Check Stack Overflow ☐
Enable Function-Level Link ☑
No Default Library ☐
Use DLL run time library ☐
Embed Library Named: ☐
Use Pascal Calling Convention ☐
Use Stdcall Calling Convention ☐
Gen In-Line 8087 Code ☐
Fast Floating Point ☐
Virt. Func. Tables in Far Data ☐
Set Data Threshold: ☐ 65535

Target CPU: 386
Struct Alignment: Byte

Code Segment:
Generate New Segment for Each Function ☐
Override Default Name ☐ _TEXT
Put Switch Tables in Code Segment ☐
Put Expression Strings in Code Segment ☐

Current Option Set: 56adm-so.OPN

22 Click **Memory Models from** the *Topics* field and ensure that the options match those shown in the following screen:

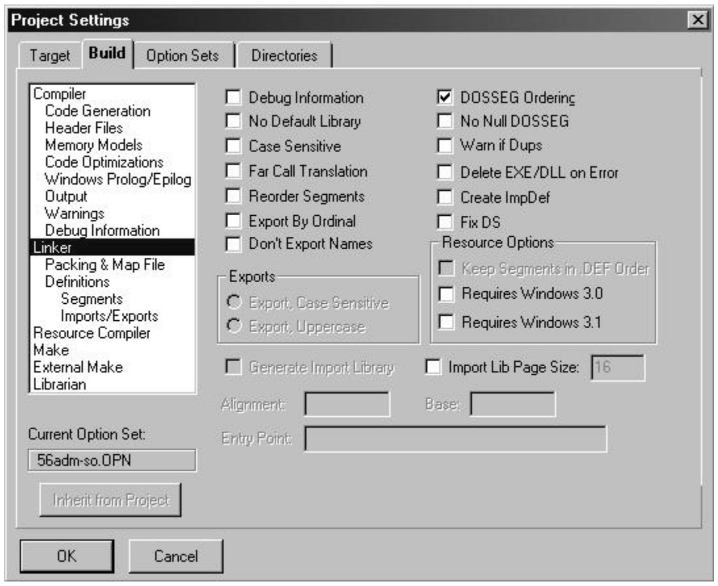


Memory Model: Tiny [ ] Compact [ ] Small [ ] [✓ Large] Medium [ ] Flat [ ]

Data Segment: [✓ Assume SS == DS] Always Reload DS [ ]

Current Option Set: 56adm-si.OPN

23 Click **Linker from** the *Topics* field and ensure that the options match those shown in the following screen:



Debug Information [ ]
No Default Library [ ]
Case Sensitive [ ]
Far Call Translation [ ]
Reorder Segments [ ]
Export By Ordinal [ ]
Don't Export Names [ ]
[✓ DOSSEG Ordering]
No Null DOSSEG [ ]
Warn if Dups [ ]
Delete EXE/DLL on Error [ ]
Create ImpDef [ ]
Fix DS [ ]

Exports: Export, Case Sensitive [ ] Export, Uppercase [ ]

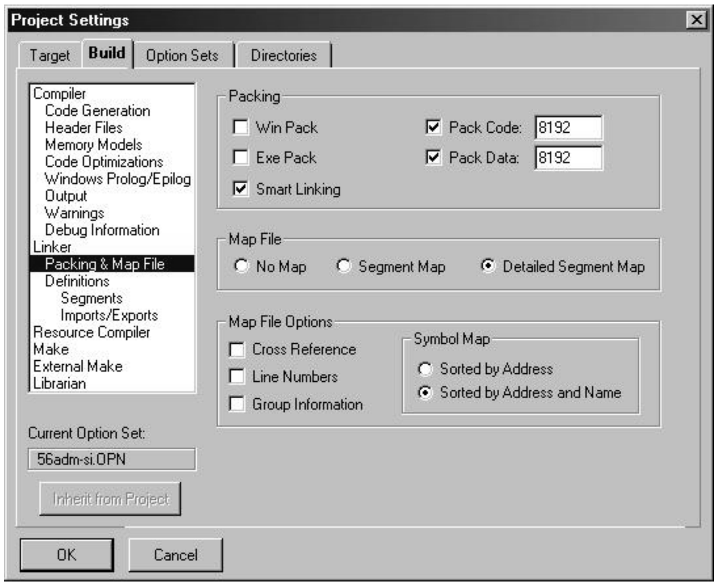
Resource Options: Keep Segments in .DEF Order [ ] Requires Windows 3.0 [ ] Requires Windows 3.1 [ ]

Generate Import Library [ ]
Import Lib Page Size: [ ] 16

Alignment:
Base:
Entry Point:

Current Option Set: 56adm-so.OPN

24 Click **Packing & Map File from** the *Topics* field and ensure that the options match those shown in the following screen:



Packing
- [ ] Win Pack
- [ ] Exe Pack
- [✓] Smart Linking
- [✓] Pack Code: 8192
- [✓] Pack Data: 8192

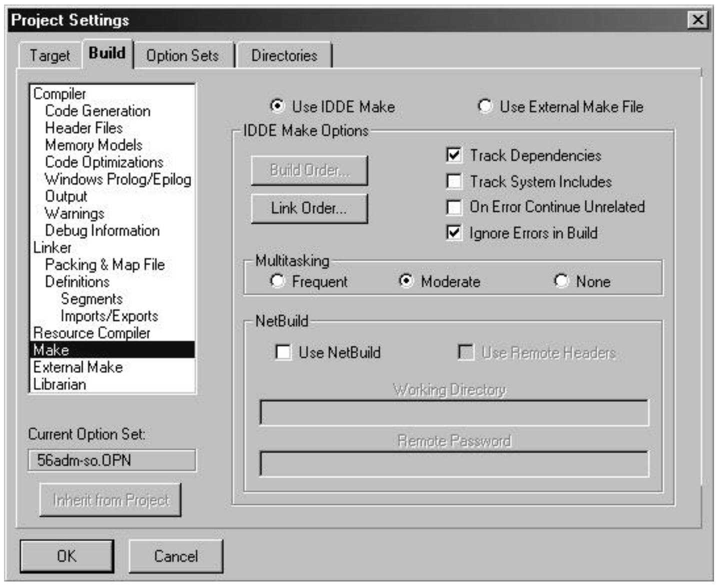
Map File
No Map Segment Map [✓ Detailed Segment Map]

Map File Options
- [ ] Cross Reference
- [ ] Line Numbers
- [ ] Group Information

Symbol Map
Sorted by Address [✓ Sorted by Address and Name]

Current Option Set: 56adm-si.OPN

25 Click **Make from** the *Topics* field and ensure that the options match those shown in the following screen:



[✓ Use IDDE Make] Use External Make File

IDDE Make Options
- [✓] Track Dependencies
- [ ] Track System Includes
- [ ] On Error Continue Unrelated
- [✓] Ignore Errors in Build

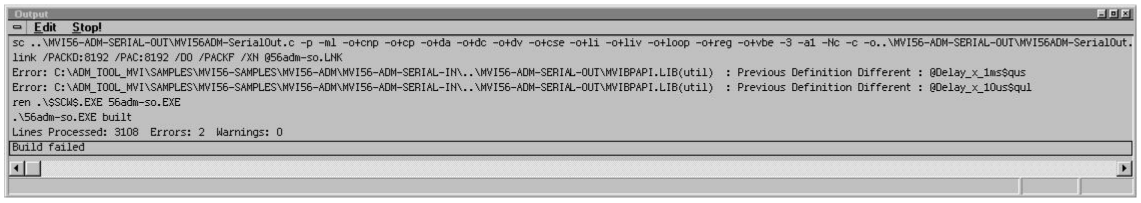
Multitasking
Frequent [✓ Moderate] None

NetBuild
- [ ] Use NetBuild
- [ ] Use Remote Headers

Current Option Set: 56adm-so.OPN

26 Click **OK**.
27 Click **Parse** → **Update All** from the Project Window *Menu*. The new settings may not take effect unless the project is updated and reparsed.
28 Click **Project** → **Build All** from the Main Menu.

29 When complete, the build results will appear in the Output window:



```
sc ..\MVI56-ADM-SERIAL-OUT\MVI56ADM-SerialOut.c -p -ml -o+cnp -o+cp -o+da -o+dc -o+dv -o+cse -o+li -o+liv -o+loop -o+reg -o+vbe -3 -a1 -Nc -c -o..\MVI56-ADM-SERIAL-OUT\MVI56ADM-SerialOut.
link /PACKD:8192 /PAC:8192 /DO /PACKF /XN @56adm-so.LNK
Error: C:\ADM_TOOL_MVI\SAMPLES\MVI56-SAMPLES\MVI56-ADM\MVI56-ADM-SERIAL-IN\..\MVI56-ADM-SERIAL-OUT\MVIBPAPI.LIB(util)  : Previous Definition Different : @Delay_x_1ms$qus
Error: C:\ADM_TOOL_MVI\SAMPLES\MVI56-SAMPLES\MVI56-ADM\MVI56-ADM-SERIAL-IN\..\MVI56-ADM-SERIAL-OUT\MVIBPAPI.LIB(util)  : Previous Definition Different : @Delay_x_10us$qul
ren .\$SCW$.EXE 56adm-so.EXE
.\56adm-so.EXE built
Lines Processed: 3108  Errors: 2  Warnings: 0
Build failed
```

The executable file will be located in the directory listed in the Compiler Output Directory box of the Directories tab (that is, C:\ADM_TOOL_PTQ\SAMPLES\…). The *Project Settings* window can be accessed by clicking **Project** → **Settings** from the *Main Menu.*

Porting Notes: *The Digital Mars compiler classifies duplicate library names as Level 1 Errors rather than warnings. These errors will manifest themselves as "Previous Definition Different: function name". Level 1 errors are non-fatal and the executable will build and run. The architecture of the ADM libraries will cause two or more of these errors to appear when the executable is built. This is a normal occurrence. If you are building existing code written for a different compiler you may have to replace calls to run-time functions with the Digital Mars equivalent. Refer to the Digital Mars documentation on the Run-time Library for the functions available.*

8.1.2 Configuring Borland C++5.02

The following procedure allows you to successfully build the sample ADM code supplied by ProSoft Technology. using Borland C++ 5.02. After verifying that the sample code can be successfully compiled and built, you can modify the sample code to work with your application.

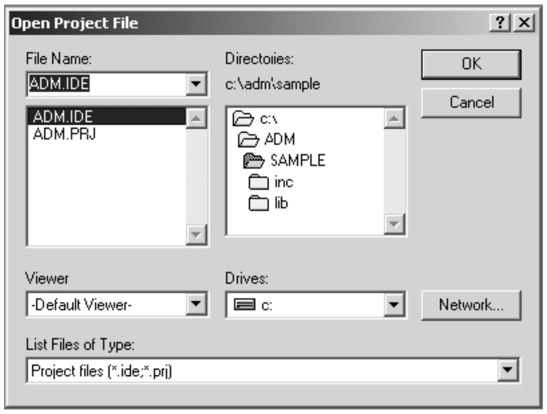
Note: This procedure assumes that you have successfully installed Borland C++ 5.02 on your workstation.

Downloading the Sample Program

The sample code files are located in the ADM_TOOL_PTQ.ZIP file. This zip file is available from the CD-ROM shipped with your system or from the www.prosoft-technology.com web site. One the file is unzipped, you can find the sample code files in \ADM_TOOL_PTQ\Samples\

Building an Existing Borland C++ 5.02 ADM Project

1 Start Borland C++ 5.02, then click **Project → Open Project** from the *Main Menu*.

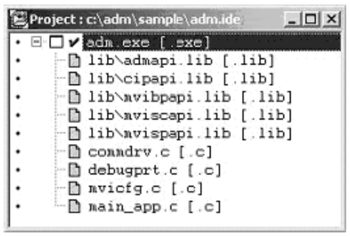

Open Project File

File Name: ADM.IDE
ADM.IDE
ADM.PRJ

Directoies: c:\adm\sample
c:\
ADM
SAMPLE
inc
lib

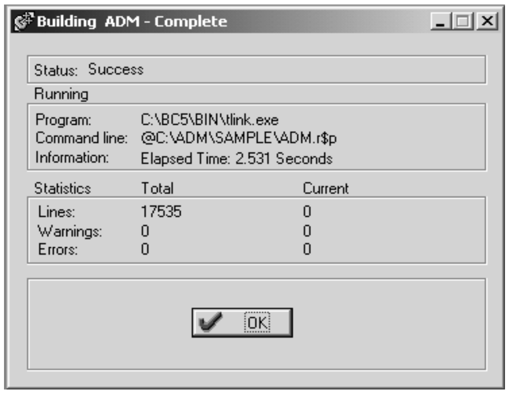
Viewer: -Default Viewer-
Drives: c:
List Files of Type: Project files (*.ide;*.prj)

2 From the *Directories* field, navigate to the directory that contains the project (C:\adm\sample).
3 In the *File Name* field, click on the project name (adm.ide).
4 Click **OK**. The *Project* window appears:


Project : c:\adm\sample\adm.ide

```
adm.exe [.exe]
  lib\admapi.lib [.lib]
  lib\cipapi.lib [.lib]
  lib\mvibpapi.lib [.lib]
  lib\mviscapi.lib [.lib]
  lib\mvispapi.lib [.lib]
  commdrv.c [.c]
  debugprt.c [.c]
  mvicfg.c [.c]
  main_app.c [.c]
```

5 Click **Project → Build All** from the *Main Menu* to create the .exe file. The *Building ADM* window appears when complete:

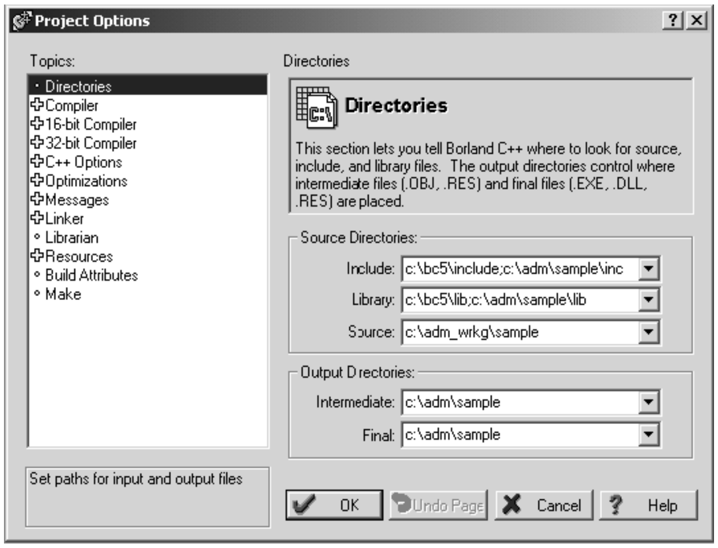

Building ADM - Complete

Status: Success
Running
Program: C:\BC5\BIN\tlink.exe
Command line: @C:\ADM\SAMPLE\ADM.r$p
Information: Elapsed Time: 2.531 Seconds

Statistics	Total	Current
Lines:	17535	0
Warnings:	0	0
Errors:	0	0

6 When Success appears in the *Status* field, click **OK**.

The executable file will be located in the directory listed in the *Final* field of the Output Directories (that is, C:\adm\sample). The *Project Options* window can be accessed by clicking **Options → Project Menu** from the *Main Menu*.



Directories

This section lets you tell Borland C++ where to look for source, include, and library files. The output directories control where intermediate files (.OBJ, .RES) and final files (.EXE, .DLL, .RES) are placed.

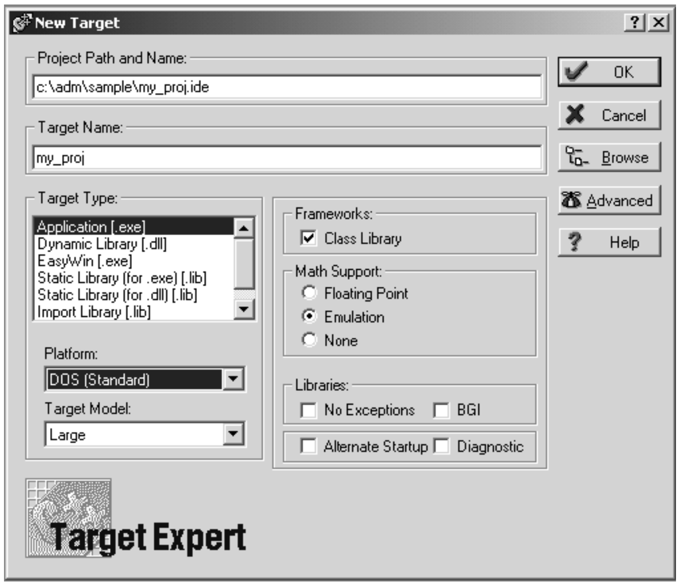
Source Directories:
Include: c:\bc5\include;c:\adm\sample\inc
Library: c:\bc5\lib;c:\adm\sample\lib
Source: c:\adm_wrkg\sample

Output Directories:
Intermediate: c:\adm\sample
Final: c:\adm\sample

Set paths for input and output files

Creating a New Borland C++ 5.02 ADM Project

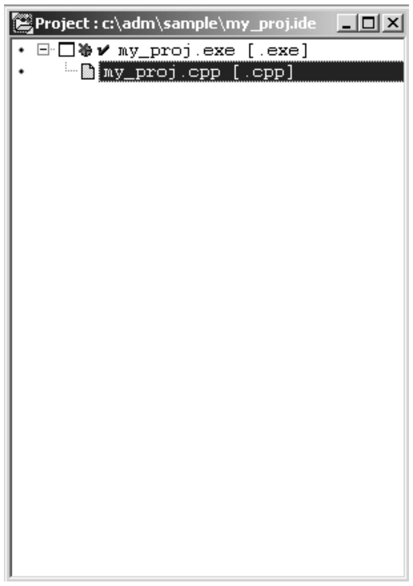
1 Start Borland C++ 5.02, then click File → Project from the Main Menu.



Project Path and Name: c:\adm\sample\my_proj.ide
Target Name: my_proj
Target Type: Application [.exe]
Platform: DOS (Standard)
Target Model: Large
Frameworks: [✓] Class Library
Math Support: ( ) Floating Point (●) Emulation ( ) None
Libraries: [ ] No Exceptions [ ] BGI
[ ] Alternate Startup [ ] Diagnostic

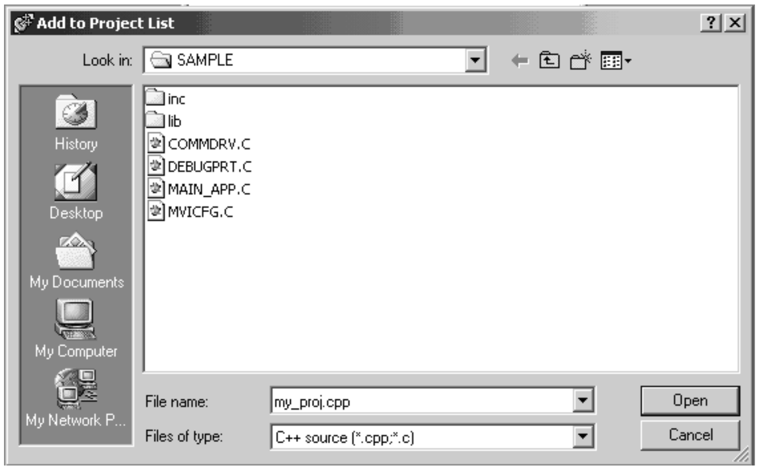
2 Type in the Project Path and Name. The Target Name is created automatically.
3 In the Target Type field, choose Application (.exe).
4 In the Platform field, choose DOS (Standard).
5 In the Target Model field, choose Large.
6 Ensure that Emulation is checked in the Math Support field.

7 Click OK. A Project window appears:



8 Click on the .cpp file created and press the Delete key. Click Yes to delete the .cpp file.

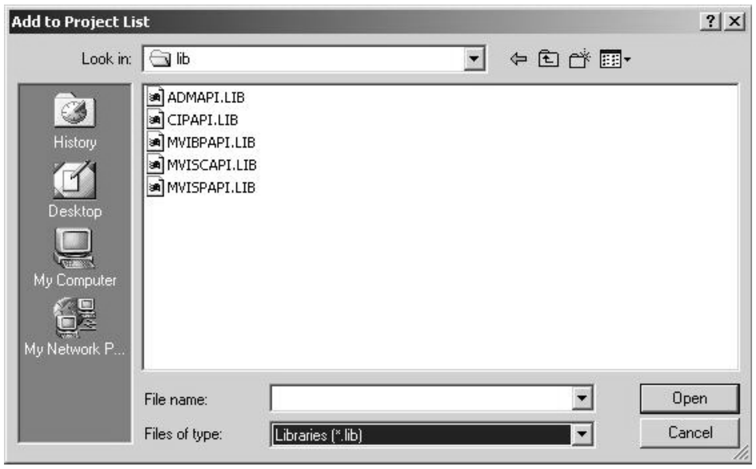
9 Right click on the .exe file listed in the Project window and choose the Add Node menu selection. The following window appears:



10 Click source file, then click Open to add source file to the project. Repeat this step for all source files needed for the project.

11 Repeat the same procedure for all library files (.lib) needed for the project.

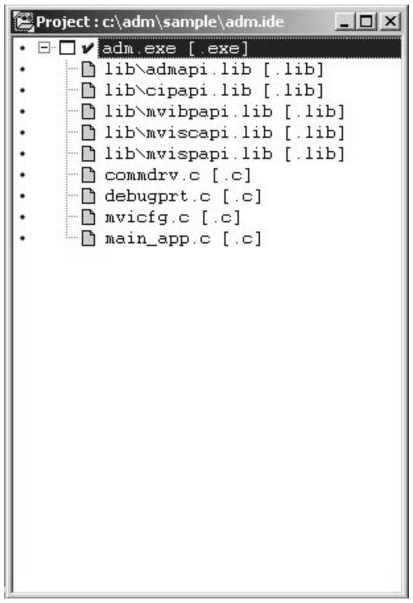
12 Choose Libraries (*.lib) from the Files of Type field to view all library files:



ADMAPI.LIB
CIPAPI.LIB
MVIBPAPI.LIB
MVISCAPI.LIB
MVISPAPI.LIB

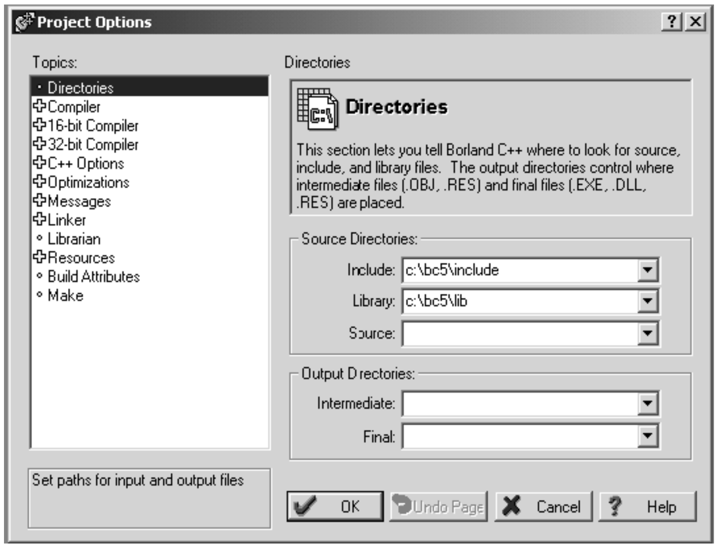
File name:
Files of type: Libraries (*.lib)

13 The Project window should now contain all the necessary source and library files as shown in the following window:



```
adm.exe [.exe]
  lib\admapi.lib [.lib]
  lib\cipapi.lib [.lib]
  lib\mvibpapi.lib [.lib]
  lib\mviscapi.lib [.lib]
  lib\mvispapi.lib [.lib]
  commdrv.c [.c]
  debugprt.c [.c]
  mvicfg.c [.c]
  main_app.c [.c]
```

14 Click Options → Project from the Main Menu.



Directories

Directories

This section lets you tell Borland C++ where to look for source, include, and library files. The output directories control where intermediate files (.OBJ, .RES) and final files (.EXE, .DLL, .RES) are placed.

Source Directories:

Include: c:\bc5\include

Library: c:\bc5\lib

Source:

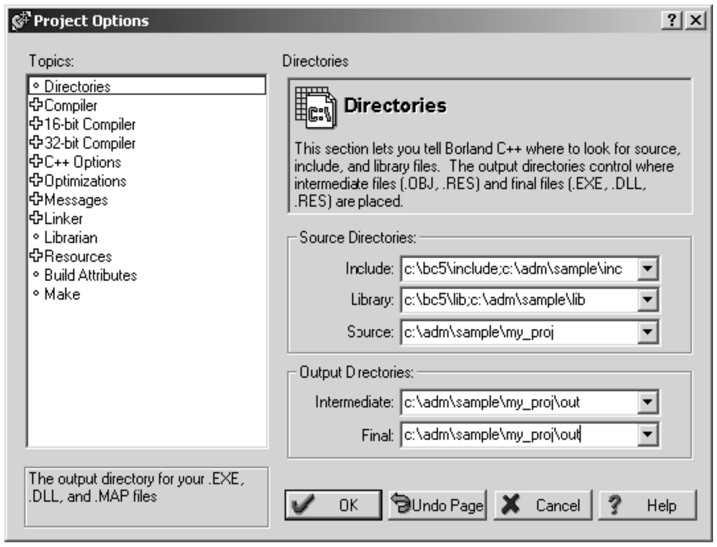
Output Directories:

Intermediate:

Final:

Set paths for input and output files

15 Click Directories from the Topics field and fill in directory information as required by your project's directory structure.



Directories

Directories

This section lets you tell Borland C++ where to look for source, include, and library files. The output directories control where intermediate files (.OBJ, .RES) and final files (.EXE, .DLL, .RES) are placed.

Source Directories:

Include: c:\bc5\include;c:\adm\sample\inc

Library: c:\bc5\lib;c:\adm\sample\lib

Source: c:\adm\sample\my_proj

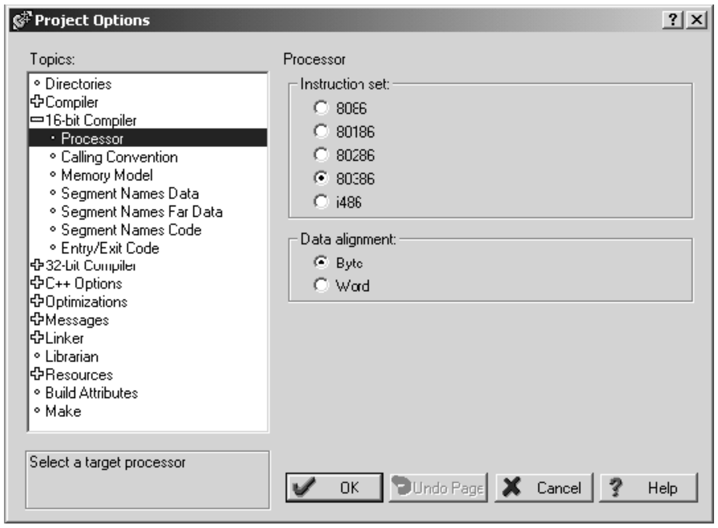
Output Directories:

Intermediate: c:\adm\sample\my_proj\out

Final: c:\adm\sample\my_proj\out

The output directory for your .EXE, .DLL, and .MAP files

16 Double-click on the Compiler header in the Topics field, and choose the Processor selection. Confirm that the settings match those shown in the following screen:



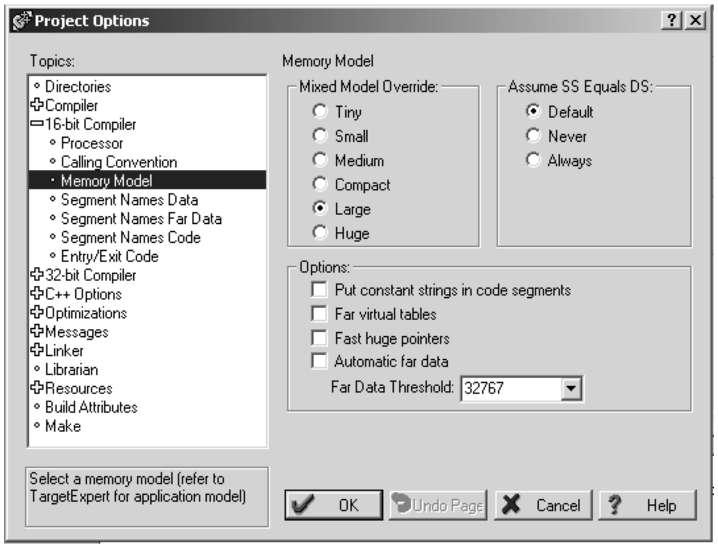
Processor

Instruction set: 8086 80186 80286 [✓ 80386] i486

Data alignment: [✓ Byte] Word

Select a target processor

17 Click Memory Model from the Topics field and ensure that the options match those shown in the following screen:



Memory Model

Mixed Model Override: Tiny Small Medium Compact [✓ Large] Huge

Assume SS Equals DS: [✓ Default] Never Always

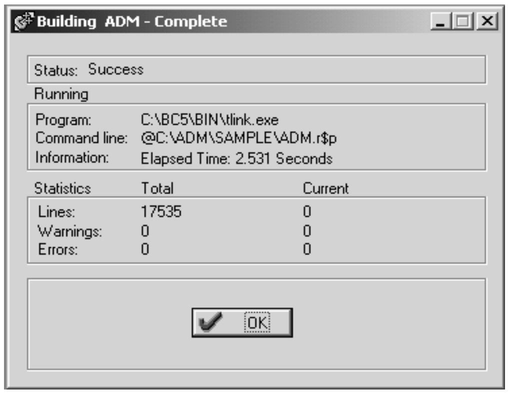
Options:
- [ ] Put constant strings in code segments
- [ ] Far virtual tables
- [ ] Fast huge pointers
- [ ] Automatic far data

Far Data Threshold: 32767

Select a memory model (refer to TargetExpert for application model)

18 Click OK.
19 Click Project → Build All from the Main Menu.

20 When complete, the Success window appears:



```
Status:  Success
Running
Program:        C:\BC5\BIN\tlink.exe
Command line:   @C:\ADM\SAMPLE\ADM.r$p
Information:    Elapsed Time: 2.531 Seconds

Statistics      Total           Current
Lines:          17535           0
Warnings:       0               0
Errors:         0               0
```

21 Click OK. The executable file will be located in the directory listed in the Final box of the Output Directories (that is, C:\adm\sample). The Project Options window can be accessed by clicking Options → Project from the Main Menu.

8.2 Creating a ROM Disk Image

To change the contents of the ROM disk, a new disk image must be created using the WINIMAGE utility.

The WINIMAGE utility for creating disk images is described in the following topics.

8.2.1 WINIMAGE - Windows Disk Image Builder

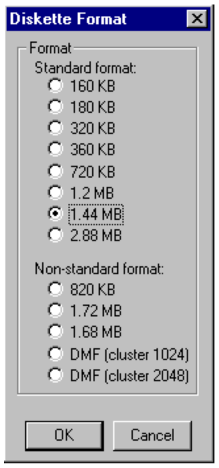
WINIMAGE is a Win9x/NT utility that may be used to create disk images for downloading to the PTQ module. It does not require the use of a floppy diskette. Also, it is not necessary to estimate the disk image size, since WINIMAGE does this automatically and can truncate the unused portion of the disk. In addition, WINIMAGE will de-fragment a disk image so that files may be deleted and added to the image without resulting in wasted space.

To install WINIMAGE, unzip the winima40.zip file in a subdirectory on your PC running Win9x or NT 4.0. To start WINIMAGE, run WINIMAGE.EXE.

Follow these steps to build a disk image:

1 Start WINIMAGE.

2 Select **File**, **New** and choose a disk format as shown in the following diagram. Any format will do, as long as it is large enough to contain your files. The default is 1.44Mb, which is fine for our purposes. Click on **OK**.

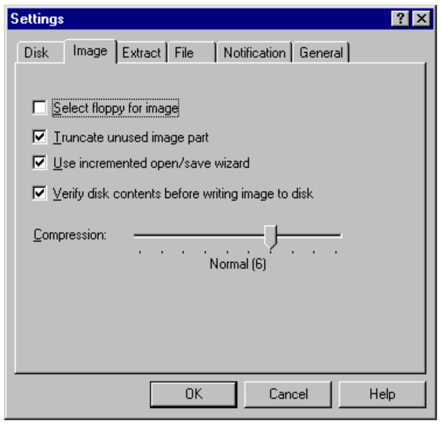


Standard format:
160 KB
180 KB
320 KB
360 KB
720 KB
1.2 MB
[✓ 1.44 MB]
2.88 MB

Non-standard format:
820 KB
1.72 MB
1.68 MB
DMF (cluster 1024)
DMF (cluster 2048)

3 Drag and drop the files you want in your image to the WINIMAGE window.

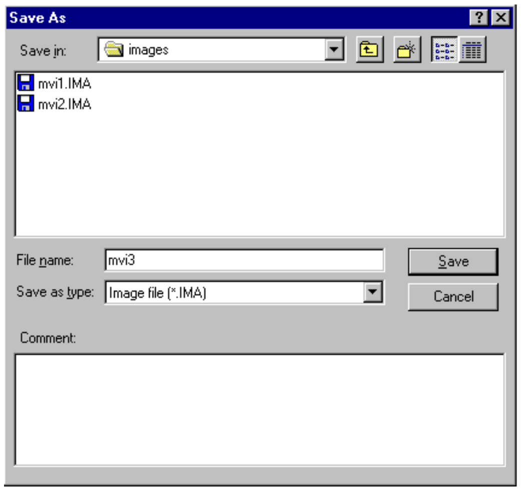
4 Click on **Options**, **Settings** and make sure the **Truncate unused image part** option is selected, as shown in the following figure. Click on **OK**.



☐ Select floppy for image
☑ Truncate unused image part
☑ Use incremented open/save wizard
☑ Verify disk contents before writing image to disk

Compression: Normal (6)

5 Click on **File**, **Save As**, and choose a directory and filename for the disk image file. The image must be saved as an uncompressed disk image, so be sure to select **Save as type: Image file (*.IMA)** as shown in the following figure.



Save in: images

mvi1.IMA
mvi2.IMA

File name: mvi3
Save as type: Image file (*.IMA)

6 Check the disk image file size to be sure it does not exceed the maximum size of the PTQ module's ROM disk (896K bytes). If it is too large, use WINIMAGE to remove some files from the image, then de-fragment the image and try again (**Note:** To de-fragment an image, click on **Image**, **Defrag current image**.

The disk image is now ready to be downloaded to the PTQ module.

For more information on using WINIMAGE, refer to the documentation included with it.

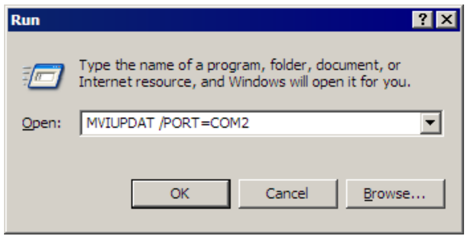
Note: WINIMAGE is a shareware utility. If you find this program useful, please register it with the author.

8.3 Downloading a ROM Disk Image

8.3.1 MVIUPDAT

MVIUPDAT.EXE is a DOS-compatible utility for downloading a ROM disk image from a host PC to the PTQ-ADM module. MVIUPDAT.EXE uses a serial port on the PC to communicate with the module. Follow the steps below to download a ROM disk image:

1 Connect a null-modem serial cable between the serial port on the PC and PRT1 on the PTQ module.
2 If you are using HyperTerm or a similar terminal program for the PTQ-ADM module console, exit or disconnect from the serial port before running the MVI Flash Update tool.
3 Turn off power to the PTQ module. Install the Setup Jumper as described in the Installation Instructions.
4 Click the **START** button, and then choose **RUN.**
5 In the **OPEN:** field, enter `MVIUPDAT`. Specify the PC port on the command line as shown in the following illustration. The default is COM1.


Run
?
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
OK
Cancel
Browse...

MVIUPDAT /PORT=COM2

6 Turn on power to the PTQ module. You should see the following menu shown on the host PC.

```
+-----------------------------+
| Main Menu                   |
|-----------------------------|
| Verify Module Connection    |
| Update Flash Disk Image     |
| Reboot Module               |
+-----------------------------+
```

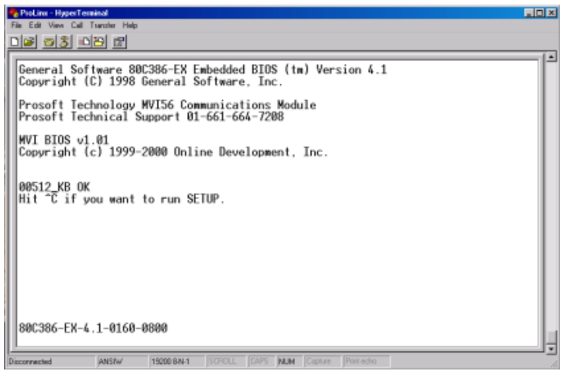
7 Select **VERIFY MODULE CONNECTION** to verify the connection to the PTQ module. If the connection is working properly, the message "Module Responding" will be displayed.

Note: If an error occurs, check your serial port assignments and cable connections. You may also need to cycle power more than once before the module responds.

8 Select **UPDATE FLASH DISK IMAGE** to download the ROM disk image. Type the image file name when prompted. The download progress is displayed as the file is being transmitted to the module.
9 After the disk image has been transferred, reboot the PTQ module by selecting the **REBOOT MODULE** menu item.
10 Exit the MVIUPDAT.EXE utility by pressing **[ESC].**

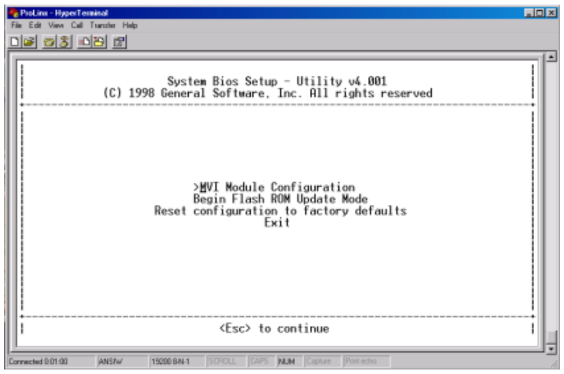
8.4 PTQ System BIOS Setup

The BIOS Setup for the PTQ products contains module configuration settings and allows for placing the PTQ module in a flash update mode. To access the BIOS Setup, attach a null modem cable from the PC COM port to the Status/Debug port on the PTQ module. Start Hyper Term with the appropriate communication settings for the Debug port. Press **[CTRL][C]** during the memory test portion in the booting of the module.



```
General Software 80C386-EX Embedded BIOS (tm) Version 4.1
Copyright (C) 1998 General Software, Inc.

Prosoft Technology MVI56 Communications Module
Prosoft Technical Support 01-661-664-7208

MVI BIOS v1.01
Copyright (c) 1999-2000 Online Development, Inc.


00512_KB OK
Hit ^C if you want to run SETUP.









80C386-EX-4.1-0160-0800
```

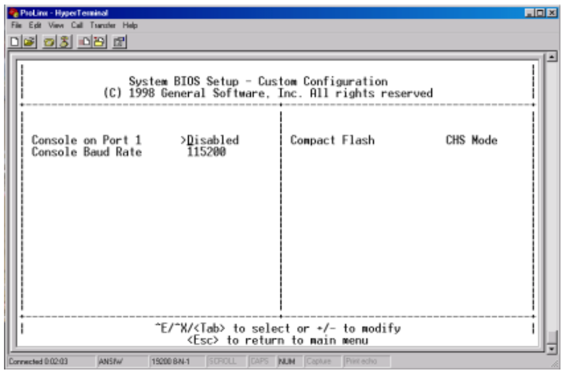
It may be necessary to install the setup jumper in order to access the BIOS Setup. The setup jumper will be necessary if the Console is disabled. The following illustration shows the BIOS Setup screen.



```
                    System Bios Setup - Utility v4.001
              (C) 1998 General Software, Inc. All rights reserved

                      >MVI Module Configuration
                       Begin Flash ROM Update Mode
                  Reset configuration to factory defaults
                                  Exit

                          <Esc> to continue
```

The PTQ module can be placed in a mode where it is waiting to receive a new flash image by selecting the Begin Flash ROM Update Mode option.

Select PTQ Module Configuration to set the Console, Console Baud Rate and Compact Flash mode. The Console allows keyboard entry and text output to the debug port. The baud rate of the console port is selected by the Console Baud Rate option. In order to use a Compact Flash disk in the PTQ module the Compact Flash option must be set to CHS mode.


ProLinx - HyperTerminal
File Edit View Call Transfer Help
Connected 0:02:03 ANSIW 19200 8-N-1 SCROLL CAPS NUM Capture Print echo

```
              System BIOS Setup - Custom Configuration
          (C) 1998 General Software, Inc. All rights reserved

 Console on Port 1      >Disabled       Compact Flash          CHS Mode
 Console Baud Rate       115200

                ^E/^X/<Tab> to select or +/- to modify
                     <Esc> to return to main menu
```

8.5 Transferring Files to and from the Module with HyperTerminal

You can transfer individual files to and from the Compact Flash drive on the ADMNET module using the utilities RY.exe (Receive Ymodem) and SY.exe (Send Ymodem). These two programs work with a terminal client (for example HyperTerminal) on your desktop PC to connect to the module and transfer files.

RY.exe and SY.exe are included in the sample ADM_TOOL.zip file for your hardware platform (inRAx, ProLinx or ProTalk).

Important: The embedded operating system in the ADM/ADMNET module restricts file names to eight "DOS legal" characters or fewer, with a three character extension. For more information on creating filenames in the proper format refer to pages 17 through 20 of the DOS 6-XL Reference manual.

8.5.1 Required Hardware

You can connect directly from your computer's serial port to the serial port on the module to send (upload) or receive (download) files.

ProSoft Technology recommends the following minimum hardware to connect your computer to the module:

- 80486 based processor (Pentium preferred)
- 1 megabyte of memory
- At least one UART hardware-based serial communications port available. USB-based virtual UART systems (USB to serial port adapters) often do not function reliably, especially during binary file transfers, such as when uploading/downloading configuration files or module firmware upgrades.
- A null modem serial cable.

8.5.2 Required Software

In order to send and receive data over the serial port (COM port) on your computer to the module, you must use a communication program (terminal emulator).

A simple communication program called HyperTerminal is pre-installed with recent versions of Microsoft Windows operating systems. If you are connecting from a machine running DOS, you must obtain and install a compatible communication program. The following table lists communication programs that have been tested by ProSoft Technology.

DOS	ProComm, as well as several other terminal emulation programs
Windows 3.1	Terminal
Windows 95/98	HyperTerminal
Windows NT/2000/XP	HyperTerminal

The RY and SY programs use the Ymodem file transfer protocol to send (upload) and receive (download) configuration files from your module. If you use a communication program that is not on the list above, please be sure that it supports Ymodem file transfers.

8.5.3 Connecting to the Module

To connect to the module’s Configuration/Debug port:

1 Connect your computer to the module’s port using a null modem cable.
2 Start the communication program on your computer and configure the communication parameters with the following settings:

Baud Rate	19200
Parity	None
Data Bits	8
Stop Bits	1
Software Handshaking	None

3 Open the connection. Send the necessary command to terminate the module’s program.

If there is no response from the module, follow these steps:

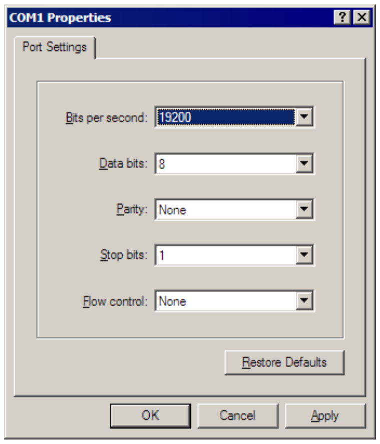
1 Verify that the null modem cable is connected properly between your computer’s serial port and the module. A regular serial cable will not work.
2 Verify that your communication software is using the correct settings for baud rate, parity and handshaking.
3 On computers with more than one serial port, verify that your communication program is connected to the same port that is connected to the module.
4 If you are still not able to establish a connection, you can contact ProSoft Technology Technical Support for further assistance.

8.5.4 Enabling the Console

Before you can use RY and SY from the command prompt, you must enable the console in the ADM module's BIOS.

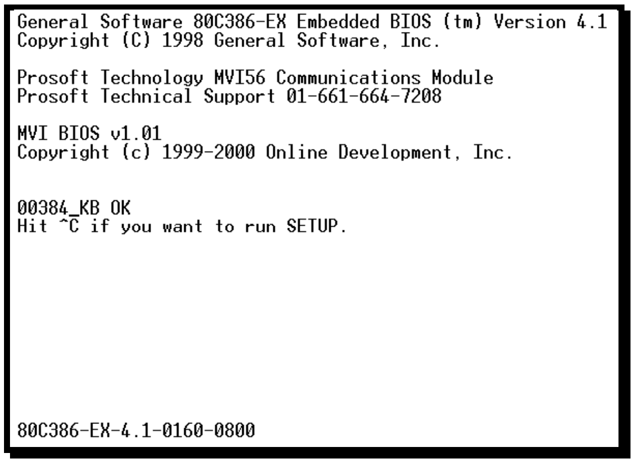
To change BIOS settings

1 Remove the module from the rack and install the Setup jumper.
2 Return the module to the rack.
3 Connect to the module using HyperTerminal at 19,200 bps, and then cycle power to reboot the module.

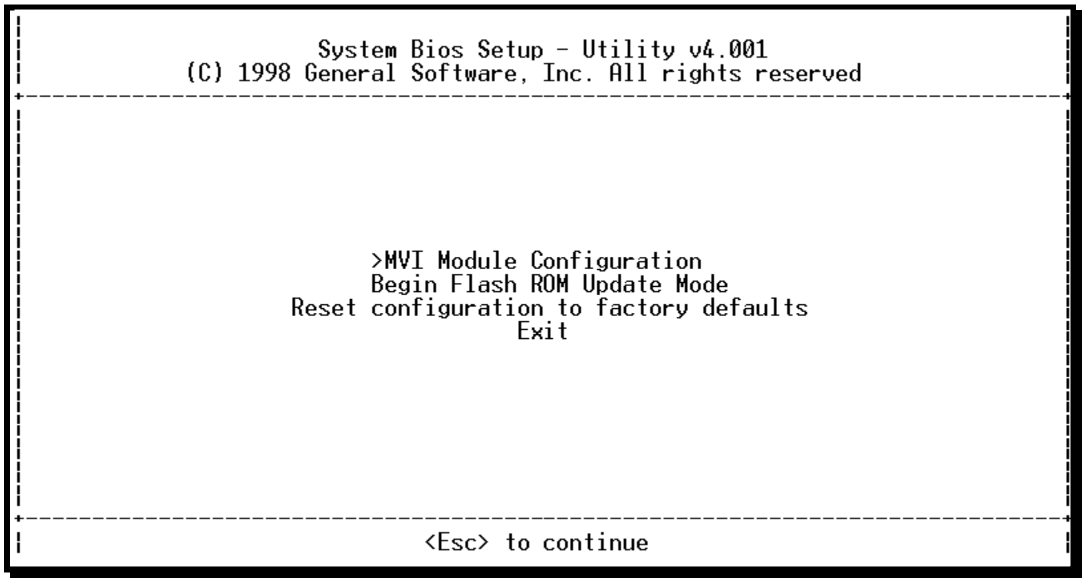


Bits per second: 19200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

4 During the memory check portion of the module's boot sequence, press **[Ctrl][C]** to enter the BIOS configuration menu.

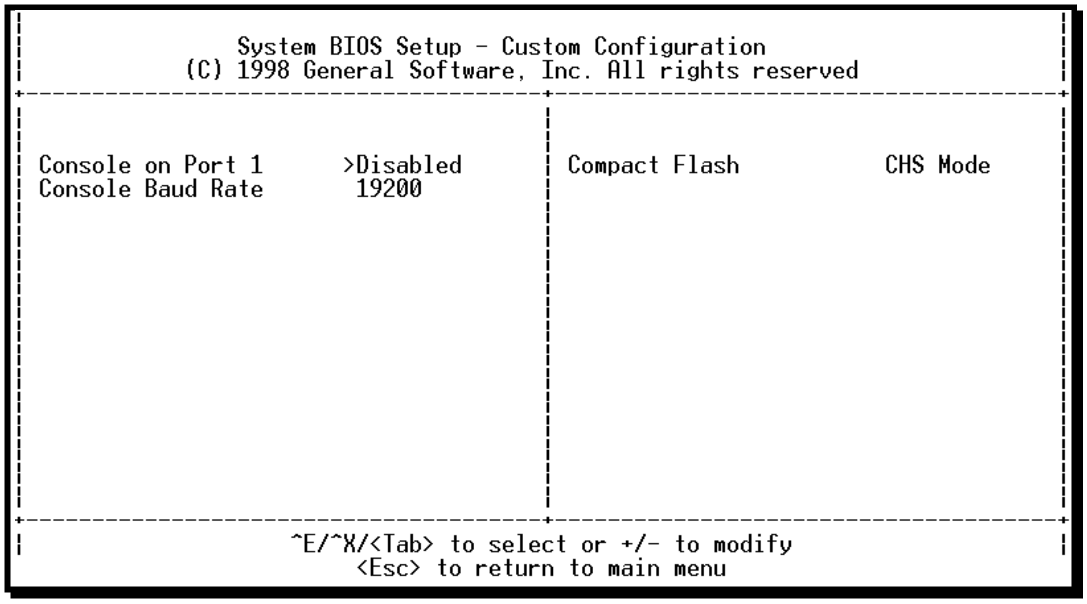


```
General Software 80C386-EX Embedded BIOS (tm) Version 4.1
Copyright (C) 1998 General Software, Inc.

Prosoft Technology MVI56 Communications Module
Prosoft Technical Support 01-661-664-7208

MVI BIOS v1.01
Copyright (c) 1999-2000 Online Development, Inc.


00384_KB OK
Hit ^C if you want to run SETUP.









80C386-EX-4.1-0160-0800
```



```
                  System Bios Setup - Utility v4.001
           (C) 1998 General Software, Inc. All rights reserved
+-----------------------------------------------------------------------+






                       >MVI Module Configuration
                        Begin Flash ROM Update Mode
                   Reset configuration to factory defaults
                                  Exit








+-----------------------------------------------------------------------+
                            <Esc> to continue
```

5 Press **[Enter]** to enter the PTQ-ADM module Configuration menu.



```
                  System BIOS Setup - Custom Configuration
             (C) 1998 General Software, Inc. All rights reserved

 Console on Port 1      >Disabled     Compact Flash          CHS Mode
 Console Baud Rate       19200

                  ^E/^X/<Tab> to select or +/- to modify
                       <Esc> to return to main menu
```

6 On the BIOS configuration menu, use the **[Tab]** key to navigate through the menu options, and then use the **[+]** key to toggle the choices.

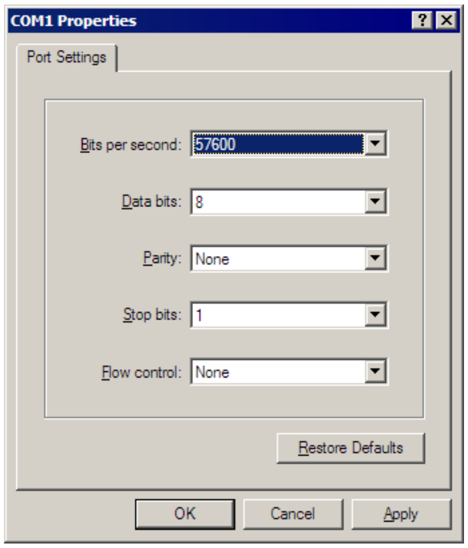
The options to change are:

- Console on Port 1: change to Enabled
- Console Baud Rate: change to 57600

7 Press **[Esc]** to return to the Main Menu.
8 Press **[Esc]** again to apply your changes and reboot the module.
9 Remove the module from the rack and disable the Setup jumper.

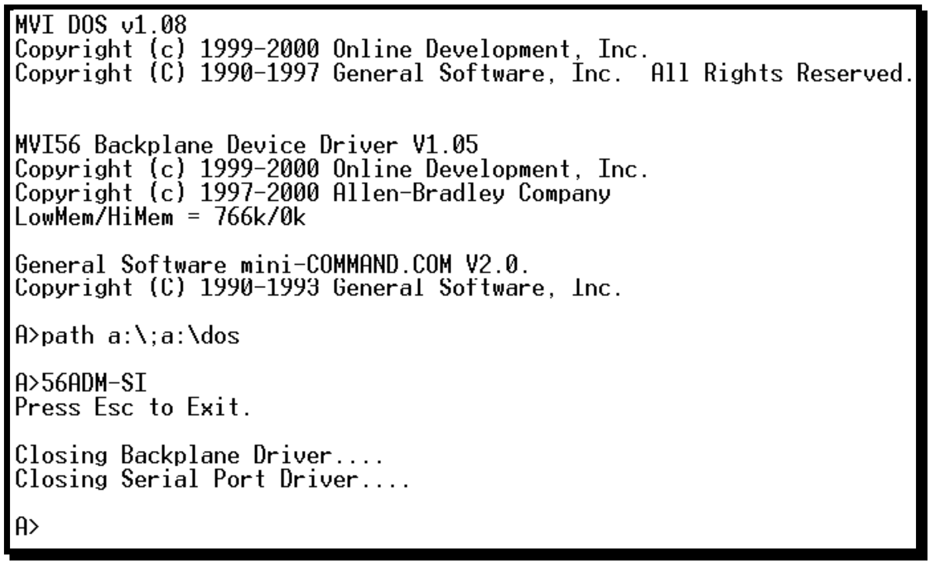
To communicate with the module in Console mode

1 Change the connection settings in HyperTerminal from 19200 to 57600, and then reconnect to the module.



Bits per second: 57600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

2 Press **[Esc]** to exit the program and return to the command prompt.



```
MVI DOS v1.08
Copyright (c) 1999-2000 Online Development, Inc.
Copyright (C) 1990-1997 General Software, Inc.  All Rights Reserved.


MVI56 Backplane Device Driver V1.05
Copyright (c) 1999-2000 Online Development, Inc.
Copyright (c) 1997-2000 Allen-Bradley Company
LowMem/HiMem = 766k/0k

General Software mini-COMMAND.COM V2.0.
Copyright (C) 1990-1993 General Software, Inc.

A>path a:\;a:\dos

A>56ADM-SI
Press Esc to Exit.

Closing Backplane Driver....
Closing Serial Port Driver....

A>
```

Important: The autoexec.bat in the image file must allow the application to exit to a DOS prompt.

8.5.5 Installing RY.exe and SY.exe

To install RY.exe and SY.exe on the module, remove the Compact Flash card from the module, and then use a Compact Flash card reader on your PC to copy the files to the root directory of the Compact Flash card. When you reinsert the Compact Flash card in the module, use the following syntax to send or receive files.

```
C:\RY
```

or

```
C:\SY "filename.ext"
```

The filename and path must be in quotes.

Important: You cannot copy files directly to the A:\ drive on the module. To update files on the A drive, you must create a new ROM image and download the image to the module using MVIFlashUpdate. (page 84) The following procedures show how to send and receive files from the module’s Compact Flash card (drive C:\).

8.5.6 Downloading Files From a PC to the ADM Module

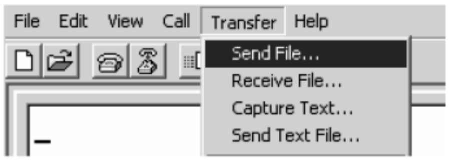
In order to download files to the module, the ADM module's running program must be interrupted. To transfer files to the module, run the RY.EXE program which uses the YModem protocol.

1 In HyperTerminal, connect to the module at 57600 baud and type the command to halt the program (for example **[Esc]** or **[Ctrl][C]**; your application must be written to allow itself to exit to the command prompt on request).

2 At the command prompt, type

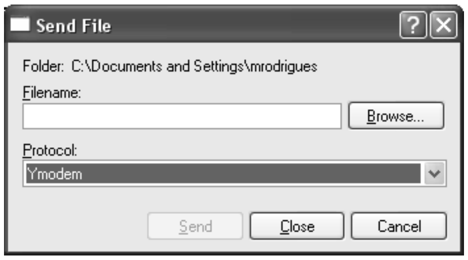
```
C:\RY
```

3 In HyperTerminal, open the Transfer menu, and then choose Send File.

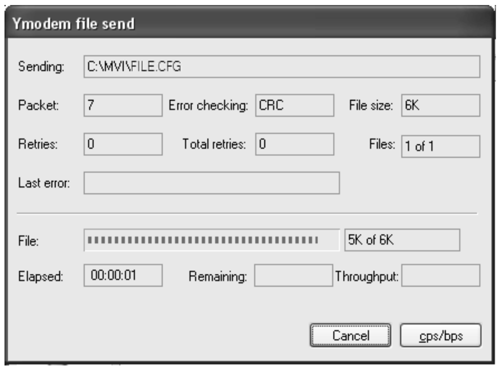


4 Click the Browse button to navigate to the folder and file to send to the module.

5 Chose Ymodem from the Protocol dropdown list, and then click Send.


Send File
Folder: C:\Documents and Settings\mrodrigues
Filename:
Protocol: Ymodem

6 The Ymodem File Send dialog box shows the file transfer size and remaining time.


Ymodem file send
Sending: C:\MVI\FILE.CFG
Packet: 7 Error checking: CRC File size: 6K
Retries: 0 Total retries: 0 Files: 1 of 1
Last error:
File: 5K of 6K
Elapsed: 00:00:01 Remaining: Throughput:

When the file has been transferred to the module, the dialog box will indicate that the transfer is complete.

8.5.7 Uploading files from the ADM module to a PC

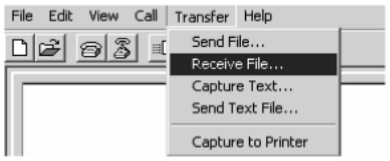
In order to upload files from the module, the ADM module's running program must be interrupted. You must run the SY.EXE program which uses the YModem protocol.

1 In HyperTerminal, connect to the module at 57600 baud and type the command to halt the program (for example **[Esc]** or **[Ctrl][C]**; your application must be written to allow itself to exit to the command prompt on request).

2 At the command prompt, type

```
C:\SY "filename.ext"
```

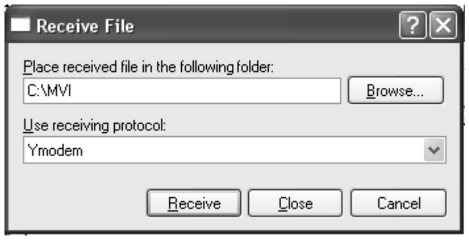
The filename and path must be in quotes.



3 From the **Transfer** menu in HyperTerminal, select **Receive File**. This action opens the Receive File dialog box.

4 Use the Browse button to choose a folder on your computer to save the file,

5 Select Ymodem as the receiving protocol, and then click the Receive button.


Receive File

Place received file in the following folder:
C:\MVI

Use receiving protocol:
Ymodem

When the file has been transferred to your PC, the dialog box will indicate that the transfer is complete.

8.6 Debugging Strategies

For simple debugging, printf's may be inserted into the module application to display debugging information on the console connected to PRT1.

9 Application Development Libraries

In This Chapter

9.1 ADM API Functions

This section provides detailed programming information for each of the ADM API library functions. The calling convention for each API function is shown in 'C' format.

API library routines are categorized according to functionality.

Function Category	Function Name	Description
Initialization	ADM_Open	Initialize access to the API
	ADMClose	Terminate access to the API
Debug Port	ADM_ProcessDebug	Debug port user interface
	ADM_DAWriteSendCtl	Writes a data analyzer Tx control symbol
	ADM_DAWriteRecvCtl	Writes a data analyzer Rx control symbol
	ADM_DAWriteSendData	Writes a data analyzer Tx data byte
	ADM_DAWriteRecvData	Writes a data analyzer Rx data byte
	ADM_ConPrint	Outputs characters to Debug port
	ADM_CheckDBPort	Checks for character input on Debug port
Database	ADM_DBOpen	Initializes database
	ADM_DBClose	Closes database
	ADM_DBZero	Zeros database
	ADM_DBGetBit	Read a bit from the database
	ADM_DBSetBit	Write a 1 to a bit to the database
	ADM_DBClearBit	Write a 0 to a bit to the database
	ADM_DBGetByte	Read a byte from the database
	ADM_DBSetByte	Write a byte to the database
	ADM_DBGetWord	Read a word from the database
	ADM_DBSetWord	Write a word to the database
	ADM_DBGetLong	Read a double word from the database
	ADM_DBSetLong	Write a double word to the database
	ADM_DBGetFloat	Read a floating-point number from the database
	ADM_DBSetFloat	Write a floating-point number to the database
	ADM_DBGetDFloat	Read a double floating-point number from the database
	ADM_DBSetDFloat	Write a double floating-point number to the database
	ADM_DBGetBuff	Reads a character buffer from the database
	ADM_DBSetBuff	Writes a character buffer to the database
	ADM_DBGetRegs	Read multiple word registers from the database
	ADM_DBSetRegs	Write multiple word registers to the database
	ADM_DBGetString	Read a string from the database
	ADM_DBSetString	Write a string to the database
	ADM_DBSwapWord	Swaps bytes within a word in the database
	ADM_DBSwapDWord	Swaps bytes within a double word in the database

Function Category	Function Name	Description
	ADM_GetDBCptr	Get a pointer to a character in the database
	ADM_GetDBIptr	Get a pointer to a word in the database
	ADM_GetDBInt	Returns an integer from the database
	ADM_DBChanged	Tests a database register for a change
	ADM_DBBitChanged	Tests a database bit for a change
	ADM_DBOR_Byte	Inclusive OR a byte with a database byte
	ADM_DBNOR_Byte	Inclusive NOR a byte with a database byte
	ADM_DBAND_Byte	AND a byte with a database byte
	ADM_DBNAND_Byte	NAND a byte with a database byte
	ADM_DBXOR_Byte	Exclusive OR a byte with a database byte
	ADM_DBXNOR_Byte	Exclusive NOR a byte with a database byte
Timer	ADM_StartTimer	Initialize a timer
	ADM_CheckTimer	Check current timer value
Backplane	ADM_BtOpen	Opens and initializes backplane interface
	ADM_BtClose	Closes backplane interface
	ADM_BtNext	Sets next write block number
	ADM_ReadBtCfg	Reads configuration from the processor
	ADM_BtFunc	Handles backplane transfers
	ADM_SetStatus	Writes status to Error/Status table
	ADM_SetBtStatus	Writes status to processor
LED	ADM_SetLed	Turn user LED indicators on or off
Flash	ADM_FileGetString	Searches for a string in a config file
	ADM_FileGetInt	Searches for an integer in a config file
	ADM_FileGetChar	Searches for a char in a config file
	ADM_GetVal	Gets an integer from a buffer
	ADM_GetStr	Gets a string from a buffer
	ADM_Getc	Gets a char from a buffer
	ADM_SkipToNext	Skips white space
Miscellaneous	ADM_GetVersionInfo	Get the ADM API version information
	ADM_SetConsolePort	Enable the console on a port
	ADM_SetConsoleSpeed	Set the console port baud rate
RAM	ADM_EEPROM_ReadConfiguration	Read configuration file.
	ADM_RAM_Find_Section	Find section in the configuration file.
	ADM_RAM_GetString	Get String under topic name.
	ADM_RAM_GetInt	Get Integer under topic name.
	ADM_RAM_GetLong	Get Long under topic name.
	ADM_RAM_GetFloat	Get Float under topic name.
	ADM_RAM_GetDouble	Get Double under topic name.
	ADM_RAM_GetChar	Get Char under topic name.
	ADM_Get_BP_Data_Exchange	Get Control Data Exchange

9.2 ADM API Initialization Functions

ADM_Open

Syntax

```
int ADM_Open(ADMHANDLE *adm_handle);
```

Parameters

adm_handle	Pointer to variable of type ADMHANDLE

Description

ADM_Open acquires access to the ADM API and sets *adm_handle* to a unique ID that the application uses in subsequent functions. This function must be called before any of the other API functions can be used.

IMPORTANT: After the API has been opened, ADM_Close should always be called before exiting the application.

Return Value

ADM_SUCCESS	API was opened successfully
ADM_ERR_REOPEN	API is already open
ADM_ERR_NOACCESS	API cannot run on this hardware

Note: ADM_ERR_NOACCESS will be returned if the hardware is not from ProSoft Technology.

Example

```
ADMHANDLE      adm_handle;
   if(ADM_Open(&adm_handle) != ADM_SUCCESS)
   {
      printf("\nFailed to open ADM API... exiting program\n");
      exit(1);
   }
```

See Also

ADM_Close (page 101)

ADM_Close

Syntax

```
int ADM_Close(ADMHANDLE adm_handle);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open

Description

This function is used by an application to release control of the API. *adm_handle* must be a valid handle returned from ADM_Open.

IMPORTANT: After the API has been opened, this function should always be called before exiting the application.

Return Value

ADM_SUCCESS	API was closed successfully
ADM_ERR_NOACCESS	*adm_handle* does not have access

Example

```
ADMHANDLE      adm_handle;
   ADM_Close(adm_handle);
```

See Also

ADM_Open (page 100)

9.3 ADM API Debug Port Functions

ADM_ProcessDebug

Syntax

```
int ADM_ProcessDebug(ADMHANDLE adm_handle, ADM_INTERFACE
*adm_interface_ptr);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures

Description

This function provides a module user interface using the debug port. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access or user pressed **Esc** to exit program

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_INTERFACE    interface;
   interface_ptr = &interface;
ADM_ProcessDebug(adm_handle, interface_ptr);
```

ADM_DAWriteSendCtl

Syntax

```
int ADM_DAWriteSendCtl(ADMHANDLE adm_handle, ADM_INTERFACE
*adm_interface_ptr, int app_port, int marker);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure which contains structure pointers needed by the function
app_port	Application serial port referenced
marker	Flow control symbol to output to the data analyzer screen

Description

This function may be used to send a transmit flow control symbol to the data analyzer screen. The control symbol will appear between two angle brackets: <R+>, <R->, <CS>.

adm_handle must be a valid handle returned from ADM_Open.

Valid values for marker are:

RTSOFF	<R->
RTSON	<R+>
CTSRCV	<CS>

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*adm_handle* does not have access
MVI_ERR_BADPARAM	Value of marker is not valid

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
ADM_DAWriteSendCtl(adm_handle, interface_ptr, app_port, RTSON);
```

See Also

ADM_DAWriteRecvCtl (page 104)

ADM_DAWriteRecvCtl

Syntax

```
int ADM_DAWriteRecvCtl(ADMHANDLE adm_handle, ADM_INTERFACE
*adm_interface_ptr, int app_port, int marker);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure which contains structure pointers needed by the function
app_port	Application serial port referenced
marker	Flow control symbol to output to the data analyzer screen

Description

This function may be used to send a receive flow control symbol to the data analyzer screen. The control symbol will appear between two square brackets: [R+], [R-], [CS].

adm_handle must be a valid handle returned from ADM_Open.

Valid values for marker are:

RTSOFF	[R-]
RTSON	[R+]
CTSRCV	[CS]

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*adm_handle* does not have access
MVI_ERR_BADPARAM	Value of marker is not valid

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
ADM_DAWriteRecvCtl(adm_handle, interface_ptr, app_port, RTSON);
```

See Also

ADM_DAWriteSendCtl (page 103)

ADM_DAWriteSendData

Syntax

```
int ADM_DAWriteSendData(ADMHANDLE adm_handle, ADM_INTERFACE
*adm_interface_ptr, int app_port, int length, char *data_buff);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure which contains structure pointers needed by the function
app_port	Application serial port referenced
length	The number of data characters to send to the data analyzer
data_buff	The buffer holding the transmit data

Description

This function may be used to send transmit data to the data analyzer screen. The data will appear between two angle brackets: <data>.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*adm_handle* does not have access

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_PORT      ports[MAX_APP_PORTS];
Int         app_port;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
ADM_DAWriteSendData(adm_handle, interface_ptr, app_port,
ports[app_port].len, ports[app_port].buff);
```

See Also

ADM_DAWriteRecvData (page 106)

ADM_DAWriteRecvData

Syntax

```
int ADM_DAWriteRecvData(ADMHANDLE adm_handle, ADM_INTERFACE
*adm_interface_ptr, int app_port, int length, char *data_buff);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure which contains structure pointers needed by the function
app_port	Application serial port referenced
length	The number of data characters to send to the data analyzer
data_buff	The buffer holding the receive data

Description

This function sends receive data to the data analyzer screen. The data will appear between two square brackets: [data].

adm_handle must be a valid handle returned from ADM_Open.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*adm_handle* does not have access

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_PORT      ports[MAX_APP_PORTS];
Int         app_port;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
ADM_DAWriteRecvData(adm_handle, interface_ptr, app_port,
ports[app_port].len, ports[app_port].buff);
```

See Also

ADM_DAWriteSendData (page 105)

ADM_ConPrint

Syntax

```
nt ADM_ConPrint(ADMHANDLE adm_handle, ADM_INTERFACE *adm_interface_ptr);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures

Description

This function outputs characters to the debug port. This function will buffer the output and allow other functions to run. The buffer is serviced with each call to ADM_ProcessDebug and can be serviced by the user's program. When sending data to the debug port, if printf statements are used, other processes will be held up until the printf function completes execution. Two variables in the interface structure must be set when data is loaded. The first, buff_ch is the offset of the next character to print. This should be set to 0. The second is buff_len. This should be set to the length of the string placed in the buffer.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_ERR_NOACCESS	*adm_handle* does not have access
	Number of characters left in the buffer

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
sprintf(interface.buff,"MVI ADM\n");
   interface.buff_ch = 0;
   interface.buff_len = strlen(interface.buff);
/* write buffer to console */
   while(interface.buff_len)
   {
      interface.buff_len = ADM_ConPrint(adm_handle, interface_ptr);
   }
```

ADM_CheckDBPort

Syntax

```
int ADM_CheckDBPort(ADMHANDLE adm_handle);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open

Description

This function checks for input characters on the debug port. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

ADM_ERR_NOACCESS	*adm_handle* does not have access

Returns the character input to the debug port

Example

```
int         key;
key = ADM_CheckDBPort(adm_handle);
printf("key = %i\n", key);
```

9.4 ADM API Database Functions

ADM_DBOpen

Syntax

```
int  ADM_DBOpen(ADMHANDLE adm_handle, unsigned short max_size)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
max_size	Maximum number of words in the database

Description

This function creates a database in the RAM area of the PTQ-ADM module. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_DB_MAX_SIZE	max_size has exceeded the maximum allowed
ADM_ERR_REG_RANGE	max_size requested was zero
ADM_ERR_OPEN	Database already created
ADM_ERR_MEMORY	Insufficient memory for database

Example

```
ADMHANDLE      adm_handle;
if(ADM_DBOpen(adm_handle, ADM_MAX_DB_REGS) != ADM_SUCCESS)
      printf("Error setting up Database!\n");
```

See Also

ADM_DBClose (page 110)

ADM_DBClose

Syntax

```
int  ADM_DBClose(ADMHANDLE adm_handle)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open

Description

This function closes a database previously created by ADM_DBOpen.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access

Example

```
ADMHANDLE      adm_handle;
ADM_DBClose(adm_handle);
```

See Also

ADM_DBOpen (page 109)

ADM_DBZero

Syntax

```
int  ADM_DBZero(ADMHANDLE adm_handle)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open

Description

This function writes zeros to a database previously created by ADM_DBOpen. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated

Example

```
ADMHANDLE      adm_handle;
ADM_DBZero(adm_handle);
```

See Also

ADM_DBOpen (page 109)

ADM_DBGetBit

Syntax

```
int  ADM_DBGetBit(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Bit offset into database

Description

This function reads a bit from the database at a specified bit offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Requested bit

ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
if(ADM_DBGetBit(adm_handle, offset))
   printf("bit is set");
else
   printf("bit is clear");
```

ADM_DBSetBit

Syntax

```
int  ADM_DBSetBit(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Bit offset into database

Description

This function sets a bit to a 1 in the database at a specified bit offset. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
ADM_DBSetBit(adm_handle, offset);
```

See Also

ADM_DBClearBit (page 114)

ADM_DBClearBit

Syntax

```
int  ADM_DBClearBit(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Bit offset into database

Description

This function clears a bit to a 0 in the database at a specified bit offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
ADM_DBClearBit(adm_handle, offset);
```

See Also

ADM_DBSetBit (page 113)

ADM_DBGetByte

Syntax

```
char  ADM_DBGetByte(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database

Description

This function reads a byte from the database at a specified byte offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Requested byte

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
int         i;
i = ADM_DBGetByte(adm_handle, offset);
```

See Also

ADM_DBSetByte (page 116)

ADM_DBSetByte

Syntax

```
int  ADM_DBSetByte(ADMHANDLE adm_handle, unsigned short offset, const char
val)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database
val	Value to be written to the database

Description

This function writes a byte to the database at a specified byte offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
const char      val;
ADM_DBSetByte(adm_handle, offset, val);
```

See Also

ADM_DBGetByte (page 115)

ADM_DBGetWord

Syntax

```
int  ADM_DBGetWord(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Word offset into database

Description

This function reads a word from the database at a specified word offset. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

Requested word

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
int         i;
i = ADM_DBGetWord(adm_handle, offset);
```

See Also

ADM_DBSetWord (page 118)

ADM_DBSetWord

Syntax

```
int  ADM_DBSetWord(ADMHANDLE adm_handle, unsigned short offset, const short
val)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Word offset into database
val	Value to be written to the database

Description

This function writes a word to the database at a specified word offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
const short      val;
ADM_DBSetWord(adm_handle, offset, val);
```

See Also

ADM_DBGetWord (page 117)

ADM_DBGetLong

Syntax

```
long  ADM_DBGetLong(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Long int offset into database

Description

This function reads a long int from the database at a specified long int offset. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

Requested long int

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
long         l;
l = ADM_DBGetLong(adm_handle, offset);
```

See Also

ADM_DBSetLong (page 120)

ADM_DBSetLong

Syntax

```
int  ADM_DBSetLong(ADMHANDLE adm_handle, unsigned short offset, const long
val)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Long int offset into database
val	Value to be written to the database

Description

This function writes a long int to the database at a specified long int offset. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
const long      val;
ADM_DBSetLong(adm_handle, offset, val);
```

See Also

ADM_DBGetLong (page 119)

ADM_DBGetFloat

Syntax

```
float  ADM_DBGetFloat(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	float offset into database

Description

This function reads a floating-point number from the database at a specified float offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Requested floating-point number.

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
float         f;
f = ADM_DBGetFloat(adm_handle, offset);
```

See Also

ADM_DBSetFloat (page 122)

ADM_DBSetFloat

Syntax

```
int  ADM_DBSetFloat(ADMHANDLE adm_handle, unsigned short offset, const float
val)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	float offset into database
val	Value to be written to the database

Description

This function writes a floating-point number to the database at a specified float offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
const float      val;
ADM_DBSetFloat(adm_handle, offset, val);
```

See Also

ADM_DBGetFloat (page 121)

ADM_DBGetDFloat

Syntax

```
double  ADM_DBGetDFloat(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	double float offset into database

Description

This function reads a double floating-point number from the database at a specified double float offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Requested double floating-point number

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
double         d;
d = ADM_DBGetDFloat(adm_handle, offset);
```

See Also

ADM_DBSetDFloat (page 124)

ADM_DBSetDFloat

Syntax

```
int ADM_DBSetDFloat(ADMHANDLE adm_handle, unsigned short offset, const
double val)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	double float offset into database
val	Value to be written to the database

Description

This function writes a double floating-point number to the database at a specified double float offset.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
unsigned short   offset;
const double      val;
ADM_DBSetDFloat(adm_handle, offset, val);
```

See Also

ADM_DBGetDFloat (page 123)

ADM_DBGetBuff

Syntax

```
char * ADM_DBGetBuff(ADMHANDLE adm_handle, unsigned short offset, const
unsigned short count, char * str)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Character offset into database where the buffer starts
count	Number of characters to retrieve
str	String buffer to receive characters

Description

This function copies a buffer of characters in the database to a character buffer. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE           adm_handle;
unsigned short        offset;
const unsigned short   char_count;
char                *string_buff;
ADM_DBGetBuff(adm_handle, offset, char_count, string_buff);
```

See Also

ADM_DBSetBuff (page 126)

ADM_DBSetBuff

Syntax

```
int  ADM_DBSetBuff(ADMHANDLE adm_handle, unsigned short offset, const
unsigned short count, char * str)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Character offset into database where the buffer starts
count	Number of characters to write
str	String buffer to copy characters from

Description

This function copies a buffer of characters to the database.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

NULL	*adm_handle* has no access, the database is not allocated, or count + offset is beyond the max size of the database
	Characters from buffer

Example

```
ADMHANDLE         adm_handle;
unsigned short      offset;
const unsigned short   char_count;
char            *string_buff = "MVI ADM";
char_count = strlen(string_buff);
ADM_DBSetBuff(adm_handle, offset, char_count, string_buff);
```

See Also

ADM_DBGetBuff (page 125)

ADM_DBGetRegs

Syntax

```
unsigned short  * ADM_DBGetRegs(ADMHANDLE adm_handle, unsigned short offset,
const unsigned short count, unsigned short * buff)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Character offset into database where the buffer starts
count	Number of integers to retrieve
buff	Register buffer to receive integers

Description

This function copies a buffer of registers in the database to a register buffer.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Returns NULL if not successful.

Returns buff if successful.

Example

```
ADMHANDLE         adm_handle;
unsigned short      offset;
const unsigned short   reg_count;
unsigned short      *reg_buff;
ADM_DBGetRegs(adm_handle, offset, reg_count, reg_buff);
```

See Also

ADM_DBSetRegs (page 128)

ADM_DBSetRegs

Syntax

```
int  ADM_DBSetRegs(ADMHANDLE adm_handle, unsigned short offset, const
unsigned short count, unsigned short * buff)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Character offset into database where the buffer starts
count	Number of integers to write
buff	Register buffer from which integers are copied

Description

This function copies a buffer of registers to the database.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE         adm_handle;
unsigned short      offset;
const unsigned short   reg_count;
unsigned short      *reg_buff;
ADM_DBSetRegs(adm_handle, offset, reg_count, reg_buff);
```

See Also

ADM_DBGetRegs (page 127)

ADM_DBGetString

Syntax

```
char  * ADM_DBGetString(ADMHANDLE adm_handle, unsigned short offset, const
unsigned short maxcount, char * str)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Character offset into database where the buffer starts
maxcount	Maximum number of characters to retrieve
str	String buffer to receive characters

Description

This function copies a string from the database to a string buffer.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Returns NULL if not successful.

Returns str if string is copy is successful.

Example

```
ADMHANDLE         adm_handle;
unsigned short      offset;
const unsigned short   maxcount;
char            *string_buff;
ADM_DBGetString(adm_handle, offset, maxcount, str);
```

See Also

ADM_DBSetString (page 130)

ADM_DBSetString

Syntax

```
int  ADM_DBSetString(ADMHANDLE adm_handle, unsigned short offset, const
unsigned short maxcount, char * str)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Character offset into database where the buffer starts
maxcount	Maximum number of characters to write
str	String buffer to copy string from

Description

This function copies a string to the database from a string buffer.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE         adm_handle;
unsigned short      offset;
const unsigned short   maxcount;
char            *string_buff;
ADM_DBSetString(adm_handle, offset, maxcount, str);
```

See Also

ADM_DBGetString (page 129)

ADM_DBSwapWord

Syntax

```
int ADM_DBSwapWord(ADMHANDLE adm_handle, unsigned short offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Word offset into database where swapping is to be performed

Description

This function swaps bytes within a database word.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE          adm_handle;
unsigned short       offset;
ADM_DBSwapWord(adm_handle, offset);
```

ADM_DBSwapDWord

Syntax

```
int ADM_DBSwapDWord(ADMHANDLE adm_handle, unsigned short offset, int type)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	long offset into database where swapping is to be performed
type	If type = 3 then bytes will be swapped in pairs within the long.

Description

This function swaps bytes within a database long word.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE         adm_handle;
unsigned short      offset;
ADM_DBSwapDWord(adm_handle, offset, 3);
```

ADM_GetDBCptr

Syntax

```
char * ADM_GetDBCptr(ADMHANDLE adm_handle, int offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Word offset into database

Description

This function obtains a pointer to char corresponding to the database + offset location. Because offset is a word offset, the pointer will always reference a character on a word boundary.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Returns NULL if not successful.

Returns pointer to char if successful.

Example

```
ADMHANDLE      adm_handle;
int         offset;
char         c;
c = *(ADM_GetDBCptr(adm_handle, offset));
```

ADM_GetDBIptr

Syntax

```
int * ADM_GetDBIptr(ADMHANDLE adm_handle, int offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Word offset into database

Description

This function obtains a pointer to int corresponding to the database + offset location.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Returns NULL if not successful.

Returns pointer to int if successful.

Example

```
ADMHANDLE      adm_handle;
int         offset;
int         i;
i = *(ADM_GetDBIptr(adm_handle, offset));
```

ADM_GetDBInt

Syntax

```
int ADM_GetDBIptr(ADMHANDLE adm_handle, int offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Word offset into database

Description

This function obtains an int corresponding to the database + offset location. *adm_handle* must be a valid handle returned from ADM_Open.

Return Value

Returns 0 if not successful.

Returns int requested if successful.

Example

```
ADMHANDLE      adm_handle;
int         offset;
int         i;
i = ADM_GetDBInt(adm_handle, offset);
```

ADM_DBChanged

Syntax

```
int ADM_DBChanged(ADMHANDLE adm_handle, int offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Word offset into database

Description

This function checks to see if a register has changed since the last call to ADM_DBChanged.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

0	No change
1	Register has changed

Example

```
ADMHANDLE      adm_handle;
int         offset;
if(ADM_DBChanged(adm_handle, offset))
   printf("Data has changed");
else
   printf("Data is unchanged");
```

ADM_DBBitChanged

Syntax

```
int ADM_DBBitChanged(ADMHANDLE adm_handle, int offset)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Bit offset into database

Description

This function checks to see if a bit has changed since the last call to ADM_DBBitChanged.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

0	No change
1	Bit has changed

Example

```
ADMHANDLE      adm_handle;
int         offset;
if(ADM_DBBitChanged(adm_handle, offset))
   printf("Bit has changed");
else
   printf("Bit is unchanged");
```

ADM_DBOR_Byte

Syntax

```
int ADM_DBOR_Byte(ADMHANDLE adm_handle, int offset, unsigned char bval)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database
bval	Bit mask to be ORed with the byte at offset

Description

This function ORs a byte in the database with a byte-long bit mask.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
int         offset;
unsigned char      bval = 0x55;
ADM_DBOR_Byte(adm_handle, offset, bval);
```

ADM_DBNOR_Byte

Syntax

```
int ADM_DBNOR_Byte(ADMHANDLE adm_handle, int offset, unsigned char bval)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database
bval	Bit mask to be NORed with the byte at offset

Description

This function NORs a byte in the database with a byte-long bit mask.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
int         offset;
unsigned char      bval = 0x55;
ADM_DBNOR_Byte(adm_handle, offset, bval);
```

ADM_DBAND_Byte

Syntax

```
int ADM_DBAND_Byte(ADMHANDLE adm_handle, int offset, unsigned char bval)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database
bval	Bit mask to be ANDed with the byte at offset

Description

This function ANDs a byte in the database with a byte-long bit mask.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
int         offset;
unsigned char      bval = 0x55;
ADM_DBAND_Byte(adm_handle, offset, bval);
```

ADM_DBNAND_Byte

Syntax

```
int ADM_DBNAND_Byte(ADMHANDLE adm_handle, int offset, unsigned char bval)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database
bval	Bit mask to be NANDed with the byte at offset

Description

This function NANDs a byte in the database with a byte-long bit mask.
adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
int         offset;
unsigned char      bval = 0x55;
ADM_DBNAND_Byte(adm_handle, offset, bval);
```

ADM_DBXOR_Byte

Syntax

```
int ADM_DBXOR_Byte(ADMHANDLE adm_handle, int offset, unsigned char bval)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database
bval	Bit mask to be XORed with the byte at offset

Description

This function XORs a byte in the database with a byte-long bit mask.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
int         offset;
unsigned char      bval = 0x55;
ADM_DBXOR_Byte(adm_handle, offset, bval);
```

ADM_DBXNOR_Byte

Syntax

```
int ADM_DBXNOR_Byte(ADMHANDLE adm_handle, int offset, unsigned char bval)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
offset	Byte offset into database
bval	Bit mask to be XNORed with the byte at offset

Description

This function XNORs a byte in the database with a byte-long bit mask.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_MEMORY	database is not allocated
ADM_ERR_REG_RANGE	offset is out of range

Example

```
ADMHANDLE      adm_handle;
int         offset;
unsigned char      bval = 0x55;
ADM_DBXNOR_Byte(adm_handle, offset, bval);
```

9.5 ADM API Clock Functions

ADM_StartTimer

Syntax

```
unsigned short ADM_StartTimer(ADMHANDLE adm_handle)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open

Description

ADM_StartTimer can be used to initialize a variable with a starting time with the current time from a microsecond clock. A timer can be created by making a call to ADM_StartTimer and by using ADM_CheckTimer to check to see if timeout has occurred. For multiple timers call ADM_StartTimer using a different variable for each timer.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Current time value from millisecond clock

Example

Initialize 2 timers.

```
ADMHANDLE      adm_handle;
unsigned short   timer1;
unsigned short   timer2;
timer1 = ADM_StartTimer(adm_handle);
timer2 = ADM_StartTimer(adm_handle);
```

See Also

ADM_CheckTimer (page 145)

ADM_CheckTimer

Syntax

```
int ADM_CheckTimer(ADMHANDLE adm_handle, unsigned short *adm_tmlast, long
*adm_tmout)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open.
adm_tmlast	Starting time of timer returned from call to ADM_StartTimer.
adm_tmout	Timeout value in microseconds.

Description

ADM_CheckTimer checks a timer for a timeout condition. Each time the function is called, ADM_CheckTimer updates the current timer value in *adm_tmlast* and the time remaining until timeout in adm_tmout. If *adm_tmout* is less than 0, then a 1 is returned to indicate a timeout condition. If the timer has not expired, a 0 will be returned.

adm_handle must be a valid handle returned from ADM_Open.

Return Value

Timer not expired.

Timer expired.

Example

Check 2 timers.

```
ADMHANDLE      adm_handle;
unsigned short   timer1;
unsigned short   timer2;
long         timeout1;
long         timeout2;
timeout1 = 10000000L;   /* set timeout for 10 seconds */
timer1 = ADM_StartTimer(adm_handle);
/* wait until timer 1 times out */
while(!ADM_CheckTimer(adm_handle, &timer1, &timeout1))
timeout2 = 5000000L;   /* set timeout for 5 seconds */
timer2 = ADM_StartTimer(adm_handle);
/* wait until timer 2 times out */
while(!ADM_CheckTimer(adm_handle, &timer2, &timeout2))
```

See Also

ADM_StartTimer (page 144)

9.6 ADM API Backplane Functions

ADM_BtOpen

Syntax

```
int ADM_BtOpen(ADMHANDLE adm_handle, ADM_INTERFACE * adm_interface_ptr, int
verbose)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures
verbose	Switch to enable status messages to the debug port. A 1 will enable messages and a 0 will disable the messages.

Description

This function opens and initializes the backplane interface.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
Backplane error number	If there is an error writing to the backplane during initialization, the error code is returned.

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
int         verbose = 1;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
ADM_BtOpen(adm_handle, interface_ptr, verbose);
```

See Also

ADM_BtClose (page 147)

ADM_BtClose

Syntax

```
int ADM_BtClose(ADMHANDLE adm_handle, ADM_INTERFACE * adm_interface_ptr)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures

Description

This function closes the backplane interface.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
   ADM_BtClose(adm_handle, interface_ptr);
```

See Also

ADM_BtOpen (page 146)

ADM_BtNext

Syntax

```
int ADM_BtNext(ADMHANDLE adm_handle, ADM_INTERFACE * adm_interface_ptr)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures

Description

This function sets the next write block number.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_NOTSUPPORTED	Function is not supported on this platform

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
   ADM_BtNext(adm_handle, interface_ptr);
```

See Also

ADM_BtOpen (page 146)

ADM_ReadBtCfg

Syntax

```
int ADM_ReadBtCfg(ADMHANDLE adm_handle, ADM_INTERFACE * adm_interface_ptr,
int verbose)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures
verbose	Switch to enable status messages to the debug port. A 1 will enable messages and a 0 will disable the messages.

Description

This function reads the module configuration from the processor. The function will make a call to the function pointed to by `interface.process_cfg_ptr`. The user function can be used to perform boundary checking on the configuration parameters.

Return Value

ADM_SUCCESS	No errors were encountered
ADM_ERR_NOACCESS	*adm_handle* does not have access, or configuration was interrupted by operator.
ADM_ERR_NOTSUPPORTED	This function is not supported on this platform

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
int         verbose = 1;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
ADM_ReadBtCfg(adm_handle, interface_ptr, verbose);
```

See Also

ADM_BtOpen (page 146)

ADM_BtFunc

Syntax

```
int ADM_BtFunc(ADMHANDLE adm_handle, ADM_INTERFACE * adm_interface_ptr, int
verbose)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures
verbose	Switch to enable status messages to the debug port. A 1 will enable messages and a 0 will disable the messages.

Description

This function handles the transfer of data across the backplane.

Return Value

0	Block transfer was successful
1	Invalid block number received

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
int         verbose = 1;
ADM_INTERFACE      interface;
   interface_ptr = &interface;
   /* call backplane transfer logic */
   ADM_BtFunc(adm_handle, interface_ptr, verbose);
```

See Also

ADM_BtOpen (page 146)

ADM_SetStatus

Syntax

```
int ADM_SetStatus(ADMHANDLE adm_handle, ADM_INTERFACE * adm_interface_ptr,
int pass_cnt)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures
pass_cnt	Counter from user code to indicate module health. This counter could be updated in the main loop of the program.

Description

This function writes status data to the database at the location set by Error/Status Pointer in the module configuration. The data is written in the following order:

pass_cnt (in the ADM_INTERFACE structure)

ADM_PRODUCT (structure)

ADM_PORT_ERRORS (structure, 1 time for each application port)

ADM_BLK_ERRORS (structure)

Return Value

ADM_SUCCESS	The function has completed successfully.
ADM_ERR_NOACCESS	*adm_handle* does not have access

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
int         pass_cnt;
ADM_INTERFACE    interface;
   interface_ptr = &interface;
   ADM_SetStatus(adm_handle, interface_ptr, interface.pass_cnt);
```

See Also

ADM_SetBtStatus (page 152)

ADM_SetBtStatus

Syntax

```
int ADM_SetBtStatus(ADMHANDLE adm_handle, ADM_INTERFACE * adm_interface_ptr,
int pass_cnt)
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to ADM_INTERFACE structure to allow API access to structures
pass_cnt	Counter from user code to indicate module health. This counter could be updated in the main loop of the program.

Description

This function writes status data to the processor at word 202 in the input image and to the database at location 6670. The data is written in the following order:

pass_cnt (in the ADM_INTERFACE structure)

ADM_PRODUCT (structure)

ADM_PORT_ERRORS (structure, 1 time for each application port)

ADM_BLK_ERRORS (structure)

CurErr (port 1, from ADM_PORT structure)

LastErr (port 1, from ADM_PORT structure)

CurErr (port 2, from ADM_PORT structure)

LastErr (port 2, from ADM_PORT structure)

Return Value

ADM_SUCCESS	The function has completed successfully.
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_NOTSUPPORTED	This function is not supported on this platform

Example

```
ADMHANDLE      adm_handle;
ADM_INTERFACE      *interface_ptr;
int        pass_cnt;
ADM_INTERFACE    interface;
   interface_ptr = &interface;
   ADM_SetBtStatus(adm_handle, interface_ptr, interface.pass_cnt);
```

See Also

ADM_SetStatus (page 151)

9.7 ADM LED Functions

ADM_SetLed

Syntax

```
int ADM_SetLed(ADMHANDLE adm_handle, ADM_INTERFACE *adm_interface_ptr, int
led, int state);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_interface_ptr	Pointer to the interface structure
led	Specifies which of the user LED indicators is being addressed
state	Specifies whether the LED will be turned on or off

Description

ADM_SetLed allows an application to turn the user LED indicators on and off.

adm_handle must be a valid handle returned from ADM_Open.

led must be set to ADM_LED_USER1, ADM_LED_USER2 or ADM_LED_STATUS for User LED 1, User LED 2 or Status LED, respectively.

state must be set to ADM_LED_OK, ADM_LED_FAULT to turn the Status LED green or red, respectively. For User LED 1 and User LED 2 state must be set to ADM_LED_OFF or ADM_LED_ON to turn the indicator On or Off, respectively.

Return Value

ADM_SUCCESS	The LED has successfully been set.
ADM_ERR_NOACCESS	*adm_handle* does not have access
ADM_ERR_BADPARAM	led or state is invalid.

Example

```
ADMHANDLE      adm_handle;
/* Set Status LED OK, turn User LED 1 off and User LED 2 on */
ADM_SetLed(adm_handle, interface_ptr, ADM_LED_STATUS, ADM_LED_OK);
   ADM_SetLed(adm_handle, interface_ptr, ADM_LED_USER1, ADM_LED_OFF);
   ADM_SetLed(adm_handle, interface_ptr, ADM_LED_USER2, ADM_LED_ON);
```

)

9.8 ADM API Miscellaneous Functions

ADM_GetVersionInfo

Syntax

```
int ADM_GetVersionInfo(ADMHANDLE adm_handle, ADMVERSIONINFO *adm_verinfo);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
adm_verinfo	Pointer to structure of type ADMVERSIONINFO

Description

ADM_GetVersionInfo retrieves the current version of the ADM API library. The information is returned in the structure adm_verinfo. adm_handle must be a valid handle returned from ADM_Open.

The ADMVERSIONINFO structure is defined as follows:

```
typedef struct
{
   char   APISeries[4];
   short   APIRevisionMajor;
   short   APIRevisionMinor;
   long   APIRun;
}ADMVERSIONINFO;
```

Return Value

ADM_SUCCESS	The version information was read successfully.
ADI_ERR_NOACCESS	adm_handle does not have access

Example

```
ADMHANDLE      adm_handle;
ADMVERSIONINFO   verinfo;

/* print version of API library */
   ADM_GetVersionInfo(adm_handle, &adm_version);
printf("Revision %d.%d\n", verinfo.APIRevisionMajor,
verinfo.APIRevisionMinor);
```

ADM_SetConsolePort

Syntax

```
void ADM_SetConsolePort(int Port);
```

Parameters

Port	Com port to use as the console (COM1=0, COM2=1, COM3=2)

Description

ADM_SetConsolePort sets the specified communication port as the console. This allows the console to be disabled in the BIOS setup and the application can still configure the console for use.

Return Value

None

Example

```
/* enable console on COM1 */
ADM_SetConsolePort(COM1);
```

See Also

ADM_SetConsoleSpeed (page 156)

ADM_SetConsoleSpeed

Syntax

```
void ADM_SetConsoleSpeed(int Port, long Speed);
```

Parameters

Port	Com port to use as the console (COM1=0, COM2=1, COM3=2)
Speed	Baud rate for console port.
Available settings are: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600 and 115200.	

Description

ADM_SetConsoleSpeed sets the specified communication port to the baud rate specified.

Return Value

None

Example

```
/* set console to 115200 baud */
ADM_SetConsoleSpeed (COM1, 115200L);
```

See Also

ADM_SetConsolePort (page 155)

9.9 ADM API RAM Functions

ADM_RAM_GetString

Syntax

```
char huge ADM_RAM_GetString (ADMHANDLE adm_handle, char huge * mydata, char
* Topic);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
mydata	Pointer return from ADM_RAM_Find_Section.
Topic	Pointer to name of a variable.

Description

ADM_RAM_GetString tries to find the Topic name passed to the function in the file.

Return Value

Pointer to the string found in the file or NULL if the sub-section is not found.

Example

```
cptr = (char*)ADM_RAM_GetString(adm_handle, tptr, "Module Name");
if(cptr == NULL)
   strcpy(module.name, "No Module Name");
else
{
   if(strlen(cptr) > 80)
      *(cptr+80) = 0;
   strcpy(module.name, cptr);
   if(module.name[strlen(module.name)-1] < 32)
      module.name[strlen(module.name)-1] = 0;
}
```

ADM_RAM_GetInt

Syntax

```
unsigned short ADM_RAM_GetInt(ADMHANDLE adm_handle, char huge * mydata, char
* Topic);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
mydata	Pointer return from ADM_RAM_Find_Section.
Topic	Pointer to name of a variable.

Description

ADM_RAM_GetInt tries to find the Topic name passed to the function in the file.

Return Value

Value of type Integer found under the Topic name or 0 if the sub-section is not found.

Example

```
module.err_offset = ADM_RAM_GetInt(adm_handle, tptr, "Baud Rate");
if(module.err_offset < 0 || module.err_offset > module.max_regs-61)
{
   module.err_offset = -1;
   module.err_freq   =  0;
}
else
{
   module.err_freq   =  500;
}
```

ADM_RAM_GetLong

Syntax

```
unsigned long ADM_RAM_GetLong (ADMHANDLE adm_handle, char huge * mydata,
char * Topic);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
mydata	Pointer return from ADM_RAM_Find_Section.
Topic	Pointer to name of a variable.

Description

ADM_RAM_GetLong tries to find the Topic name passed to the function in the file.

Return Value

Value of a type Long found under the Topic name or 0 if the sub-section is not found.

Example

```
module.err_offset = ADM_RAM_GetLong(adm_handle, tptr, "Baud Rate");
if(module.err_offset < 0 || module.err_offset > module.max_regs-61)
{
   module.err_offset = -1;
   module.err_freq   =  0;
}
else
{
   module.err_freq   =  500;
}
```

ADM_RAM_GetFloat

Syntax

```
float ADM_RAM_GetFloat (ADMHANDLE adm_handle, char huge * mydata, char *
Topic);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
mydata	Pointer return from ADM_RAM_Find_Section.
Topic	Pointer to name of a variable.

Description

ADM_RAM_GetFloat tries to find the Topic name passed to the function in the file.

Return Value

Value of a type Float found under the Topic name or 0 if the sub-section is not found.

Example

```
module.time = ADM_RAM_GetFloat(adm_handle, tptr, "Time");
if(module.time < 0 || module.time > module.max_regs-61)
{
   module.time = -1;
   module.err_freq   =  0;
}
else
{
   module.err_freq   =  500;
}
```

ADM_RAM_GetDouble

Syntax

```
double ADM_RAM_GetDouble(ADMHANDLE adm_handle, char huge * mydata, char *
Topic);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
mydata	Pointer return from ADM_RAM_Find_Section.
Topic	Pointer to name of a variable.

Description

ADM_RAM_GetDouble tries to find the Topic name passed to the function in the file.

Return Value

Value of a type Double found under the Topic name or 0 if the sub-section is not found.

Example

```
module.time = ADM_RAM_GetDouble(adm_handle, tptr, "Time");
if(module.time < 0 || module.time > module.max_regs-61)
{
   module.time = -1;
   module.err_freq   =  0;
}
else
{
   module.err_freq   =  500;
}
```

ADM_RAM_GetChar

Syntax

```
unsigned char ADM_RAM_GetChar (ADMHANDLE adm_handle, char huge * mydata,
char * Topic);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open
mydata	Pointer return from ADM_RAM_Find_Section.
Topic	Pointer to name of a variable.

Description

ADM_RAM_GetChar tries to find the Topic name passed to the function in the file.

Return Value

Character found under the Topic name or ' ' if the sub-section is not found.

Example

```
module.enable = ADM_RAM_GetChar(adm_handle, tptr, "Enable");
if(module.enable == ' ')
{
   module.time = -1;
   module.err_freq   =  0;
}
else
{
   module.err_freq   =  500;
}
```

ADM_Get_BP_Data_Exchange

Syntax

```
void ADM_Get_BP_Data_Exchange (ADMHANDLE adm_handle);
```

Parameters

adm_handle	Handle returned by previous call to ADM_Open

Description

ADM_Get_BP_Data_Exchange read the whole, [Backplane Data Exchange], section, and load all variable for communication between Quantum PLC and module.

Return Value

None

Example

```
ADM_Get_BP_Data_Exchange(adm_handle);
```

10 Backplane API Functions

In This Chapter

This section provides detailed programming information for each of the API library functions. The calling convention for each API function is shown in 'C' format.

The API library routines are categorized according to functionality as follows:

Initialization

MVIbp_Open
MVIbp_Close

Configuration

MVIbp_GetIOConfig
MVIbp_SetIOConfig

Synchronization

MVIbp_WaitForInputScan
MVIbp_WaitForOutputScan

Direct I/O Access

MVIbp_ReadOutputImage
MVIbp_WriteInputImage

Messaging

MVIbp_ReceiveMessage
MVIbp_SendMessage

Miscellaneous

MVIbp_GetVersionInfo
MVIbp_ErrorString
MVIbp_SetUserLED
MVIbp_SetModuleStatus
MVIbp_GetSetupMode
MVIbp_GetConsoleMode
MVIbp_SetConsoleMode
MVIbp_GetModuleInfo
MVIbp_GetProcessorStatus
MVIbp_Sleep

Platform Specific

MVIbp_WriteModuleFile
MVIbp_ReadModuleFile
MVIbp_SetModuleInterrupt

10.1 Backplane API Initialization Functions

MVIbp_Open

Syntax

```
int MVIbp_Open(MVIHANDLE *handle);
```

Parameters

handle	Pointer to variable of type MVIHANDLE

Description

MVIbp_Open acquires access to the API and sets handle to a unique ID that the application uses in subsequent functions. This function must be called before any of the other API functions can be used.

IMPORTANT: After the API has been opened, MVIbp_Close should always be called before exiting the application.

Return Value

MVI_SUCCESS	API was opened successfully
MVI_ERR_REOPEN	API is already open
MVI_ERR_NODEVICE	Backplane driver could not be accessed

Note: MVI_ERR_NODEVICE will be returned if the backplane device driver is not loaded.

Example

```
MVIHANDLE    Handle;

if ( MVIbp_Open(&Handle) != MVI_SUCCESS) {
    printf("Open failed!\n");
} else {
    printf("Open succeeded\n");
}
```

See Also

MVIbp_Close (page 168)

MVIbp_Close

Syntax

```
int MVIbp_Close(MVIHANDLE handle);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open

Description

This function is used by an application to release control of the API. handle must be a valid handle returned from MVIbp_Open.

IMPORTANT: After the API has been opened, this function should always be called before exiting the application.

Return Value

MVI_SUCCESS	API was closed successfully
MVI_ERR_NOACCESS	Handle does not have access

Example

```
MVIHANDLE    Handle;

MVIbp_Close(Handle);
```

See Also

MVIbp_Open (page 167)

10.2 Backplane API Configuration Functions

MVIbp_GetIOConfig

Syntax

```
int MVIbp_GetIOConfig(MVIHANDLE handle, MVIBPIOCONFIG *ioconfig);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
ioconfig	Pointer to MVIBPIOCONFIG structure to receive configuration information

Description

This function obtains the I/O configuration of the PTQ module. handle must be a valid handle returned from MVIbp_Open.

The MVIBPIOCONFIG structure is defined as shown:

```
typedef struct tagMVIBPIOCONFIG
{
  WORD   TotalInputSize;    // Size of entire input image in words
  WORD   TotalOutputSize;   // Size of entire output image in words
  WORD   DirectInputSize;   // Input words available for direct access
  WORD   DirectOutputSize;  // Output words available for direct access
  WORD   MsgRcvBufSize;     // Max size in words for received messages
  WORD   MsgSndBufSize;     // Max size in words for sent messages
} MVIBPIOCONFIG;
```

The sizes in words of the module's input and output images are returned in the MVIBPIOCONFIG structure pointed to by ioconfig. The TotalInputSize and TotalOutputSize members are set equal to the size of the entire input or output image, respectively. The DirectInputSize and DirectOutputSize members are set equal to the number of words of the respective image that is available for direct access via the MVIbp_WriteInputImage or MVIbpReadOutputImage functions. By default, the direct and total sizes are equal. Refer to the MVIbp_SetIOConfig function for more information.

The MsgRcvBufSize and MsgSndBufSize members indicate the maximum size in words for received or sent messages, respectively. By default, these values are both zero, indicating that messaging is disabled. Refer to the MVIbp_SetIOConfig function for more information.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	handle does not have access

Example

```
MVIHANDLE          handle;
MVIBPIOCONFIG      ioconfig;

MVIbp_GetIOConfig(handle, &ioconfig);
printf("%d words of input image available\n", ioconfig.DirectInputSize);
printf("%d words of output image available\n", ioconfig.DirectOutputSize);
```

See Also

MVIbp_SetIOConfig (page 171)

MVIbp_SetIOConfig

Syntax

```
int MVIbp_SetIOConfig(MVIHANDLE handle, MVIBPIOCONFIG *ioconfig);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
ioconfig	Pointer to MVIBPIOCONFIG structure which contains configuration information

Description

This function defines the portion of the module's I/O images that will be used for direct I/O access, and to enable messaging. handle must be a valid handle returned from MVIbp_Open.

By default, all of the module's I/O image is available for direct I/O access, and messaging is disabled. The MVIbp_SetIOConfig may be used to limit the amount of I/O image available for direct access to only that which the application expects to use. Attempts to access I/O outside of the range defined by this function will result in an error.

If the application is to use the messaging functions (MVIbp_SendMessage and MVIbp_ReceiveMessage), MVIbp_SetIOConfig must be called to enable messaging and setup the maximum message size that will be allowed. The message size is expressed in words.

The MVIBPIOCONFIG structure is defined as shown:

```
typedef struct tagMVIBPIOCONFIG
{
  WORD   TotalInputSize;     // Size of entire input image in words
  WORD   TotalOutputSize;    // Size of entire output image in words
  WORD   DirectInputSize;    // Input words available for direct access
  WORD   DirectOutputSize;   // Output words available for direct access
  WORD   MsgRcvBufSize;      // Max size in words for received messages
  WORD   MsgSndBufSize;      // Max size in words for sent messages
} MVIBPIOCONFIG;
```

The TotalInputSize and TotalOutputSize members are ignored by the API, since the total I/O image sizes cannot be changed by the application. The DirectInputSize and DirectOutputSize members should be set equal to the number of words of the respective image that will be used for direct access via the MVIbp_WriteInputImage or MVIbpReadOutputImage functions.

To enable the module to receive messages from the control processor via the MVIbp_ReceiveMessage function, the MsgRcvBufSize member should be set to the maximum message size expected. Likewise, to enable the module to send messages to the control processor via the MVIbp_SendMessage function, the MsgSndBufSize member should be set to the maximum message size expected. The message sizes are expressed in words. The combined maximum message size is 2048 words. If the sum of MsgRcvBufSize and MsgSndBufSize exceeds 2048, the error MVI_ERR_BADCONFIG will be returned.

Notes

If messaging is enabled, a portion of the input and output images must be reserved for use by the messaging protocol. One word of input and one word of output is required for messaging control. At least one additional word of input and/or output is required for messaging data, depending upon the messaging direction(s) enabled. To receive messages from the control processor, at least one word of output image is required for messaging data. To send messages to the control processor, at least one word of input image is required for messaging data. Therefore, for bi-directional messaging, at least two words of input and two words of output image must be left unallocated when the direct I/O sizes are specified. If messaging is enabled and insufficient I/O image is available for messaging, the error MVI_ERR_BADCONFIG will be returned.

For best messaging performance, set the direct I/O sizes as small as possible.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_BADCONFIG	Configuration is not valid
MVI_ERR_NOTSUPPORTED	Always returns this error

Example

```
MVIHANDLE       handle;
MVIBPIOCONFIG      ioconfig;

ioconfig.DirectInputSize = 2;    // 2 words used for input
ioconfig.DirectOutputSize = 1;   // 1 word used for output
MsgSndBufSize = 256;             // Enable 256 word (max) messages to
processor
MsgRcvBufSize = 0;               // Received messages not enabled
if (MVI_SUCCESS != MVIbp_SetIOConfig(handle, &ioconfig))
    printf("Error: I/O configuration failed\n");
```

See Also

MVIbp_GetIOConfig (page 169)

10.3 Backplane API Synchronization Functions

MVIbp_WaitForInputScan

Syntax

```
int MVIbp_WaitForInputScan(MVIHANDLE handle, WORD timeout);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
timeout	Maximum number of milliseconds to wait for scan

Description

MVIbp_WaitForInputScan allows an application to synchronize with the scan of the module's input image. This function will return immediately after the input image has been read.

handle must be a valid handle returned from MVIbp_Open. timeout specifies the number of milliseconds that the function will wait for the input scan to occur.

Return Value

MVI_SUCCESS	The input scan has occurred.
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_TIMEOUT	The timeout expired before an input scan occurred.

Example

```
MVIHANDLE           Handle;

/* Wait here until input scan, 50ms timeout */
rc = MVIbp_WaitForInputScan(Handle, 50);
if (rc == MVI_ERR_TIMEOUT)
    printf("Input scan did not occur within 50 milliseconds\n");
else
    printf("Input scan has occurred\n");
```

See Also

MVIbp_WaitForOutputScan (page 174)

MVIbp_WaitForOutputScan

Syntax

```
int MVIbp_WaitForOutputScan(MVIHANDLE handle, WORD timeout);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
timeout	Maximum number of milliseconds to wait for scan

Description

MVIbp_WaitForInputScan allows an application to synchronize with the scan of the module's output image. This function will return immediately after the module's output image has been written.

handle must be a valid handle returned from MVIbp_Open. timeout specifies the number of milliseconds that the function will wait for the output scan to occur.

Return Value

MVI_SUCCESS	The output scan has occurred.
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_TIMEOUT	The timeout expired before an output scan occurred.
MVI_ERR_BADCONFIG	The data connection is not open

Example

```
MVIHANDLE    Handle;
int       rc;

/* Wait here until output scan, 50ms timeout */
rc = MVIbp_WaitForOutputScan(Handle, 50);
if (rc == MVI_ERR_TIMEOUT)
    printf("Output scan did not occur within 50ms\n");
else
    printf("Output scan has occurred\n");
```

See Also

MVIbp_WaitForInputScan (page 173)

10.4 Backplane API Direct I/O Access

MVIbp_ReadOutputImage

Syntax

```
int MVIbp_ReadOutputImage(MVIHANDLE handle, WORD *buffer, WORD offset, WORD
length);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
buffer	Pointer to buffer to receive data from output image
offset	Word offset into output image at which to begin reading
length	Number of words to read

Description

MVIbp_ReadOutputImage reads from the module's output image. handle must be a valid handle returned from MVIbp_Open.

buffer must point to a buffer of at least length words in size.

offset specifies the word in the output image to begin reading, and length specifies the number of words to read. The error MVI_ERR_BADPARAM will be returned if an attempt is made to access the output image beyond the range configured for direct I/O. Refer to the MVIbp_SetIOConfig function for more information.

The output image is written by the control processor and read by the module.

Return Value

MVI_SUCCESS	The data was read from the output image successfully.
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_BADPARAM	Parameter contains invalid value
MVI_ERR_BADCONFIG	The data connection is not open.

Example

```
MVIHANDLE    Handle;
WORD      buffer[8];
int       rc;

/* Read 8 words of data from the output image, starting with word 2 */
rc = MVIbp_ReadOutputImage(Handle, buffer, 2, 8);
if (rc != MVI_SUCCESS)
    printf("ERROR: MVIbp_ReadOutputImage failed");
```

See Also

MVIbp_SetIOConfig (page 171)

MVIbp_WriteInputImage (page 176)

MVIbp_WriteInputImage

Syntax

```
int MVIbp_WriteInputImage(MVIHANDLE handle, WORD *buffer, WORD offset, WORD
length);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
buffer	Pointer to buffer of data to be written to input image
offset	Word offset into input image at which to begin writing
length	Number of words to write

Description

MVIbp_WriteInputImage writes to the module's input image. handle must be a valid handle returned from MVIbp_Open.

buffer must point to a buffer of at least length words in size.

offset specifies the word in the input image to begin writing, and length specifies the number of words to write. The error MVI_ERR_BADPARAM will be returned if an attempt is made to access the input image beyond the range configured for direct I/O. If this error is returned, no data will be written to the input image. Refer to the MVIbp_SetIOConfig function for more information.

The input image is written by the module and read by the control processor.

Return Value

MVI_SUCCESS	The data was written to the input image successfully.
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_BADPARAM	Parameter contains invalid value
MVI_ERR_BADCONFIG	The data connection is not open

Example

```
MVIHANDLE    Handle;
WORD      buffer[2];
int       rc;

/* Write 2 words of data to the input image, starting with word 0 */
rc = MVIbp_WriteInputImage(Handle, buffer, 0, 2);
if (rc != MVI_SUCCESS)
    printf("ERROR: MVIbp_WriteInputImage failed");
```

See Also

MVIbp_SetIOConfig (page 171)

MVIbp_ReadOutputImage (page 175)

10.5 Backplane API Messaging Functions

MVIbp_ReceiveMessage

Syntax

```
int MVIbp_ReceiveMessage(MVIHANDLE handle, WORD *buffer, WORD *length, WORD
reserved, WORD timeout);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
buffer	Pointer to buffer to receive message data from processor
length	Pointer to a variable containing the maximum message length in words. When this function is called, this should be set to the size of the indicated buffer. Upon successful return, this variable will contain the actual received message length.
reserved	Must be set to 0
timeout	Maximum number of milliseconds to wait for message

Description

This function retrieves a message sent from the control processor. handle must be a valid handle returned from MVIbp_Open.

Upon calling this function, length should contain the maximum message size in words to be received. buffer must point to a buffer of at least length words in size. Upon successful return, length will contain the actual length of the message received.

If length exceeds the maximum message size specified by the value MsgRcvBufSize (refer to the MVIbp_SetIOConfig function), MVI_ERR_BADPARAM will be returned.

reserved is not used and must be set to zero. MVI_ERR_BADPARAM will be returned if reserved is not zero.

timeout specifies the number of milliseconds that the function will wait for a message. To poll for a message without waiting, set timeout to zero. If no message has been received, MVI_ERR_TIMEOUT will be returned.

Before this function can be used, messaging must be enabled with the MVIbp_SetIOConfig function. If messaging has not been enabled, MVI_ERR_BADCONFIG will be returned.

If the message received from the control processor is larger than length, the message will be truncated to length words and MVI_ERR_MSGTOOBIG will be returned.

The MVIbp_ReceiveMessage function retrieves data written to the PTQ-ADM module by the processor via a MSG instruction. The MSG instruction must be configured as shown in table A. The MSG instruction implements a "put attribute' command to the PTQ module’s assembly object. The MSG instruction will fail if a message has already been written to the PTQ module but application has not yet retrieved the message via MVIbp_ReceiveMessage.

Receive MSG Instruction Configuration

Field	Value	Description
Message Type	CIP Generic	Specify CIP message type
Service Code	10 (Hex)	Set_Attribute_Single service
Object Type	4	Assembly object class code
Object ID	8	Output message instance number
Object Attribute	3	Data attribute
Num Elements	Application dependent	Size of message to be written
Path	Application dependent	Path to PTQ module

Return Value

MVI_SUCCESS	A message has been received.
MVI_ERR_NOACCESS	handle does not have access.
MVI_ERR_TIMEOUT	The timeout occurred before a message was received.
MVI_ERR_BADPARAM	A parameter is invalid.
MVI_ERR_BADCONFIG	Receive messaging is not enabled.
MVI_ERR_MSGTOOBIG	The received message is too big for the buffer.

Example

```
MVIHANDLE    Handle;
int       rc;
WORD      buffer[256];
WORD      length;

length = 256;        // maximum message size that can be received
// Wait up to 5 seconds for a message
rc = MVIbp_ReceiveMessage(Handle, buffer, &length, 0, 5000);
if (rc == MVI_SUCCESS)
    printf("Message received. Length is %d words\n", length);
```

See Also

MVIbp_SetIOConfig (page 171)
MVIbp_SendMessage (page 179)

MVIbp_SendMessage

Syntax

```
int MVIbp_SendMessage(MVIHANDLE handle, WORD *buffer, WORD length,
```

WORD reserved, WORD timeout);

Parameters

handle	Handle returned by previous call to MVIbp_Open
buffer	Pointer to buffer of data to send to processor
length	The length in words of the message to send.
reserved	Must be set to 0
timeout	Maximum number of milliseconds to wait for processor to read message

Description

This function sends a message to the control processor. handle must be a valid handle returned from MVIbp_Open.

Upon calling this function, length should contain the message size in words. buffer must point to a buffer of at least length words in size.

If length exceeds the maximum message size specified by the value MsgSndBufSize (refer to the MVIbp_SetIOConfig function), MVI_ERR_BADPARAM will be returned.

timeout specifies the number of milliseconds that the function will wait for the message to transfer to the control processor. If the timeout occurs before the message has been transferred, MVI_ERR_TIMEOUT will be returned.

If timeout is 0, the function will return immediately. If the message was successfully queued to be sent, MVI_SUCCESS will be returned. If the message was not queued (for example, a previous message is being sent), MVI_ERR_TIMEOUT will be returned and the message must be re-tried at a later time. A timeout of 0 allows an application to perform other tasks while the message is being transmitted.

Before this function can be used, messaging must be enabled with the MVIbp_SetIOConfig function. If messaging has not been enabled, MVI_ERR_BADCONFIG will be returned.

Notes

The MVIbp_SendMessage function copies the message data into a buffer to be retrieved by the processor via a MSG instruction. The MSG instruction must be configured as shown in table B. The MSG instruction implements a "get attribute" command to the PTQ module's assembly object. The MSG instruction will fail if a message has not already been written by the application via MVIbp_SendMessage.

Send MSG Instruction Configuration

Field	Value	Description
Message Type	CIP Generic	Specify CIP message type
Service Code	OE (Hex)	Get_Attribute_Single service
Object Type	4	Assembly object class code
Object ID	7	Output message instance number
Object Attribute	3	Data attribute
Num Elements	Application dependent	Size of message to be written
Path	Application dependent	Path to PTQ module

Return Value

MVI_SUCCESS	A message has been received.
MVI_ERR_NOACCESS	handle does not have access.
MVI_ERR_TIMEOUT	The timeout occurred before the message was transferred.
MVI_ERR_BADPARAM	A parameter is invalid.
MVI_ERR_BADCONFIG	Send messaging is not enabled.

Example

```
MVIHANDLE    Handle;
int       rc;
WORD      buffer[256];

// Wait 5 seconds for the message to be sent
rc = MVIbp_SendMessage(Handle, buffer, 256, 5000);
if (rc == MVI_SUCCESS)
    printf("Message sent\n");
```

See Also

MVIbp_SetIOConfig (page 171)
MVIbp_ReceiveMessage (page 177)

10.6 Backplane API Miscellaneous Functions

MVIbp_GetVersionInfo

Syntax

```
int MVIbp_GetVersionInfo(MVIHANDLE handle, MVIBPVERSIONINFO *verinfo);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
verinfo	Pointer to structure of type MVIBPVERSIONINFO

Description

MVIbp_GetVersionInfo retrieves the current version of the API library and the backplane device driver. The information is returned in the structure verinfo. handle must be a valid handle returned from MVIbp_Open.

The MVIBPVERSIONINFO structure is defined as follows:

```
typedef struct tagMVIBPVERSIONINFO
{
WORD  APISeries;   /* API series */
WORD  APIRevision; /* API revision */
WORD  BPDDSeries;/* Backplane device driver series */
WORD  BPDDRevision; /* Backplane device driver revision */
BYTE  Reserved[8];    /* Reserved */
} MVIBPVERSIONINFO;
```

Return Value

MVI_SUCCESS	The version information was read successfully.
MVI_ERR_NOACCESS	handle does not have access

Example

```
MVIHANDLE           Handle;
MVIBPVERSIONINFO      verinfo;

/* print version of API library */
MVIbp_GetVersionInfo(Handle,&verinfo);
printf("Library Series %d, Rev %d\n", verinfo.APISeries,
verinfo.APIRevision);
printf("Driver Series %d, Rev %d\n", verinfo.BPDDSeries,
verinfo.BPDDRevision);
```

MVIbp_GetModuleInfo

Syntax

```
int MVIbp_GetModuleInfo(MVIHANDLE handle, MVIBPMODULEINFO *modinfo);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
modinfo	Pointer to structure of type MVIBPMODULEINFO

Description

MVIbp_GetModuleInfo retrieves identity information for the module. The information is returned in the structure modinfo. *handle* must be a valid handle returned from MVIbp_Open.

The MVIBPMODULEINFO structure is defined as follows:

```
typedef struct tagMVIBPMODULEINFO
{
 WORD    VendorID;           // Reserved
 WORD    DeviceType;         // Reserved
 WORD    ProductCode;        // Device model code
 BYTE    MajorRevision;      // Device major revision
 BYTE    MinorRevision;      // Device minor revision
 DWORD   SerialNo;           // Serial number
 BYTE    Name[32];           // Device name (string)
 BYTE    Month;              // Date of manufacture - month
 BYTE    Day;                // Date of manufacture - day
 WORD    Year;               // Date of manufacture - year
} MVIBPMODULEINFO;
```

Return Value

MVI_SUCCESS	The version information was read successfully.
MVI_ERR_NOACCESS	handle does not have access

Example

```
MVIHANDLE           Handle;
MVIBPMODULEINFO      modinfo;

/* print module name */
MVIbp_GetModuleInfo(Handle,&modinfo);
printf("Name is %s\n", modinfo.Name);
```

MVIbp_ErrorString

Syntax

```
int MVIbp_ErrorString(int errcode, char *buf);
```

Parameters

errcode	Error code returned from an API function
buf	Pointer to user buffer to receive message

Description

MVIbp_ErrorStr returns a text error message associated with the error code errcode. The null-terminated error message is copied into the buffer specified by buf. The buffer should be at least 80 characters in length.

Return Value

MVI_SUCCESS	Message returned in buf
MVI_ERR_BADPARAM	Unknown error code

Example

```
char buf[80];
int rc;

/* print error message */
MVIbp_ErrorString(rc, buf);
printf("Error: %s", buf);
```

MVIbp_SetUserLED

Syntax

```
int MVIbp_SetUserLED(MVIHANDLE handle, int lednum, int ledstate);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
lednum	Specifies which of the user LED indicators is being addressed

Description

MVIbp_SetUserLED allows an application to turn the user LED indicators on and off. handle must be a valid handle returned from MVIbp_Open.

lednum must be set to MVI_LED_USER1 or MVI_LED_USER2 to select User LED 1 or User LED 2, respectively.

ledstate must be set to MVI_LED_STATE_ON or MVI_LED_STATE_OFF to turn the indicator On or Off, respectively.

Return Value

MVI_SUCCESS	The input scan has occurred.
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_BADPARAM	lednum or ledstate is invalid.

Example

```
MVIHANDLE           Handle;

/* Turn User LED 1 on and User LED 2 off */
MVIbp_SetUserLED(Handle, MVI_LED_USER1, MVI_LED_STATE_ON);
MVIbp_SetUserLED(Handle, MVI_LED_USER2, MVI_LED_STATE_OFF);
```

MVIbp_SetModuleStatus

Syntax

```
int MVIbp_SetModuleStatus(MVIHANDLE handle, int status);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
status	Module status, OK or Faulted

Description

MVIbp_SetModuleStatus allows an application set the state of the module to OK or Faulted. handle must be a valid handle returned from MVIbp_Open.

state must be set to MVI_MODULE_STATUS_OK or MVI_MODULE_STATUS_FAULTED. If the state is OK, the module status LED indicator will be set to Green. If the state is Faulted, the status indicator will be set to Red.

Note: The MVI hardware can set the OK LED to Red if any of the following occurs:

- an unrecoverable fault
- hardware failure
- backplane driver failure
- Neither the PTQ hardware, nor the Set ModuleStatus call has priority. Either can overwrite the other.

Return Value

MVI_SUCCESS	The input scan has occurred.
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_BADPARAM	lednum or ledstate is invalid.

Example

```
MVIHANDLE          Handle;

/* Set the Status indicator to Red */
MVIbp_SetModuleStatus(Handle, MVI_MODULE_STATUS_FAULTED);
```

MVIbp_GetConsoleMode

Syntax

```
int MVIbp_GetConsoleMode(MVIHANDLE handle, int *mode, int *baud);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
mode	Pointer to an integer that is set to 1 if the console is installed, or 0 if the console is not enabled.
baud	Pointer to an integer that is set to the console baud rate index if the console is enabled.

Description

This function is used to query the state of the console. handle must be a valid handle returned from MVIbp_Open.

mode is a pointer to an integer. When this function returns, mode will be set to 1 if the console is enabled, or 0 if the console is disabled.

baud is a pointer to an integer. When this function returns, baud will be set to the console's baud index value if the console is enabled. baud is not set if the console is disabled.

It may be useful for an application to detect that the console is enabled and allow user interaction.

Note: If the Setup Jumper is installed, the console is enabled at 19200 baud.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	handle does not have access

Example

```
MVIHANDLE          handle;
int  mode;

MVIbp_GetConsoleMode(handle, &mode);
if (mode)
   // Console is enabled - allow user interaction
else
   // Console is not available - normal operation
```

MVIbp_GetSetupMode

Syntax

```
int MVIbp_GetSetupMode(MVIHANDLE handle, int *mode);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
mode	Pointer to an integer that is set to 1 if the Setup Jumper is installed, or 0 if the Setup Jumper is not installed.

Description

This function is used to query the state of the Setup Jumper. handle must be a valid handle returned from MVIbp_Open.

mode is a pointer to an integer. When this function returns, mode will be set to 1 if the module is in Setup Mode, or 0 if not.

If the Setup Jumper is installed, the module is considered to be in Setup Mode. It may be useful for an application to detect Setup Mode and perform special configuration or diagnostic functions.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	handle does not have access

Example

```
MVIHANDLE          handle;
int  mode;

MVIbp_GetSetupMode(handle, &mode);
if (mode)
   // Setup Jumper is installed - perform configuration/diagnostic
else
   // Not in Setup Mode - normal operation
```

MVIbp_GetProcessorStatus

Syntax

```
int MVIbp_GetProcessorStatus(MVIHANDLE handle, WORD *pstatus);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
pstatus	Pointer to a word that will be updated with the current processor status.

Description

This function is used to query the state of the processor. handle must be a valid handle returned from MVIbp_Open.

pstatus is a pointer to an word. When this function returns, certain bits in this word will be set to indicate the current processor status, as shown in Figure 6.

Processor Status Bits

Bit	Name	Description
0	MVI_PROCESSOR_STATUS_RUN	Set if processor is in Run Mode
1	MVI_DATA_CONNECTION_OPEN	Set if data connection is open
2	MVI_STATUS_CONNECTION_OPEN	Set if status connection is open

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	handle does not have access
MVI_ERR_BADCONFIG	The data connection is not open

Example

```
MVIHANDLE handle;
WORD status;
MVIbp_GetProcessorStatus(handle, &status);
if (status & MVI_PROCESSOR_STATUS_RUN)
// Processor is in Run Mode
else
// Processor is not in Run Mode or there is no connection
```

MVIbp_Sleep

Syntax

```
int MVIbp_Sleep( MVIHANDLE handle, WORD msdelay );
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
msdelay	Time in milliseconds to suspend task

Description

MVIbp_Sleep suspends the calling thread for at least msdelay milliseconds. The actual delay may be several milliseconds longer than msdelay, due to system overhead and the system timer granularity (5ms).

Return Value

MVI_SUCCESS	Success
MVI_ERR_NOACCESS	handle does not have access

Example

```
MVIHANDLE handle;
int timeout=200;
// Simple timeout loop
while(timeout--)
{
// Poll for data, and so on.
// Break if condition is met (no timeout)
// Else sleep a bit and try again
MVIbp_Sleep(10);
}
```

MVIbp_SetConsoleMode

Syntax

```
int MVIbp_SetConsoleMode(MVIHANDLE handle, int mode, int baud);
```

Parameters

handle	Handle returned by previous call to MVIbp_Open
mode	An integer that is set to 1 if the console is to be enabled, or 0 if the console is not enabled.
baud	An integer that is set to the desired console baud rate index if the console is enabled.

Description

This function sets the state of the console. handle must be a valid handle returned from MVIbp_Open.

mode is an integer that contains the desired state of the console. mode should be set to 1 if the console is to be enabled, or 0 if the console is to be disabled.

baud is an integer that contains the desired baud rate of the console. baud should be set to the console's baud index value if the console is enabled. The baud index values are shown in Table 3.

The state of the console is normally configured with the BIOS setup menu and is saved in battery-backed memory. If the module is removed from power for a period of time and the battery discharges, then the state information is lost. This function allows an application to store a desired console state into the battery-backed memory.

Note: If the Setup Jumper is installed, the console is enabled at 19200 baud.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	handle does not have access

Example

```
MVIHANDLE handle;
int mode,baud;
mode = 1; // enable the console
baud = 8; // set baud rate to 19200 baud
MVIbp_SetConsoleMode(handle, mode, baud);
```

11 Serial Port Library Functions

In This Chapter

This section provides detailed programming information for each of the API library functions. The calling convention for each API function is shown in 'C' format.

The API library routines are categorized according to functionality as follows:

Initialization

MVIsp_Open

MVIsp_Close

MVIsp_OpenAlt

Configuration

MVIsp_Config

MVIsp_SetHandshaking

Port Status

MVIsp_SetRTS, MVIsp_GetRTS

MVIsp_SetDTR, MVIsp_GetDTR

MVIsp_GetCTS

MVIsp_GetDSR

MVIsp_GetDCD

MVIsp_GetLineStatus

Communications

MVIsp_Putch

MVIsp_Puts

MVIsp_PutData

MVIsp_Getch

MVIsp_Gets

MVIsp_GetData

MVIsp_GetCountUnsent

MVIsp_GetCountUnread

MVIsp_PurgeDataUnsent

MVIsp_PurgeDataUnread

Miscellaneous

MVIsp_GetVersionInfo

11.1 Serial Port API Initialization Functions

MVIsp_Open

Syntax

```
int MVIsp_Open(int comport, BYTE baudrate, BYTE parity, BYTE wordlen,
BYTE stopbits);
```

Parameters

comport	Communications Port to open
baudrate	Baud rate for this port
parity	Parity setting for this port
wordlen	Number of bits for each character
stopbits	Number of stop bits for each character

Description

MVIsp_Open acquires access to a communications port. This function must be called before any of the other API functions can be used.

comport specifies which port is to be opened. The valid values for the module are COM1 (corresponds to PRT1), COM2 (corresponds to PRT2), and COM3 (corresponds to PRT3)..

baudrate is the desired baud rate. The allowable values for baudrate are shown in the following table.

Baud Rate	**Value**
BAUD_110	0
BAUD_150	1
BAUD_300	2
BAUD_600	3
BAUD_1200	4
BAUD_2400	5
BAUD_4800	6
BAUD_9600	7
BAUD_19200	8
BAUD_28800	9
BAUD_38400	10
BAUD_57600	11
BAUD_115200	12

Valid values for *parity* are PARITY_NONE, PARITY_ODD, PARITY_EVEN, PARITY_MARK, and PARITY_SPACE.

wordlen sets the word length in number of bits per character. Valid values for word length are WORDLEN5, WORDLEN6, WORDLEN7, and WORDLEN8.

The number of stop bits is set by *stopbits*. Valid values for stop bits are STOPBITS1 and STOPBITS2.

The handshake lines DTR and RTS of the port specified by *comport* are turned on by MVIsp_Open.

Note: If the console is enabled or the Setup jumper is installed, the baud rate for COM1 is set as configured in BIOS Setup and cannot be changed by MVIsp_Open. MVIsp_Open will return MVI_SUCCESS, but the baud rate will not be affected. It is recommended that the console be disabled in BIOS Setup if COM1 is to be accessed with the serial API.
IMPORTANT: After the API has been opened, MVIsp_Close should always be called before exiting the application.Return Value

MVI_SUCCESS	Port was opened successfully
MVI_ERR_REOPEN	Port is already open
MVI_ERR_NODEVICE	UART not found on port

Note: MVI_ERR_NODEVICE will be returned if the port is not supported by the module.

Example

```
if ( MVIsp_Open(COM1,BAUD_9600,PARITY_NONE,WORDLEN8,STOPBITS1) !=
MVI_SUCCESS) {
    printf("Open failed!\n");
} else {
    printf("Open succeeded\n");
}
```

See Also

MVIsp_Close (page 197)

MVIsp_OpenAlt

Syntax

```
int MVIsp_ OpenAlt(int comport, MVISPALTSETUP *altsetup);
```

Parameters

comport	Communications port to open
altsetup	pointer to structure of type MVISPALTSETUP

Description

MVIsp_OpenAlt provides an alternate method to acquire access to a communications port.

With MVIsp_OpenAlt, the sizes of the serial port data queues can be set by the application.

See MVIsp_Open for any considerations about opening a port.

Comport specifies which port is to be opened. See MVIsp_Open for valid values.

Altsetup points to a MVISPALTSETUP structure that contains the configuration information for the port.

The MVISPALTSETUP structure is defined as follows

```
typedef struct tagMVISPALTSETUP
{
BYTE baudrate;
BYTE parity;
BYTE wordlen;
BYTE stopbits;
int txquesize; /* Transmit queue size */
int rxquesize; /* Receive queue size */
BYTE fifosize; /* UART Internal FIFO size */
} MVISPALTSETUP;
```

See MVIsp_Open for valid values for the baudrate, parity, wordlen, and stopbits members of the structure. The txquesize and rxquesize members determine the size of the data buffers used to queue serial data. Valid values for the queue sizes can be any value from MINQSIZE to MAXQSIZE. The MVIsp_Open function uses a queue size of DEFQSIZE. These values are defined as:

```
#define MINQSIZE 512 /* Minimum Queue Size */
#define DEFQSIZE 1024 /* Default Queue Size */
#define MAXQSIZE 16384 /* Maximum Queue Size */
```

By default, the API sets the UART’s internal receive fifo size to 8 characters to permit greater reliability at higher baud rates. In certain serial protocols, this buffering of characters can cause character timeouts and can be changed or disabled to meet these requirements. Most applications should set the fifosize to the default RXFIFO_DEFAULT.

Either MVIsp_OpenAlt or MVIsp_Open must be called before any of the other API functions can be used.

Return Value

MVI_SUCCESS	Port was opened successfully
MVI_ERR_REOPEN	Port is already open
MVI_ERR_NODEVICE	UART not found for port

Example

```
MVISPALTSETUP altsetup;
altsetup.baudrate = BAUD_9600;
altsetup.parity = PARITY_NONE;
altsetup.wordlen = WORDLEN8;
altsetup.stopbits = STOPBITS1;
altsetup.txquesize = DEFQSIZE;
altsetup.rxquesize = DEFQSIZE * 2;
if (MVIsp_OpenAlt(COM1, &altsetup) != MVI_SUCCESS)
{
printf("Open failed!\n");
} else {
printf("Open succeeded!\n");
}
```

See Also

MVIsp_Open (page 193)

MVIsp_Close

Syntax

```
int MVIsp_Close(int comport);
```

Parameters

comport	Port to close

Description

This function is used by an application to release control of the a communications port. comport must be previously opened with MVIsp_Open.

comport specifies which port is to be closed. The valid values for the module are COM1 (corresponds to PRT1), COM2 (corresponds to PRT2), and COM3 (corresponds to PRT3).

The handshake lines DTR and RTS of the port specified by comport are turned off by MVIsp_Close.

IMPORTANT: After the API has been opened, this function should always be called before exiting the application.

Return Value

MVI_SUCCESS	Port was closed successfully
MVI_ERR_NOACCESS	Comport has not been opened

Example

```
MVIsp_Close(COM1);
```

See Also

MVIsp_Open (page 193)

11.2 Serial Port API Configuration Functions

MVIsp_Config

Syntax

```
int MVIsp_Config(int comport, BYTE baudrate, BYTE parity,
```
BYTE wordlen, BYTE stopbits);

Parameters

comport	Communications port to open
baudrate	Baud rate for this port
parity	Parity setting for this port
wordlen	Number of bits for each character
stopbits	Number of stop bits for each character
baudrate	Pointer to DWORD to receive baudrate

Description

MVIsp_Config allows the configuration of a serial port to be changed after it has been opened.

comport specifies which port is to be opened.

baudrate is the desired baud rate.

Valid values for parity are PARITY_NONE, PARITY_ODD, PARITY_EVEN, PARITY_MARK, and PARITY_SPACE.

wordlen sets the word length in number of bits per character. Valid values for word length are WORDLEN5, WORDLEN6, WORDLEN7, and WORDLEN8.

The number of stop bits is set by stopbits. Valid values for stop bits are STOPBITS1 and STOPBITS2.

Note: If the console is enabled or the Setup jumper is installed, the baud rate for COM1 is set as configured in BIOS Setup and cannot be changed by MVIsp_Open. MVIsp_Config will return MVI_SUCCESS, but the baud rate will not be affected.

Return Value

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	Comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
if (MVIsp_Config(COM1,BAUD_9600,PARITY_NONE,WORDLEN8,STOPBITS1) !=
MVI_SUCCESS) {
    printf("Config failed!\n");
} else {
    printf("Config succeeded\n");
}
```

See Also

MVIsp_Open (page 193)

MVIsp_SetHandshaking

Syntax

```
int MVIsp_SetHandshaking(int comport, int shake);
```

Parameters

comport	Port for which handshaking is to be set
shake	Desired handshake mode

Description

This function enables handshaking for a port after it has been opened. comport must be previously opened with MVIsp_Open.

shake is the desired handshake mode. Valid values for shake are HSHAKE_NONE, HSHAKE_XONXOFF, HSHAKE_RTSCTS, and HSHAKE_DTRDSR.

Use HSHAKE_XONXOFF to enable software handshaking for a port. Use HSHAKE_RTSCTS or HSHAKE_DTRDSR to enable hardware handshaking for a port. Hardware and software handshaking cannot be used together.

Handshaking is supported in both the transmit and receive directions.

Important: If hardware handshaking is enabled, using the MVIsp_SetRTS and MVIsp_SetDTR functions will cause unpredictable results.
If software handshaking is enabled, ensure that the XON and XOFF ASCII characters are not transmitted as data from a port or received into a port because this will be treated as handshaking controls.

Return Values

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid handshaking mode

Example

```
if (MVI_SUCCESS != MVIsp_SetHandshaking(COM1, HSHAKE_RTSCTS))
    printf("Error: Set Handshaking failed\n");
```

11.3 Serial Port API Status Functions

MVIsp_SetRTS

Syntax

```
int MVIsp_SetRTS(int comport, int state);
```

Parameters

comport	Port for which RTS is to be changed
state	Desired RTS state

Description

This functions allows the state of the RTS signal to be controlled. comport must be previously opened with MVIsp_Open.

state specifies desired state of the RTS signal. Valid values for state are ON and OFF.

Note: If RTS/CTS hardware handshaking is enabled, using the MVIsp_SetRTS function will cause unpredictable results.

Return Value

MVI_SUCCESS	The RTS signal was set successfully.
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid state

Example

```
int rc;

rc = MVIsp_SetRTS(COM1, ON);
if (rc != MVI_SUCCESS)
    printf("SetRTS failed\n ");
```

See Also

MVIsp_GetRTS (page 201)

MVIsp_GetRTS

Syntax

```
int MVIsp_GetRTS(int comport, int *state);
```

Parameters

comport	Port for which RTS is requested
state	Pointer to int for desired state

Description

This function allows the state of the RTS signal to be determined. comport must be previously opened with MVIsp_Open.

The current state of the RTS signal is copied to the int pointed to by state.

Return Value

MVI_SUCCESS	The RTS state was read successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
int   state;

if (MVIsp_GetRTS(COM1, &state) == MVI_SUCCESS)
{
   if (state == ON)
      printf("RTS is ON\n");
   else
      printf("RTS is OFF\n");
}
```

See Also

MVIsp_SetRTS (page 200)

MVIsp_SetDTR

Syntax

```
int MVIsp_SetDTR(int comport, int state);
```

Parameters

comport	Port for which DTR is to be changed
state	Desired state

Description

This function allows the state of the DTR signal to be controlled. comport must be previously opened with MVIsp_Open.

state is the desired state of the DTR signal. Valid values for state are ON and OFF.

Note: If DTR/DSR handshaking is enabled, changing the state of the DTR signal with MVIsp_SetDTR will cause unpredictable results.

Return Value

MVI_SUCCESS	The DTR signal was set successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid state

Example

```
if (MVIsp_SetDTR(COM1, ON) != MVI_SUCCESS)
printf("Set DTR failed\n");
```

See Also

MVIsp_GetDTR (page 203)

MVIsp_GetDTR

Syntax

```
int MVIsp_GetDTR(int comport, int *state);
```

Parameters

comport	Port for which DTR is requested
state	Pointer to int for desired state

Description

This function allows the state of the DTR signal to be determined. comport must be previously opened with MVIsp_Open. The current state of the DTR signal is copied to the int pointed to by state.

Return Values

MVI_SUCCESS	The DTR state was read successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
int   state;

if (MVIsp_GetDTR(COM1, &state) == MVI_SUCCESS)
{
   if (state == ON)
      printf("DTR is ON\n");
   else
      printf("DTR is OFF\n");
}
```

See Also

MVIsp_SetDTR (page 202)

MVIsp_GetCTS

Syntax

```
int MVIsp_GetCTS(int comport, int *state);
```

Parameters

comport	Port for which CTS is requested
state	Pointer to int for desired state

Description

This function allows the state of the CTS signal to be determined. comport must be previously opened with MVIsp_Open. The current state of the CTS signal is copied to the int pointed to by state.

Return Value

MVI_SUCCESS	The CTS state was read successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
int   state;

if (MVIsp_GetCTS(COM1, &state) == MVI_SUCCESS)
{
   if (state == ON)
      printf("CTS is ON\n");
   else
      printf("CTS is OFF\n");
}
```

MVIsp_GetDSR

Syntax

```
int MVIsp_GetDSR(int comport, int *state);
```

Parameters

comport	Port for which DSR is requested
state	Pointer to int for desired state

Description

This function allows the state of the DSR signal to be determined. comport must be previously opened with MVIsp_Open. The current state of the DSR signal is copied to the int pointed to by state.

Return Value

MVI_SUCCESS	The DSR state was read successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
int   state;

if (MVIsp_GetDSR(COM1, &state) == MVI_SUCCESS)
{
   if (state == ON)
      printf("DSR is ON\n");
   else
      printf("DSR is OFF\n");
}
```

MVIsp_GetDCD

Syntax

```
int MVIsp_GetDCD(int comport, int *state);
```

Parameters

comport	Port for which DCD is requested
state	Pointer to int for desired state

Description

This function allows the state of the DCD signal to be determined. comport must be previously opened with MVIsp_Open. The current state of the DCD signal is copied to the int pointed to by state.

Return Value

MVI_SUCCESS	The DCD state was read successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
int   state;

if (MVIsp_GetDCD(COM1, &state) == MVI_SUCCESS)
{
   if (state == ON)
      printf("DCD is ON\n");
   else
      printf("DCD is OFF\n");
}
```

MVIsp_GetLineStatus

Syntax

```
int MVIsp_GetLineStatus(int comport, BYTE *status);
```

Parameters

comport	Port for which line status is requested
status	Pointer to BYTE to receive line status

Description

MVIsp_GetLineStatus returns any line status errors received over the serial port. The status returned indicates if any overrun, parity, or framing errors or break signals have been detected.

comport is the desired serial port and must be previously opened with MVIsp_Open.

status points to a BYTE that will receive a set of flags that indicate errors received over the serial port. If the returned status is 0, no errors have been detected. If status is non-zero, it can be logically and'ed with the line status error flags LSERR_OVERRUN, LSERR_PARITY, LSERR_FRAMING, LSERR_BREAK, and/or QSERR_OVERRUN to determine the exact cause of the error. The corresponding error flag will be set for each error type detected.

Note: The QSERR_OVERRUN bit indicates that a receive queue overflow has occurred.

After returning the bit flags in status, line status errors are cleared. Therefore, MVIsp_GetLineStatus actually returns line status errors detected since the previous call to this function.

Return Value

MVI_SUCCESS	The line status was read successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
BYTE   sts;

if (MVIsp_GetGetLineStatus(COM2,&sts) == MVI_SUCCESS)
{
   if (sts == 0)
      printf("No Line Status Errors Received\n");
   else if ( (sts & LSERR_BREAK) != 0)
      printf("A Break Signal was Received\n");
   else
      printf("A Line Status Error was Received\n");
}
```

11.4 Serial Port API Communications

MVIsp_Putch

Syntax

```
int MVIsp_Putch(int comport, BYTE ch, DWORD timeout);
```

Parameters

comport	Port to which data is to be sent
ch	Character to be sent
timeout	Amount of time to wait to send character

Description

This function is used to transmit a single character across a serial port. comport must be previously opened with MVIsp_Open.

ch is the byte to be sent.

All data sent to a port is queued before transmission across the serial port. Therefore, some delay may occur between the time after this function returns and the actual time that the character is transmitted across the serial line. This function attempts to insert the character into the transmission queue, and return values correspond accordingly.

timeout specifies the amount of time in milliseconds to wait. If timeout is TIMEOUT_ASAP, the function will return immediately if the character cannot be queued immediately. If timeout is TIMEOUT_FOREVER, the function will not return until the character is queued successfully.

If the character can be queued immediately, MVIsp_Putch returns MVI_SUCCESS. If the character cannot be queued immediately, MVIsp_Putch tries to queue the character until the timeout elapses. If the timeout elapses before the character can be queued, MVI_ERR_TIMEOUT is returned.

Note: If handshaking is enabled and the receiving serial device has paused transmission, timeouts may occur after the queue becomes full.

Return Value

MVI_SUCCESS	The char was sent successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid parameter
MVI_ERR_TIMEOUT	Timeout elapsed before character sent

Example

```
if (MVIsp_Putch(COM1, ';', 1000L) != MVI_SUCCESS)
   printf("Semicolon could not be sent in 1 second\n");
```

See Also

MVIsp_GetCh (page 209)

MVIsp_Puts (page 210)

MVIsp_PutData (page 212)

MVIsp_Getch

Syntax

```
int MVIsp_Getch(int comport, BYTE *ch, DWORD timeout);
```

Parameters

comport	Port from which data is to be received
ch	Pointer to BYTE to receive character
timeout	Amount of time to wait to receive character

Description

This function receives a single character from a serial port. comport must be previously opened with MVIsp_Open.

ch points to a BYTE that will receive the character.

All data received from a port is queued after reception from the serial port. Therefore, some delay may occur between the time a character is received across the serial line and the time the character is returned by MVIsp_Getch. This function attempts to retrieve a character from the reception queue, and return values correspond accordingly.

timeout specifies the amount of time in milliseconds to wait. If timeout is TIMEOUT_ASAP, the function will return immediately if the queue is empty. If timeout is TIMEOUT_FOREVER, the function will not return until a character is retrieved from the reception queue successfully.

If the reception queue is not empty, the oldest character is retrieved from the queue and MVIsp_Getch returns MVI_SUCCESS. If the queue is empty, MVIsp_Getch tries to retrieve a character from the queue until the timeout elapses. If the timeout elapses before a character can be retrieved, MVI_ERR_TIMEOUT is returned.

Return Value

MVI_SUCCESS	A char was retrieved successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer
MVI_ERR_TIMEOUT	Timeout elapsed before character retrieved

Example

```
BYTE ch;

if (MVIsp_Getch(COM1, &ch, 1000L) == MVI_SUCCESS)
   putch((char)ch);
```

See Also

MVIsp_PutCh (page 208)

MVIsp_Gets (page 214)

MVIsp_Puts

Syntax

```
int MVIsp_Puts(int comport, BYTE *str, BYTE term, int *len, DWORD timeout);
```

Parameters

comport	Port to which data is to be sent
str	String of characters to be sent
term	Termination character of string
len	Pointer to BYTE to receive number of characters sent
timeout	Amount of time to wait to send character

Description

This function is used to transmit a string of characters across a serial port. comport must be previously opened with MVIsp_Open.

str is a pointer to an array of characters (or is a string) to be sent.

MVIsp_Puts sends each char in the array str to the serial port until it encounters the termination character term. Therefore, the character array must end with the termination character. The termination character is not sent to the serial port.

All data sent to a port is queued before transmission across the serial port. Therefore, some delay may occur between the time this function returns and the actual time that the characters are transmitted across the serial line. This function attempts to insert the characters into the transmission queue, and return values correspond accordingly.

timeout specifies the amount of time in milliseconds to wait. If timeout is TIMEOUT_ASAP, the function will return immediately if any of the characters cannot be queued immediately. If timeout is TIMEOUT_FOREVER, the function will not return until all the characters are queued successfully.

If all the characters can be queued immediately, MVIsp_Puts returns MVI_SUCCESS. If the characters cannot be queued immediately, MVIsp_Puts tries to queue the characters until the timeout elapses. If the timeout elapses before the characters can be queued, MVI_ERR_TIMEOUT is returned.

If len is not NULL, MVIsp_Puts writes to the int pointed to by len the number of characters queued successfully. len is written for successfully sent characters as well as timeouts.

Note: If handshaking is enabled and the receiving serial device has paused transmission, timeouts may occur after the queue becomes full.

Return Value

MVI_SUCCESS	The characters were sent successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid parameter
MVI_ERR_TIMEOUT	Timeout elapsed before characters sent

Example

```
char str[ ] = "Hello, World!";
int nn;

if (MVIsp_Puts(COM1, str, '\0', &nn, 1000L) != MVI_SUCCESS)
   printf("%d characters were sent\n",nn);
```

See Also

MVIsp_Gets (page 214)
MVIsp_PutCh (page 208)
MVIsp_PutData (page 212)

MVIsp_PutData

Syntax

```
int MVIsp_PutData(int comport, BYTE *data, int *len, DWORD timeout);
```

Parameters

comport	Port to which data is to be sent
data	Pointer to array of bytes to be sent
len	Pointer to number of bytes to send / bytes sent
timeout	Amount of time to wait to send byte

Description

This function is used to transmit an array of bytes across a serial port. comport must be previously opened with MVIsp_Open.

data is a pointer to an array of bytes to be sent.

MVIsp_PutData sends each byte in the array data to the serial port. len should point to the number of bytes in the array data to be sent.

All data sent to a port is queued before transmission across the serial port. Therefore, some delay may occur between the time this function returns and the actual time that the bytes are transmitted across the serial line. This function attempts to insert the bytes into the transmission queue, and return values correspond accordingly.

timeout specifies the amount of time in milliseconds to wait. If timeout is TIMEOUT_ASAP, the function will return immediately if any of the bytes cannot be queued immediately. If timeout is TIMEOUT_FOREVER, the function will not return until all the bytes are queued successfully.

If all the bytes can be queued immediately, MVIsp_PutData returns MVI_SUCCESS. If the characters cannot be queued immediately, MVIsp_PutData tries to queue the bytes until the timeout elapses. If the timeout elapses before the bytes can be queued, MVI_ERR_TIMEOUT is returned.

When MVIsp_PutData returns, it writes to the int pointed to by len the number of bytes queued successfully. len is written for successfully sent bytes as well as timeouts.

Note: If software handshaking is enabled on the external serial device, sending data that contains XOFF characters may stop transmission from the external serial device.

If handshaking is enabled and the receiving serial device has paused transmission, timeouts may occur after the queue becomes full.

Return Value

MVI_SUCCESS	The bytes were sent successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid parameter
MVI_ERR_TIMEOUT	Timeout elapsed before bytes sent

Example

```
BYTE dd[5] = { 10, 20, 30, 40, 50 };
int nn;

nn = 5;
if (MVIsp_PutData(COM1, &dd[0], &nn, 1000L) != MVI_SUCCESS)
   printf("%d bytes were sent\n",nn);
```

See Also

MVIsp_PutCh (page 208)

MVIsp_Puts (page 210)

MVIsp_Gets

Syntax

```
int MVIsp_Gets(int comport, BYTE *str, BYTE term, int *len, DWORD timeout);
```

Parameters

comport	Port from which data is to be received
str	Pointer to array of bytes to receive data
term	Termination character of data
len	Number of bytes to receive / bytes received
timeout	Amount of time to wait to receive character

Description

This function receives an array of bytes from a serial port. comport must be previously opened with MVIsp_Open.

str points to an array of bytes that will receive the data.

len points to the number of bytes to receive.

MVIsp_Gets retrieves bytes from the reception queue until either a byte is equal to the termination character or the number of bytes pointed to by len are retrieved. If a byte is retrieved that equals the termination character, the byte is copied into the array str and the function returns.

All data received from a port is queued after reception from the serial port. Therefore, some delay may occur between the time a character is received across the serial line and the time the character is returned by MVIsp_Gets. This function attempts to retrieve characters from the reception queue, and return values correspond accordingly.

timeout specifies the amount of time in milliseconds to wait. If timeout is TIMEOUT_ASAP, the function will return immediately if the queue is empty. If timeout is TIMEOUT_FOREVER, the function will not return until an array of bytes is retrieved from the reception queue successfully.

If the timeout elapses before the termination character or len bytes are received, MVI_ERR_TIMEOUT is returned.

When MVIsp_Gets returns, it writes to the int pointed to by len the number of bytes retrieved. len is written for successfully retrieved bytes as well as timeouts. If the function returns because a termination character was retrieved, len includes the termination character in the length.

Note: If handshaking is enabled and the reception queue is full, this API may pause transmissions from the external device, and timeouts may then occur.

Return Value

MVI_SUCCESS	Bytes were retrieved successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer
MVI_ERR_TIMEOUT	Timeout elapsed before bytes retrieved

Example

```
BYTE str[10];
int   nn;

nn = 10;
if (MVIsp_Gets(COM1, &str[0], '\r', &nn, 1000L) == MVI_SUCCESS)
   printf("%d bytes were received\n",nn);
```

See Also

MVIsp_Getch (page 209)

MVIsp_Puts (page 210)

MVIsp_PutData (page 212)

MVIsp_GetData

Syntax

```
int MVIsp_GetData(int comport, BYTE *data, int *len, DWORD timeout);
```

Parameters

comport	Port from which data is to be received
data	Pointer to array of bytes to receive data
len	Number of bytes to receive / bytes received
timeout	Amount of time to wait to receive character

Description

This function receives an array of bytes from a serial port. comport must be previously opened with MVIsp_Open.

data points to an array of bytes that will receive the data.

len points to the number of bytes to receive.

MVIsp_GetData retrieves bytes from the reception queue until either the number of bytes pointed to by len are retrieved or the timeout elapses.

All data received from a port is queued after reception from the serial port. Therefore, some delay may occur between the time a character is received across the serial line and the time the character is returned by MVIsp_GetData. This function attempts to retrieve characters from the reception queue, and return values correspond accordingly.

timeout specifies the amount of time in milliseconds to wait. If timeout is TIMEOUT_ASAP, the function will return immediately if the queue is empty. If timeout is TIMEOUT_FOREVER, the function will not return until an array of bytes is retrieved from the reception queue successfully.

If the timeout elapses before the termination character or len bytes are received, MVI_ERR_TIMEOUT is returned.

When MVIsp_GetData returns, it writes to the int pointed to by len the number of bytes retrieved. len is written for successfully retrieved bytes as well as timeouts.

Return Value

MVI_SUCCESS	bytes were retrieved successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	invalid pointer
MVI_ERR_TIMEOUT	timeout elapsed before bytes retrieved

Example

```
BYTE data[10];
int   nn;

nn = 10;
if (MVIsp_GetData(COM1, data, &nn, 1000L) == MVI_SUCCESS)
   printf("%d bytes were received\n",nn);
```

See Also

MVIsp_Gets (page 214)

MVIsp_Getch (page 209)

MVIsp_PutData (page 212)

MVIsp_GetCountUnsent

Syntax

```
int MVIsp_GetCountUnsent(int comport, int *count);
```

Parameters

comport	Desired communications port
count	Pointer to int to receive unsent character count

Description

MVIsp_GetCountUnsent returns the number of characters in the transmit queue that are waiting to be sent. Since data sent to a port is queued before transmission across a serial port, the application may need to determine if all characters have been transmitted or how many characters remain to be transmitted.

comport is the desired serial port and must be previously opened with MVIsp_Open.

count points to an int that will receive the number of characters that have been sent to the serial port but not transmitted. If the returned count is 0, all data has been transmitted. If it is non-zero, it contains the number of characters put into the queue with MVIsp_Putch, MVIsp_Puts, or MVIsp_PutData but that have not been transmitted.

Return Value

MVI_SUCCESS	Count retrieved successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
int count;

if (MVIsp_GetCountUnsent(COM2,&count) == MVI_SUCCESS)
{
   if (count == 0)
      printf("All chars sent\n");
   else
      printf("%d characters remaining\n",count);
}
```

See Also

MVIsp_Putch (page 208)

MVIsp_Puts (page 210)

MVIsp_PutData (page 212)

MVIsp_GetCountUnread

Syntax

```
int MVIsp_GetCountUnread(int comport, int *count);
```

Parameters

comport	Desired communications port
count	Pointer to int to receive unread character count

Description

MVIsp_GetCountUnread returns the number of characters in the receive queue that are waiting to be read. Since data received from a port is queued after reception from a serial port, the application may need to determine if all characters have been read or how many characters remain to be read.

comport is the desired serial port and must be previously opened with MVIsp_Open.

count points to an int that will receive the number of characters that have been received from the serial port but not read by the application. If the returned count is 0, all received data has been read. If it is non-zero, it contains the number of characters placed into the receive queue after reception from a serial port but that have not been read from the queue with MVIsp_Getch, MVIsp_Gets, or MVIsp_GetData.

Return Value

MVI_SUCCESS	Count retrieved successfully
MVI_ERR_NOACCESS	comport has not been opened
MVI_ERR_BADPARAM	Invalid pointer

Example

```
int count;

if (MVIsp_GetCountUnread(COM2,&count) == MVI_SUCCESS)
{
   if (count == 0)
      printf("All chars read\n");
   else
      printf("%d characters remaining\n",count);
}
```

See Also

MVIsp_Getch (page 209)

MVIsp_Gets (page 214)

MVIsp_GetData (page 216)

MVIsp_PurgeDataUnsent

Syntax

```
int MVIsp_PurgeDataUnsent(int comport);
```

Parameters

comport	Port whose transmit data is to be purged

Description

MVIsp_PurgeDataUnsent deletes all data waiting in the transmit queue. The data is discarded

and is not transmitted.

Comport specifies the port whose transmit queue is to be purged.

Return Value

MVI_SUCCESS	The data was purged successfully
MVI_ERR_BADPARAM	invalid comport
MVI_ERR_NOACCESS	The comport has not been opened

Example

```
if (MVIsp_PurgeDataUnsent(COM1) == MVI_SUCCESS)
printf("Transmit Data purged.\n");
```

See Also

MVIsp_PurgeDataUnread (page 221)

MVIsp_PurgeDataUnread

Syntax

```
int MVIsp_PurgeDataUnread(int comport)
```

Parameters

comport	Port whose receive data is to be purged

Description

MVIsp_PurgeDataUnread deletes all data waiting in the receive queue. The data is discarded and is no longer available for reading.

Note: If handshaking is enabled and the transmitting serial device has been paused, this function will release the transmitting serial device to resume transmission.

Return Value

MVI_SUCCESS	The data was purged successfully
MVI_ERR_BADPARAM	invalid comport
MVI_ERR_NOACCESS	The comport has not been opened

Example

```
if (MVIsp_PurgeDataUnread(COM1) == MVI_SUCCESS)
printf("Transmit Data purged.\n");
```

See Also

MVIsp_PurgeDataUnsent (page 220)

11.5 Serial Port API Miscellaneous Functions

MVIsp_GetVersionInfo

Syntax

```
int MVIsp_GetVersionInfo(MVISPVERSIONINFO *verinfo);
```

Parameters

verinfo	Pointer to structure of type MVISPVERSIONINFO

Description

MVIsp_GetVersionInfo Retrieves the current version of the API. The version information is returned in the structure verinfo.

The MVISPVERSIONINFO structure is defined as follows:

```
typedef struct tagMVISPVERSIONINFO
{
   WORD   APISeries;      /* API series */
   WORD   APIRevision;   /* API revision */
} MVISPVERSIONINFO;
```

Return Value

MVI_SUCCESS	The version information was read successfully.

Example

```
MVISPVERSIONINFO      verinfo;

/* print version of API library */
MVIsp_GetVersionInfo(&verinfo);
printf("Library Series %d, Rev %d\n", verinfo.APISeries,
verinfo.APIRevision);
```

12 Product Specifications

In This Chapter

The PTQ Application Development Module is an backplane compatible module that allows user-developed 'C' applications to operate on the platform. A great way to speed up custom ASCII data communications or to protect a proprietary algorithm, the ADM is a powerful tool for the platform.

Powerful platform for developing and running 'C' applications on Schneider Electric's processors. The PTQ-ADM module is a single slot, backplane compatible solution for the platform. This module is a powerful and programmable solution supporting two fully isolated serial ports allowing the many serial field devices to be integrated into the platform.

The PTQ-ADM module has three serial ports, two of which are isolated for field interfaces:

- CFG: Debug/configuration RS-232
- PRT1: Application RS-232, RS-422 or RS-485
- PRT2: Application RS-232, RS-422 or RS-485

PRT1 and PRT2 are jumper configured for direct or multi-drop field communication. The application program can be written to control the two application ports independently, allowing maximum flexibility in the design.

12.1 General Specifications

- Single Slot - Quantum backplane compatible
- The module is recognized as an Options module and has access to PLC memory for data transfer
- Configuration data is stored in non-volatile memory in the ProTalk module
- Up to six modules can be placed in a rack
- Local rack - The module must be placed in the same rack as processor
- Compatible with all common Quantum programming packages, including Concept (version 2.6 or higher), Unity Pro (version 2.2 or higher), ProWORX (version 2.20 or later), and ModSoft
- Quantum data types supported: 3x, 4x
- High speed data transfer across backplane provides quick data update times
- Sample ladder file available

12.2 Hardware Specifications

Specification	Value
Backplane Current Load	1100 mA maximum @ 5 Vdc ± 5%
Operating Temperature	0°C to 60°C (32°F to 140°F)
Storage Temperature	-40°C to 85°C (-40°F to 185°F)
Relative Humidity	5% to 95% (without condensation)
Vibration	Sine vibration 4-100 Hz in each of the 3 orthogonal axes
Shock	30G, 11 mSec. in each of the 3 orthogonal axes
Dimensions (HxWxD), Approx.	250 x 103.85 x 40.34 mm 9.84 x 4.09 x 1.59 in
LED Indicators	Module Status Backplane Transfer Status Serial Port Activity LED Serial Activity and Error LED Status
Configuration Serial Port (PRT1)	DB-9M PC Compatible RS-232 only No hardware handshaking
Application Serial Ports	(PRT2, PRT3) DB-9M PC Compatible RS-232/422/485 jumper selectable RS-422/485 screw termination included RS-232 handshaking configurable 500V Optical isolation from backplane

12.3 Functional Specifications

The PTQ ADM API Suite allows software developers to access the Quantum backplane and serial ports without needing detailed knowledge of the module's hardware design.

Serial Port API Functions

The serial port API provides a common interface to the serial ports across all of the PTQ hardware platforms. Functions include configuring, opening, closing, controlling and monitoring the serial port, and sending and receiving serial data

Backplane API Functions

The backplane API provides an interface to transfer data between the module and the processor over the backplane. Functions include initialization, configuration, direct I/O access, synchronization, messaging, and control of the console and LEDs.

ADM API Functions

The ADM API provides an interface to initialize the API, control the debug port, read and write data to the database, start and check timers, transfer data over the backplane, parse configuration files, set user LED indicators, and configure the console.

Module Specifications

Module

- User-definable module memory usage, supporting the storage and transfer of up to 5000 registers to/from the control processor
- Floating-point data movement support

Development Environment

- Operating system: General software DOS 6-XL
- Compatible compilers (16-bit DOS target)
  - Digital Mars C++ V8.49 (included)
  - Borland C++ V5.02

13 DOS 6 XL Reference Manual

The DOS 6 XL Reference Manual makes reference to compilers other than Digital Mars C++ or Borland Compilers. The PTQ-ADM and ADMNET modules only support Digital Mars C++ and Borland C/C++ Compiler Version 5.02. References to other compilers should be ignored.

14 Support, Service & Warranty

In This Chapter

14.1 Contacting Technical Support

ProSoft Technology, Inc. (ProSoft) 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, an after-hours answering system allows 24-hour/7-days-a-week pager access to one of our qualified Technical and/or Application Support Engineers. Detailed contact information for all our worldwide locations is available on the following page.*

Internet	Web Site: www.prosoft-technology.com/support E-mail address: support@prosoft-technology.com
Asia Pacific (location in Malaysia)	Tel: +603.7724.2080, E-mail: asiapc@prosoft-technology.com Languages spoken include: Chinese, English
Asia Pacific (location in China)	Tel: +86.21.5187.7337 x888, E-mail: asiapc@prosoft-technology.com Languages spoken include: Chinese, English
Europe (location in Toulouse, France)	Tel: +33 (0) 5.34.36.87.20, E-mail: support.EMEA@prosoft-technology.com Languages spoken include: French, English
Europe (location in Dubai, UAE)	Tel: +971-4-214-6911, E-mail: mea@prosoft-technology.com Languages spoken include: English, Hindi
North America (location in California)	Tel: +1.661.716.5100, E-mail: support@prosoft-technology.com Languages spoken include: English, Spanish
Latin America (Oficina Regional)	Tel: +1-281-2989109, E-Mail: latinam@prosoft-technology.com Languages spoken include: Spanish, English
Latin America (location in Puebla, Mexico)	Tel: +52-222-3-99-6565, E-mail: soporte@prosoft-technology.com Languages spoken include: Spanish
Brasil (location in Sao Paulo)	Tel: +55-11-5083-3776, E-mail: brasil@prosoft-technology.com Languages spoken include: Portuguese, English

14.2 Warranty Information

Complete details regarding ProSoft Technology’s TERMS AND CONDITIONS OF SALE, WARRANTY, SUPPORT, SERVICE AND RETURN MATERIAL AUTHORIZATION INSTRUCTIONS can be found at www.prosoft-technology.com/warranty.

Documentation is subject to change without notice.

Glossary of Terms

A

API

Application Program Interface

B

Backplane

Refers to the electrical interface, or bus, to which modules connect when inserted into the rack. The module communicates with the control processor(s) through the processor backplane.

BIOS

Basic Input Output System. The BIOS firmware initializes the module at power up, performs self-diagnostics, and provides a DOS-compatible interface to the console and Flashes the ROM disk.

Byte

8-bit value

C

CIP

Control and Information Protocol. This is the messaging protocol used for communications over the ControlLogix backplane. Refer to the ControlNet Specification for information.

Connection

A logical binding between two objects. A connection allows more efficient use of bandwidth, because the message path is not included after the connection is established.

Consumer

A destination for data.

Controller

The PLC or other controlling processor that communicates with the module directly over the backplane or via a network or remote I/O adapter.

D

DLL

Dynamic Linked Library

E

Embedded I/O

Refers to any I/O which may reside on a CAM board.

ExplicitMsg

An asynchronous message sent for information purposes to a node from the scanner.

H

HSC

High Speed Counter

I

Input Image

Refers to a contiguous block of data that is written by the module application and read by the controller. The input image is read by the controller once each scan. Also referred to as the input file.

L

Library

Refers to the library file containing the API functions. The library must be linked with the developer's application code to create the final executable program.

Linked Library

Dynamically Linked Library. See Library.

Local I/O

Refers to any I/O contained on the CPC base unit or mezzanine board.

Long

32-bit value.

M

Module

Refers to a module attached to the backplane.

Mutex

A system object which is used to provide mutually-exclusive access to a resource.

MVI Suite

The MVI suite consists of line products for the following platforms:

- Flex I/O
- ControlLogix
- SLC
- PLC

- CompactLogix

MVI46

MVI46 is sold by ProSoft Technology under the MVI46-ADM product name.

MVI56

MVI56 is sold by ProSoft Technology under the MVI56-ADM product name.

MVI69

MVI69 is sold by ProSoft Technology under the MVI69-ADM product name.

MVI71

MVI71 is sold by ProSoft Technology under the MVI71-ADM product name.

MVI94

MVI94 and MVI94AV are the same modules. The MVI94AV is now sold by ProSoft Technology under the MVI94-ADM product name

O

Originator

A client that establishes a connection path to a target.

Output Image

Table of output data sent to nodes on the network.

P

Producer

A source of data.

PTO

Pulse Train Output

PTQ Suite

The PTQ suite consists of line products for Schneider Electronics platforms: Quantum (ProTalk)

S

Scanner

A DeviceNet node that scans nodes on the network to update outputs and inputs.

Side-connect

Refers to the electronic interface or connector on the side of the PLC-5, to which modules connect directly through the PLC using a connector that provides a fast communication path between the - module and the PLC-5.

T

Target

The end-node to which a connection is established by an originator.

Thread

Code that is executed within a process. A process may contain multiple threads.

W

Word

16-bit value

Index