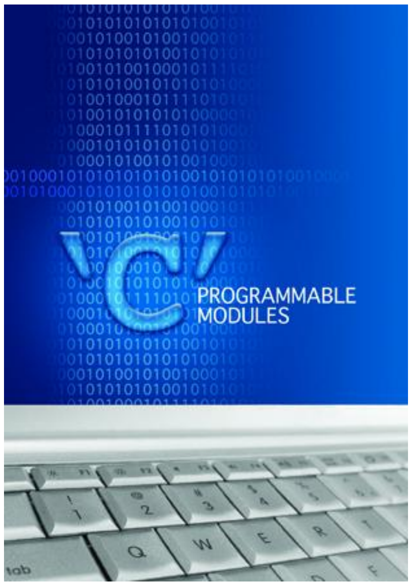
ProSoft TECHNOLOGY

Where Automation Connects.



ProTalk®

PTQ-ADMNET

'C' Programmable

‘C’ Programmable Network Interface with Ethernet for Quantum

February 20, 2013

DEVELOPER'S 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

Copyright © 2013 ProSoft Technology, Inc., all rights reserved.

PTQ-ADMNET Developer's Guide

February 20, 2013

ProSoft Technology ®, ProLinx ®, inRAx ®, ProTalk ®, and RadioLinx ® are Registered Trademarks of ProSoft Technology, Inc. All other brand or product names are or may be trademarks of, and are used to identify products and services of, their respective owners.

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

This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither ProSoft Technology nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. Information in this document including illustrations, specifications and dimensions may contain technical inaccuracies or typographical errors. ProSoft Technology makes no warranty or representation as to its accuracy and assumes no liability for and reserves the right to correct such inaccuracies or errors at any time without notice. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of ProSoft Technology. All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components. When devices are used for applications with technical safety requirements, the relevant instructions must be followed. Failure to use ProSoft Technology software or approved software with our hardware products may result in injury, harm, or improper operating results. Failure to observe this information can result in injury or equipment damage.

© 2013 ProSoft Technology. All rights reserved.

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%; 3mA @ 24 Vdc
- Operating Temperature: 0°C to 60°C (32°F to 140°F)
- Storage Temperature: -40°C to 85°C (-40°F to 185°F)
- Shock: 30 g operational; 50 g non-operational; Vibration: 5 g from 10 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

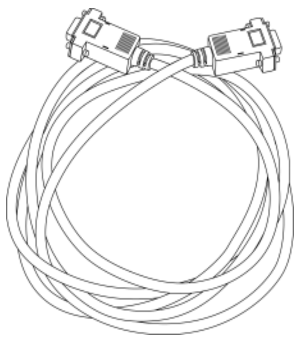
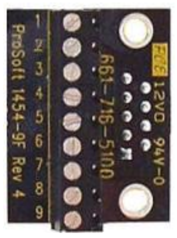
In This Chapter

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.

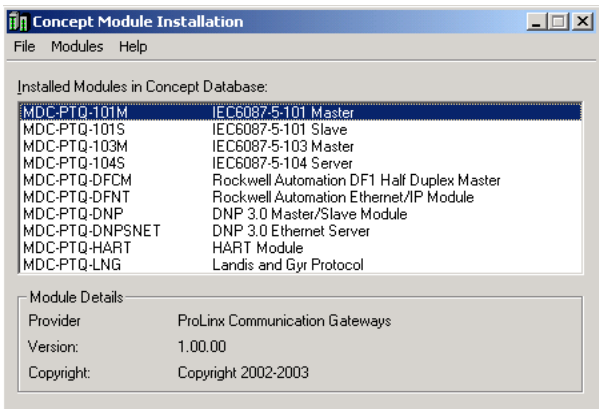
1.2 Information for Concept Version 2.6 Users

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.


Concept Module Installation
File
Modules
Help
Installed Modules in Concept Database:

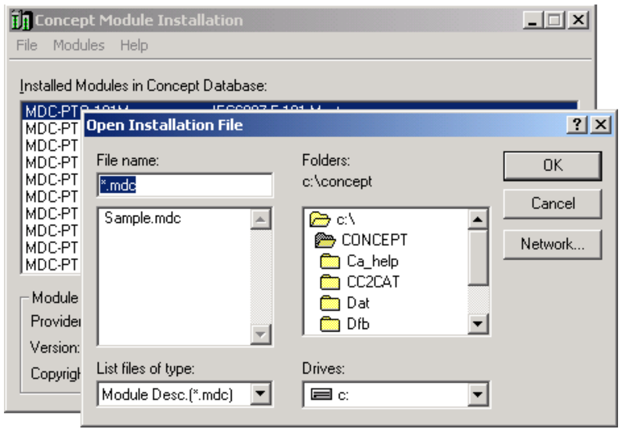
MDC-PTQ-101M	IEC6087-5-101 Master
MDC-PTQ-101S	IEC6087-5-101 Slave
MDC-PTQ-103M	IEC6087-5-103 Master
MDC-PTQ-104S	IEC6087-5-104 Server
MDC-PTQ-DFCM	Rockwell Automation DF1 Half Duplex Master
MDC-PTQ-DFNT	Rockwell Automation Ethernet/IP Module
MDC-PTQ-DNP	DNP 3.0 Master/Slave Module
MDC-PTQ-DNPSNET	DNP 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:

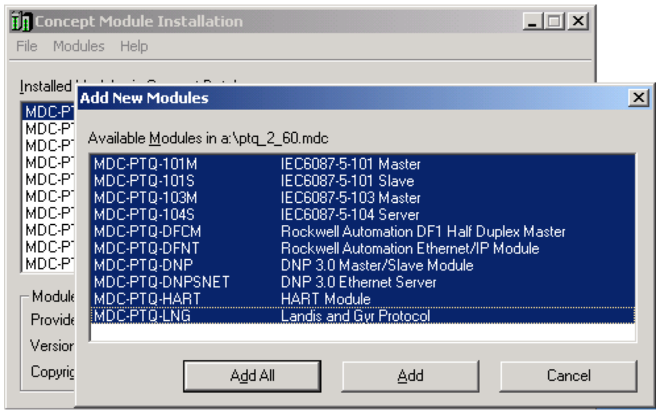


3 If you are using a Quantum processor, you will need the MDC files. In the Open Installation File dialog box, navigate to the **MDC FILES** directory on the ProTalk CD.

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

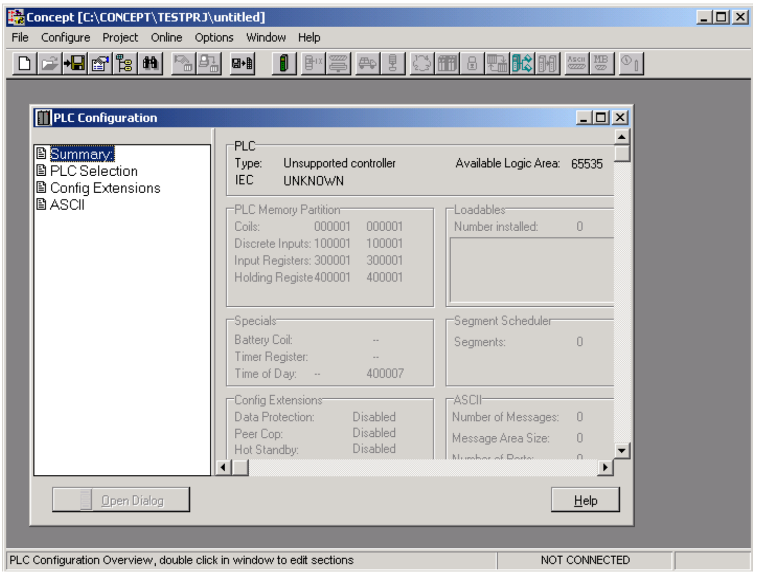
The following steps are designed to ensure that the processor is able to transfer data successfully with the PTQ module. As part of this procedure, you will use Concept configuration software from Schneider Electric to create a project, add the PTQ module to the project, set up data memory for the project,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

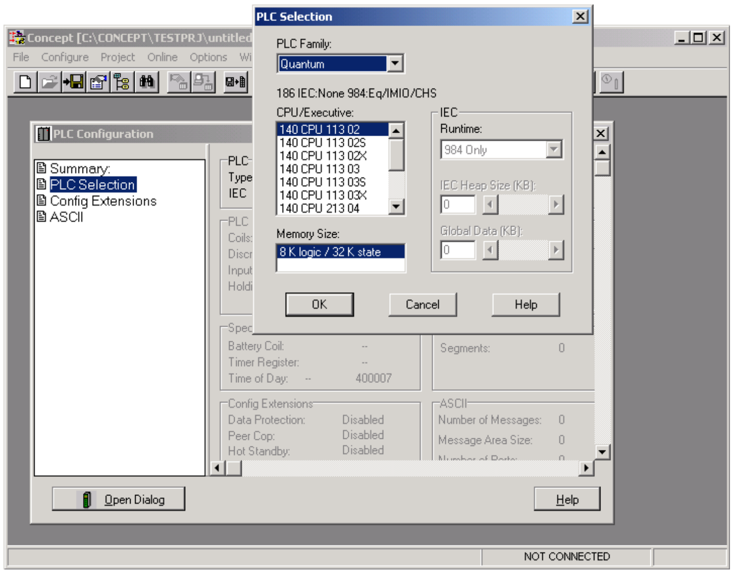
Segment Scheduler
Segments: 0

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

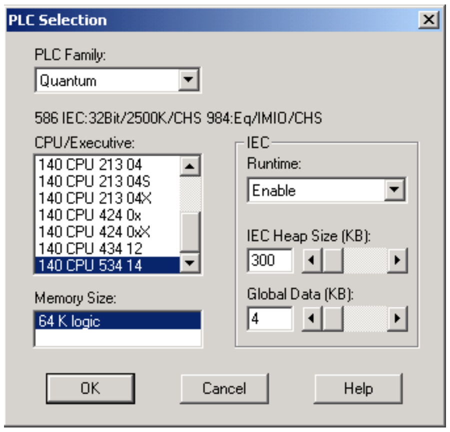
186 IEC:None 984:Eq/IMIO/CHS

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 Family: Quantum

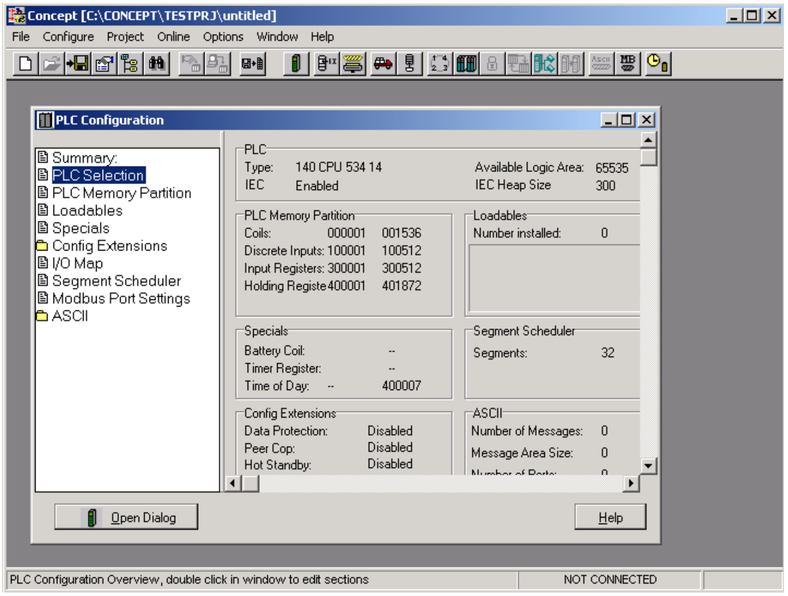
586 IEC:32Bit/2500K/CHS 984:Eq/IMIO/CHS

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

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

Memory Size: 64 K logic

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



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

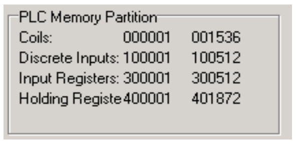
Segment Scheduler
Segments: 32

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

ASCII
Number of Messages: 0
Message Area Size: 0

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.

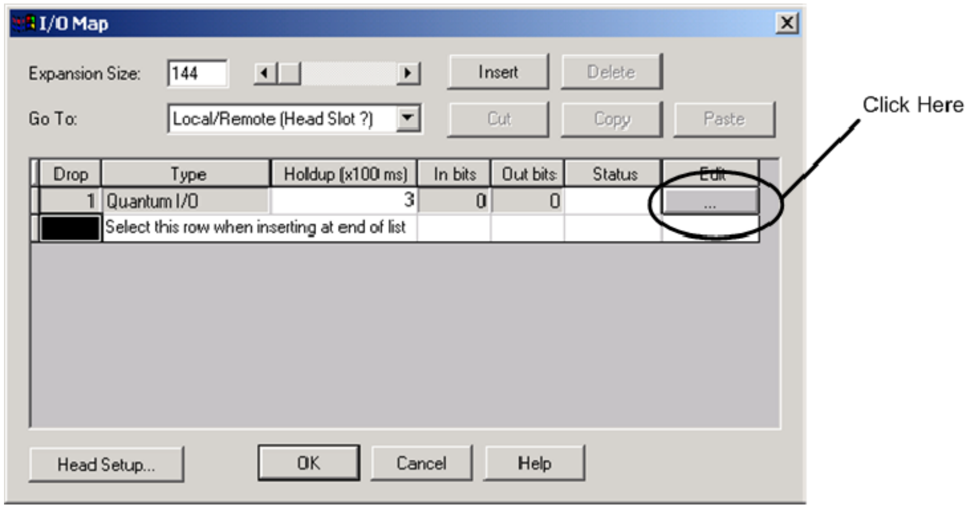


PLC Memory Partition

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

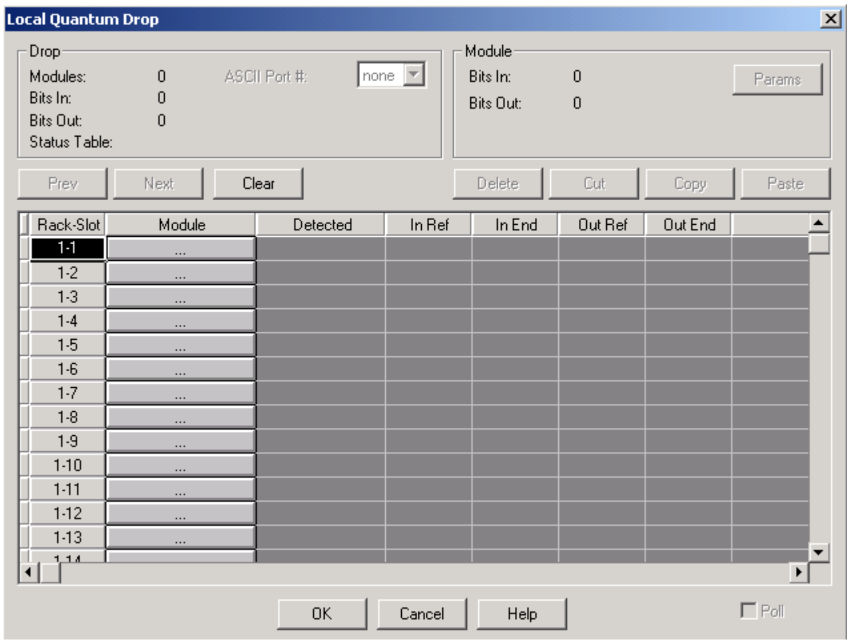
2.2 Add the PTQ Module to the Project

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.



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.

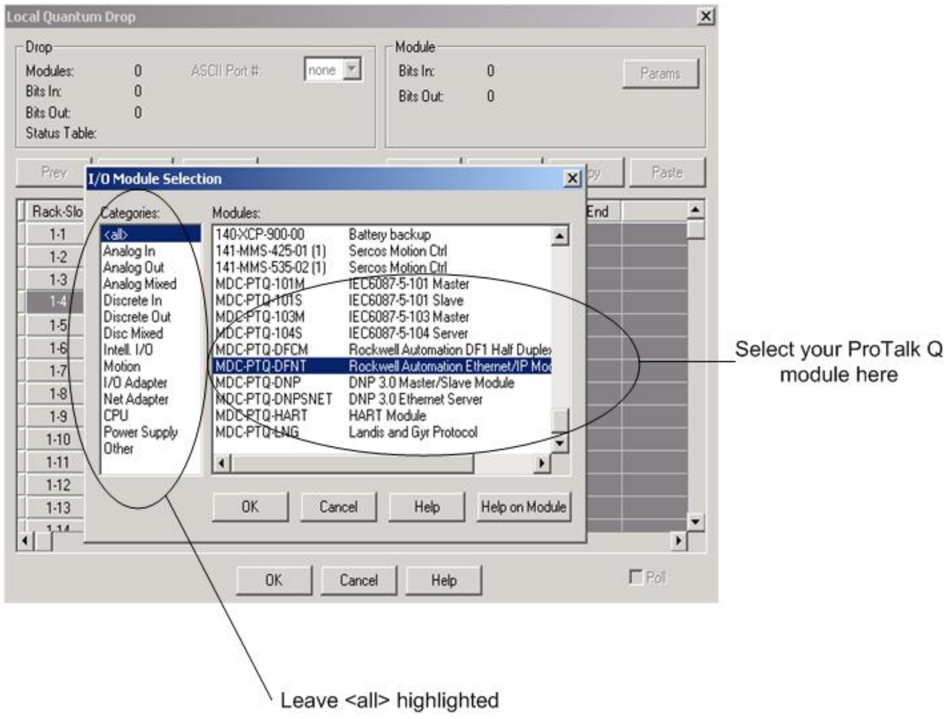


Modules:	0
Bits In:	0
Bits Out:	0
Status Table:	

Bits In:	0
Bits Out:	0

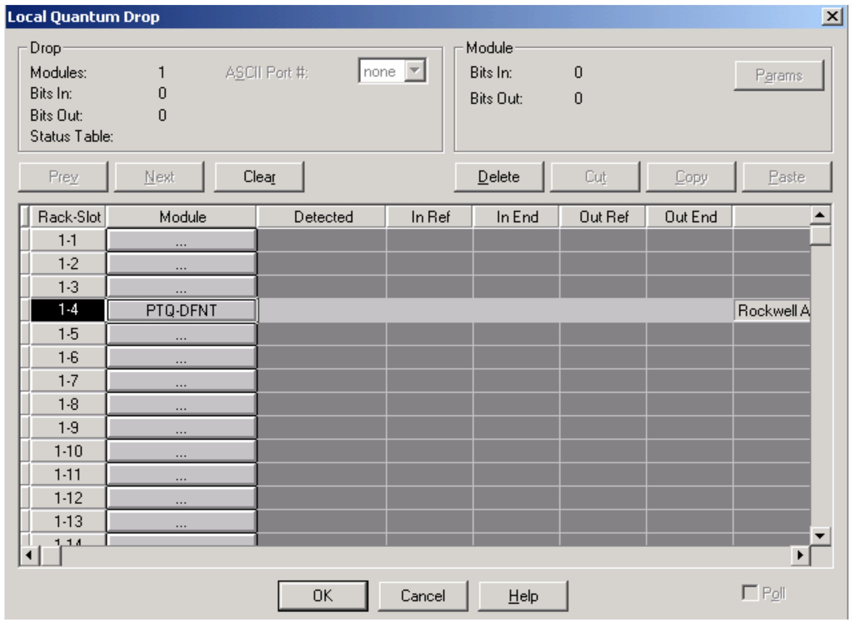
Rack-Slot	Module	Detected	In Ref	In End	Out Ref	Out End
1-1	...					
1-2	...					
1-3	...					
1-4	...					
1-5	...					
1-6	...					
1-7	...					
1-8	...					
1-9	...					
1-10	...					
1-11	...					
1-12	...					
1-13	...					

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.



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.



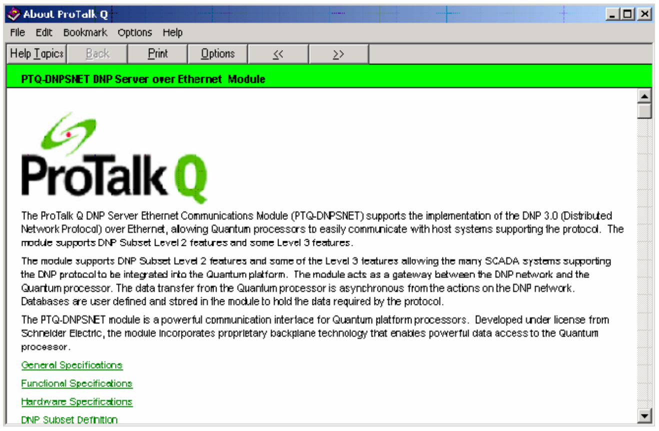
Drop
Modules: 1
Bits In: 0
Bits Out: 0
Status Table:
ASCII Port #: none

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.


About ProTalk Q
File Edit Bookmark Options Help
Help Topics Back Print Options << >>
ProTalk Q

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

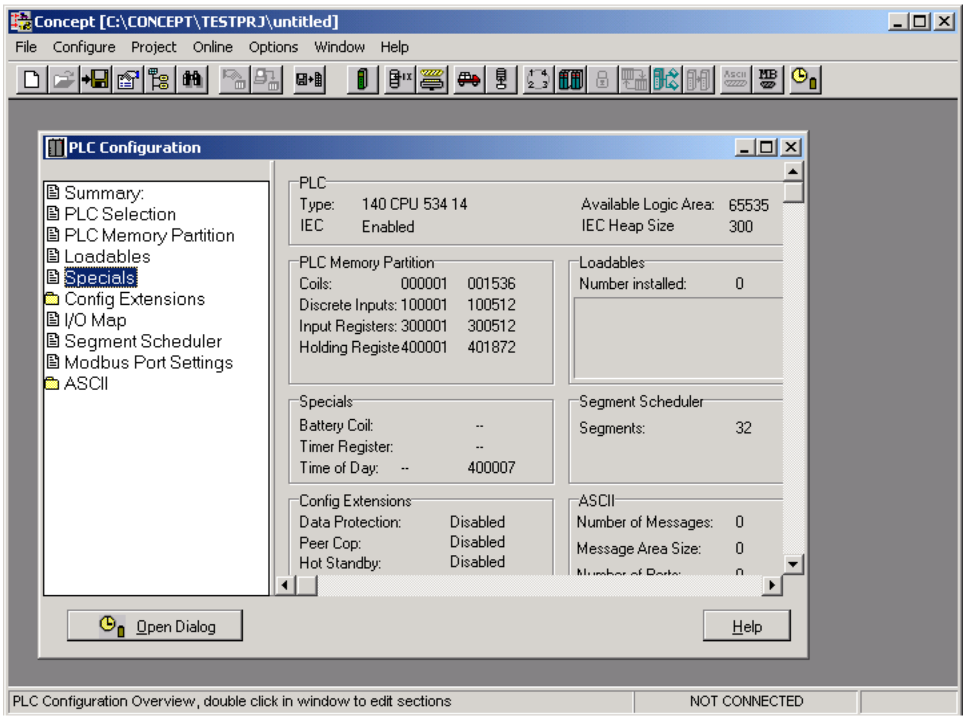
Functional Specifications

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

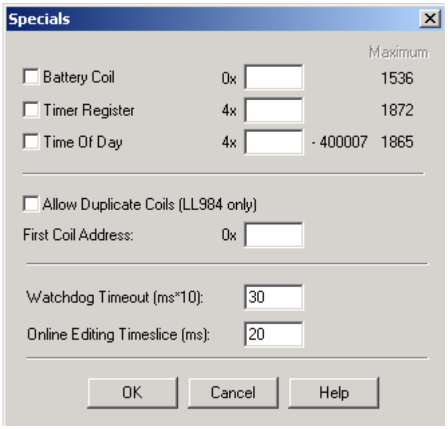
Segment Scheduler
Segments: 32

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

ASCII
Number of Messages: 0
Message Area Size: 0

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

2 This action opens the **SPECIALS** dialog box.



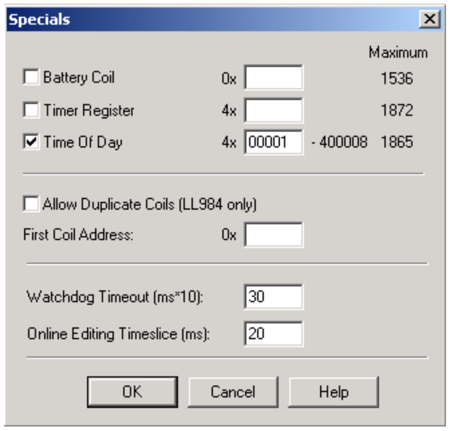
☐ Battery Coil	0x			1536
☐ Timer Register	4x			1872
☐ Time Of Day	4x		- 400007	1865

☐ Allow Duplicate Coils (LL984 only)
First Coil Address: 0x

Watchdog Timeout (ms*10): 30
Online Editing Timeslice (ms): 20

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

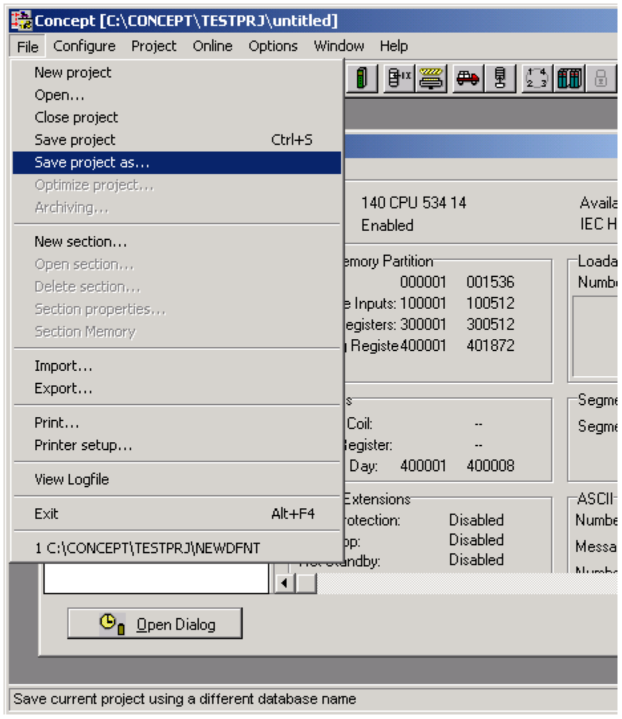
☐ Allow Duplicate Coils (LL984 only)
First Coil Address: 0x

Watchdog Timeout (ms*10): 30
Online Editing Timeslice (ms): 20

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.**



Concept [C:\CONCEPT\TESTPRJ\untitled]

140 CPU 534 14
Enabled

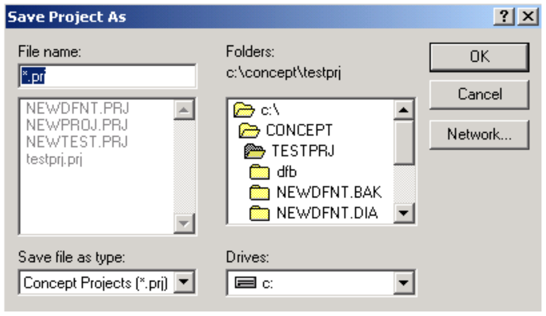
emory Partition
000001 001536
e Inputs: 100001 100512
egisters: 300001 300512
Registe 400001 401872

Coil: --
egister: --
Day: 400001 400008

Extensions
otection: Disabled
p: Disabled
ndby: Disabled

Save current project using a different database name

2 This action opens the **SAVE PROJECT AS** dialog box.

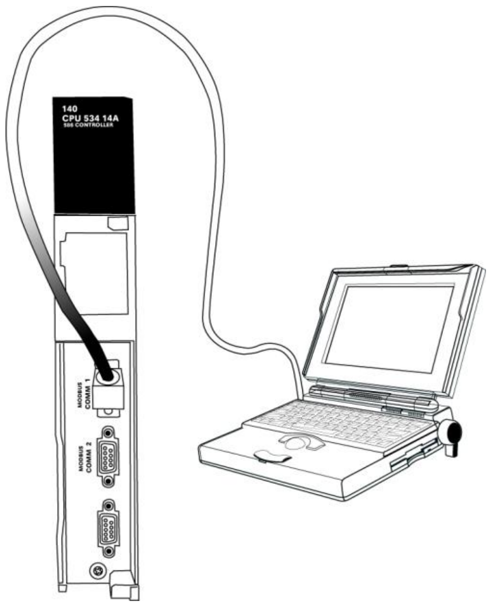


3 Name the project, and then click **OK** to save the project to a file.

2.4 Download the Project to the Processor

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

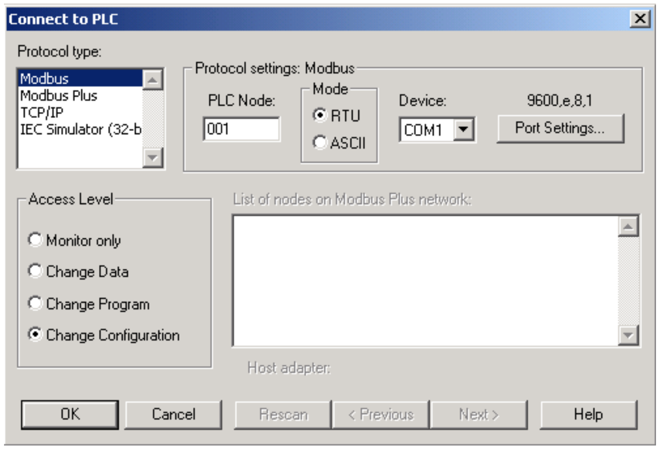
1 Use the null modem cable to connect your PC's serial port to the Quantum processor, as shown in the following illustration.



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.



Protocol settings: Modbus

PLC Node: 001

Mode: [✓ RTU] ASCII

Device: COM1

9600,e,8,1

Port Settings...

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

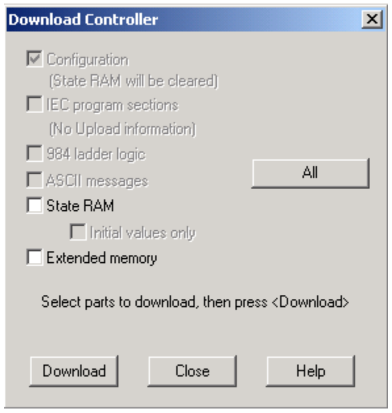
List of nodes on Modbus Plus network:

Host adapter:

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

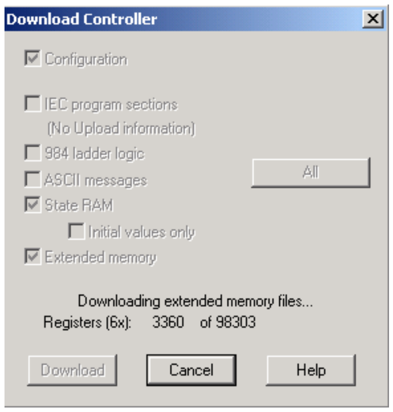
Note: Click **OK** to dismiss any message boxes that appear during the connection process.

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.



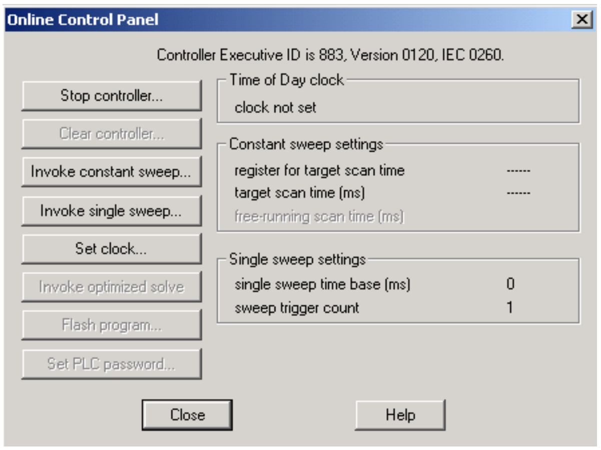
Downloading extended memory files...
Registers (6x): 3360 of 98303

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.



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

Time of Day clock
clock not set

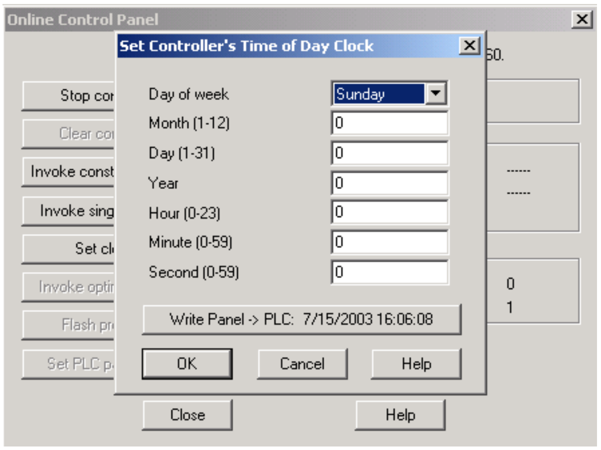
Constant sweep settings

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.



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

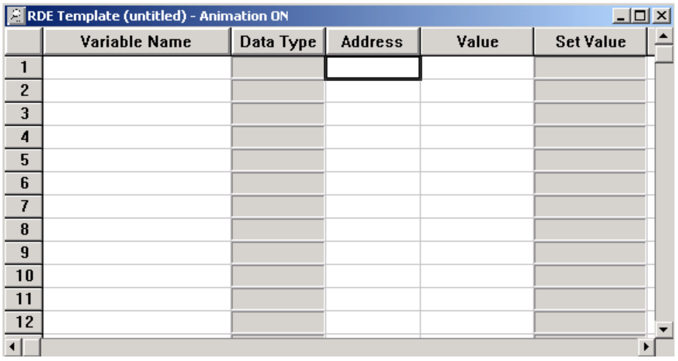
Write Panel -> PLC: 7/15/2003 16:06:08

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.

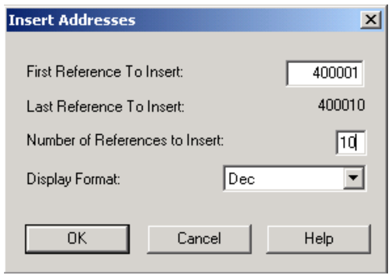
6 Place the cursor over the first address field, as shown in the following illustration.



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

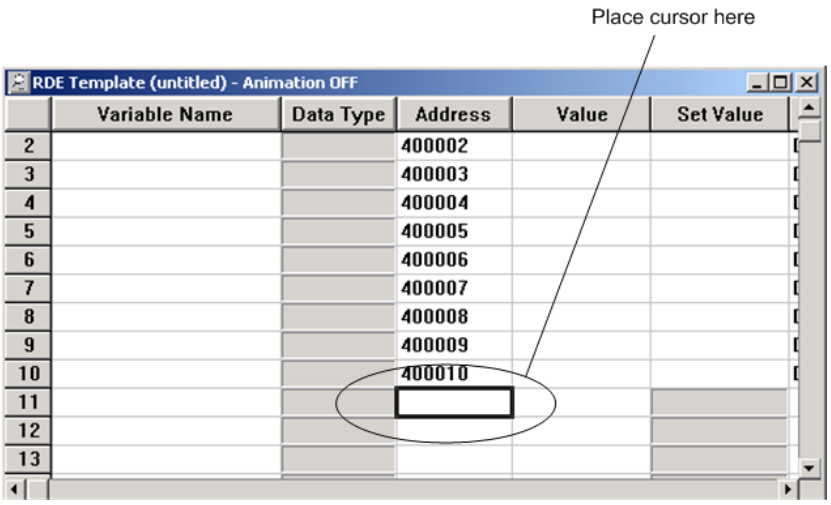
7 In the **PLC CONFIGURATION** window, open the **TEMPLATES** menu, and then choose **INSERT ADDRESSES.** This action opens the Insert addresses dialog box.

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



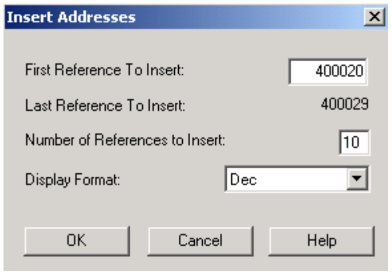
First Reference To Insert: 400001
Last Reference To Insert: 400010
Number of References to Insert: 10
Display Format: Dec

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					

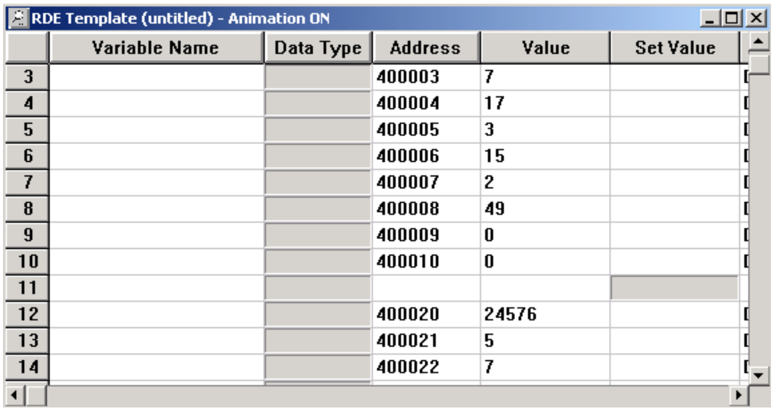
10 Repeat steps 6 through 9, using the values in the following illustration:



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.

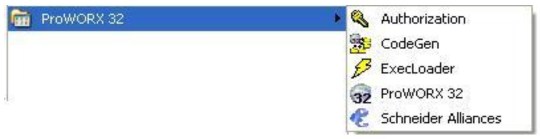
You are now ready to complete the installation and setup of the ProTalk module.

3 Configuring the Processor with ProWORX

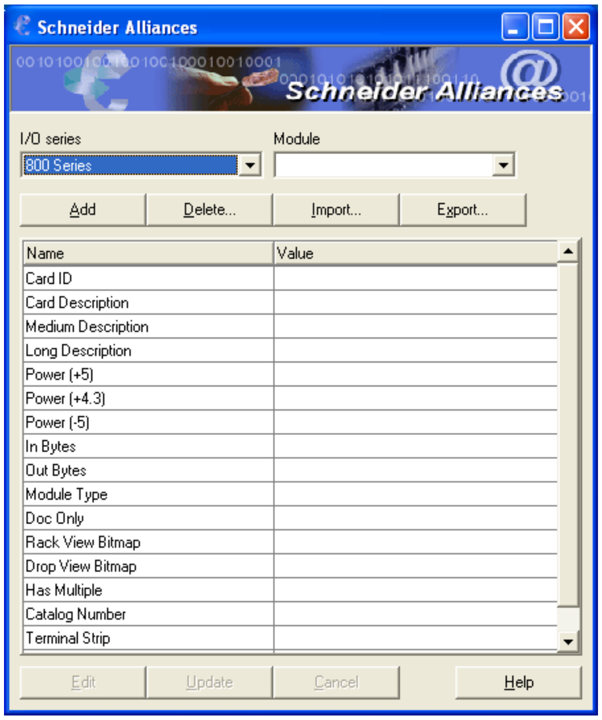
When you use ProWORX 32 software to configure the processor, use the example SAF file provided on the ProTalk Solutions CD-ROM.

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:

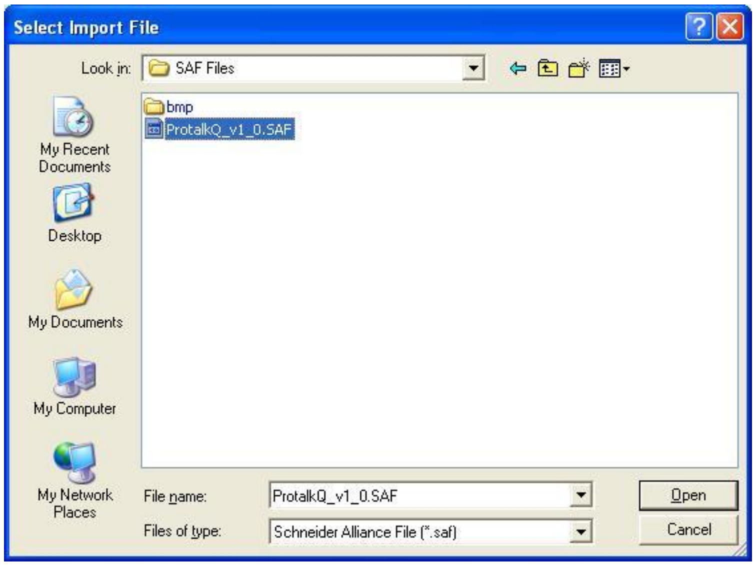


2 Click on **IMPORT…**



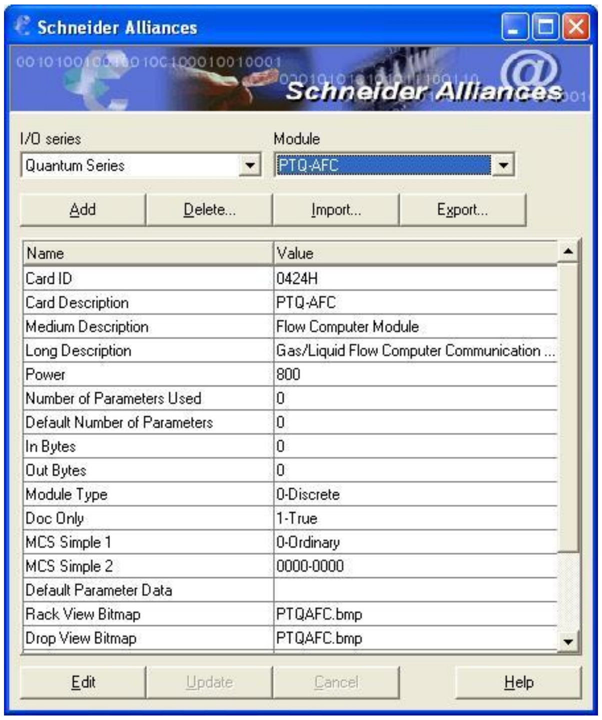
Name	Value
Card ID	
Card Description	
Medium Description	
Long Description	
Power (+5)	
Power (+4.3)	
Power (-5)	
In Bytes	
Out Bytes	
Module Type	
Doc Only	
Rack View Bitmap	
Drop View Bitmap	
Has Multiple	
Catalog Number	
Terminal Strip	

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



File name: ProtalkQ_v1_0.SAF
Files of type: Schneider Alliance File (*.saf)

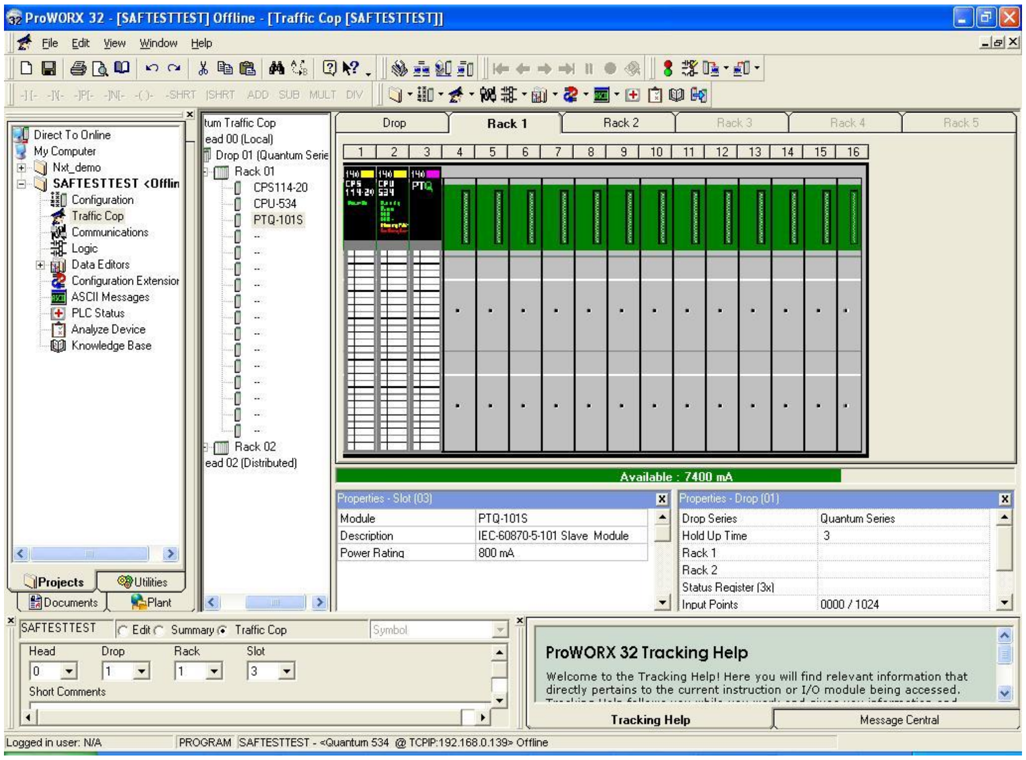
4 After you click on **OPEN** you should see the PTQ modules imported (select **I/O SERIES** as **QUANTUM**):



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

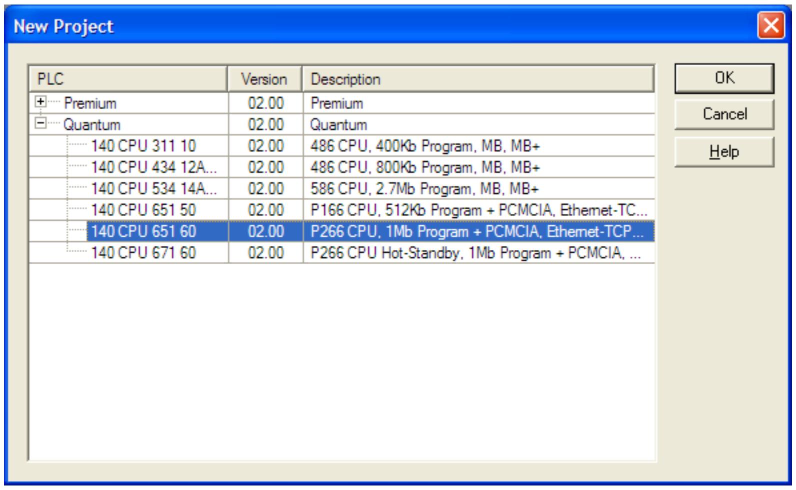
In This Chapter

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

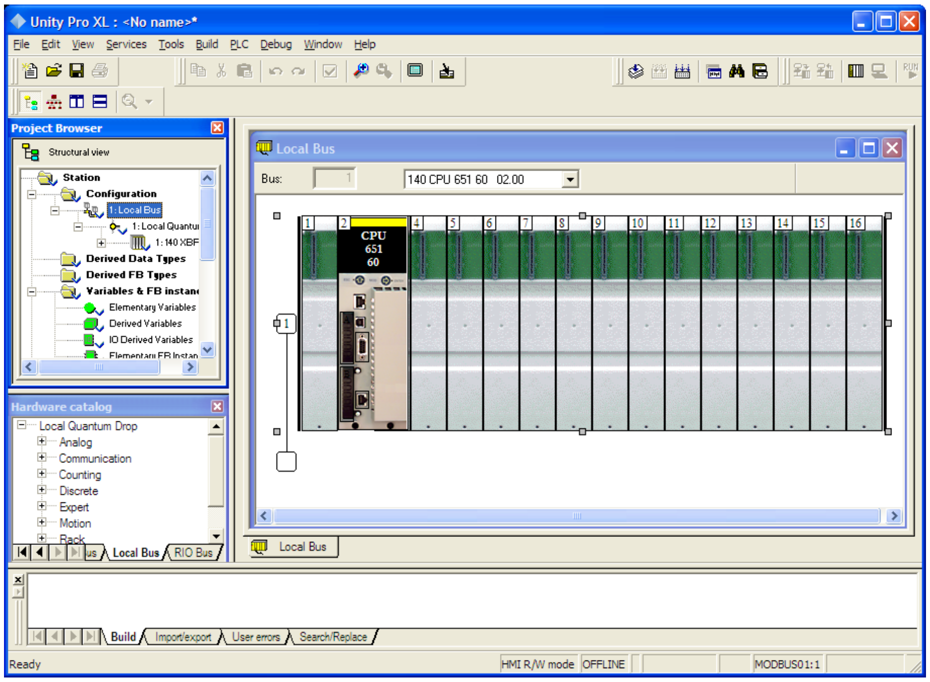
The first step is to open Unity Pro and create a new project.

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.

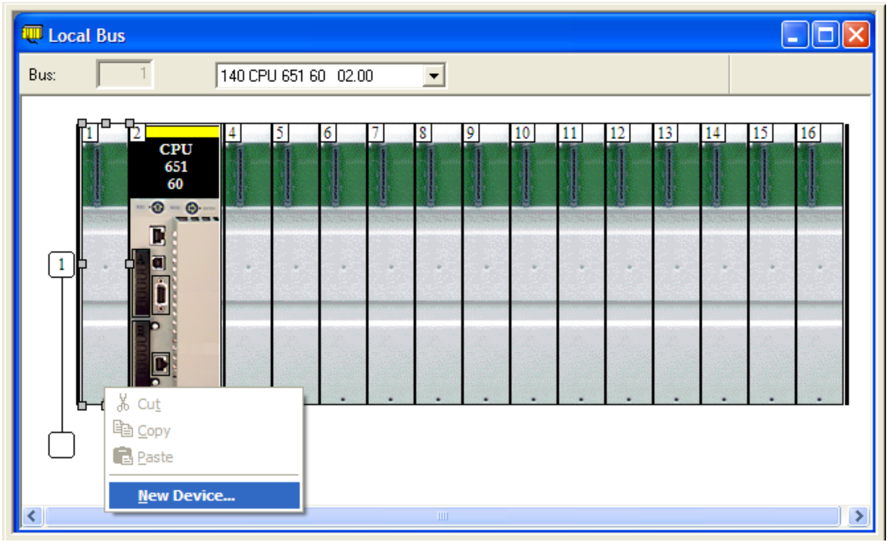


PLC	Version	Description
⊞ Premium	02.00	Premium
⊟ Quantum	02.00	Quantum
140 CPU 311 10	02.00	486 CPU, 400Kb Program, MB, MB+
140 CPU 434 12A...	02.00	486 CPU, 800Kb Program, MB, MB+
140 CPU 534 14A...	02.00	586 CPU, 2.7Mb Program, MB, MB+
140 CPU 651 50	02.00	P166 CPU, 512Kb Program + PCMCIA, Ethernet-TC...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

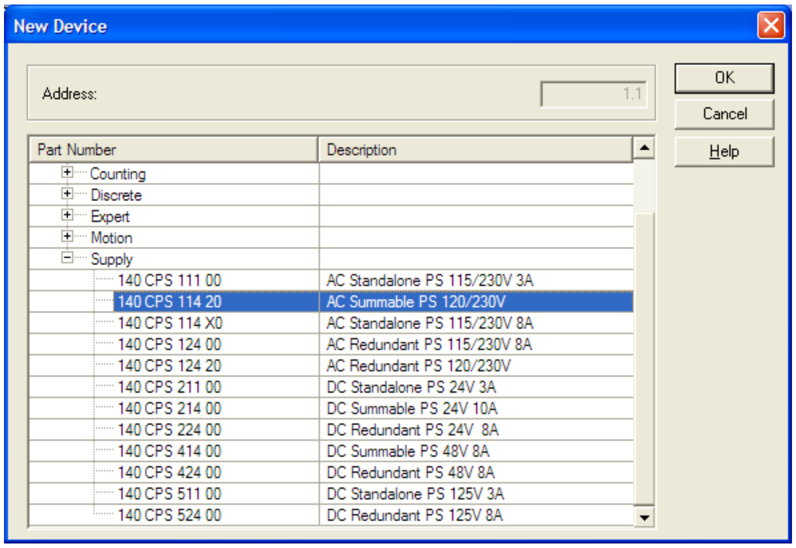
2. Next, add a power supply to the project. In the **PROJECT BROWSER**, expand the **CONFIGURATION** folder, and then double-click the **1:LOCALBUS** icon. This action opens a graphical window showing the arrangement of devices in your Quantum rack.



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.



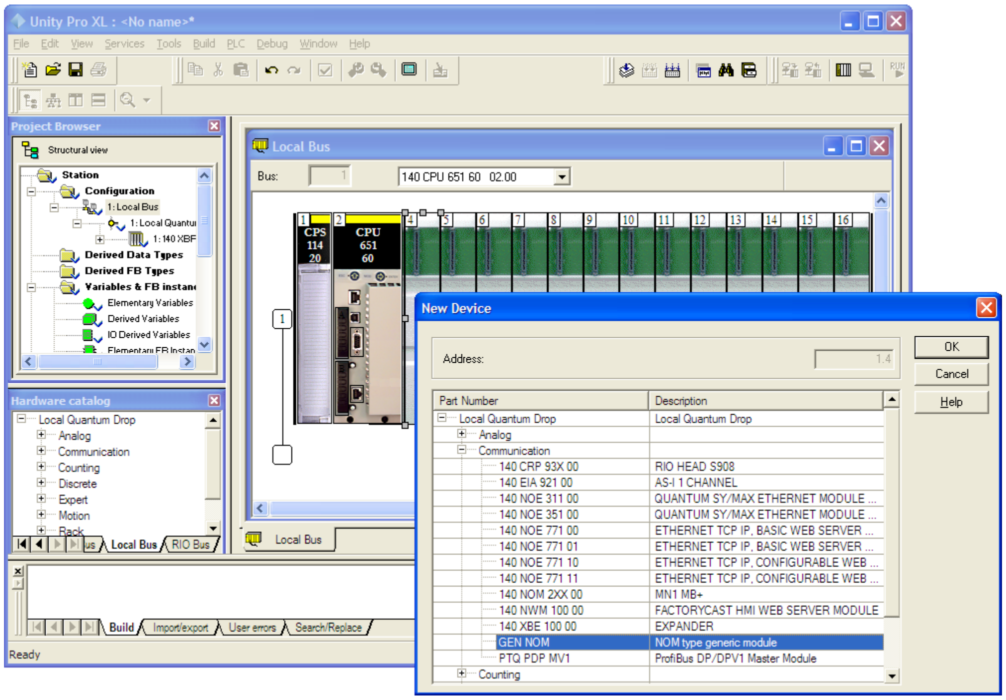
Address: 1.1

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

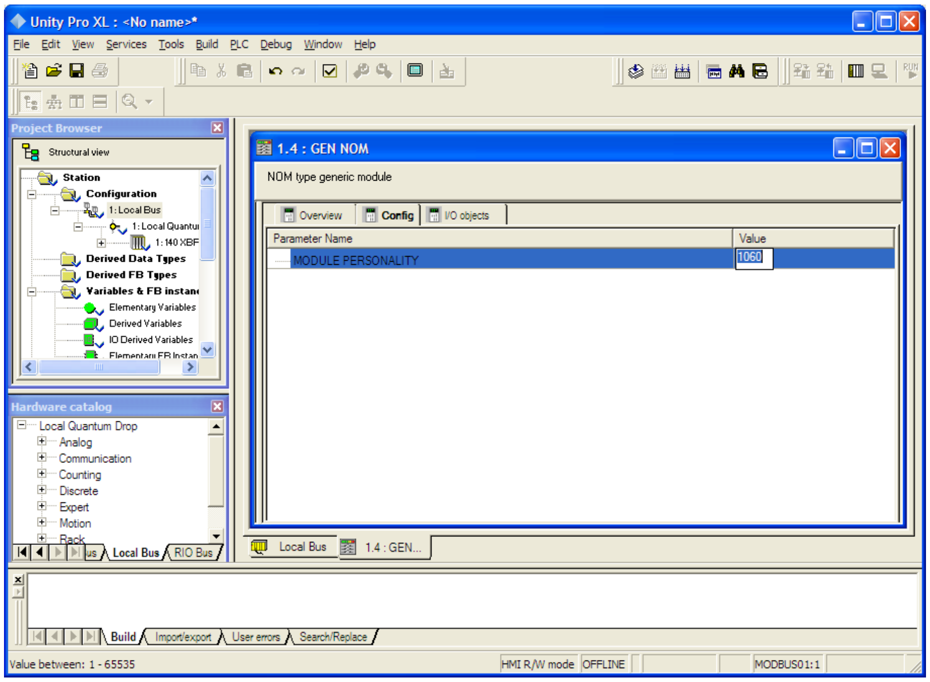
1 Expand the **COMMUNICATION** tree, and select **GEN NOM**. This module type provides extended communication capabilities for the Quantum system, and allows communication between the PLC and the PTQ module without requiring additional programming.



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

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.

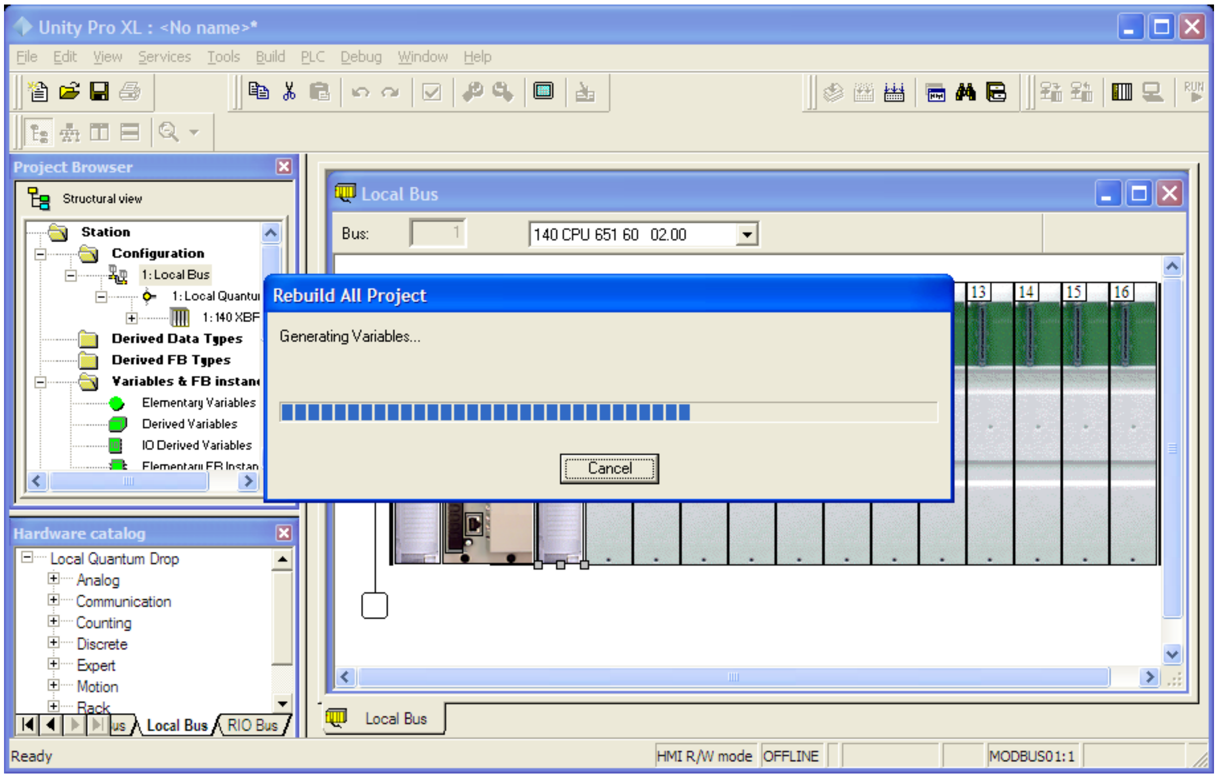
4.3 Build the Project

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.

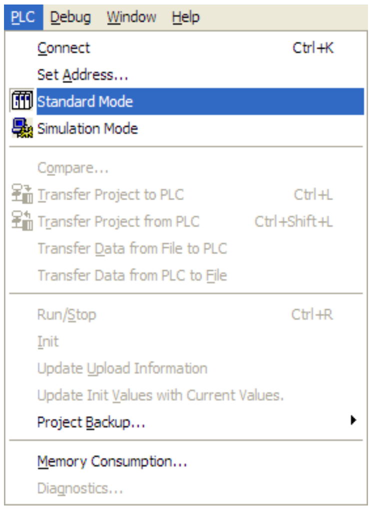
4.4 Connect Your PC to the Processor

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.

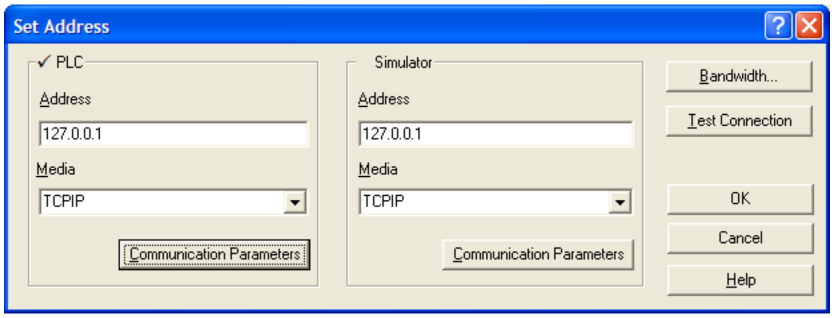
Note: If you have never connected from the PC to your processor before, you must verify that the necessary port drivers are installed and available to Unity Pro.

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...

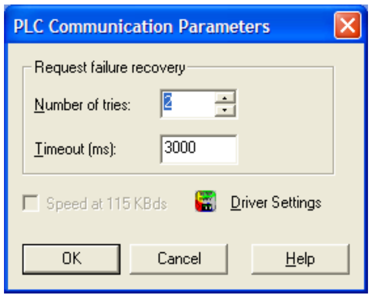
2 Open the **PLC** menu, and choose **SET ADDRESS...** This action opens the **SET ADDRESS** dialog box. Open the **MEDIA** dropdown list and choose the connection type to use (TCPIP or USB).



✓ 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.

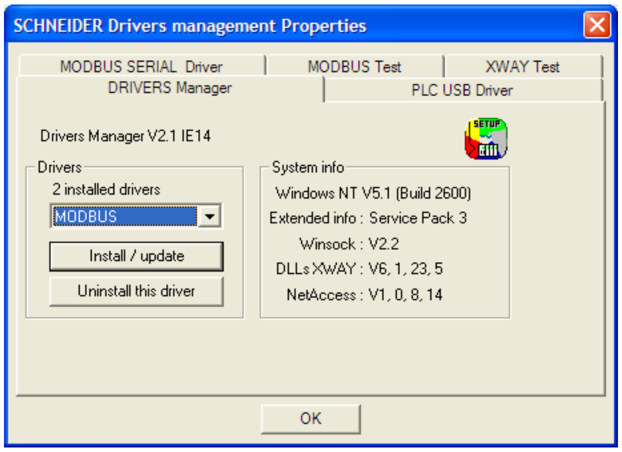

PLC Communication Parameters

Request failure recovery

Number of tries: 2

Timeout (ms): 3000

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)

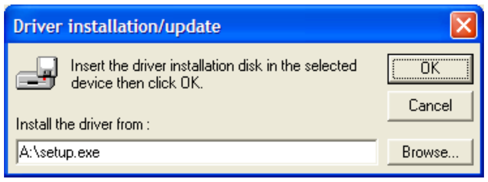
Extended info : Service Pack 3

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.

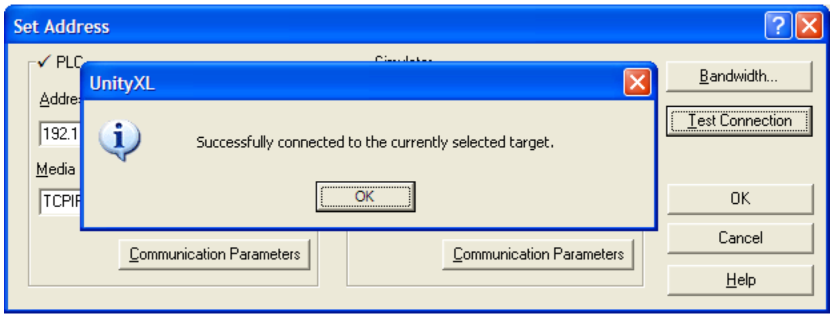
4.4.1 Connecting to the Processor with TCPIP

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.

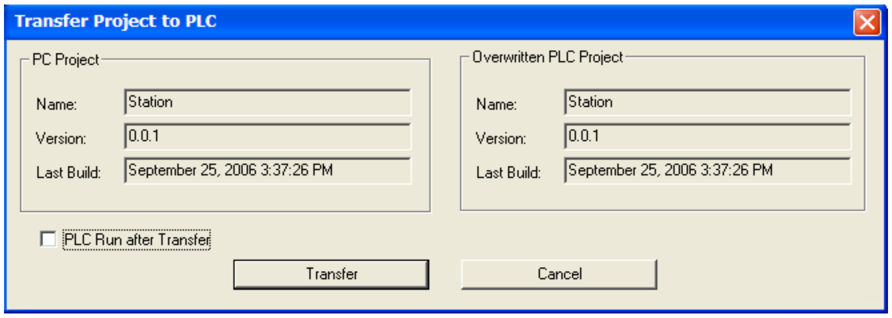


Successfully connected to the currently selected target.

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.


Transfer Project to PLC

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

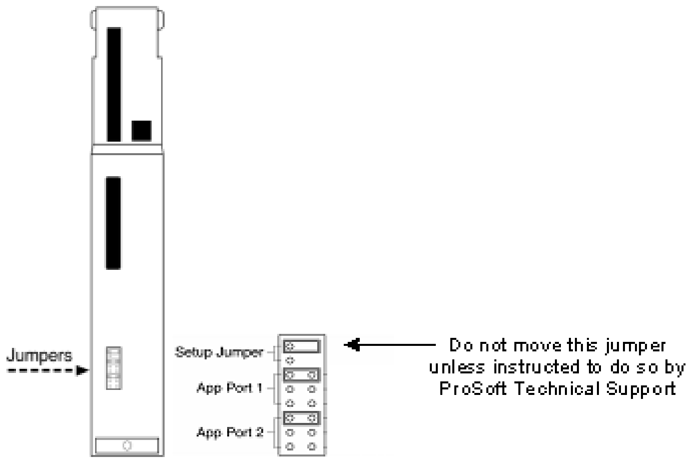
After you complete the following procedures, the ProTalk module will actively be transferring data bi-directionally with the processor.

5.1 Install the ProTalk Module in the Quantum Rack

5.1.1 Verify Jumper Settings

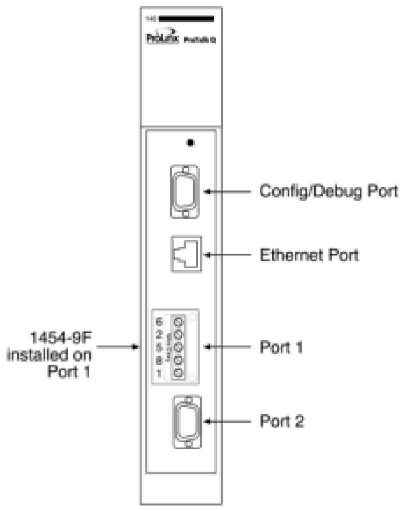
ProTalk modules are configured for RS-232 serial communications by default. To use RS-422 or RS-485, you must change the jumpers.

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

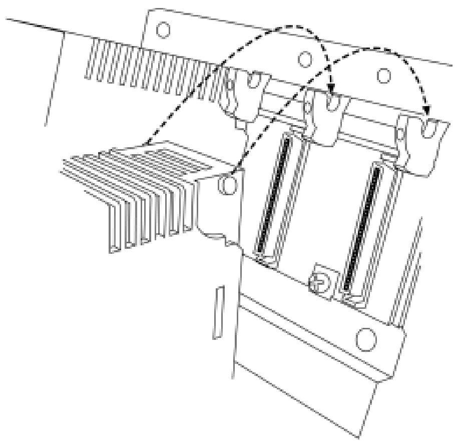
Insert the 1454-9F connector as shown. Wiring locations are shown in the table:



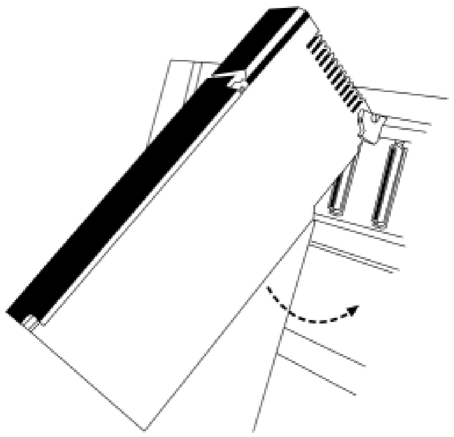
5.1.3 Install the ProTalk Module in the Quantum Rack

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 PTQ-ADMNET module has the following communication connections on the module:

- One Ethernet port (RJ45 connector)
- One RS-232 Configuration/Debug port (DB9 connector)

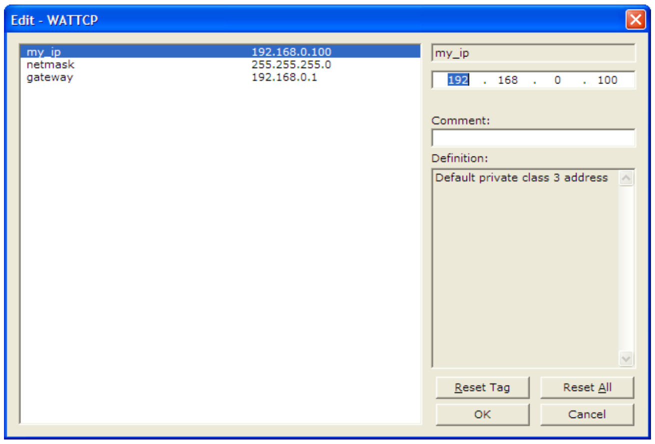
Ethernet Connection

The PTQ-ADMNET module has an RJ45 port located on the front of the module, labeled *Ethernet*, for use with the TCP/IP network. The module is connected to the Ethernet network using an Ethernet cable between the module’s Ethernet port and an Ethernet switch or hub.

Note: Depending on hardware configuration, you may see more than one RJ45 port on the module. The Ethernet port is labeled *Ethernet*.
Warning: The PTQ-ADMNET module is NOT compatible with Power Over Ethernet (IEEE802.3af / IEEE802.3at) networks. Do NOT connect the module to Ethernet devices, hubs, switches or networks that supply AC or DC power over the Ethernet cable. Failure to observe this precaution may result in damage to hardware, or injury to personnel.
Important: The module requires a static (fixed) IP address that is not shared with any other device on the Ethernet network. Obtain a list of suitable IP addresses from your network administrator BEFORE configuring the Ethernet port on this module.

Ethernet Port Configuration - wattcp.cfg

The wattcp.cfg file must be set up properly in order to use a TCP/IP network connection. You can view the current network configuration using an ASCII terminal by selecting **[@]** (Network Menu) and **[V]** (View) options when connected to the Debug port.



```
my_ip        192.168.0.100
netmask      255.255.255.0
gateway      192.168.0.1
```

my_ip

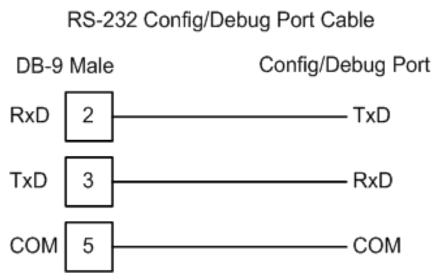
192 . 168 . 0 . 100

Comment:

Definition:
Default private class 3 address

RS-232 Configuration/Debug Port

This port is physically an RJ45 connection. An RJ45 to DB-9 adapter cable is included with the module. 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:



6 Setting Up Your Development Environment

In This Chapter

6.1 Setting Up Your Compiler

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

6.1.1 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.

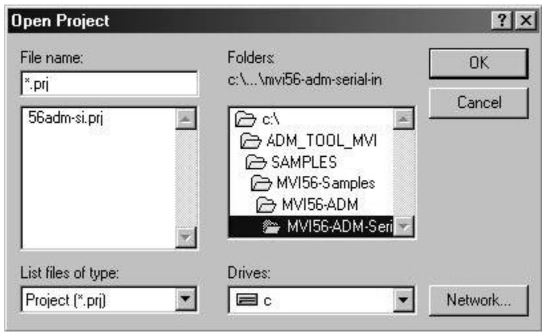
Downloading the Sample Program

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

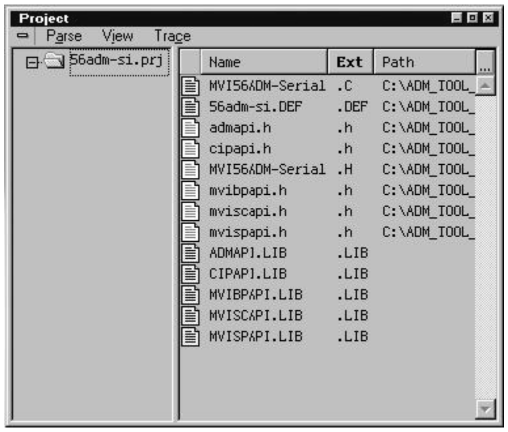
1 Start Digital Mars C++ 8.49, and then click **Project** → **Open** from the *Main Menu*.



2 From the *Folders* field, navigate to the folder that contains the project (C:\ADM_TOOL_PTQ\SAMPLES\…).

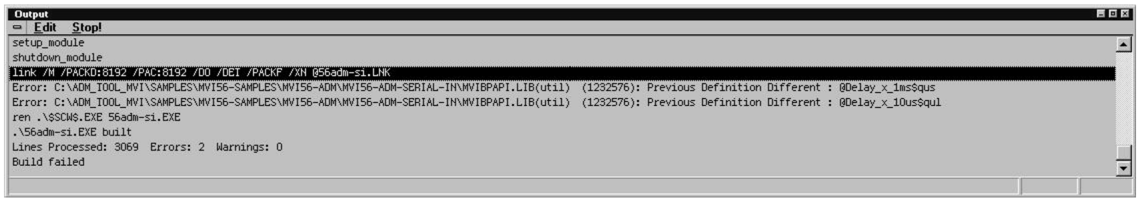
3 In the *File Name* field, click on the project name (56adm-si.prj).

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	

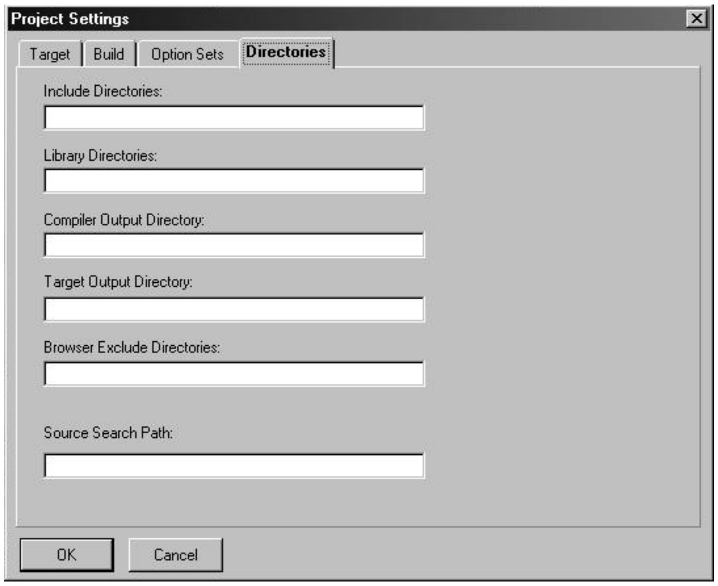
5 Click **Project** → **Rebuild All** from the *Main Menu* to create the .exe file. The status of the build will appear in the Output window:



```
setup_module
shutdown_module
link /M /PACKD:8192 /PAC:8192 /DO /DET /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*.



Include Directories:

Library Directories:

Compiler Output Directory:

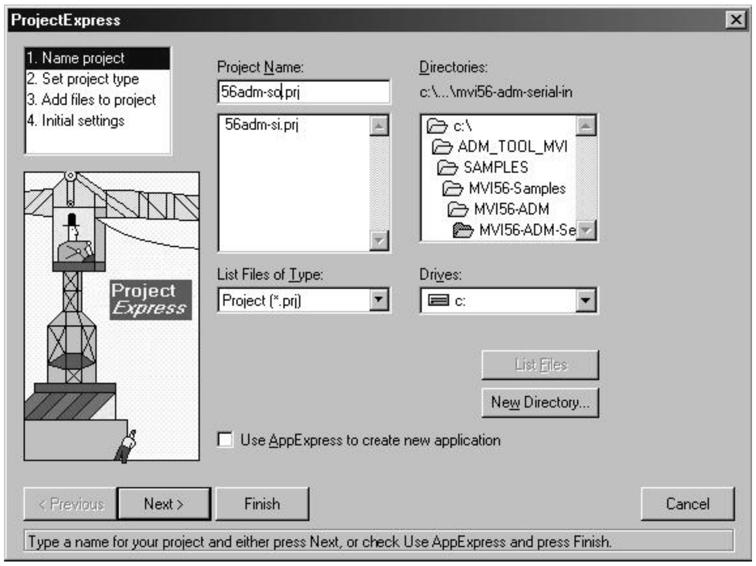
Target Output Directory:

Browser Exclude Directories:

Source Search Path:

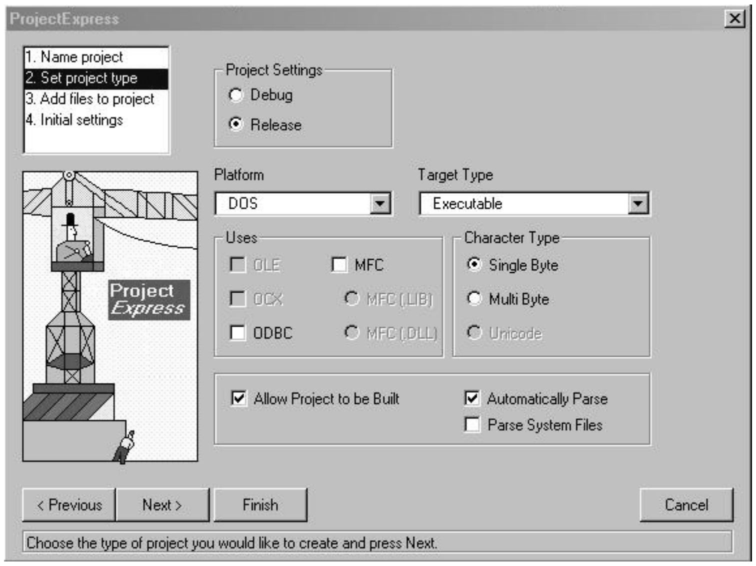
Creating a New Digital Mars C++ 8.49 ADM Project

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



Project Name: 56adm-so.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.

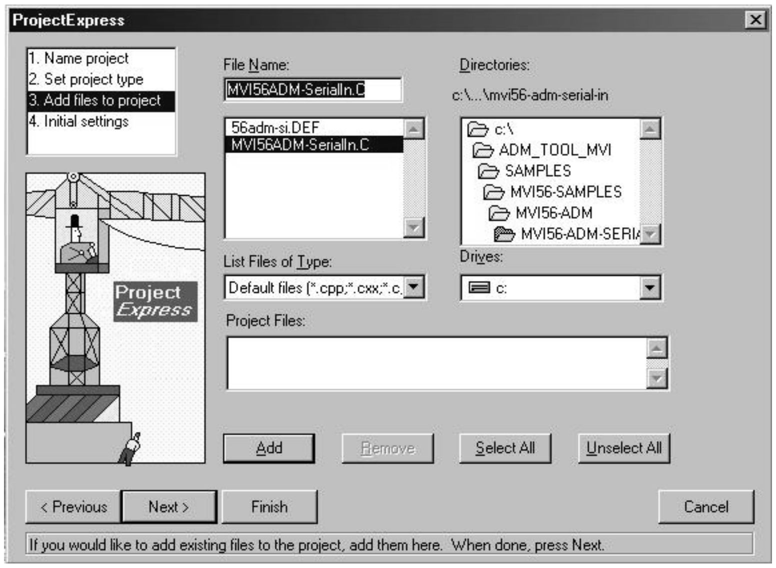
2 Select the path and type in the **Project Name**.
3 Click Next.



Project Settings: ( ) Debug [✓ Release]
Platform: DOS
Target Type: Executable
Uses: ☐ OLE ☐ MFC ☐ OCX ( ) MFC (LIB) ☐ ODBC ( ) 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

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

56adm-si.DEF
MVI56ADM-SerialIn.C

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

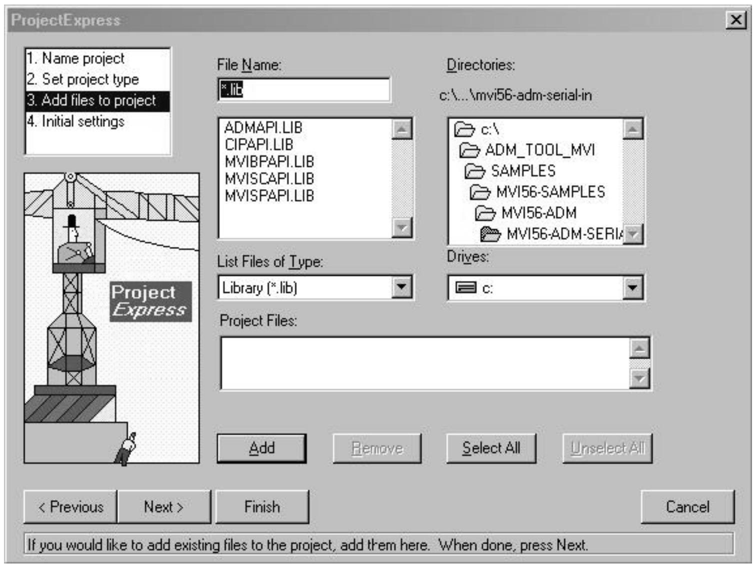
List Files of Type: Default files (*.cpp;*.cxx;*.c,

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

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

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

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

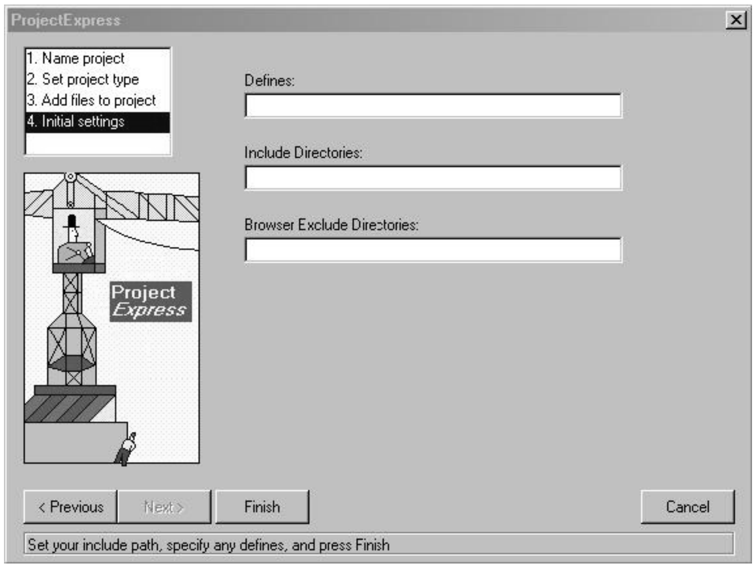
List Files of Type: Library (*.lib)

Drives: c:

Project Files:

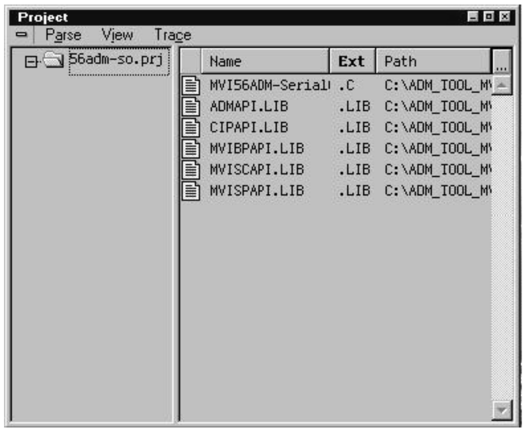
If you would like to add existing files to the project, add them here. When done, press Next.

12 Click Next.



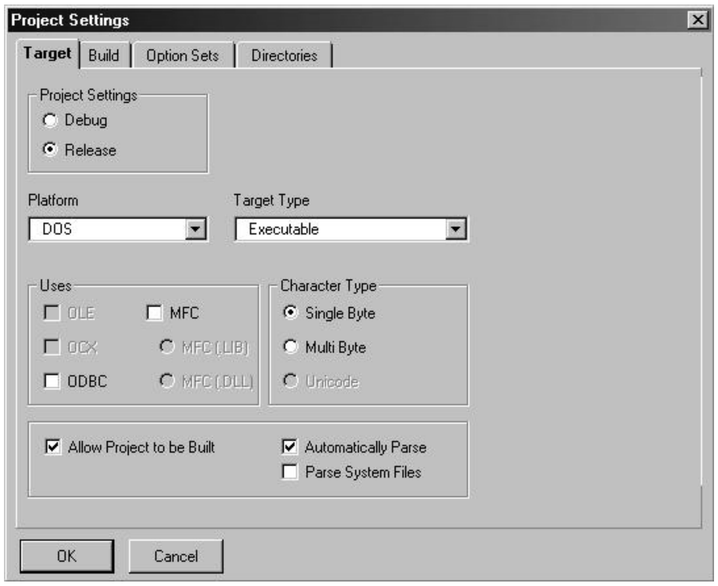
Set your include path, specify any defines, and press Finish

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-Seriall	.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, OCX, ODBC, MFC, MFC (LIB), MFC (DLL)

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

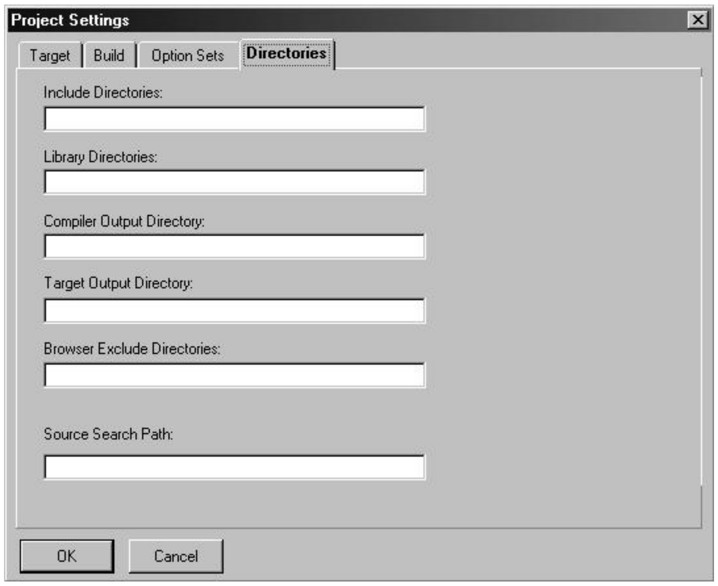
[✓] Allow Project to be Built

[✓] 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:

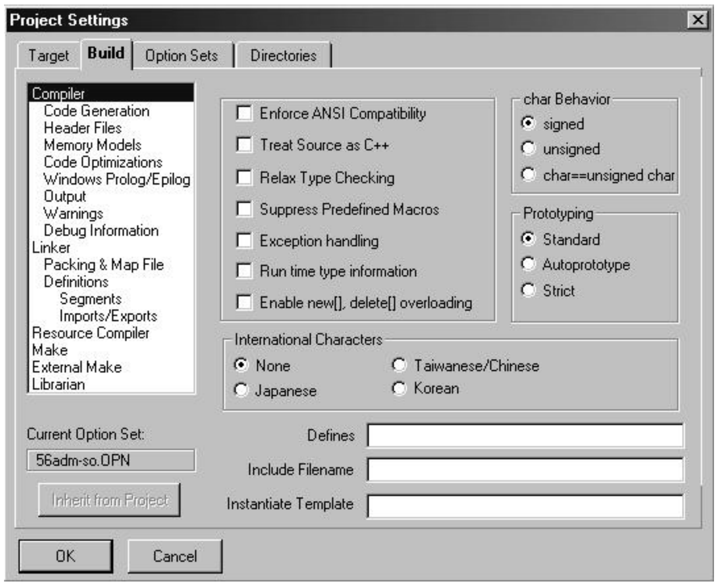
Target Output Directory:

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:



Current Option Set: 56adm-so.OPN

- [ ] 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

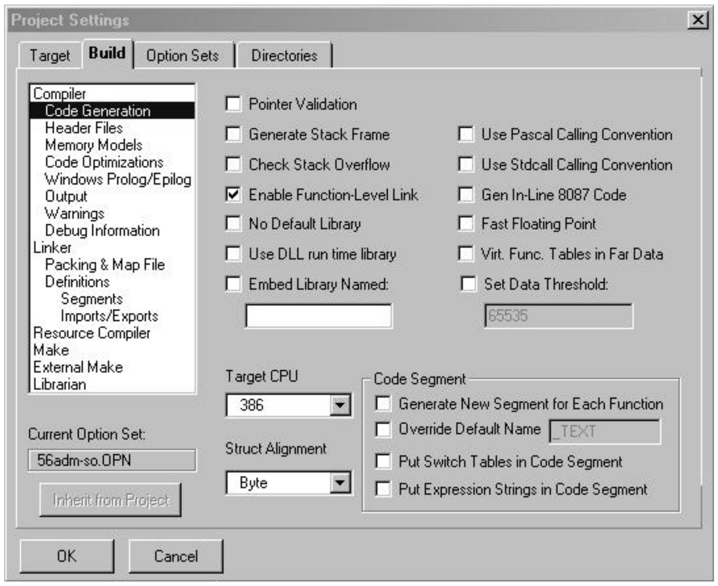
International Characters: [✓ None] Taiwanese/Chinese Japanese Korean

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:



Current Option Set: 56adm-so.OPN

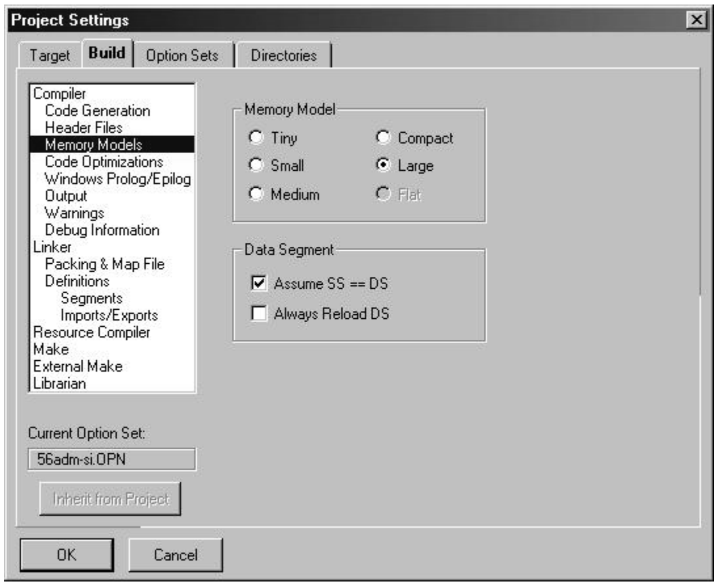
- [ ] Pointer Validation
- [ ] Generate Stack Frame
- [ ] Check Stack Overflow
- [✓] Enable Function-Level Link
- [ ] No Default Library
- [ ] Use DLL run time library
- [ ] Embed Library Named:
- [ ] Use Pascal Calling Convention
- [ ] Use Stdcall Calling Convention
- [ ] Gen In-Line 8087 Code
- [ ] 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

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


Project Settings

Target | **Build** | Option Sets | Directories

Compiler
  Code Generation
  Header Files
  Memory Models
  Code Optimizations
  Windows Prolog/Epilog
  Output
  Warnings
  Debug Information
Linker
  Packing & Map File
  Definitions
    Segments
    Imports/Exports
Resource Compiler
Make
External Make
Librarian

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

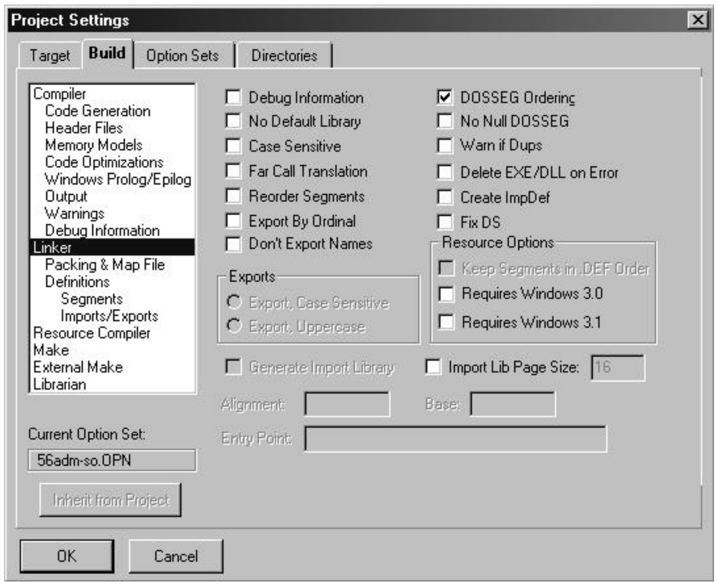
Data Segment
[✓] Assume SS == DS
[ ] Always Reload DS

Current Option Set: 56adm-si.OPN

Inherit from Project

OK | Cancel

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


Project Settings

Target | **Build** | Option Sets | Directories

Compiler
  Code Generation
  Header Files
  Memory Models
  Code Optimizations
  Windows Prolog/Epilog
  Output
  Warnings
  Debug Information
Linker
  Packing & Map File
  Definitions
    Segments
    Imports/Exports
Resource Compiler
Make
External Make
Librarian

[ ] 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

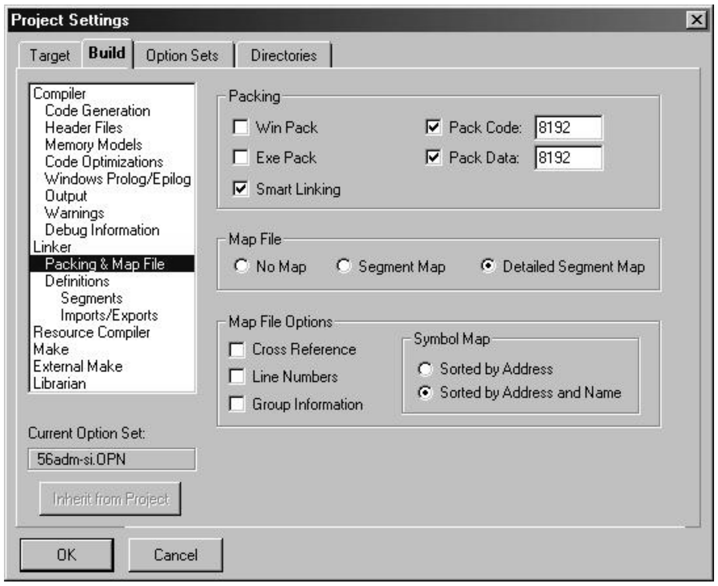
Alignment:
Base:
Entry Point:

Current Option Set: 56adm-so.OPN

Inherit from Project

OK | Cancel

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


Project Settings

Target | **Build** | Option Sets | Directories

Compiler
- Code Generation
- Header Files
- Memory Models
- Code Optimizations
- Windows Prolog/Epilog
- Output
- Warnings
- Debug Information

Linker
- **Packing & Map File**
- Definitions
  - Segments
  - Imports/Exports

Resource Compiler
Make
External Make
Librarian

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

Map File
- ○ No Map
- ○ Segment Map
- ● Detailed Segment Map

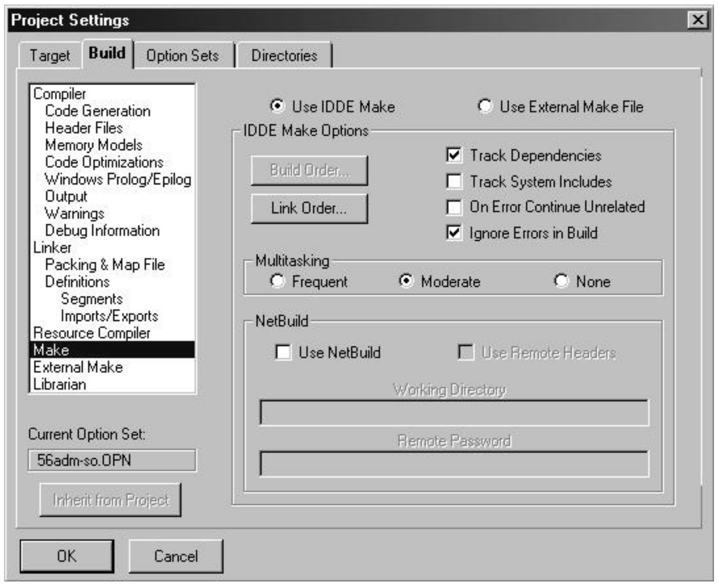
Map File Options
- [ ] Cross Reference
- [ ] Line Numbers
- [ ] Group Information

Symbol Map
- ○ Sorted by Address
- ● Sorted by Address and Name

Current Option Set: 56adm-si.OPN

Inherit from Project | OK | Cancel

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


Project Settings

Target | **Build** | Option Sets | Directories

Compiler
- Code Generation
- Header Files
- Memory Models
- Code Optimizations
- Windows Prolog/Epilog
- Output
- Warnings
- Debug Information

Linker
- Packing & Map File
- Definitions
  - Segments
  - Imports/Exports

Resource Compiler
Make
External Make
Librarian

- ● Use IDDE Make
- ○ Use External Make File

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

Multitasking
- ○ Frequent
- ● Moderate
- ○ None

NetBuild
- [ ] Use NetBuild
- [ ] Use Remote Headers
- Working Directory
- Remote Password

Current Option Set: 56adm-so.OPN

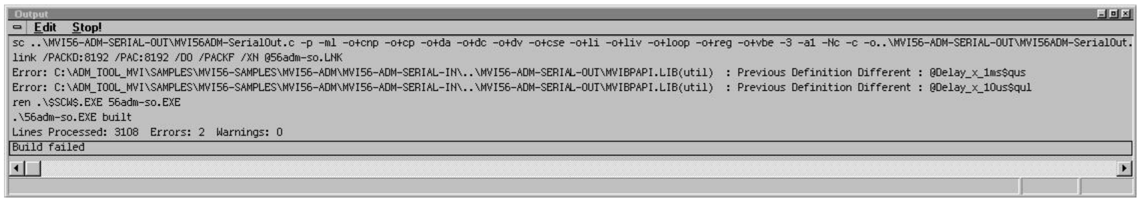
Inherit from Project | OK | Cancel

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.*

6.1.2 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.

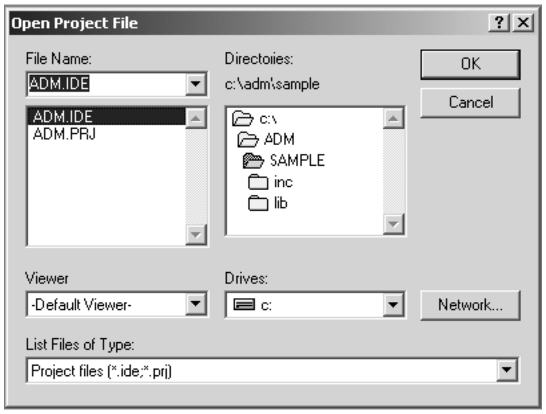
Downloading the Sample Program

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. 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 Borland C++ 5.02 ADM Project

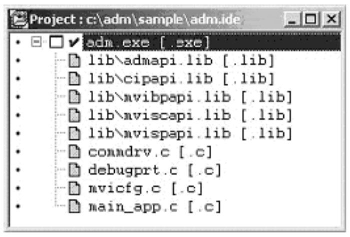
1 Start Borland C++ 5.02, then click **Project → Open Project** from the *Main Menu*.



2 From the *Directories* field, navigate to the directory that contains the project (C:\adm\sample).

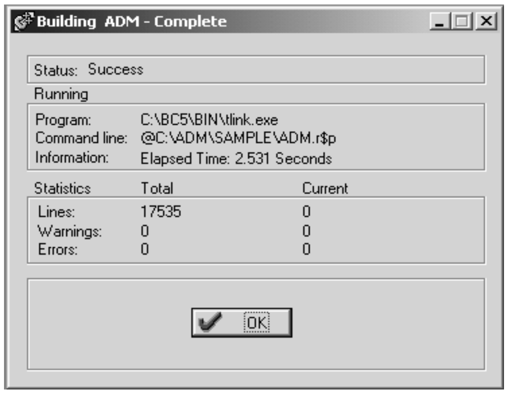
3 In the *File Name* field, click on the project name (adm.ide).

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



```
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:



Status: Success

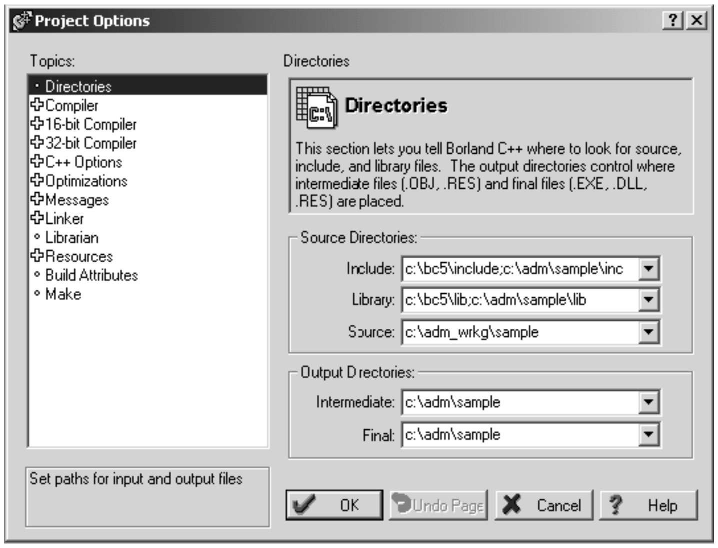
Running

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*.



Project Options

Topics: Directories, Compiler, 16-bit Compiler, 32-bit Compiler, C++ Options, Optimizations, Messages, Linker, Librarian, Resources, Build Attributes, Make

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.

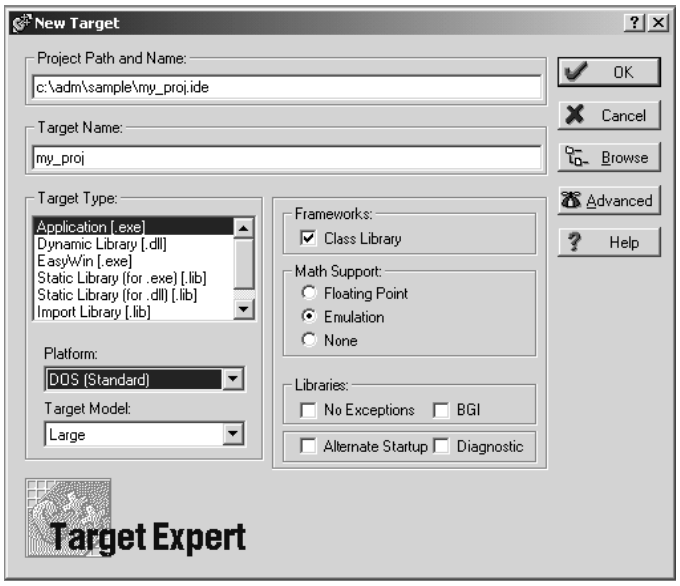
Source Directories:
Include: c:\bc5\include;c:\adm\sample\inc
Library: c:\bc5\lib;c:\adm\sample\lib
Source: c:\adm_wrkg\sample

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, and then click **File** → **Project** from the *Main Menu*.



New Target

Project Path and Name: c:\adm\sample\my_proj.ide

Target Name: my_proj

Target Type: Application [.exe] (selected), Dynamic Library [.dll], EasyWin [.exe], Static Library (for .exe) [.lib], Static Library (for .dll) [.lib], Import Library [.lib]

Platform: DOS (Standard)

Target Model: Large

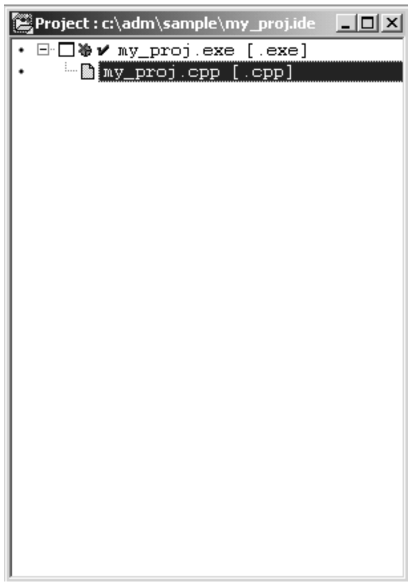
Frameworks: [✓] Class Library

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.

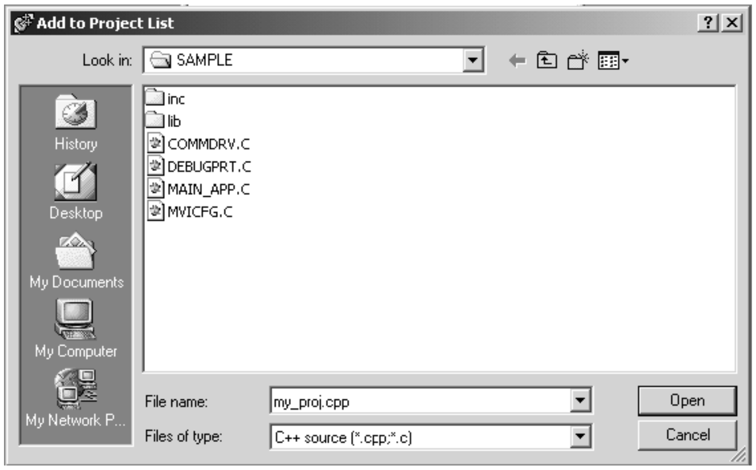
7 Click **OK**. A *Project* window appears:



```
my_proj.exe [.exe]
  my_proj.cpp [.cpp]
```

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:



Look in: SAMPLE

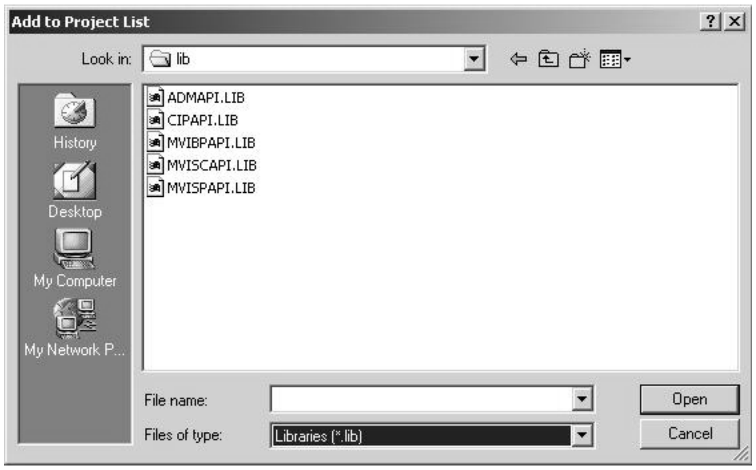
inc
lib
COMMDRV.C
DEBUGPRT.C
MAIN_APP.C
MVICFG.C

File name: my_proj.cpp
Files of type: C++ source (*.cpp;*.c)

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.

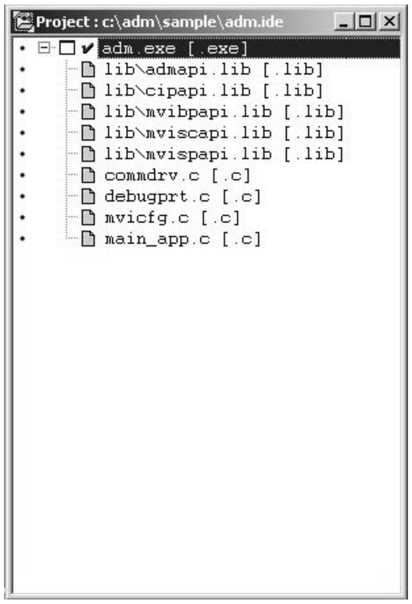
12 Choose Libraries (*.lib) from the *Files of Type* field to view all library files:



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

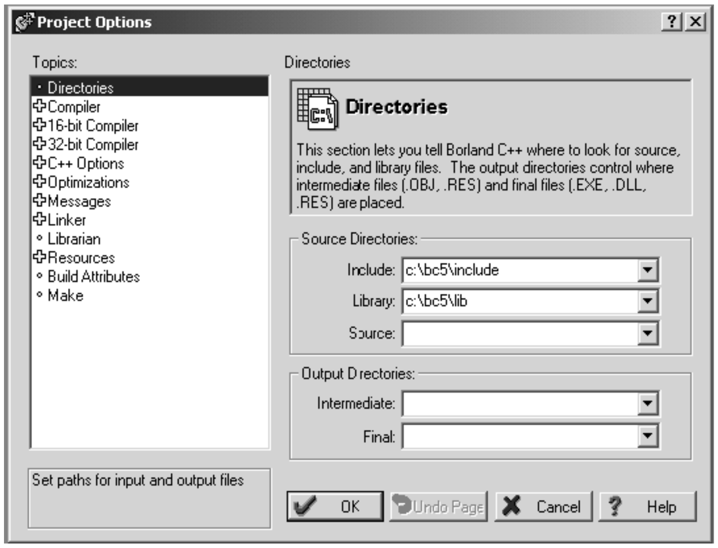
File name:
Files of type: Libraries (*.lib)

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



```
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]
```

14 Click **Options** → **Project** 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:

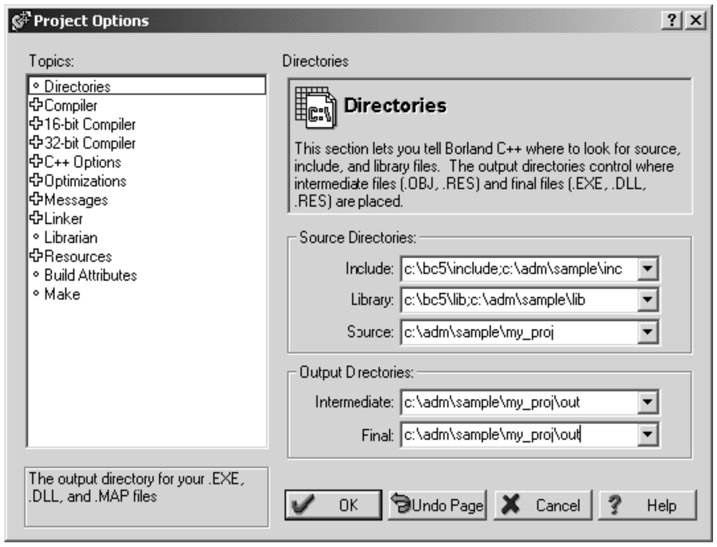
Include: c:\bc5\include
Library: c:\bc5\lib
Source:

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

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:

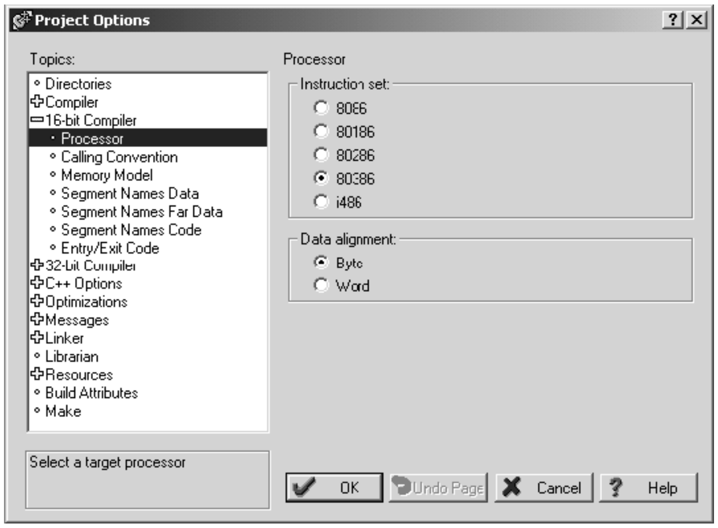
Include: c:\bc5\include;c:\adm\sample\inc
Library: c:\bc5\lib;c:\adm\sample\lib
Source: c:\adm\sample\my_proj

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:



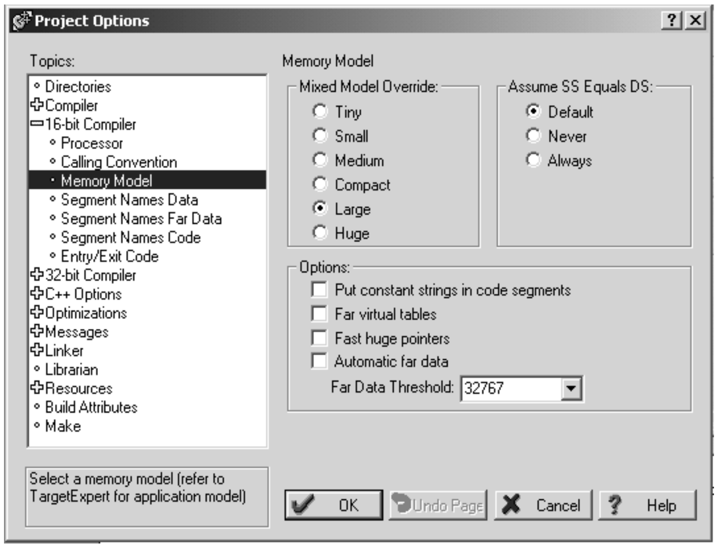
Processor

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

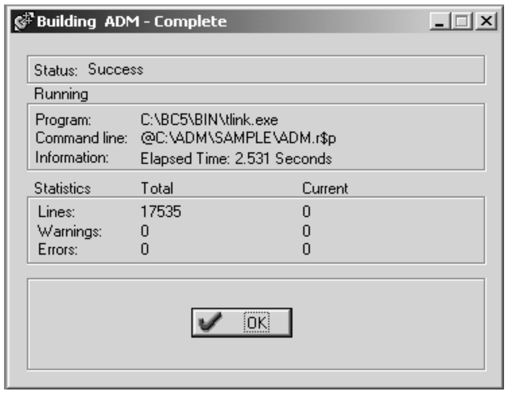
Far Data Threshold: 32767

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:



```
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
```

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.*

6.2 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.

6.2.1 WINIMAGE: Windows Disk Image Builder

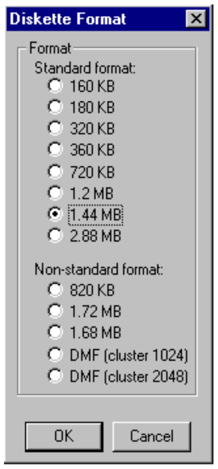
WINIMAGE is a Win9x/NT utility that may be used to create disk images for downloading to the PTQ-ADMNET module. It does not require the use of a floppy diskette. Also, it is not necessary to estimate the disk image size, since WINIMAGE does this automatically and can truncate the unused portion of the 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**.

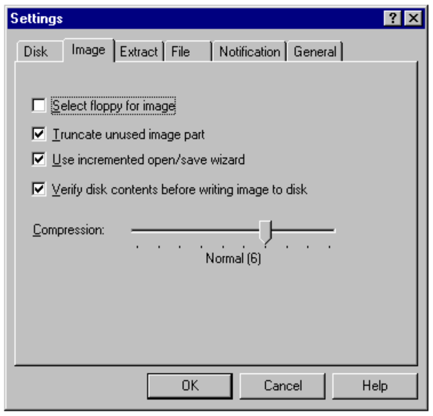

Diskette Format

Format
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**.

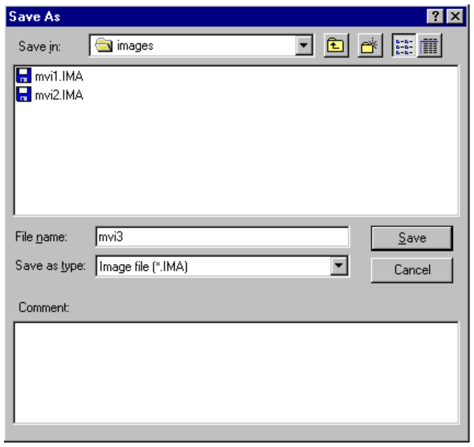

Settings

Disk | Image | Extract | File | Notification | General

☐ 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 As

Save in: images

mvi1.IMA
mvi2.IMA

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

6 Check the disk image file size to be sure it does not exceed the maximum size of the PTQ-ADMNET module's ROM disk (896K bytes, 384K bytes for MVI94). 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**.

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

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

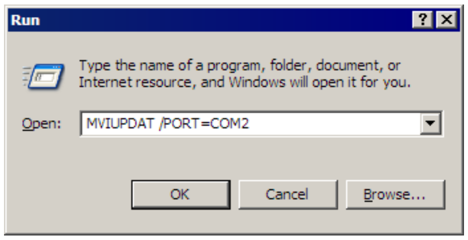
Note: WINIMAGE is a shareware utility. If you find this program useful, please register it with the author.

6.3 Downloading a ROM Disk Image

6.3.1 MVIUPDAT

MVIUPDAT.EXE is a DOS-compatible utility for downloading a ROM disk image from a host PC to the PTQ-ADMNET 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-ADMNET 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              |
+----------------------------+
```

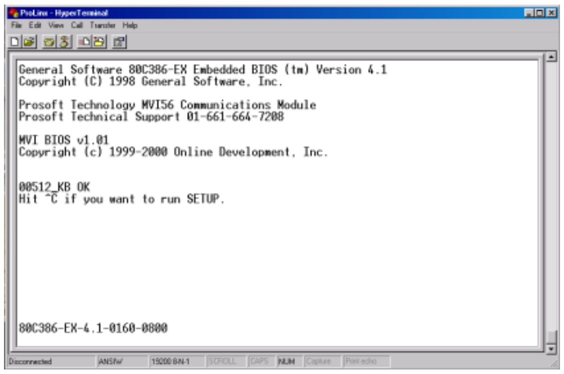
7 Select **VERIFY MODULE CONNECTION** to verify the connection to the PTQ module. If the connection is working properly, the message "Module Responding" will be displayed.

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].**

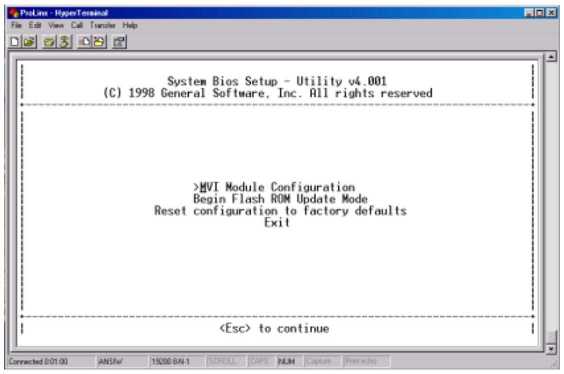
6.4 PTQ System BIOS Setup

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
```

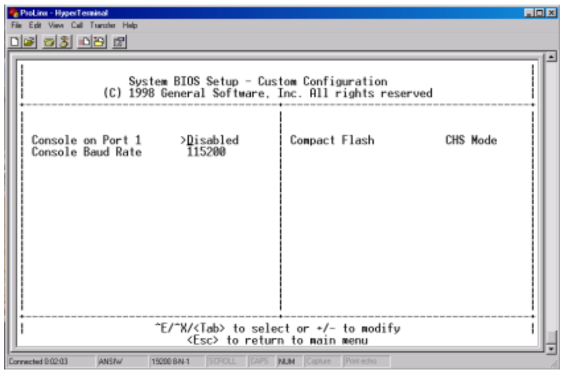
It may be necessary to install the setup jumper in order to access the BIOS Setup. The setup jumper will be necessary if the Console is disabled. The following illustration shows the BIOS Setup screen.



```
                    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.



```
              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
```

6.5 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.

6.5.1 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.

6.5.2 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.

6.5.3 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.

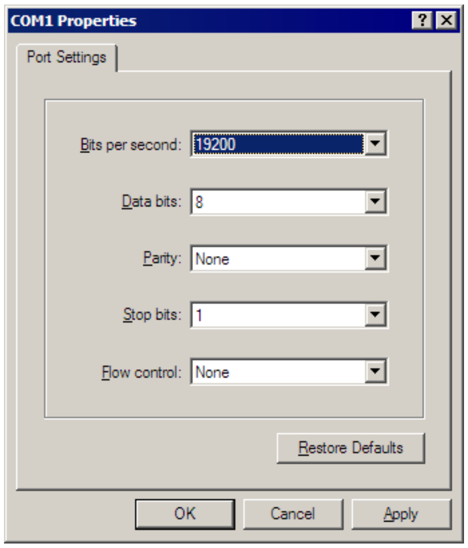
6.5.4 Enabling the Console

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.

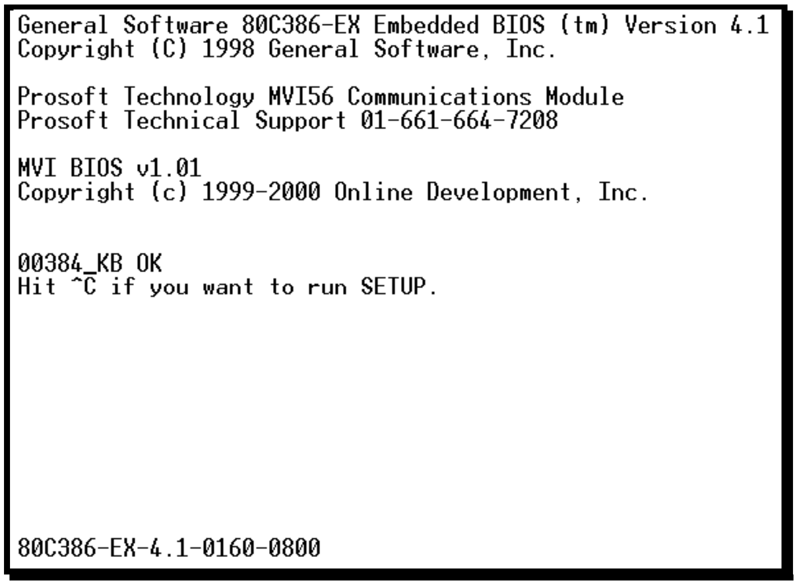
3 Connect to the module using HyperTerminal at 19,200 bps, and then cycle power to reboot the module.


COM1 Properties

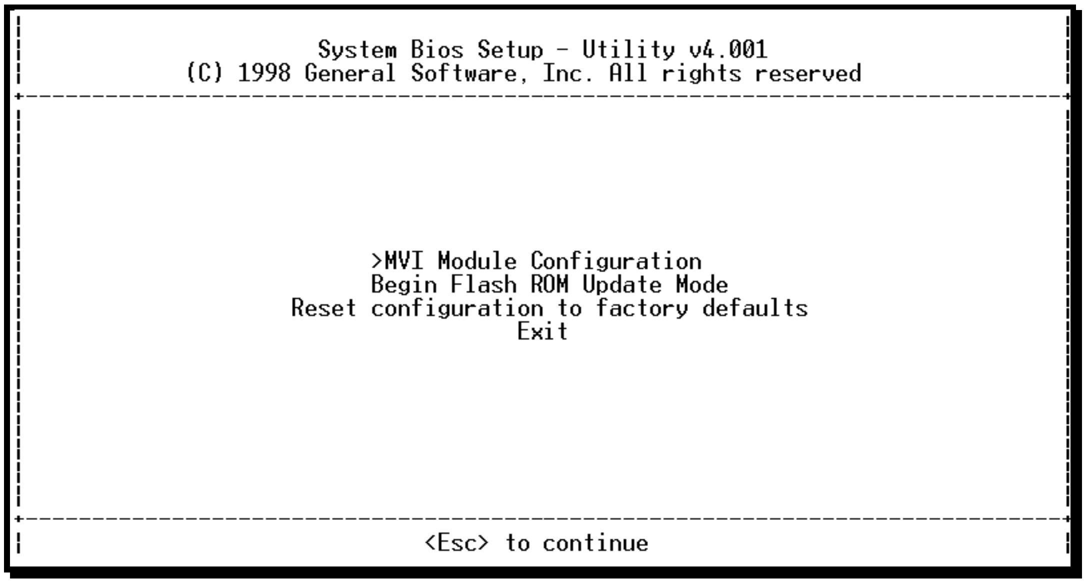
Port Settings

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.

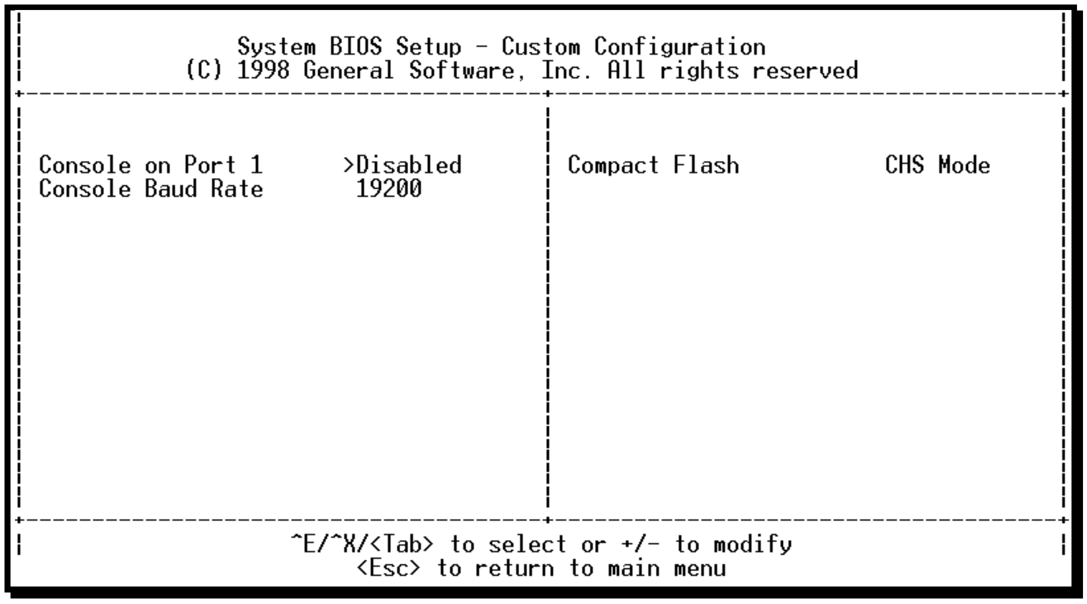


```
General Software 80C386-EX Embedded BIOS (tm) Version 4.1
Copyright (C) 1998 General Software, Inc.

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-ADMNET 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

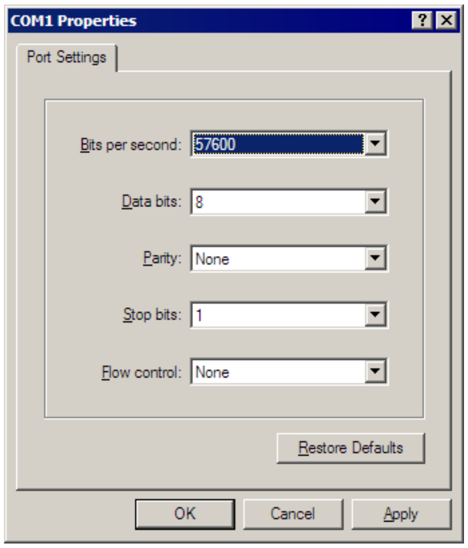
7 Press **[Esc]** to return to the Main Menu.

8 Press **[Esc]** again to apply your changes and reboot the module.

9 Remove the module from the rack and disable the Setup jumper.

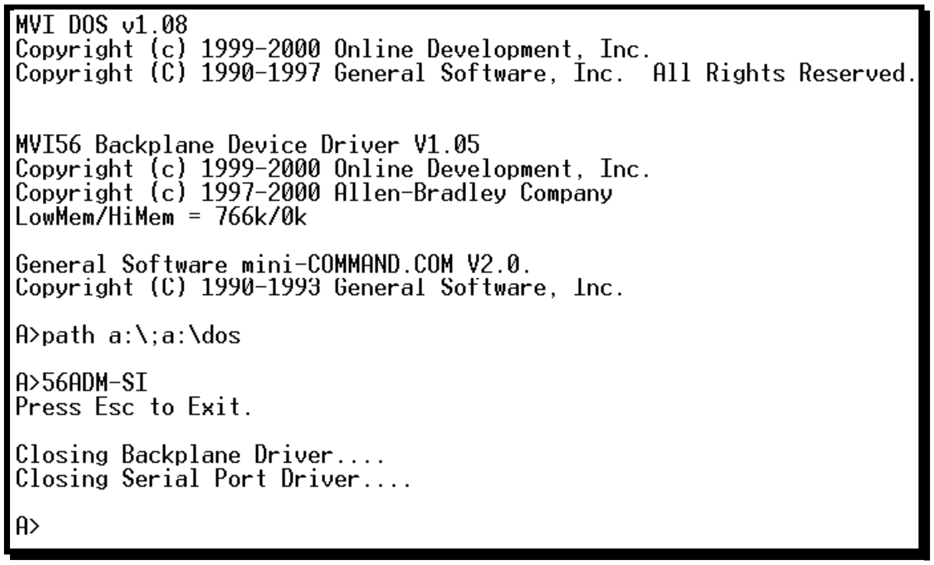
To communicate with the module in Console mode

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.

6.5.5 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 (page 68) and download the image to the module using MVIFlashUpdate. (page 70) The following procedures show how to send and receive files from the module's Compact Flash card (drive C:\).

6.5.6 Downloading Files From a PC to the ADM Module

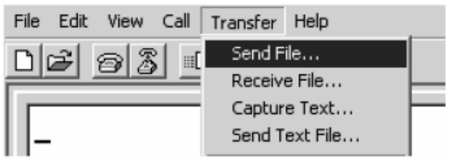
In order to download files to the module, the ADM module's running program must be interrupted. To transfer files to the module, run the RY.EXE program which uses the YModem protocol.

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

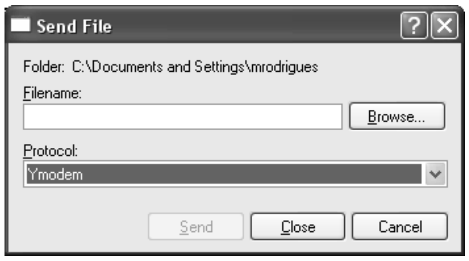
```
C:\RY
```

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



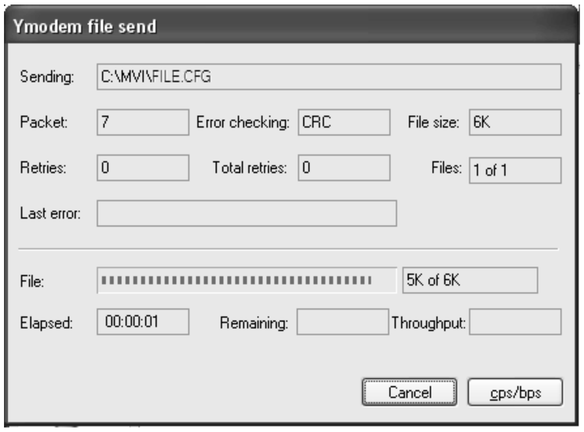
4 Click the Browse button to navigate to the folder and file to send to the module.

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



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

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



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.

6.5.7 Uploading files from the ADM module to a PC

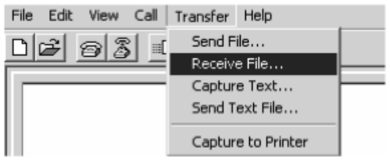
In order to upload files from the module, the ADM module's running program must be interrupted. You must run the SY.EXE program which uses the YModem protocol.

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"
```

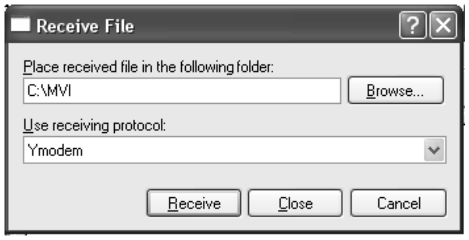
The filename and path must be in quotes.



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.



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.

7 Understanding the PTQ-ADMNET API

In This Chapter

The PTQ ADM API Suite allows software developers access to the top layer of the serial and Ethernet ports. The PTQ-ADMNET API suite accesses the Ethernet port. Both APIs can be easily used without having detailed knowledge of the module's hardware design. The PTQ ADMNET API Suite consists of the Ethernet Port API. The Ethernet Port API provides access to the Ethernet network.

Applications for the PTQ ADMNET 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 ADMNET module.

7.1 Introduction

This document provides information needed to develop application programs for the PTQ ADMNET Ethernet Serial Communication Module. The PTQ suite of modules is designed to allow devices with a serial and Ethernet port to be accessed by a PLC.

The modules are programmable to accommodate devices with unique Serial-Ethernet protocols.

This document includes information about the available ethernet communication software API libraries, programming information, and example code. For tools, module configuration, serial communication software API, serial communication programming information, and example code for both the module and the PLC, refer to *PTQ ADM Developer’s Guide*.

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 Schneider Electronics programmable controllers and the PLC platform.

7.2 Operating System

The PTQ module includes General Software Embedded DOS 6-XL. This operating system provides DOS compatibility along with real-time multitasking 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 or Borland compilers. User programs may be executed automatically by loading them from either the CONFIG.SYS file or an AUTOEXEC.BAT file. In addition to PTQ-ADMNET, ADMTCP.CFG is required to assign an IP address to the module. Users can store the ADMTCP.CFG file directly to a Compact Flash.

The format of the ADMTCP.CFG is as follows:

```
# ProSoft Technology
# Default private class 3 address
my_ip=192.168.0.148
# Default class 3 network mask
netmask=255.255.255.0
# name server 1 up to 9 may be included
# nameserver=xxx.xxx.xxx.xxx
# name server 2
# nameserver=xxx.xxx.xxx.xxx
# The gateway I wish to use
gateway=192.168.0.1
# some networks (class 2) require all three parameters
# gateway,network,subnetmask
# gateway 192.168.0.1,192.168.0.0,255.255.255.0
# The name of my network
# domainslist="mynetwork.name"
```

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.

7.3 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 C++ 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.

7.3.1 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.

7.3.2 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.

7.3.3 Sample Code

A sample application is provided to illustrate the usage of the API functions. Full source for the sample application is also provided. The sample application may be compiled using Digital Mars or Borland C++.

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.

7.3.4 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.

7.4 Development Tools

An application developed for the PTQ ADM module must be stored on the module’s Flash ROM disk in order to be executed.

7.5 Theory of Operation

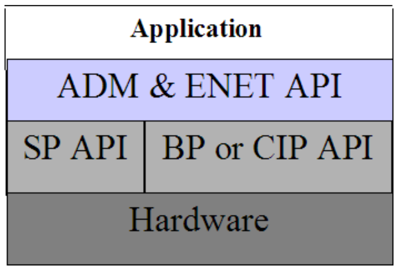
7.5.1 ADMNET API

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

7.5.2 ADMNET API Architecture

The ADMNET API is composed of a statically-linked library (called the ADMNET library). Applications using the ADMNET API must be linked with the ADMNET library.

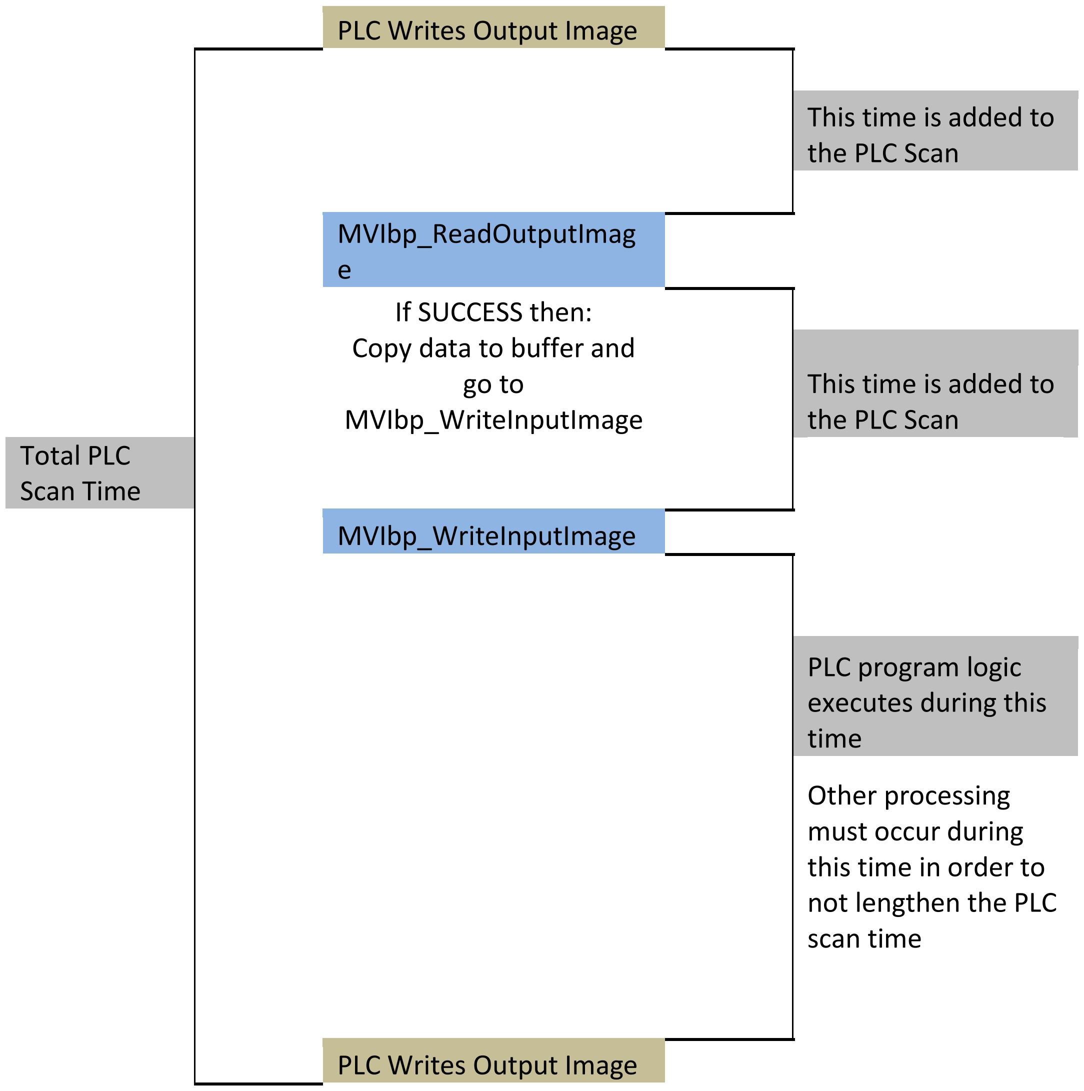
The following illustration shows the relationship between the API components.



7.5.3 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

7.6 ADMNET API Files

The following table lists the supplied API file names. These files should be copied to a convenient directory on the computer where the application is to be developed. These files need not be present on the module when executing the application.

File Name	Description
ADMNETAPI.H	Include file
ADMNETAPI.LIB	Library (16-bit OMF format)

7.6.1 ADMNET Interface Structure

The ADMNET interface structure functions mainly as a protocol UDP and TCP socket. Pointers to structures are used so that the API can access lower-level Ethernet communication. The ADMNET API requires the interface structure and the structures referenced by it. Refer to the example code section for examples of the functions.

The interface structure is as follows:

```
typedef struct   _tcp_socket {
struct        _tcp_socket *next;
word          ip_type;                   // always set to TCP_PROTO
char          *err_msg;
char          *usr_name;
void          (*usr_yield)(void);
byte           rigid;
byte           stress;
word           sock_mode;                // a logical OR of bits

longword       usertimer;                // ip_timer_set, ip_timer_timeout
dataHandler_t  dataHandler;              // called with incoming data
eth_address    hisethaddr;               // ethernet address of peer
longword       hisaddr;                  // internet address of peer
word           hisport;                  // tcp ports for this connection
longword       myaddr;
word           myport;
word           locflags;

int            queuelen;
byte           *queue;

int            rdatalen;                 // must be signed
word           maxrdatalen;
byte           *rdata;
byte           rddata[tcp_MaxBufSize+1]; // received data
longword       safetysig;
word           state;                    // connection state

longword       acknum;
longword       seqnum;                   // data ack'd and sequence num
long           timeout;                  // timeout, in milliseconds
byte           unhappy;                  // flag, indicates retransmitting
segt's
byte           recent;                   // 1 if recently transmitted
```

```
word           flags;                     // tcp flags word for last packet
sent

word           window;                    // other guy's window
int            datalen;                   // number of bytes of data to send
                                          // must be signed
int            unacked;                   // unacked data

byte           cwindow;                   // Van Jacobson's algorithm
byte           wwindow;

word           vj_sa;                     // VJ's alg, standard average
word           vj_sd;                     // VJ's alg, standard deviation
longword       vj_last;                   // last transmit time
word           rto;
byte           karn_count;                // count of packets
byte           tos;                       // priority
                                             // retransmission timeout
 procedure
                                          // these are in clock ticks
longword       rtt_lasttran;              // last transmission time
longword       rtt_smooth;                // smoothed round trip time
longword       rtt_delay;                 // delay for next transmission
longword       rtt_time;                  // time of next transmission

word           mss;
longword       inactive_to;               // for the inactive flag
int            sock_delay;

byte           data[tcp_MaxBufSize+1];    // data to send
 } tcp_Socket;

 typedef struct _udp_socket {
struct         _udp_socket *next;
word           ip_type;                   // always set to UDP_PROTO
char           *err_msg;                  // null when all is ok
char           *usr_name;
void          (*usr_yield)( void );
byte           rigid;
byte           stress;
word           sock_mode;                 // a logical OR of bits
longword       usertimer;                 // ip_timer_set, ip_timer_timeout
dataHandler_t  dataHandler;
eth_address    hisethaddr;                // peer's ethernet address
longword       hisaddr;                   // peer's internet address
word           hisport;                   // peer's UDP port
longword       myaddr;
word           myport;
word           locflags;

int            queuelen;
byte           *queue;

int            rdatalen;                  // must be signed
word           maxrdatalen;
byte           *rdata;
byte           rddata[ tcp_MaxBufSize + 1];  // if dataHandler = 0, len ==
512
longword       safetysig;
} udp_Socket;
```

8 Application Development Function Library - ADMNET API

In This Chapter

8.1 ADMNET API Functions

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

The same set of API functions is supported for all of the modules in the PTQ family.

API library routines are categorized according to functionality.

Function Category	Function Name	Description
Initialize Socket	ADM_init_socket	Initialize number of sockets used on each port number and assign name to each port.
	ADM_open_sk	Open and reopen each socket separately after socket is initialized or closed.
Release Socket	ADM_release_sockets	Release all sockets that have been initialized using ADM_init_socket.
	ADM_close_sk	Close each socket separately without release socket.
Send Socket	ADM_send_socket	Send socket according to name assign throughout initialization process as either UDP or TCP. This function also takes care of opening socket connection.
	ADM_send_sk	Send socket with previously open with function ADM_open_sk.
Receive Socket	ADM_receive_socket	Receive socket according to name assigned throughout initialization process as either UDP or TCP. This function also takes care of opening socket connection.
	ADM_receive_sk	Receive socket with previously open with function ADM_open_sk.
Miscellaneous	ADM_NET_GetVersionInfo	Get ADMNET API version information.
	ADM_is_sk_open	Test if the socket is still open.

8.2 ADMNET API Initialize Functions

The following topics describe the ADMNET API Initialize functions.

ADM_init_socket

Syntax

```
int ADM_init_socket(int numSK, int portNum, int buffSize, char *name);
```

Parameters

numSK	Variable indicating how many sockets to use.
portNum	Port Number.
buffSize	The size of the buffer available in each socket.
name	The name of the socket.

Description

ADM_init_socket acquires access to the ADMNET API and dynamically generates a set of sockets according to numSK and assigns portNum, buffSize, then names each socket that the application will use 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_Release_Sockets should always be called before exiting the application.

Return Value

SK_SUCCESS	API has successfully initialized variables.
SK_PORT_NOT_ALLOW	API does not allow port number used.
SK_CANNOT_ALLOCATE_MEMORY	API cannot allocate memory.

Example

```
int numSK = 5;
int portNum = 5757;
int buffSize = 1000;

if(ADM_init_socket(numSK, portNum, buffSize, "ReceiveSK") != SK_SUCCESS)
{
printf("\nFailed to open ADM API... exiting program\n");
ADM_release_sockets();
}
```

See Also

ADM_release_sockets (page 99)

ADM_open_sk

Syntax

```
int ADM_open_sk(char *skName, char *ServerIPAddress, int protocol);
```

Parameters

skName	Name of the socket that has been initialized and used to send data.
ServerIPAddress	IP address that will be used to send data to.
protocol	Specified protocol to send over Ethernet (USE_TCP or USE_UDP).

Description

ADM_open_sk opens a socket according to the name previously initialized, skName, with ADM_init_socket given, and assigns IP address, ServerIPAddress for send function with specific protocol, either UDP or TCP. ADM_init_socket must be used before this function.

IMPORTANT: After the API has been opened, ADM_close_sk should always be called for closing the socket. 0.0.0.0 passes as ServerIPAddress to open socket as a server to listen to a message from client.

Return Value

SK_SUCCESS	API has successfully opened socket.
SK_PROCESS_SOCKET	Open is still in process.
SK_NOT_FOUND	API could not find an initialized socket with the name passed to the function.
SK_TIMEOUT	Time out opening socket.
SK_OPEN_FAIL	Socket could not be opened.

Example

```
char sockName1[ ] = "SendSocket";
int buffSize1 = 4096;
int port_1 = 6565;
int numSocket1 = 1;
int result;

sock_init();    //initialize the socket interface
ADM_init_socket(numSocket1, port_1, buffSize1, sockName1);

while ((result = ADM_open_sk(sockName1, "0.0.0.0",
USE_TCP))==SK_PROCESS_SOCKET);

if (result==SK_SUCCESS)
{
   printf("successfully Opened a connection!\n");
} else {
   printf("Error Opening a connection!  %d\n", result);
}
```

See Also

ADM_close_sk (page 100)

8.3 ADMNET API Release Socket Functions

This section describes the ADMNET API Release Socket Functions.

ADM_release_sockets

Syntax

```
int ADM_release_sockets(void);
```

Parameters

none

Description

This function is used by an application to release all sockets created by ADM_init_socket.

IMPORTANT: After a socket has been generated, this function should always be called before exiting the application.

Return Value

SK_SUCCESS	API was successfully released all the sockets.

Example

```
ADM_release_sockets();
```

See Also

ADM_init_socket (page 97)

ADM_close_sk

Syntax

```
int ADM_close_sk(char *skName);
```

Parameters

skName	Name of the socket that has been initialized and used to send data.

Description

This function is used by an application to close socket opened by ADM_open_sk.

IMPORTANT: After a socket has been opened, this function should always be called to close socket, but not release socket.

Return Value

SK_SUCCESS	API was successfully released all the sockets.
SK_NOT_FOUND	API could not find an initialized socket with the name passed to the function.

Example

```
char sockName1[ ] = "SendSocket";

ADM_close_sk(sockName1);
printf ("Connection Closed!\n");
```

See Also

ADM_init_socket (page 97)

8.4 ADMNET API Send Socket Functions

This section describes the ADMNET API Send Socket functions.

ADM_send_socket

Syntax

```
int ADM_send_socket(char *skName, char *holdSendPtr, int *sendLen, char
*ServerIPAddress, int protocol);
```

Parameters

skName	Name of the socket that has been initialized and used to send data.
holdSendPtr	Pointer to a string of data that will be sent to the ServerIPAddress
sendLen	Number of data specified to send.
ServerIPAddress	IP address that will be used to send data to.
protocol	Specified protocol to send over Ethernet (USE_TCP or USE_UDP).

Description

To simplify a program, this function opens connection and sends message. *skName* must be a valid name that has been initialized with ADM_init_socket.

Return Value

SK_SUCCESS	Socket is successfully sent.
SK_NOT_FOUND	Socket could not be found.
SK_PROCESS_SOCKET	Socket is in the process of sending.

Example

```
int sendLen = 10;
int se;

se = ADM_send_socket("sendSK", "1234567890", &sendLen, "192.168.0.148",
USE_UDP);
if(se == SK_SUCCESS)
{
printf("send Success\n");
}
```

See Also

ADM_receive_socket (page 103)

ADM_send_sk

Syntax

```
int ADM_send_sk(char *skName, char *holdSendPtr, int *sendLen);
```

Parameters

skName	Name of the socket that has been initialized and used to send data.
holdSendPtr	Pointer to a string of data that will be sent to the ServerIPAddress
sendLen	Number of data specified to send.

Description

ADM_ send _sk sends with a socket previously open using ADM_open_sk.

Return Value

SK_SUCCESS	API has successfully open socket.
SK_PROCESS_SOCKET	Open process is still in
SK_NOT_FOUND	API could not find an initialized socket with the name passed to the function.

Example

```
char sockName1[ ] = "SendSocket";
char holdingReg[100];
int buffSize1 = 4096;
int port_1 = 6565;
int numSocket1 = 1;
int result;

sock_init();    //initialize the socket interface
ADM_init_socket(numSocket1, port_1, buffSize1, sockName1);

sprintf(holdingReg,"abcdefghijklmnopqrstuvwxyz-");
sendLen = 27;

while ((result = ADM_send_sk(sockName1, holdingReg, &sendLen)) ==
SK_PROCESS_SOCKET);

if(result == SK_SUCCESS)
{
printf("Data: %s Sent \n", holdingReg);
} else {
printf("Error sending data\n");
}
```

See Also

ADM_receive_sk (page 104)

8.5 ADMNET API Receive Socket Functions

This section describes the ADMNET API Receive Socket functions.

ADM_receive_socket

Syntax

```
int ADM_receive_socket(char *skName, char *holdRecPtr, int *readLen, int
protocol);
```

Parameters

skName	Name of the socket that has been initialized and used to receive data.
holdRecPtr	Pointer to a buffer to hold data that will be received by the API.
readLen	Length of data received by the API.
protocol	Specified protocol to receive over Ethernet (USE_TCP or USE_UDP).

Description

To simplify a program, this function opens connection and receives message.

Return Value

SK_SUCCESS	Socket is successfully sent.
SK_NOT_FOUND	Socket could not be found.
SK_PROCESS_SOCKET	Socket is in the process of sending.

Example

```
char hold[5000];
int readLen;
int se, i;

se = ADM _receive_socket("receiveSK", holdingReg, &readLen, USE_UDP);
if(se == SK_SUCCESS)
{
printf("Length == %d\n", readLen);
for (i=0; i<readLen; i++)
{
  printf("%02X ", *(holdingReg+i));
  if(i%10 == 0) printf("\n");
}
printf("\n");
}
```

See Also

ADM_send_socket (page 101)

ADM_receive_sk

Syntax

```
int ADM_receive_sk(char *skName, char *holdRecPtr, int *readLen, char
*fromIP);
```

Parameters

skName	Name of the socket that has been initialized and used to receive data.
holdRecPtr	Pointer to a buffer to hold data that will be received by the API.
readLen	Length of data received by the API.
fromIP	Pointer to character array which in turn return with client IP.

Description

This function receives socket after ADM_open_sk is used. skName must be a valid name that has been initialized with ADM_init_socket.

Return Value

SK_SUCCESS	Socket is successfully sent.
SK_NOT_FOUND	Socket could not be found.
SK_PROCESS_SOCKET	Socket is in the process of sending.
SK_TIMEOUT	Time out opening socket.

Example

```
char sockName1[ ] = "SendSocket";
char holdingReg[100];
int result;

while ((result=ADM_receive_sk(sockName1, holdingReg, &readLen, fromIP)) == S
K_PROCESS_SOCKET);

if(result == SK_SUCCESS){
printf("Received data!\n");
printf("Length == %d\n", readLen);
for (i=0; i<readLen; i++)
{
  printf("%c", *(holdingReg+i));
}
   printf("\n");

} else {
   printf("Received no data Error: %d\n",result);
}
```

See Also

ADM_send_socket (page 101)

8.6 ADMNET API Miscellaneous Functions

ADM_NET_GetVersionInfo

Syntax

```
void ADM_NET_GetVersionInfo(ADMNETVERSIONINFO* admnet_verinfo);
```

Parameters

admnet_verinfo	Pointer to structure of type ADMNETVERSIONINFO.

Description

ADM_GetVersionInfo retrieves the current version of the ADMNET API library. The information is returned in the structure admnet_verinfo.

The ADMVERSIONINFO structure is defined as follows:

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

Return Value

None

Example

```
ADMNETVERSIONINFO verinfo;
/* print version of API library */

ADM_NET_GetVersionInfo(& verinfo);

printf("Revision %d.%d\n", verinfo.APIRevisionMajor,
verinfo.APIRevisionMinor);
```

ADM_is_sk_open

Syntax

```
int ADM_is_sk_open(char *skName);
```

Parameters

skName	Name of the socket that has been initialized and used to receive data.

Description

ADM_is_sk_open tests if connection is still valid or not.

Return Value

SK_SUCCESS	Socket is successfully sent.
SK_NOT_FOUND	Socket could not be found.
SK_SOCKET_CLOSE	Socket is closed.

Example

```
char sockName1[ ] = "SendSocket";

if(ADM_is_sk_open(sockName1) != SK_SUCCESS) {
  printf("Socket not Opened\n");
} else {
  printf("Socket Opened\n");
}
```

9 WATTCP API Functions

In This Chapter

9.1 WATTCP API Functions

This API is a TCP/IP stack, which is used on ADMNET API. Parts of this document are brought from Waterloo TCP by Erik Engelke. Each section provides detailed programming information for each WATTCP API library function. The calling convention for each API function is shown in 'C' format.

The API library routines are categorized according to functionality as shown in the following table.

Function Category	Function Name	Description
Initialize Socket	sock_init	TCP/IP system initialization.
System Functionality	tcp_tick	Determine socket connection.
	tcp_open & tcp_open_fast	Generate socket session to a host computer for TCP protocol. tcp_open_fast will have no wait for if the host computer is not found.
	udp_open & udp_open_fast	Generate socket session to a host computer for UDP protocol. udp_open_fast will have no wait for if the host computer is not found.
	resolve	Convert string IP Address into a longword.
	sock_mode	Setup socket protocol transfer mode for the particular use (UDP or TCP).
	sock_established	Check if connect has been established.
	ip_timer_init	Initialize timing.
	ip_timer_expired	Check if timer has been expired.
	set_timeout	Set timer.
	chk_timeout	Check timer if expired.
	sockerr	Return ASCII error message if there is any.
	sockstate	Return ASCII message what is the current state.
	gethostid	Returned value is the IP address in host format.
Release Socket	sock_exit	Release all the TCP/IP system initialized by sock_init.
	sock_abort	Abort a connection.
	sock_close	Close a connection.
Send Socket	sock_write & sock_fastwrite	Write data out to a port. sock_fastwrite will have no check for data written out to the socket.
	sock_flush	Flush data out to the socket to make sure all the data has been sent.
	sock_flushnext	Call before write the data out to make sure that after write the data out to the socket, buffer will be flushed.
	sock_puts	Put string onto the buffer.
	sock_putc	Put a character onto the buffer.
Receive Socket	sock_read & sock_fastread	Read data coming into a port.

Function Category	Function Name	Description
	tcp_listen	Listen to a message coming in to a specified port.
	sock_gets	Get String
	sock_getc	Get Character
	sock_dataready	Return the number data ready to be read.
	rip	Remove carriage returns and line feeds.
Miscellaneous	inet_ntoa	Build ASCII representation of an IP address with a user supply string from decimal representation of the IP address.
	inet_addr	Convert string dot address to host format.
	ntohs	Convert network word to host word
	htons	Convert host word to network word
	ntohl	Convert network longword to host longword
	htonl	Convert host longword to network longword

9.2 ADMNET API Initialize Functions

The following topics detail the ADMNET API Initialize functions.

sock_init

Syntax

```
void sock_init(void);
```

Parameters

None

Description

This function will read a stored TCP/IP configuration file and prepare a variable.

Return Value

SK_SUCCESS	API has successfully initialized variables.
SK_PORT_NOT_ALLOW	API does not allow port number used.
SK_CANNOT_ALLOCATE_MEMORY	API cannot allocate memory.

Example

```
int numSK = 5;
int portNum = 5757;
int buffSize = 1000;

sock_init();    //initialize the socket interface

/* initialize each socket */
if(ADM_init_socket(numSK, portNum, buffSize, "ReceiveSK") != SK_SUCCESS)
{
printf("\nFailed to open ADM API... exiting program\n");
ADM_release_sockets();
}
```

See Also

sock_exit (page 126)

9.3 ADMNET API System Functionality

The following topics describe the ADMNET API System Functionality calls.

tcp_tick

Syntax

```
int tcp_tick( sock_type *skType );
```

Parameters

skType	Current socket Type or NULL for all sockets.

Description

This function is used by an application to determine the connection status of the sockets.

Return Value

0	disconnected or reset.
>0	connected.

Example

```
sock_type *socket;

. . .

if(tcp_tick(socket))  //check socket
{
  printf("Connected\n");
}
```

tcp_open

Syntax

```
int tcp_open( tcp_Socket *sk, word lPort, longword ina, word port,
dataHandler_t datahandler );
```

Parameters

sk	Pointer to the socket that has been initialized.
lPort	Local port number.
ina	Host IP Address.
port	Host port number.
datahandler	Data Handler. Not used in this version. Use NULL for this parameter.

Description

This function opens a TCP socket connection to a host machine using parameters passed to it. *lPort* is an option parameter. Most of the time, *lPort* can be set to 0. The API will find an available port number for the socket. *ina* is a host IP address passed as a longword. Function resolve can be used to convert an IP address into longword-formatted variable.

Return Value

	Connection cannot be made
>0	Connection is made

Example

```
tcp_Socket *socket;

. . .

if(tcp_open(socket, 0, resolve("192.168.0.1"), 5656, NULL))
{
printf("Open Successfully\n");
}
```

See Also

resolve (page 116)

tcp_open_fast

Syntax

```
int tcp_open_fast( tcp_Socket *sk, word lPort, longword ina, word port,
dataHandler_t datahandler );
```

Parameters

sk	Pointer to the socket that has been initialized.
lPort	Local port number.
ina	Host IP Address.
port	Host port number.
datahandler	Data Handler. Not used in this version. Use NULL for this parameter.

Description

This function opens a TCP socket connection to a host machine using parameters passed to it. For this function, there is no wait to resolve the IP address. *lPort* is an option parameter. Most of the time, *lPort* can be set to 0. The API will find an available port number for the socket. *ina* is a host IP address passed as a longword. Function resolve can be used to convert an IP address into a longword-formatted variable.

Return Value

	Connection cannot be made
>0	Connection is made

Example

```
tcp_Socket *socket;

. . .

if(tcp_open_fast(socket, 0, resolve("192.168.0.1"), 5656, NULL))
{
printf("Open Successfully\n");
}
```

See Also

resolve (page 116)

udp_open

Syntax

```
int udp_open( udp_Socket *sk, word lPort, longword ina, word port,
dataHandler_t datahandler );
```

Parameters

sk	Pointer to the socket that has been initialized.
lPort	Local port number.
ina	Host IP Address.
port	Host port number.
datahandler	Data Handler. Not used in this version. Use NULL for this parameter.

Description

This function opens a UDP socket connection to a host machine using parameters passed to it. *lPort* is an option parameter. Most of the time, *lPort* can be set to 0. The API will find an available port number for the socket. *ina* is a host IP address passed as a longword. Function resolve can be use to convert an IP address into a longword-formatted variable.

Return Value

	Connection cannot be made
>0	Connection is made

Example

```
udp_Socket *socket;

. . .

if(udp_open(socket, 0, resolve("192.168.0.1"), 5656, NULL))
{
printf("Open Successfully\n");
}
```

See Also

resolve (page 116)

udp_open_fast

Syntax

```
int udp_open_fast( tcp_Socket *sk, word lPort, longword ina, word port,
dataHandler_t datahandler );
```

Parameters

sk	Pointer to the socket that has been initialized.
lPort	Local port number.
ina	Host IP Address.
port	Host port number.
datahandler	Data Handler. Not used in this version. Use NULL for this parameter.

Description

This function opens a UDP socket connection to a host machine using parameters passed to it. For this function, there is no wait to resolve the IP address that passes the function. *lPort* is an option parameter. Most of the time, *lPort* can be set to 0. The API will find an available port number for the socket. *ina* is a host IP address passed as a longword. Function resolve can be used to convert an IP address into a longword-formatted variable.

Return Value

	Connection cannot be made
>0	Connection is made

Example

```
udp_Socket *socket;

. . .

if(udp_open_fast(socket, 0, resolve("192.168.0.1"), 5656, NULL))
{
printf("Open Successfully\n");
}
```

See Also

resolve (page 116)

resolve

Syntax

```
longword resolve( char *name );
```

Parameters

name	String IP Address.

Description

This function converts a string IP Address into a long.

Return Value

longword	Value of the IP Address in a long format.

Example

```
resolve("192.168.0.1");
```

sock_mode

Syntax

```
word sock_mode( sock_type *skType, word mode);
```

Parameters

skType	Current socket Type that will be used to set up socket mode.		
mode	The following is the available mode:		
	TCP_BINARY	0	default
	TCP_ASCII	1	treat as ASCII data
	UDP_CRC	0	checksum enable
	UDP_NOCRC	2	checksum disable
	TCP_NAGLE	0	default
	TCP_NONAGLE	4	used for real time application.

Description

This function is used set the socket transfer protocol mode.

Return Value

Current mode.

Example

```
sock_type *socket;

. . .

sock_mode(socket, TCP_MODE_NONAGLE);
```

sock_established

Syntax

```
int sock_established( sock_type *skType );
```

Parameters

skType	Current socket Type that will be used to check the connection.

Description

This function is used check if the socket has been established.

Return Value

	Not established.
1	Establish

Example

```
sock_type *socket;

. . .

if(sock_established(socket))
{
printf("Socket has been established\n");
}
```

ip_timer_init

Syntax

```
void ip_timer_init( sock_type *skType, word second );
```

Parameters

skType	Current socket Type that will be used to check the connection.
second	Number of second to set the timer. 0 mean no timer out.

Description

This function is used initialize the timer.

Return Value

None

Example

```
sock_type *socket;

. . .

ip_timer_init (socket, 100);
```

ip_timer_expired

Syntax

```
word ip_timer_expired( sock_type *skType );
```

Parameters

skType	Current socket Type that will be used to check the connection.

Description

This function is used check if the timer has been expired.

Return Value

1	timer has been expired.

Example

```
sock_type *socket;

. . .

if(ip_timer_expired (socket))
{
printf("time's up\n");
}
```

set_timeout

Syntax

```
longword set_timeout( word seconds );
```

Parameters

seconds	Number of second to set the timer.

Description

This function is used set the timer.

Return Value

Number of timeout.

Example

```
set_timeout (100);
```

chk_timeout

Syntax

```
word chk_timeout( longword timeout );
```

Parameters

timeout	Number of timeout return from set_timerout.

Description

This function is used check if the time is out.

Return Value

1	timeout

Example

```
int timeout = set_timeout (100);

 While(!chk_timeout (timeout))
printf("Not timeout yet\n");
```

sockerr

Syntax

```
char *sockerr ( sock_type *skType );
```

Parameters

skType	Current socket Type that will be used to check the connection.

Description

This function returns ASCII error message if there is any. Otherwise, NULL is returned.

Return Value

String message or NULL if there is no error.

Example

```
sock_type *socket;
char *p;

. . .

if(p = sockerr(socket) != NULL)
{
printf("Error: %s\n", p);
}
```

sockstate

Syntax

```
char *sockstate ( sock_type *skType );
```

Parameters

skType	Current socket Type that will be used to check the connection.

Description

This function returns ASCII message indicating current state.

Return Value

String message.

Example

```
sock_type *socket;
char *p;

. . .

if(p = sockstate(socket) != NULL)
{
printf("State: %s\n", p);
}
```

gethostid

Syntax

```
char *gethostid ( void );
```

Parameters

None

Description

This function returns value of the IP address in host format.

Return Value

String IP Address.

Example

```
sock_type *socket;
char *p;

. . .

if(p = gethostid(socket) != NULL)
{
printf("My IP: %s\n", p);
}
```

9.4 ADMNET API Release Socket Functions

This section describes the ADMNET API Release Socket Functions.

sock_exit

Syntax

```
void sock_exit( void );
```

Parameters

None

Description

This function is used by an application to release all the TCP/IP variables created by sock_init.

Return Value

None

Example

```
sock_exit();
```

See Also

sock_init (page 110)

sock_abort

Syntax

```
void sock_abort( sock_type *skType);
```

Parameters

skType	Current socket Type that will be used to abort the connection.

Description

This function is used abort a connection. This function is common for TCP connections.

Return Value

None

Example

```
sock_type *socket;

. . .

sock_abort(socket);
```

See Also

sock_close (page 128)

sock_close

Syntax

```
void sock_close ( sock_type *skType);
```

Parameters

skType	Current socket Type that will be used to close the connection.

Description

This function is used to permanently close a connection. This function is common for UDP connections.

Return Value

None

Example

```
sock_type *socket;

. . .

sock_close(socket);
```

See Also

sock_abort (page 127)

9.5 ADMNET API Send Socket Functions

This section describes the ADMNET API Send Socket functions.

sock_write

Syntax

```
int sock_write( sock_type *skType, byte *data, int len);
```

Parameters

skType	Socket that will be used to send data.
data	Pointer to a buffer that contains data that will be sent to a server.
len	Length of the data specified to send.

Description

This function writes data to the socket being passed to the function. The function will wait until the all the data is written.

Return Value

Number of Bytes that are written to the socket or -1 if an error occurs.

Example

```
sock_type *socket;
char theBuffer [512];
int len, bytes_sent;

. . .

bytes_sent = sock_write(socket, (byte*)theBuffer, len);
```

See Also

sock_fastwrite (page 130)

sock_fastwrite

Syntax

```
int sock_fastwrite( sock_type *skType, byte *data, int len);
```

Parameters

skType	Current socket that will be used to send data.
data	Pointer to a buffer that contains data that will be sent to a server.
len	Length of data specified to send.

Description

This function writes data to the socket being passed to the function. The function will not check to the data written out to the socket.

Return Value

Number of bytes that are written to the socket or -1 if an error occurs.

Example

```
sock_type *socket;
char theBuffer [512];
int len, bytes_sent;

. . .

bytes_sent = sock_fastwrite(socket, (byte*)theBuffer, len);
```

See Also

sock_write (page 129)

sock_flush

Syntax

```
void sock_flush( sock_type *skType );
```

Parameters

skType	Current socket that will be used to flush all the data out of the buffer.

Description

This function is used to flush all the data that is still in the buffer out to the socket. This function has no effect for UDP, since UDP is a connectionless protocol.

Return Value

None

Example

```
sock_type *socket;

. . .

sock_flush(socket);  // Flush the output
```

See Also

sock_flushnext (page 132)

sock_flushnext

Syntax

```
void sock_flushnext( sock_type *skType );
```

Parameters

skType	Current socket that will be used to flush all the data in the buffer out.

Description

This function is used after the write function is called to ensure that the data in a buffer is flushed immediately.

Return Value

None

Example

```
sock_type *socket;

. . .

sock_flushnext(socket);  // Flush the output
```

See Also

sock_flush (page 131)

sock_puts

Syntax

```
int sock_puts( sock_type *skType, byte *data);
```

Parameters

e	Socket that will be used to put string data to.
data	Pointer to the string that will be sent.

Description

This function sends a string to the socket. Character new line "\n", will be attached to the end of the string.

Return Value

The length that is written to the socket.

Example

```
sock_type *socket;
char data [512];
int len;

. . .

len = sock_puts(socket, data);
printf("Put %d\n", len);
```

See Also

sock_putc (page 134)

sock_putc

Syntax

```
byte sock_putc( sock_type *skType, byte character);
```

Parameters

skType	Socket that will be used to get string data from.
character	A character that is used.

Description

This function is used to put one character at a time to the socket.

Return Value

Character put in is returned.

Example

```
sock_type *socket;
char in;

. . .


in = sock_putc(socket, 'A');
printf("%c", in);
```

See Also

sock_puts (page 133)

9.6 ADMNET API Receive Socket Functions

This section describes the ADMNET API Receive Socket functions.

sock_read

Syntax

```
int sock_read( sock_type *skType, byte *data, int len);
```

Parameters

skType	Socket that will be used to receive data.
data	Pointer to a buffer that contains data that is received.
len	Length of the data specified to receive.

Description

This function reads data from the socket being passed to the function. The function will wait until the all the data is read.

Return Value

Number of Bytes that are read to the socket or -1 if an error occurs.

Example

```
sock_type *socket;
char theBuffer [512];
int len, bytes_receive;

. . .

bytes_receive = sock_read(socket, (byte*)theBuffer, len);
```

See Also

sock_fastread (page 136)

sock_fastread

Syntax

```
int sock_fastread( sock_type *skType, byte *data, int len);
```

Parameters

skType	Current socket that will be used to receive data.
data	Pointer to a buffer that contains data that is received to a server.
len	Length of data specified to receive.

Description

This function reads data to the socket being passed to the function. The function will not check to the data read into the socket.

Return Value

Number of bytes that are read to the socket or -1 if an error occurs.

Example

```
sock_type *socket;
char theBuffer [512];
int len, bytes_receive;

. . .

bytes_receive = sock_fastread(socket, (byte*)theBuffer, len);
```

See Also

sock_read (page 135)

tcp_listen

Syntax

```
int tcp_listen( tcp_Socket *sk, word lPort, longword ina, word port,
dataHandler_t datahandler, word timeout );
```

Parameters

sk	Pointer to the socket that has been initialized.
lPort	Local port number.
datahandler	Data Handler. Not used in this version. Use NULL for this parameter.
ina	Host IP Address.
port	Host port number.
timeout	Value used to set the period of time to wait for data. 0 is set to indicate no timeout.

Description

This function is used for listening to an incoming message. *port* is an option parameter. Most of the time, port can be set to 0. The API will find an available port number for the socket. *ina* is a host IP address passed as a longword. Function resolve can be used to convert an IP address into a longword-formatted variable. 0 can be passed as an *ina* value if there is no specific IP Address to listen too.

Example

```
tcp_Socket *socket;
int port = 5656;

. . .

tcp_listen(socket, port, 0L, 0, NULL, 0);
```

See Also

ADM_send_socket (page 101)

sock_gets

Syntax

```
int sock_gets( sock_type *skType, byte *data, int len);
```

Parameters

skType	Socket that will be used to get string data from.
data	Pointer to the string return.
len	Specified length for the function to get the string.

Description

This function is used for obtaining a string from the socket. The *len* parameter specifies how long the string will be read.

Return Value

The length read from the socket is returned.

Example

```
sock_type *socket;
char data [512];
int len;

. . .

len = sock_gets(socket, data, 100);
printf("Get %d\n", len);
```

See Also

sock_getc (page 139)

sock_getc

Syntax

```
int sock_getc( sock_type *skType);
```

Parameters

skType	Socket that will be used to get string data from.

Description

This function gets one character at a time from the socket.

Return Value

Character read in is returned.

Example

```
Sock_type *socket;
char in;

. . .


in = sock_getc(socket);
printf("%c", in);
```

See Also

sock_gets (page 138)

sock_dataready

Syntax

```
int sock_dataready( sock_type *skType );
```

Parameters

skType	Current socket that will be used to check if data is ready to be read.

Description

This function is used check if there is data ready to be read.

Return Value

Number of bytes ready to be read or -1 if error occurs.

Example

```
int in;
sock_type *socket;

. . .


in = sock_dataready(socket);
printf("%d", in);
```

rip

Syntax

```
Char * rip( char *String );
```

Parameters

String	Array of character string.

Description

This function is used to strip out carriage return and line feed. If there are more than one carriage return or line feed, the first one will be replace with 0 and the rest of them will not be defined.

Return Value

Pointer to the new string.

Example

```
char s;

. . .


s = sock_dataready("This is a test\n\r");
printf("%s", s);
```

inet_ntoa

Syntax

```
Char * inet_ntoa( char *String, longword IP );
```

Parameters

String	Array of character string.
IP	Decimal representation of IP address.

Description

This function builds ASCII representation of an IP address with a user supply string from decimal representation of the IP address. The size of the buffer has to be at least 16 byte.

Return Value

Pointer to the new string.

Example

```
char buffer[ 20 ];

sock_init();

printf("My IP address is %s\n", inet_ntoa( buffer, gethostid()));
```

inet_addr

Syntax

```
longword * inet_addr( char *String);
```

Parameters

String	Array of character string.

Description

This function converts string dot address to host format.

Return Value

Host IP address format.

Example

```
char buffer[ ] = "192.168.0.1";

sock_init();

printf("My IP address is %ld\n", inet_addr( buffer ));
```

10 Support, Service & Warranty

In This Chapter

10.1 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

10.2 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