ProLinx®
MCM4-ADM4
ProLinx Standalone
'C' Programmable Modbus Communication Module

February 20, 2013

DEVELOPER'S GUIDE
Important Installation Instructions

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

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

All ProLinx® Products

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.

Markings

<table>
<thead>
<tr>
<th>UL/cUL</th>
<th>ISA 12.12.01 Class I, Div 2 Groups A, B, C, D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cUL</td>
<td>C22.2 No. 213-M1987</td>
</tr>
</tbody>
</table>

Markings:
- CL I Div 2 GPs A, B, C, D
- Temp Code T5
- II 3 G
- Ex nA nL IIC T5 X
- $0^\circ C \leq Ta \leq 60^\circ C$
- 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.
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 the product, 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.

MCM4-ADM4 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
Latin America: +1.281.298.9109
## Contents

### Important Installation Instructions
- Page 2

### Your Feedback Please
- Page 3

### Content Disclaimer
- Page 3

### 1 Introduction
- Page 7
  - 1.1 Operating System

### 2 Preparing the PLX-MCM4 Module
- Page 9
  - 2.1 Package Contents
  - 2.2 Setting Port 0 Configuration Jumpers
  - 2.3 Mounting the gateway on the DIN-rail
  - 2.4 Connecting Power to the Unit
  - 2.5 Cable Connections

### 3 Setting Up Your Development Environment
- Page 17
  - 3.1 Setting Up Your Compiler

### 4 Programming the Module
- Page 37
  - 4.1 Debugging Strategies
  - 4.2 RS-485 Programming Note

### 5 Understanding the ADM API
- Page 39
  - 5.1 API Libraries
  - 5.2 Development Tools
  - 5.3 Theory of Operation
  - 5.4 ADM Functional Blocks
  - 5.5 ADM API Files

### 6 Application Development Function Library - ADM API
- Page 45
  - 6.1 ADM API Functions
  - 6.2 Core Functions
  - 6.3 Database Functions
  - 6.4 Clock Functions
  - 6.5 Console Port Functions
  - 6.6 LED Functions
  - 6.7 Serial Port Functions

### 7 Reference
- Page 103
  - 7.1 Product Specifications
  - 7.2 MCM Database Definition
7.3 Configuration Data ................................................................. 106
7.4 Modbus Error and Status Data Area Addresses .................. 109
7.5 Error Codes ........................................................................... 112
7.6 LED Indicators ...................................................................... 114

8 DOS 6 XL Reference Manual .................................................. 117

9 Glossary of Terms .................................................................. 119

10 Support, Service & Warranty ................................................. 123
   10.1 Contacting Technical Support .......................................... 123
   10.2 Warranty Information ....................................................... 124

Index ......................................................................................... 125
1 Introduction

In This Chapter

- Operating System .......................................................... 7

This document provides information needed for development of application programs for the MCM4-ADM4 Serial Communication Module.

The modules are programmable to accommodate devices with unique serial protocols.

Included in this document is information about the available software API libraries and tools, module configuration and programming information, and example code for the module.

1.1 Operating System

The 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 Borland compilers.

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

Refer to the General Software Embedded DOS 6-XL Developer’s Guide (page 117) on the MCM4-ADM4 CD-ROM for more information.
2 Preparing the PLX-MCM4 Module

In This Chapter

- Package Contents ................................................................. 9
- Setting Port 0 Configuration Jumpers .................................. 10
- Mounting the gateway on the DIN-rail ............................... 11
- Connecting Power to the Unit ............................................... 11
- Cable Connections ............................................................... 12

2.1 Package Contents

The following components are included with your MCM4-ADM4 gateway, and are all required for installation and configuration.

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

<table>
<thead>
<tr>
<th>Qty.</th>
<th>Part Name</th>
<th>Part Number</th>
<th>Part Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MCM4-ADM4 gateway</td>
<td>PLX-####</td>
<td>ProLinx communication gateway gateway</td>
</tr>
<tr>
<td>1</td>
<td>Cable</td>
<td>Cable #15, RS232 Null Modem</td>
<td>For RS232 Connection from a PC to the CFG Port of the gateway</td>
</tr>
<tr>
<td>Varies</td>
<td>Cable</td>
<td>Cable #9, Mini-DIN8 to DB9 Male Adapter</td>
<td>For DB9 Connection to gateway’s Port. One DIN to DB-9M cable included per configurable serial port, plus one for gateway configuration</td>
</tr>
<tr>
<td>Varies</td>
<td>Adapter</td>
<td>1454-9F</td>
<td>Adapters, DB9 Female to Screw Terminal. For RS422 or RS485 Connections to each serial application port of the gateway</td>
</tr>
<tr>
<td>1</td>
<td>ProSoft Solutions CD</td>
<td></td>
<td>Contains sample programs, utilities and documentation for the MCM4-ADM4 gateway.</td>
</tr>
</tbody>
</table>

If any of these components are missing, please contact ProSoft Technology Support for replacements.
2.2 Setting Port 0 Configuration Jumpers

Before installing the module on the DIN-rail, you must set the jumpers for the Port 0 application port.

**Note:** Ethernet-only ProLinx modules do not use the serial port jumper settings. The serial configuration jumper settings on an Ethernet-only module have no effect.

**Note:** The presence of Port 0 depends on the specific combination of protocols in your ProLinx module. If your module does not have a Port 0, the following jumper settings do not apply.

Port 0 is preconfigured for RS-232. You can move the port configuration jumper on the back of the module to select RS-485 or RS-422.

**Note:** Some ProLinx modules do not correctly report the position of the port 0 jumper to the Port Configuration page on the Config/Debug menu. In cases where the reported configuration differs from the known jumper configuration, the physical configuration of the jumper is correct.

The following illustration shows the jumper positions for Port 0:

![ProLinx 5000/6000 Series Module](image)
2.3 Mounting the gateway on the DIN-rail

![Diagram of ProLinx 5000/6000 Series gateway]

2.4 Connecting Power to the Unit

![Diagram of power connection details]

**WARNING:** Ensure that you do not reverse polarity when applying power to the gateway. This will cause damage to the gateway’s power supply.
2.5 Cable Connections

The application ports on the MCM4-ADM4 module support RS-232, RS-422, and RS-485 interfaces. Please inspect the module to ensure that the jumpers are set correctly to correspond with the type of interface you are using.

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

### 2.5.1 RS-232

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

RS-232 Application Port Cable
(No Handshaking)

<table>
<thead>
<tr>
<th>DB-9 Male</th>
<th>RS-232 Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>RxD</td>
<td>2</td>
</tr>
<tr>
<td>TxD</td>
<td>3</td>
</tr>
<tr>
<td>COM</td>
<td>5</td>
</tr>
</tbody>
</table>

**RS-232: Modem Connection**

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

RS-232 Application Port Cable
(Modem Connection)

<table>
<thead>
<tr>
<th>DB-9 Male</th>
<th>RS-232 Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD</td>
<td>3</td>
</tr>
<tr>
<td>RxD</td>
<td>2</td>
</tr>
<tr>
<td>RTS</td>
<td>7</td>
</tr>
<tr>
<td>CTS</td>
<td>8</td>
</tr>
<tr>
<td>Signal</td>
<td>5</td>
</tr>
<tr>
<td>Common</td>
<td></td>
</tr>
<tr>
<td>DTR</td>
<td>4</td>
</tr>
</tbody>
</table>

The "Use CTS Line" parameter for the port configuration should be set to 'Y' for most modem applications.
RS-232: Null Modem Connection (Hardware Handshaking)

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

RS-232 Application Port Cable
(Hardware Handshaking)

<table>
<thead>
<tr>
<th>DB-9 Male</th>
<th>RS-232 Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD</td>
<td>3</td>
</tr>
<tr>
<td>RxD</td>
<td>2</td>
</tr>
<tr>
<td>RTS</td>
<td>7</td>
</tr>
<tr>
<td>CTS</td>
<td>8</td>
</tr>
<tr>
<td>Signal Common</td>
<td>5</td>
</tr>
<tr>
<td>DTR</td>
<td>4</td>
</tr>
<tr>
<td>Signal Common</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DSR</td>
</tr>
<tr>
<td></td>
<td>DCD</td>
</tr>
</tbody>
</table>

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

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

RS-232 Application Port Cable
(No Handshaking)

<table>
<thead>
<tr>
<th>DB-9 Male</th>
<th>RS-232 Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD</td>
<td>3</td>
</tr>
<tr>
<td>RxD</td>
<td>2</td>
</tr>
<tr>
<td>RTS</td>
<td>7</td>
</tr>
<tr>
<td>CTS</td>
<td>8</td>
</tr>
<tr>
<td>Signal Common</td>
<td>5</td>
</tr>
<tr>
<td>DTR</td>
<td>4</td>
</tr>
<tr>
<td>Signal Common</td>
<td></td>
</tr>
</tbody>
</table>

Note: If the port is configured with the "Use CTS Line" set to 'Y', then a jumper is required between the RTS and the CTS line on the module connection.
2.5.2 **RS-232 Configuration/Debug Port**

This port is physically a Mini-DIN connection. A Mini-DIN 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:

![RS-232 Config/Debug Port Cable Diagram](image)

2.5.3 **RS-485**

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

![RS-485 Application Port Cable Diagram](image)

**Note:** Terminating resistors are generally not required on the RS-485 network, unless you are experiencing communication problems that can be attributed to signal echoes or reflections. In this case, install a 120-ohm terminating resistor on the RS-485 line.
2.5.4 RS-422

RS-422 Application Port Cable

<table>
<thead>
<tr>
<th>DB-9 Male</th>
<th>RS-422 Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>TxD+</td>
<td>RxD+</td>
</tr>
<tr>
<td>TxD-</td>
<td>RxD-</td>
</tr>
<tr>
<td>Signal Common</td>
<td>Signal Common</td>
</tr>
<tr>
<td>RxD+</td>
<td>TxD+</td>
</tr>
<tr>
<td>RxD-</td>
<td>TxD-</td>
</tr>
</tbody>
</table>

RS-485 and RS-422 Tip

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

2.5.5 DB9 to Mini-DIN Adaptor (Cable 09)

Wiring Diagram
3 Setting Up Your Development Environment

In This Chapter

- Setting Up Your Compiler ................................................................. 17

3.1 Setting Up Your Compiler

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

3.1.1 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_MCM_TOOL_PLX.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_MCM_TOOL_PLX\SAMPLES\.

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\MCM_TOOL\PLX\SAMPLES\...).

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

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

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:

---

**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**.

![Project Settings Window](image)

**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 Express Window](image)

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

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:

12 Click Next.

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:

16 Click Project → Settings from the Main Menu.

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

![Project Settings screenshot](image)

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:

![Project Settings screenshot](image)
21 Click **Code Generation** from the *Topics* field and ensure that the options match those shown in the following screen:

![Code Generation Screen](image)

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

![Memory Models Screen](image)
23 Click **Linker** from the *Topics* field and ensure that the options match those shown in the following screen:

![Linker settings](image1.png)

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

![Packing & Map File settings](image2.png)
25 Click Make from the Topics field and ensure that the options match those shown in the following screen:

![Project Settings](image)

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:

![Build Results](image)

The executable file will be located in the directory listed in the Compiler Output Directory box of the Directories tab (that is, C:\ADM_MCM_TOOL\PLXL\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.
3.1.2 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 MCM4ADM.zip file. This zip file is available from the CD-ROM shipped with your system or from the www.prosoft-technology.com web site. Once the file is unzipped, you can find the sample code files in `\MCM4ADM\Sample`.

**Note:** ProSoft recommends using the project file MCMADM.IDE as a starting point for your project. You can then modify this file for your particular needs.

**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:

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

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.
**Creating a New Borland C++ 5.02 ADM Project**

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

2. Type in the **Project Path and Name**. The Target Name is created automatically.
3. In the **Target Type** field, choose **Application (.exe)**.
4. In the **Platform** field, choose **DOS (Standard)**.
5. In the **Target Model** field, choose **Large**.
6. Ensure that **Emulation** is checked in the **Math Support** field.
7. Click **OK**. A Project window appears:
8 Click on the .cpp file created and press the **Delete** key. Click **Yes** to delete the .cpp file.

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

![image of Add to Project List window]

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:

![image of Add to Project List window with library files]
13 The Project window should now contain all the necessary source and library files as shown in the following window:

![Project window](image)

14 Click Options → Project from the Main Menu.
15 Click **Directories** from the *Topics* field and fill in directory information as required by your project’s directory structure.

![Directories screenshot]

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

![Screen showing Memory Model settings]

18 Click **OK**.

19 Click **Project → Build All** from the *Main Menu*.

20 When complete, the **Success** window appears:

![Success window displaying build information]

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*.
3.1.3 Downloading Files to the Module

1. Connect your PC’s COM port to the ProLinx Configuration/Debug port using the Null Modem cable and ProLinx Adapter cable.

2. From the Start Menu on your PC, select Programs → Accessories → Communications → HyperTerminal. The New Connection Screen appears:

3. Enter a name and choose OK. The Connect To window appears:
4 Choose the COM port that your ProLinx module is connected to and choose OK. The COM1 Properties window appears.

5 Ensure that the settings shown on this screen match those on your PC.
6 Click OK. The HyperTerminal window appears with a DOS prompt and blinking cursor.
7 Apply power to the ProLinx module and hold down the [L] key. The screen displays information and ultimately displays the Loader menu:
This menu provides options that allow you to download a configuration file [C], a WATTCP file [W], or a new executable file [U]. You can also press [V] to view module version information.

1. Type [U] at the prompt to transfer executable files from the computer to the ProLinx unit.
2. Type [Y] when the program asks if you want to load an .exe file.
3. From the HyperTerminal menu, select Transfer → Send.

4. When the Send To screen appears, browse for the executable file to send to the module. Be sure to select Y Modem in the Protocol field.

5. Click Send. The program loads the new executable file to the ProLinx module. When the download is complete, the program returns to the Loader menu.

   If you want to load a new configuration file or a WATTCP file, select the appropriate option and perform the same steps to download these files.

6. Press [Esc], then [Y] to confirm module reboot.
4 Programming the Module

In This Chapter

- Debugging Strategies ................................................................. 37
- RS-485 Programming Note ............................................................ 37

This section describes how to get your application running on the ProLinx module. Once an application has been developed using the serial API, it must be downloaded to the ProLinx module in order to run. The application may then be run manually from the console command line, or automatically on boot from the AUTOEXEC.BAT or CONFIG.SYS files.

4.1 Debugging Strategies

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

4.2 RS-485 Programming Note

4.2.1 Hardware

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

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

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

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

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

In This Chapter

- API Libraries .......................................................... 39
- Development Tools ..................................................... 41
- Theory of Operation .................................................... 41
- ADM Functional Blocks ............................................... 41
- ADM API Files .......................................................... 43

The ADM API Suite allows software developers to access the serial ports without needing detailed knowledge of the module’s hardware design. The API provides for Modbus Master/Slave and generic serial ports.

Applications for the MCM4-ADM4 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 MCM4-ADM4 module.

5.1 API Libraries

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

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

Note: The following compiler versions are intended to be compatible with the PLX 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.

5.1.1 Calling Convention

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

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

5.1.3 Sample Code

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

5.1.4 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** libraries have been tested and are thread-safe for use in multi-threaded applications.
- **MVIsp** libraries are safe to use in multi-threaded applications with the following precautions: If you call the same `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 `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.
5.2 Development Tools

An application that is developed for the MCM4-ADM4 module must be stored on the module’s Flash ROM disk to be executed. A loader program is provided with the module, to download an executable, configuration file or wattcp.cfg file via module port 0, as needed.

5.3 Theory of Operation

5.3.1 ADM API

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

5.3.2 ADM API Architecture

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

The following illustration shows the relationship between the API components.

5.4 ADM Functional Blocks

5.4.1 Debug / Status Port

The Configuration/Debug Port allows you to transmit or receive configuration data, view database information in the module and view configuration data. Use of this port can aid in locating problems that may exist in the user configuration and attached devices. Refer to Diagnostics and Troubleshooting for information on using the Config/Debug port.
5.4.2 Serial Communications

The developer must provide the serial communication driver code. The serial API has many useful functions to facilitate writing a driver. A sample communication driver is included in the example program provided.

5.4.3 Database

The database functions of the ADM API allow the creation of a database in memory to store data to be accessed via the application ports. The database consists of word registers that can be accessed as bits, bytes, words, longs, floats or doubles. Functions are provided for reading and writing the data in the various data types. The database serves as a holding area for exchanging data with foreign devices attached to the application ports.

5.4.4 MCM4_ADM.C

The application starts by opening the MCMADM API. The console for the Debug port and the database is installed. A protocol driver for each port is registered. Two protocol drivers are available, Modbus (MCM) and user developed (ADM). When registering ADM protocol drivers the name of the user written functions must be passed to the registration function.

The startup function is then called, initializing the application and protocol drivers.

The application enters the run loop that calls the run functions of the protocol drivers.

When an ESC key is received on the Debug port the loop exits and the shutdown function is called, shutting down the application and the protocol drivers.

5.4.5 MCMADM.H

This header file contains definitions and function declarations for the MCMADM library.
5.4.6 adm_prot.c

This file contains sample ADM protocol driver functions. There are startup, run and shutdown functions for each of the ADM ports.

The ADM driver on Port 0 is an ASCII talker. On 1 second intervals a count value is retrieved from the database, incremented, and saved back to the database. This counter value is then sent out of the serial port.

The ADM driver on Port 1 is an ASCII listener. This driver receives a two byte ASCII value from the serial port (sent from Port 0), swaps the byte order and saves the value to the database.

5.4.7 adm_prot.h

This header file contains definitions and function declarations for the ADM protocol driver.

5.5 ADM API Files

Table 1 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.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcmadm.h</td>
<td>Include file</td>
</tr>
<tr>
<td>mcmadm.lib</td>
<td>Library (16-bit OMF format)</td>
</tr>
</tbody>
</table>
6 Application Development Function Library - ADM API

In This Chapter

- ADM API Functions ................................................................. 45
- Core Functions ........................................................................... 47
- Database Functions ..................................................................... 58
- Clock Functions .......................................................................... 84
- Console Port Functions .............................................................. 88
- LED Functions ............................................................................ 90
- Serial Port Functions ................................................................. 91

6.1 ADM API Functions

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

API library routines are categorized according to functionality.

<table>
<thead>
<tr>
<th>Function Category</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Functions</td>
<td>MCM_Open</td>
<td>Opens the API and enables the other functions to be used</td>
</tr>
<tr>
<td></td>
<td>MCM_RegisterProtocol</td>
<td>Registers a Modbus driver on a particular port</td>
</tr>
<tr>
<td></td>
<td>ADM_RegisterProtocol</td>
<td>Registers a ADM driver on a particular port</td>
</tr>
<tr>
<td></td>
<td>ADM_RegisterUserFunc</td>
<td>Registers a user process in the application</td>
</tr>
<tr>
<td></td>
<td>MCM_InstallDatabase</td>
<td>Creates the database area for the protocols to pass data to one another</td>
</tr>
<tr>
<td></td>
<td>MCM_InstallConsole</td>
<td>Installs the console on the Debug port</td>
</tr>
<tr>
<td></td>
<td>MCM_Startup</td>
<td>Performs the module startup process</td>
</tr>
<tr>
<td></td>
<td>MCM_Run</td>
<td>Performs the module run process</td>
</tr>
<tr>
<td></td>
<td>MCM_Shutdown</td>
<td>Performs the module shutdown process</td>
</tr>
<tr>
<td>Database Functions</td>
<td>MCM_DBGGetBit</td>
<td>Get bit value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetBit</td>
<td>Set bit</td>
</tr>
<tr>
<td></td>
<td>MCM_DBClearBit</td>
<td>Clear bit</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGGetByte</td>
<td>Get byte value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetByte</td>
<td>Set byte value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGGetWord</td>
<td>Get 16-bit word value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetWord</td>
<td>Set 16-bit word value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGGetLong</td>
<td>Get 32-bit long word value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetLong</td>
<td>Set 32-bit long word value</td>
</tr>
<tr>
<td>Function Category</td>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGetFloat</td>
<td>Get 32-bit float value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetFloat</td>
<td>Set 32-bit float value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGetDFloat</td>
<td>Get 64-bit double float value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetDFloat</td>
<td>Set 64-bit double float value</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGetBytes</td>
<td>Get multiple bytes</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetBytes</td>
<td>Set multiple bytes</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGetWords</td>
<td>Get multiple 16-bit words</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetWords</td>
<td>Set multiple 16-bit words</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGetString</td>
<td>Get ASCII string</td>
</tr>
<tr>
<td></td>
<td>MCM_DBSetString</td>
<td>Set ASCII string</td>
</tr>
<tr>
<td></td>
<td>MCM_DBGetIntPtr</td>
<td>Get a pointer to a word location</td>
</tr>
<tr>
<td></td>
<td>MCM_DBBitChanged</td>
<td>Test for bit changed</td>
</tr>
<tr>
<td></td>
<td>MCM_DBBYTEChanged</td>
<td>Test for byte changed</td>
</tr>
<tr>
<td></td>
<td>MCM_DBChanged</td>
<td>Test for 16-bit word changed</td>
</tr>
<tr>
<td></td>
<td>MCM_DBLONGChanged</td>
<td>Test for 32-bit long word changed</td>
</tr>
<tr>
<td></td>
<td>MCM_DBFloatChanged</td>
<td>Test for 32-bit float changed</td>
</tr>
<tr>
<td></td>
<td>MCM_DBDoubleChanged</td>
<td>Test for 64-bit double float changed</td>
</tr>
<tr>
<td>Clock Functions</td>
<td>MCM_ClockGetHandle</td>
<td>Gets access to a timer</td>
</tr>
<tr>
<td></td>
<td>MCM_ClockStart</td>
<td>Starts timer</td>
</tr>
<tr>
<td></td>
<td>MCM_ClockCheck</td>
<td>Check for timeout</td>
</tr>
<tr>
<td></td>
<td>MCM_ClockGetValue</td>
<td>Gets value of timer</td>
</tr>
<tr>
<td>Console Port</td>
<td>MCM_Send</td>
<td>Send characters to the console</td>
</tr>
<tr>
<td></td>
<td>MCM_GetKey</td>
<td>Get a key from the console</td>
</tr>
<tr>
<td>LED Functions</td>
<td>MCM_LED_Set</td>
<td>Activate/deactivate LEDs</td>
</tr>
<tr>
<td>Serial Port</td>
<td>MCM_SendBytes</td>
<td>Send bytes to the serial port using the built-in driver</td>
</tr>
<tr>
<td></td>
<td>MCM_SendBytesDirect</td>
<td>Send bytes directly to the serial port</td>
</tr>
<tr>
<td></td>
<td>MCM_SetRTS</td>
<td>Set the RTS level</td>
</tr>
<tr>
<td></td>
<td>MCM_SetDTR</td>
<td>Set the DTR level</td>
</tr>
<tr>
<td></td>
<td>MCM_GetCTS</td>
<td>Get the CTS level</td>
</tr>
<tr>
<td></td>
<td>MCM_GetByte</td>
<td>Get character from receive buffer</td>
</tr>
<tr>
<td></td>
<td>MCM_GetAsciiString</td>
<td>Get an ASCII string from the receive buffer</td>
</tr>
<tr>
<td></td>
<td>MCM_GetDataString</td>
<td>Get a multiple bytes from the receive buffer</td>
</tr>
<tr>
<td></td>
<td>MCM_BytesInTransmitBuffer</td>
<td>Get the number of bytes in the transmit buffer still to be sent</td>
</tr>
<tr>
<td></td>
<td>MCM_BytesInReceiveBuffer</td>
<td>Get the number of bytes in the receive buffer</td>
</tr>
<tr>
<td></td>
<td>MCM_FlushTransmitBuffer</td>
<td>Clear characters from the transmit buffer</td>
</tr>
<tr>
<td></td>
<td>MCM_FlushReceiveBuffer</td>
<td>Clear characters from the receive buffer</td>
</tr>
</tbody>
</table>
6.2 Core Functions

MCM_Open

Syntax

ADMAPIENTRY MCM_Open(void);

Parameters

None

Description

This function opens the MCMADM API. This function must be called before any of the other API functions can be used.

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

Return Value

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>API was opened successfully</td>
</tr>
<tr>
<td>ADM_ERR_REOPEN</td>
<td>API is already open</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>API cannot run on this hardware</td>
</tr>
</tbody>
</table>

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

Example

/* open MCMADM API */
if(MCM_Open() != ADM_SUCCESS)
{
    printf("\nFailed to open MCMADM API... exiting program\n");
    exit(1);
}
# MCM_RegisterProtocol

## Syntax

```c
ADMAPIENTRYW MCM_RegisterProtocol(int port);
```

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Com port to use (0 to 3)</td>
</tr>
</tbody>
</table>

## Description

This function registers and installs an MCM driver on the Com port. This function must be called in order to use the MCM protocol driver.

## Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>MCM driver was installed successfully</td>
</tr>
<tr>
<td>ADM_ERR_REOPEN</td>
<td>MCM driver is already installed</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>API is not open</td>
</tr>
<tr>
<td>ADM_ERR_BADPARAM</td>
<td>Com port specified is out of range</td>
</tr>
</tbody>
</table>

## Example

```c
MCM_RegisterProtocol(0);  // Register MCM driver on port 0
```
ADM_RegisterProtocol

Syntax

```
ADMAPIENTRYW ADM_RegisterProtocol(int port, void (*startup_func)(), void (*run_func)(), void (*shutdown_func)());
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Com port to use (0 to 3)</td>
</tr>
<tr>
<td>startup_func</td>
<td>Pointer to user startup function</td>
</tr>
<tr>
<td>run_func</td>
<td>Pointer to user run function</td>
</tr>
<tr>
<td>shutdown</td>
<td>Pointer to user shutdown function</td>
</tr>
</tbody>
</table>

Description

This function registers and installs an ADM driver on the Com port. This function must be called in order to use the ADM port driver. A pointer to a startup, run and shutdown function must be provided. These functions will be called by the system at various times. The startup function will be called once during the boot process. When the module enters the run loop the run function will be called once per loop. When shutdown of the module is requested the shutdown function will be called once.

**Note:** The run function should be written to be non-blocking to ensure timely processing of all the drivers.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>ADM driver was installed successfully</td>
</tr>
<tr>
<td>ADM_ERR_REOPEN</td>
<td>ADM driver is already installed</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>API is not open</td>
</tr>
<tr>
<td>ADM_ERR_BADPARAM</td>
<td>Com port specified is out of range</td>
</tr>
</tbody>
</table>

Example

```c
/* Set port 0 as an ADM port */
ADM_RegisterProtocol(0,
ADM_Protocol_Startup0,
ADM_Protocol_Run_Talker,
ADM_Protocol_Shutdown0);
/* startup function for port 0 */
void ADM_Protocol_Startup0(void)
{
    printf("ADM Startup0\n");
    MCM_FlushTransmitBuffer(0);
    // If clock handle does not exist get handle
    if(CountTimer == -1)
        CountTimer = MCM_ClockGetHandle();
    /* start 1 second timer */
    MCM_ClockStart(CountTimer, 1000000L);
}
```
/* run function for port 0 *
void ADM_Protocol_Run_Talker(void)
{
    /* check timer */
    if(MCM_ClockCheck(CountTimer) == TRUE)
        return;
    /* re-start clock, 1 second */
    MCM_ClockStart(CountTimer, 1000000L);
    /* get counter from database */
    Counter = MCM_DBGetWord(COUNTER_OFFSET);
    /* increment count */
    Counter++;
    /* save new count to database */
    MCM_DBSetWord(COUNTER_OFFSET, Counter);
    /* get count from database and swap bytes */
    TxBuff[1] = MCM_DBGetByte(COUNTER_OFFSET*2);
    TxBuff[0] = MCM_DBGetByte((COUNTER_OFFSET*2)+1);
    /* send count message out of port */
    MCM_SendBytes(0, TxBuff, 2);
}

/* shutdown function for port 0 */
void ADM_Protocol_Shutdown0(void)
{
    printf("ADM Shutdown0\n");
}

Note: The pointers to the user functions are the names of the functions.
ADM_RegisterUserFunc

Syntax

ADMAPIENTRYW ADM_RegisterUserFunc(void (*startup_func)(), void (*run_func)(), void(*shutdown_func)() , int (*debug_func)());

Parameters

| startup_func | Pointer to user startup function |
| run_func     | Pointer to user run function     |
| shutdown     | Pointer to user shutdown function|
| debug_func   | Pointer to user debug function   |

Description

This function registers and installs a user process. This function is useful for adding a user-defined process to the application. A pointer to a startup, run and shutdown function must be provided. These functions will be called by the system at various times. The startup function will be called once during the boot process. When the module enters the run loop the run function will be called once per loop. When shutdown of the module is requested the shutdown function will be called once.

Note: The run function should be written to be non-blocking to ensure timely processing of all the drivers.

| ADM_SUCCESS                        | ADM driver was installed successfully |
| ADM_ERR_NOACCESS                  | API is not open                        |

Example

```c
void ADM_Protocol_Startup(void)
{
    /* initialize user function */
    ...
}

void ADM_Protocol_Run(void)
{
    /* run user function */
    ...
}

void ADM_Protocol_Shutdown(void)
{
    /* close user function */
    ...
}

int ADM_Protocol_Debug(void)
{
    /* print out debugging information */
    ...
}
```
... ADM_RegisterUserFunc(
    ADM_Protocol_Startup,
    ADM_Protocol_Run,
    ADM_Protocol_Shutdown,
    ADM_Protocol_Debug);
MCM_InstallDatabase

Syntax

ADMAPIENTRYW MCM_InstallDatabase(unsigned int size);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Size of database in 16-bit registers</td>
</tr>
</tbody>
</table>

Description

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>Database was installed successfully</td>
</tr>
<tr>
<td>ADM_ERR_DB_MAX_SIZE</td>
<td>Database maximum size exceeded</td>
</tr>
<tr>
<td>ADM_ERR_MEMORY</td>
<td>Insufficient memory for database</td>
</tr>
<tr>
<td>ADM_ERR_REOPEN</td>
<td>Database is already installed</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>API is not open</td>
</tr>
<tr>
<td>ADM_ERR_BADPARAM</td>
<td>Size is less than 1000 or greater than 10000</td>
</tr>
</tbody>
</table>

Example

MCM_InstallDatabase(4000);  // Install database of 4000 registers
# MCM_InstallConsole

## Syntax

```c
ADMAPIENTRYW MCM_InstallConsole(void);
```

## Parameters

None

## Description

This function installs the console on the Debug port. This allows access to the module through a terminal emulation program such as Hyper Term.

## Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>Console was installed successfully</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>API is not open</td>
</tr>
</tbody>
</table>

## Example

```c
/* initialize console */
MCM_InstallConsole();
```
MCM_Startup

Syntax

ADMAPIENTRYW MCM_Startup(void);

Parameters

None

Description

This function performs the module initialization. The protocol drivers must be registered before the initialization is performed. During the initialization the protocol drivers will be initialized and the database will be cleared.

Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>Initialization was performed</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>API is not open</td>
</tr>
</tbody>
</table>

Example

/* Initialize processes */
MCM_Startup();
### MCM_Run

**Syntax**

```c
ADMAPIENTRYW MCM_Run(void);
```

**Parameters**

None

**Description**

This function calls startup for all of the processes. The user startup function will be called by this function. Once startup is complete, the processes will be run. The user run function will be called repeatedly while the function is running. When an ESC key is received over the Debug port, the processes will be shutdown. The user shutdown function will be called at this time. The function will then exit.

**Return Value**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>Run was performed</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>API is not open</td>
</tr>
</tbody>
</table>

**Example**

```c
/* Run protocol drivers */
MCM_Run();
```
MCM_Shutdown

Syntax

ADMAPIENTRYW  MCM_Shutdown(void);

Parameters

None

Description

This function removes the protocol drivers and closes the database.

Return Value

| ADM_SUCCESS | Shutdown was performed |

Example

MCM_Shutdown();
exit(0);
6.3 Database Functions

MCM_DBGetBit

Syntax

ADMAPIENTRY MCM_DBGetBit(unsigned short offset);

Parameters

<table>
<thead>
<tr>
<th>offset</th>
<th>Bit offset into database</th>
</tr>
</thead>
</table>

Description

This function is used to read a bit from the database at a specified bit offset.

Return Value

<table>
<thead>
<tr>
<th>Requested bit</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short offset;
/* test bit at offset 16 */
offset = 16;
if(MCM_DBGetBit(offset))
    printf("bit is set");
else
    printf("bit is clear");
```
MCM_DBSetBit

Syntax

ADMAPIENTRY MCM_DBSetBit(unsigned short offset);

Parameters

offset Bit offset into database

Description

This function is used to set a bit to a 1 in the database at a specified bit offset.

Return Value

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short offset;
/* set bit at offset 16 to 1 */
offset = 16;
MCM_DBSetBit(offset);
```
MCM_DBClearBit

Syntax

ADMAPIENTRY MCM_DBClearBit(unsigned short offset);

Parameters

| offset          | Bit offset into database |

Description

This function is used to clear a bit to a 0 in the database at a specified bit offset.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short offset;
// clear bit at offset 16 to 0 */
offset = 16;
MCM_DBClearBit(offset);
```
MCM_DBGetByte

Syntax
ADMAPIENTRYC MCM_DBGetByte(unsigned short offset);

Parameters
offset  Byte offset into database

Description
This function is used to read a byte from the database at a specified byte offset.

Return Value
Requested byte

Example
unsigned short offset;
char c;
/* get byte value at byte offset 1000 (register 500) */
offset = 1000;
c = MCM_DBGetByte(offset);
MCM_DBSetByte

Syntax

ADMAPIENTRY MCM_DBSetByte(unsigned short offset, const char val);

Parameters

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
<td>Byte offset into database</td>
</tr>
<tr>
<td>Val</td>
<td>Value to be written to the database</td>
</tr>
</tbody>
</table>

Description

This function is used to write a byte to the database at a specified byte offset.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short offset;
const char val;
/* write 25 to byte 1000 (register 500) */
offset = 1000;
val = 25;
MCM_DBSetByte(offset, val);
```
MCM_DBGetWord

Syntax

ADMAPIENTRY MCM_DBGetWord(unsigned short offset);

Parameters

offset  Word offset into database

Description

This function is used to read a word from the database at a specified word offset.

Return Value

Requested word

Example

unsigned short offset;
int i;
i = MCM_DBGetWord(offset);
MCM_DBSetWord

Syntax

ADMAPIENTRY MCM_DBSetWord(unsigned short offset, const short val);

Parameters

<table>
<thead>
<tr>
<th>offset</th>
<th>Word offset into database</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Value to be written to the database</td>
</tr>
</tbody>
</table>

Description

This function is used to write a word to the database at a specified word offset.

Return Value

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short offset;
const short val;
/* write 300 to register 1000 */
offset = 1000;
val = 300;
MCM_DBSetWord(offset, val);
```
MCM_DBGetLong

Syntax

ADMAPIENTRYL MCM_DBGetLong(unsigned short offset);

Parameters

<table>
<thead>
<tr>
<th>offset</th>
<th>Long int offset into database</th>
</tr>
</thead>
</table>

Description

This function is used to read a long int from the database at a specified offset.

Return Value

Requested long int

Example

unsigned short offset;
long l;
/* get long value at long register offset 1000 (register 2000) */
offset = 2000;
l = MCM_DBGetLong(offset);
MCM_DBSetLong

Syntax
ADMAPIENTRY MCM_DBSetLong(unsigned short offset, const long val);

Parameters

<table>
<thead>
<tr>
<th>offset</th>
<th>Long int offset into database</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Value to be written to the database</td>
</tr>
</tbody>
</table>

Description
This function is used to write a long int to the database at a specified offset.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example
unsigned short offset;
const long val;
/* write 100000 to long register 1000 (register 2000) */
offset = 2000;
val = 100000L;
MCM_DBSetLong(offset, val);
MCM_DBGetFloat

Syntax
ADMAPIENTRYF  MCM_DBGetFloat(unsigned short offset);

Parameters
offset        float offset into database

Description
This function is used to read a floating-point number from the database at a specified float offset.

Return Value
Requested floating-point number.

Example
unsigned short offset;
float f;
/* read float from float register 1000 (register 2000) */
offset = 2000;
f = MCM_DBGetFloat(offset);
MCM_DBSetFloat

Syntax

ADMAPIENTRY MCM_DBSetFloat(unsigned short offset, const float val);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>float offset into database</td>
</tr>
<tr>
<td>val</td>
<td>Value to be written to the database</td>
</tr>
</tbody>
</table>

Description

This function is used to write a floating-point number to the database at a specified float offset.

Return Value

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short ofset;
const float val;
/* write 25.3 to float register 200 (register 400) */
offset = 400;
val = 25.3;
MCM_DBSetFloat(offset, val);
```
MCM_DBGetDFloat

Syntax

ADMAPIENTRYD MCM_DBGetDFloat(unsigned short offset);

Parameters

| offset | double float offset into database |

Description

This function is used to read a double floating-point number from the database at a specified double float offset.

Return Value

Requested double floating-point number

Example

unsigned short offset;
double d;
/* get double value at double offset 1000 (register 2000) */
offset = 2000;
d = MCM_DBGetDFloat(offset);
MCM_DBSetDFloat

Syntax
ADMAPIENTRY MCM_DBSetDFloat(unsigned short offset, const double val);

Parameters
<table>
<thead>
<tr>
<th>offset</th>
<th>double float offset into database</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Value to be written to the database</td>
</tr>
</tbody>
</table>

Description
This function is used to write a double floating-point number to the database at a specified double float offset.

Return Value
| ADM_SUCCESS                         | No errors were encountered |
| ADM_ERR_NOT_OPEN                    | Database is not open       |
| ADM_ERR_REG_RANGE                   | Database register is out of range |

Example
unsigned short offset;
const double val;
/* write 300.8 to double offset 100 (register 200) */
offset = 200;
val = 300.8;
MCM_DBSetDFloat(offset, val);
MCM_DBGetBytes

Syntax

```c
ADMAPIENTRY MCM_DBGetBytes(unsigned short offset, const unsigned short count, char* pBytes);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Character offset into database where the buffer starts</td>
</tr>
<tr>
<td>count</td>
<td>Number of characters to retrieve</td>
</tr>
<tr>
<td>pBytes</td>
<td>String buffer to receive characters</td>
</tr>
</tbody>
</table>

Description

This function is used to copy a number of characters in the database to a character buffer.

Return Value

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
<tr>
<td>ADM_ERR_MEMORY</td>
<td>Insufficient memory for database</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short offset;
const unsigned short char_count;
char *string_buff;
/* get 20 bytes from byte offset 200 (register 100) */
offset = 100;
char_count = 20;
MCM_DBGetBytes(offset, char_count, string_buff);
```
MCM_DBSetBytes

Syntax
ADMAPIENTRY MCM_DBSetBytes(unsigned short offset, const unsigned short count, const char* pBytes);

Parameters
<table>
<thead>
<tr>
<th>offset</th>
<th>Character offset into database where the buffer starts</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Number of characters to write</td>
</tr>
<tr>
<td>pBytes</td>
<td>String buffer to copy characters from</td>
</tr>
</tbody>
</table>

Description
This function is used to copy a buffer of characters to the database.

Return Value
| ADM_SUCCESS     | No errors were encountered                           |
| ADM_ERR_NOT_OPEN| Database is not open                                 |
| ADM_ERR_REG_RANGE| Database register is out of range                  |

Example
unsigned short offset;
char *string_buff[] = {1,2,3,4,5};
/* set 5 bytes at byte offset 200 (register 100) */
offset = 100;
MCM_DBSetBytes(offset, 5, string_buff);
MCM_DBGetWords

Syntax

ADMAPIENTRY MCM_DBGetWords(unsigned short offset, const unsigned short count, unsigned short* pWords);

Parameters

offset          Character offset into database where the buffer starts
count           Number of integers to retrieve
pWords          Register buffer to receive integers

Description

This function is used to copy a buffer of registers in the database to a register buffer.

Return Value

ADM_SUCCESS     No errors were encountered
ADM_ERR_NOT_OPEN Database is not open
ADM_ERR_REG_RANGE Database register is out of range
ADM_ERR_MEMORY  Insufficient memory for database

Example

unsigned short reg_buff[20];
/* get 20 registers from offset 200 */
MCM_DBGetWords(200, 20, reg_buff);
MCM_DBSetWords

Syntax

ADMAPIENTRY MCM_DBSetWords(unsigned short offset, const unsigned short count, const unsigned short* pWords);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Character offset into database where the buffer starts</td>
</tr>
<tr>
<td>count</td>
<td>Number of integers to retrieve</td>
</tr>
<tr>
<td>pWords</td>
<td>Register buffer to receive integers</td>
</tr>
</tbody>
</table>

Description

This function is used to copy a buffer of registers to the database.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned short reg_buff[] = {1,2,3,4,5};
/* set 5 registers at offset 200 */
MCM_DBSetWords(200, 5, reg_buff);
```
MCM_DBGetString

Syntax

ADMAPIENTRY MCM_DBGetString(unsigned short offset, const unsigned short maxcount, char* str);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Character offset into database where the buffer starts</td>
</tr>
<tr>
<td>maxcount</td>
<td>Maximum number of characters to retrieve</td>
</tr>
<tr>
<td>str</td>
<td>String buffer to receive characters</td>
</tr>
</tbody>
</table>

Description

This function is used to copy a string from the database to a string buffer.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
<tr>
<td>ADM_ERR_MEMORY</td>
<td>Insufficient memory for database</td>
</tr>
<tr>
<td>ADM_ERR_DB_MAX_SIZE</td>
<td>maxcount is larger than database size</td>
</tr>
</tbody>
</table>

Example

```c
char string_buff[20];
/* get max of 20 bytes from offset 200 (register 100) */
MCM_DBGetString(100, 20, string_buff);
```
**MCM_DBSetString**

**Syntax**

```
ADMAPIENTRY MCM_DBSetString(unsigned short offset, const char* str);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Character offset into database where the buffer starts</td>
</tr>
<tr>
<td>str</td>
<td>String buffer to receive characters</td>
</tr>
</tbody>
</table>

**Description**

This function is used to copy a string to the database from a string buffer.

**Return Value**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>No errors were encountered</td>
</tr>
<tr>
<td>ADM_ERR_NOT_OPEN</td>
<td>Database is not open</td>
</tr>
<tr>
<td>ADM_ERR_REG_RANGE</td>
<td>Database register is out of range</td>
</tr>
<tr>
<td>ADM_ERR_MEMORY</td>
<td>Insufficient memory for database</td>
</tr>
</tbody>
</table>

**Example**

```c
char string_buff[] = {"abc"};
/* set bytes to offset 200 (register 100) */
MCM_DBSetString(100, string_buff);
```
MCM_DBGetIntPtr

Syntax
ADMAPIENTRYIP MCM_DBGetIntPtr(int offset);

Parameters

<table>
<thead>
<tr>
<th>Description</th>
<th>Word offset into database</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

- Returns NULL if not successful.
- Returns pointer to int if successful.

Example

```c
int i;
/* get the value from offset 100 using a pointer to the location */
i = *(MCM_DBGetIntPtr(100));
```
MCM_DBBitChanged

Syntax

ADMAPIENTRY MCM_DBBitChanged(int offset);

Parameters

<table>
<thead>
<tr>
<th>offset</th>
<th>Bit offset into database</th>
</tr>
</thead>
</table>

Description

This function is used to check to see if a bit has changed since the last call to MCM_DBBitChanged.

Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>Bit has changed</td>
</tr>
</tbody>
</table>

Example

```c
if(MCM_DBBitChanged(offset))
    printf("Bit has changed");
else
    printf("Bit is unchanged");
```
MCM_DBByteChanged

Syntax

ADMAPIENTRY MCM_DBByteChanged(unsigned short offset);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>Byte offset into database</td>
</tr>
</tbody>
</table>

Description

This function is used to check to see if a byte has changed since the last call to MCM_DBByteChanged.

Return Value

0 No change
1 Byte has changed

Example

if(MCM_DBByteChanged(offset))
printf("Byte has changed");
else
printf("Byte is unchanged");
MCM_DBChanged

Syntax

ADMAPIENTRY MCM_DBChanged(unsigned short offset);

Parameters

<table>
<thead>
<tr>
<th>offset</th>
<th>Word offset into database</th>
</tr>
</thead>
</table>

Description

This function is used to check to see if a register has changed since the last call to MCM_DBChanged.

Return Value

<table>
<thead>
<tr>
<th>0</th>
<th>No change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Register has changed</td>
</tr>
</tbody>
</table>

Example

/* test register 100 for change */
if(MCM_DBChanged(100))
printf("Data has changed");
else
printf("Data is unchanged");
MCM_DBLongChanged

Syntax

ADMAPIENTRY MCM_DBLongChanged(unsigned short offset);

Parameters

offset  long offset into database

Description

This function is used to check to see if a long int has changed since the last call to MCM_DBLongChanged.

Return Value

0  No change
1  long int has changed

Example

/* test long int 100 for change */
if(MCM_DBLongChanged(200))
  printf("Data has changed");
else
  printf("Data is unchanged");
MCM_DBFloatChanged

Syntax

ADMAPIENTRY MCM_DBFloatChanged(unsigned short offset);

Parameters

offset float offset into database

Description

This function is used to check to see if a float has changed since the last call to MCM_DBFloatChanged.

Return Value

0 No change
1 float has changed

Example

/* test float 100 for change */
if(MCM_DBFloatChanged(200))
printf("Data has changed");
else
printf("Data is unchanged");
MCM_DBDoubleChanged

Syntax

ADMAPIENTRY MCM_DBDoubleChanged(unsigned short offset);

Parameters

<table>
<thead>
<tr>
<th>offset</th>
<th>double offset into database</th>
</tr>
</thead>
</table>

Description

This function is used to check to see if a double has changed since the last call to MCM_DBDoubleChanged.

Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>double has changed</td>
</tr>
</tbody>
</table>

Example

/* test double 100 for change */
if (MCM_DBDoubleChanged(200))
printf("Data has changed");
else
printf("Data is unchanged");
6.4 Clock Functions

MCM_ClockGetHandle

Syntax
ADMAPIENTRY MCM_ClockGetHandle(void);

Parameters
None

Description
This function gets access to a clock. There approximately 300 clocks available to the user. This number depends on the number of MCM drivers and the system processes used.

Return Value

<table>
<thead>
<tr>
<th>ClockHandle</th>
<th>Handle for accessing clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>There are no clocks available.</td>
</tr>
</tbody>
</table>

Example

```c
int handle;
handle = MCM_ClockGetHandle();  // Get clock handle
```
MCM_ClockStart

Syntax

ADMAPIENTRY MCM_ClockStart(int ClockHandle, long MicroSecondCount);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClockHandle</td>
<td>The handle to the clock returned by MCM_ClockGetHandle</td>
</tr>
<tr>
<td>MicroSecondCount</td>
<td>The number of microseconds to run</td>
</tr>
</tbody>
</table>

Description

This function starts the clock timing for the period of MicroSecondCount.

Return Value

<table>
<thead>
<tr>
<th>ClockHandle</th>
<th>L</th>
</tr>
</thead>
</table>

Example

ClockStart(ClockHandle, 1000L); // Start clock timing for 1 millisecond
MCM_ClockCheck

Syntax

ADMAPIENTRY MCM_ClockCheck(int ClockHandle);  // returns true if clock running

Parameters

ClockHandle The handle to the clock returned by MCM_ClockGetHandle

Description

This function checks the clock to see if it has expired.

Return Value

<table>
<thead>
<tr>
<th>true</th>
<th>Clock is running.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>Clock has timed out.</td>
</tr>
</tbody>
</table>

Example

If(MCM_ClockCheck(ClockHandle) == false)
{
    printf("Clock timed out\n");
}

MCM_ClockGetValue

Syntax

ADMAPIENTRYL MCM_ClockGetValue(int ClockHandle);

Parameters

ClockHandle  The handle to the clock returned by MCM_ClockGetHandle

Description

This function gets the current microsecond value of the clock.

Return Value

The current long word microsecond value of the clock.

Example

long ClockValue;
ClockValue = MCM_ClockGetValue(ClockHandle);
### 6.5 Console Port Functions

**MCM_Send**

**Syntax**

```
ADMAPIENTRY MCM_Send(const char*p_Data);
```

**Parameters**

| p_Data | Pointer to text string to send |

**Description**

This function sends a string of text to the console (Debug port).

**Return Value**

Returns number of characters placed in the console buffer.

**Example**

```c
char text[] = "hello";
MCM_Send(text);
```
### MCM_GetKey

#### Syntax

```c
ADMAPIENTRY MCM_GetKey(char *Char);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char</td>
<td>pointer to char to hold key from console port</td>
</tr>
</tbody>
</table>

#### Description

This function will get a key from the console port if a key is waiting. If no key is waiting the function will exit without waiting for a key.

#### Return Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if no valid keypress</td>
</tr>
<tr>
<td>1</td>
<td>if valid keypress</td>
</tr>
</tbody>
</table>

#### Example

```c
char z = 0;
/* check for key press from console */
if(MCM_GetKey(&z) == 1)
{
    /* print out key received */
    printf("key: %c\n", z);
}
```
6.6 LED Functions

**MCM_LED_Set**

**Syntax**

```
ADMAPIENTRY MCM_LED_Set(unsigned short LED, int On);
```

**Parameters**

<table>
<thead>
<tr>
<th>LED</th>
<th>The LED to be controlled</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCM_LED_OFF</td>
<td>Fault, CFG, APP ERR, Port 0 ERR LED OFF</td>
</tr>
<tr>
<td>MCM_LED_FLT</td>
<td>Fault LED</td>
</tr>
<tr>
<td>MCM_LED_CFG</td>
<td>CFG LED</td>
</tr>
<tr>
<td>MCM_LED_APP</td>
<td>APP ERR LED</td>
</tr>
<tr>
<td>MCM_LED_P0</td>
<td>Port 0 ERR LED</td>
</tr>
<tr>
<td>MCM_LED_P1</td>
<td>Port 1 ERR LED</td>
</tr>
<tr>
<td>MCM_LED_P2</td>
<td>Port 2 ERR LED</td>
</tr>
<tr>
<td>MCM_LED_P3</td>
<td>Port 3 ERR LED</td>
</tr>
<tr>
<td>MCM_LED_POFF</td>
<td>Port 1 to 3 ERR LED OFF</td>
</tr>
<tr>
<td>On</td>
<td>On=ON, Off=OFF</td>
</tr>
</tbody>
</table>

**Description**

This function sets an LED to the desired on/off state.

**Return Value**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>LED was set to desired state</td>
</tr>
<tr>
<td>ADM_ERR_BADPARAM</td>
<td>Invalid LED designation</td>
</tr>
</tbody>
</table>

**Example**

```
MCM_LED_Set(MCM_LED_FLT, ON); // Set the Fault LED on
```
### 6.7 Serial Port Functions

**MCM_SendBytes**

**Syntax**

```c
ADMAPIENTRY MCM_SendBytes(int port, unsigned char *data, int len);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port to use to send bytes (0 to 3)</td>
</tr>
<tr>
<td>data</td>
<td>pointer to buffer holding bytes to send</td>
</tr>
<tr>
<td>len</td>
<td>number of bytes to send</td>
</tr>
</tbody>
</table>

**Description**

MCM_SendBytes puts bytes in the serial port state machine to be sent out of the port. The state machine handles hardware handshaking and the internal data analyzer for the port.

**Return Value**

- Number of bytes sent
- `ADM_ERR_NOACCESS` port value is out of range

**Example**

```c
unsigned char TxBuff[] = {1,2,3,4,5};
MCM_SendBytes(0, TxBuff, 5);
```
MCM_SendBytesDirect

Syntax

ADMAPIENTRY MCM_SendBytesDirect(int port, unsigned char *data, int len);

Parameters

<table>
<thead>
<tr>
<th>port</th>
<th>port to use to send bytes (0 to 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>pointer to buffer holding bytes to send</td>
</tr>
<tr>
<td>len</td>
<td>number of bytes to send</td>
</tr>
</tbody>
</table>

Description

MCM_SendBytesDirect sends a number of bytes out of the port without using the serial port state machine. Hardware handshaking has to be handled by the application.

Return Value

<table>
<thead>
<tr>
<th>Number of bytes sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_ERR_NOACCESS port value is out of range</td>
</tr>
</tbody>
</table>

Example

```c
unsigned char TxBuff[] = {1,2,3,4,5};
MCM_SendBytesDirect(0, TxBuff, 5);
```
MCM_SetRTS

Syntax

ADMAPIENTRY MCM_SetRTS(int port, int state);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port for which RTS is to be changed (0 to 3)</td>
</tr>
<tr>
<td>state</td>
<td>desired RTS state</td>
</tr>
</tbody>
</table>

Description

This function allows the state of the RTS signal to be controlled. state specifies desired state of the RTS signal. Valid values for state are ON and OFF.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>RTS was set to desired state</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>port value is out of range</td>
</tr>
</tbody>
</table>

Example

```c
int rc;
rc = MCM_SetRTS(COM1, ON);
if (rc != ADM_SUCCESS)
    printf("SetRTS failed\n ");
```
**MCM_SetDTR**

**Syntax**

```c
ADMAPIENTRY MCM_SetDTR(int port, int state);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port for which DTR is to be changed (0 to 3)</td>
</tr>
<tr>
<td>state</td>
<td>desired RTS state</td>
</tr>
</tbody>
</table>

**Description**

This function allows the state of the DTR signal to be controlled. state is the desired state of the DTR signal. Valid values for state are ON and OFF.

**Return Value**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>RTS was set to desired state</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>port value is out of range</td>
</tr>
</tbody>
</table>

**Example**

```c
int rc;
rc = MCM_SetDTR(COM1, ON);
if (rc != ADM_SUCCESS)
    printf("SetDTR failed\n");
```
MCM_GetCTS

Syntax
ADMAPIENTRY MCM_GetCTS(int port);

Parameters
| port | port for which CTS is requested (0 to 3) |

Description
This function allows the state of the CTS signal to be determined.

Return Value
The state of CTS line
| ADM_ERR_NOACCESS | port value is out of range |

Example
```c
int state;
state = MCM_GetCTS(0);
if(state == ON)
    printf("CTS is ON\n");
else
    printf("CTS is OFF\n");
```
MCM_GetByte

Syntax

```c
ADMAPIENTRY MCM_GetByte(int port);
```

Parameters

| port | port from which data is to be received |

Description

This function is used to receive a single character from a serial port. All data received from a port is queued after reception from the serial port. Therefore, some delay may occur between the time a character is received across the serial line and the time the character is returned by MCM_GetByte.

Return Value

| Byte from receive buffer of serial port |
|ADM_ERR_NOACCESS             | port value is out of range |

Example

```c
int ch;
ch = MCM_GetByte(0);
```
MCM_GetAsciiString

Syntax

ADMAPIENTRY MCM_GetAsciiString(int port, unsigned char *buffer, char endChar, int *count);

Parameters

- **port**: port from which data is to be received
- **buffer**: buffer to hold string
- **endChar**: character marking the end of the string (ex. LF)
- **count**: max number of bytes to get

Description

This function is used to get a string terminated by `endChar` from a serial port. All data received from a port is queued after reception from the serial port. Therefore, some delay may occur between the time a character is received across the serial line and the time the character is returned by `MCM_GetAsciiString`.

Return Value

- Number of bytes in string
- ADM_ERR_NOACCESS: port value is out of range

Example

```c
#define LF 0x0A
unsigned char RxBuff[21];
MCM_GetAsciiString(0, RxBuff, LF, 20);
```
**MCM_GetDataString**

**Syntax**

```c
ADMAPIENTRY MCM_GetDataString(int port, unsigned char *buffer, int count);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port from which data is to be received</td>
</tr>
<tr>
<td>buffer</td>
<td>buffer to hold string</td>
</tr>
<tr>
<td>count</td>
<td>max number of bytes to get</td>
</tr>
</tbody>
</table>

**Description**

This function is used to receive an array of bytes from a serial port. All data received from a port is queued after reception from the serial port. Therefore, some delay may occur between the time a character is received across the serial line and the time the character is returned by `MCM_GetDataString`.

**Return Value**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bytes in string</td>
<td>port value is out of range</td>
</tr>
</tbody>
</table>

**Example**

```c
unsigned char RxBuff[21];
MCM_GetDataString(0, RxBuff, 20);
```
MCM_BytesInTransmitBuffer

Syntax
ADMAPIENTRY MCM_BytesInTransmitBuffer(int port);

Parameters

| port        | port whose transmit buffer is to be queried |

Description

MCM_BytesInTransmitBuffer returns the number of characters in the transmit queue that are waiting to be sent. Since data sent to a port is queued before transmission across a serial port, the application may need to determine if all characters have been transmitted or how many characters remain to be transmitted.

Return Value

- Returns number of bytes in buffer
- ADM_ERR_NOACCESS port value is out of range

Example

```c
int count;
count = MCM_BytesInTransmitBuffer(COM2)
if(count == 0)
  printf("All chars read\n");
else
  printf("%d characters remaining\n",count);
```
MCM_BytesInReceiveBuffer

Syntax

ADMAPIENTRY MCM_BytesInReceiveBuffer(int port);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port whose receive buffer is to be queried</td>
</tr>
</tbody>
</table>

Description

MCM_BytesInReceiveBuffer returns the number of characters in the receive queue that are waiting to be read. Since data received from a port is queued after reception from a serial port, the application may need to determine if all characters have been read or how many characters remain to be read.

Return Value

Returns number of bytes in buffer

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>port value is out of range</td>
</tr>
</tbody>
</table>

Example

```c
int count;
count = MCM_BytesInReceiveBuffer(COM2)
if(count == 0)
    printf("All chars read\n");
else
    printf("%d characters remaining\n", count);
```
MCM_FlushTransmitBuffer

Syntax

```c
ADMAPIENTRY MCM_FlushTransmitBuffer(int port);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port whose transmit data is to be purged</td>
</tr>
</tbody>
</table>

Description

MCM_FlushTransmitBuffer deletes all data waiting in the transmit queue. The data is discarded and is not transmitted.

Return Value

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>the data was purged successfully</td>
</tr>
<tr>
<td>ADM_ERR_BADPARAM</td>
<td>Com port specified is out of range</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>the comport has not been opened</td>
</tr>
</tbody>
</table>

Example

```c
if (MCM_FlushTransmitBuffer (COM1) == ADM_SUCCESS)
    printf("Transmit Data purged.\n");
```
MCM_FlushReceiveBuffer

Syntax

ADMAPIENTRY MCM_FlushReceiveBuffer(int port)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>The port whose receive data is to be purged.</td>
</tr>
</tbody>
</table>

Description

MCM_FlushReceiveBuffer deletes all data waiting in the receive queue. The data is discarded and is no longer available for reading.

Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM_SUCCESS</td>
<td>the data was purged successfully</td>
</tr>
<tr>
<td>ADM_ERR_BADPARAM</td>
<td>Com port specified is out of range</td>
</tr>
<tr>
<td>ADM_ERR_NOACCESS</td>
<td>the comport has not been opened</td>
</tr>
</tbody>
</table>

Example

```c
if (MCM_FlushReceiveBuffer (COM1) == ADM_SUCCESS)
    printf("Receive Data purged.\n");
```
7 Reference

In This Chapter

- Product Specifications .......................................................... 103
- MCM Database Definition .......................................................... 106
- Configuration Data ................................................................. 106
- Modbus Error and Status Data Area Addresses .......................... 109
- Error Codes ............................................................................. 112
- LED Indicators ........................................................................ 114

7.1 Product Specifications

7.1.1 General Specifications

The MCM4-ADM4 module acts as an input/output module between the ADM4 network and the user protocol. The data transfer from the user protocol is asynchronous from the actions on the MODBUS network. A 1000 to 10,000-word register space in the module exchanges data between the user protocol and the MODBUS network.

Some of the general specifications include:

- Support for the storage and transfer of up to 10,000 registers
- Module memory usage that is completely user definable
- Four ports to emulate any combination of MODBUS master or slave device and user protocol
- Configurable MCM port parameters include:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>RTU or ASCII</td>
</tr>
<tr>
<td>Baud Rate</td>
<td>110 to 115,200 (up to 38,400 on Port 0)</td>
</tr>
<tr>
<td>Parity</td>
<td>None, Odd and Even</td>
</tr>
<tr>
<td>Data Bits</td>
<td>5 to 8</td>
</tr>
<tr>
<td>Stop Bits</td>
<td>1 or 2</td>
</tr>
<tr>
<td>RTS On and Off Timing</td>
<td>0 to 65535 milliseconds</td>
</tr>
<tr>
<td>Minimum Response Delay</td>
<td>0 to 65535 milliseconds</td>
</tr>
<tr>
<td>Use of CTS Modem Line</td>
<td>Yes or No</td>
</tr>
<tr>
<td>Floating-Point Support</td>
<td></td>
</tr>
</tbody>
</table>
Slave Functional Specifications

The MCM4-ADM4 module accepts MODBUS commands from an attached MODBUS master unit. A port configured as a virtual MODBUS slave permits a remote master to interact with all data contained in the module. This data can be derived from other MODBUS slave devices on the network through a master port or from the user protocol.

Master Functional Specifications

A port configured as a virtual MODBUS master device on the MCM4-ADM4 module will actively issue MODBUS commands to other nodes on the MODBUS network. One hundred commands are supported on each port. Additionally, the master ports have an optimized polling characteristic that will poll slaves with communication problems less frequently.

7.1.2 Hardware Specifications and Equipment Ratings

<table>
<thead>
<tr>
<th>Type</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Ports</td>
<td></td>
</tr>
<tr>
<td>Serial Port Cable (DB-9M Connector)</td>
<td>A mini-DIN to DB-9M cable is included with the unit</td>
</tr>
<tr>
<td>Debug</td>
<td>RS-232/422/485 - jumper selectable</td>
</tr>
<tr>
<td></td>
<td>DB-9M connector</td>
</tr>
<tr>
<td></td>
<td>No hardware handshaking</td>
</tr>
<tr>
<td>Serial Port Isolation</td>
<td>2500V RMS port-to-port isolation per UL 1577.</td>
</tr>
<tr>
<td></td>
<td>3000V DC min. port to ground and port to logic power isolation.</td>
</tr>
<tr>
<td>Serial Port Protection</td>
<td>RS-485/422 port interface lines TVS diode protected at +/- 27V standoff voltage.</td>
</tr>
<tr>
<td></td>
<td>RS-232 port interface lines fault protected to +/- 36V power on, +/- 40V power off.</td>
</tr>
<tr>
<td>General Signal Connections</td>
<td>For highest EMI/RFI immunity, signal connections shall use the interconnect cable as specified by the protocol in use. Interconnect cable shields shall be connected to earth ground.</td>
</tr>
<tr>
<td>Example Interconnect Cable</td>
<td>Rockwell Automation RIO and DH+ protocols use Belden 9463 type shielded cable or equivalent. Schneider Electric Modbus Plus protocol uses Belden 9841 type shielded cable or equivalent.</td>
</tr>
<tr>
<td>Types</td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td></td>
</tr>
<tr>
<td>External Power</td>
<td>Supply Voltage: 24 VDC nominal, 18 to 32 VDC allowed</td>
</tr>
<tr>
<td></td>
<td>Supply Current: 500 mA (max. at 24 VDC)</td>
</tr>
<tr>
<td></td>
<td>Center terminal shall be connected to earth ground.</td>
</tr>
<tr>
<td>Power Connector</td>
<td>+/-GND screw connectors, rated for 24 AWG to 14 AWG tinned copper, stranded, insulated wire. Use 2.5 mm screwdriver blade.</td>
</tr>
<tr>
<td>Environmental</td>
<td></td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>-20 to 60°C (-4 to 140°F)</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>-40 to 85°C (-40 to 185°F)</td>
</tr>
</tbody>
</table>
## 7.1.3 Ports

**Serial (Mini DIN 8)**

The ProLinx module serial ports are capable of supporting several protocols as either Master or Slave on up to four ports. Each port is individually configurable, thereby providing a great deal of flexibility.

When configured as a Master port, the serial ports can be used to continuously interface with slave devices over a serial Communication Interface (RS-232, RS-422, or RS-485). Each Master port supports 100 user-defined commands that determine the read/write commands issued to each slave attached to the port.

In addition, the module may be configured to place slave devices that are not responding to commands at a lower priority. If the module recognizes that a slave device has failed to respond to a message after the user-defined retry count, it marks the slave as "in communication failure" and sets the error delay counter to the user-specified value.

Alternatively, the serial port can be configured to emulate a slave device.

Every gateway module is shipped with one Mini DIN 8 to DB-9 conversion cable per configurable port on the module.

<table>
<thead>
<tr>
<th>Type</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Humidity</td>
<td>5% to 95% (non-condensing)</td>
</tr>
<tr>
<td>Shock (Unpackaged)</td>
<td>Operational - Pending testing</td>
</tr>
<tr>
<td></td>
<td>Non-operational - Pending testing</td>
</tr>
<tr>
<td>Vibration (Unpackaged)</td>
<td>Pending testing</td>
</tr>
<tr>
<td>Dimensions</td>
<td>3.71H x 6.06 W x 4.70 D inches</td>
</tr>
<tr>
<td></td>
<td>94.2 H x 153.9 W x 119.3 D mm</td>
</tr>
<tr>
<td>Weight (max.)</td>
<td>Pending</td>
</tr>
<tr>
<td>Altitude</td>
<td>Shipping and storage: up to 3000 m (9843 Feet). Operation: up to 2000 m (6562 Feet).</td>
</tr>
<tr>
<td>Corrosion Immunity</td>
<td>Rated in accordance with IEC 68.</td>
</tr>
<tr>
<td>Pollution Degree</td>
<td>Rated to pollution degree 2. Equipment may be exposed to non-conductive pollution. Occasional conductivity due to condensation may occur. Equipment may not function properly until condensation evaporates.</td>
</tr>
<tr>
<td>Overvoltage Category</td>
<td>Rated to over voltage category I. Reverse polarity, improper lead connection, and/or voltages outside of the range of 18 VDC to 36 VDC applied to the power connector may damage the equipment.</td>
</tr>
</tbody>
</table>
7.2 MCM Database Definition

This section contains a listing of the internal database of the MCM4-ADM4 module. This information can be used to interface other devices to the data contained in the module.

<table>
<thead>
<tr>
<th>Content</th>
<th>Offset from top of user data</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCM Port 0 Status</td>
<td>0</td>
</tr>
<tr>
<td>MCM Port 1 Status</td>
<td>10</td>
</tr>
<tr>
<td>MCM Port 2 Status</td>
<td>20</td>
</tr>
<tr>
<td>MCM Port 3 Status</td>
<td>30</td>
</tr>
<tr>
<td>MCM Port 0 Configuration</td>
<td>40</td>
</tr>
<tr>
<td>MCM Port 1 Configuration</td>
<td>70</td>
</tr>
<tr>
<td>MCM Port 2 Configuration</td>
<td>100</td>
</tr>
<tr>
<td>MCM Port 3 Configuration</td>
<td>130</td>
</tr>
<tr>
<td>MCM Port 0 Commands</td>
<td>160</td>
</tr>
<tr>
<td>MCM Port 1 Commands</td>
<td>960</td>
</tr>
<tr>
<td>MCM Port 2 Commands</td>
<td>1760</td>
</tr>
<tr>
<td>MCM Port 3 Commands</td>
<td>2560</td>
</tr>
</tbody>
</table>

The User Data area holds data collected from other nodes on the network (master read commands) or data received from the processor (write blocks). Detailed definition of the miscellaneous status data area can be found in Misc. Status.

Definition of the configuration data areas can be found in the data definition section of this document and in Configuration Data Definition.

7.3 Configuration Data

This section contains listings of the MCM4-ADM4 module's database that are related to the module's configuration. This data is available to any node on the network and is read from the config file when the module first initializes. Additionally, this section contains the miscellaneous status data and command control database layout.

7.3.1 MCM Port x Configuration

<table>
<thead>
<tr>
<th>Offset</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Enable</td>
<td>This parameter defines if this MODBUS port will be used. If the parameter is set to 0, the port is disabled. A value of 1 enables the port.</td>
</tr>
<tr>
<td>1</td>
<td>Type</td>
<td>This parameter specifies if the port will emulate a MODBUS master device (0), a MODBUS slave device without pass-through (1), or a MODBUS slave device with unformatted pass-through (2), or a MODBUS slave device with formatted pass-through and data swapping (3).</td>
</tr>
<tr>
<td>Offset</td>
<td>Content</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>Float Flag</td>
<td>This flag specifies if the floating-point data access functionality is to be implemented. If the float flag is set to 1, MODBUS functions 3, 6, and 16 will interpret floating-point values for registers as specified by the two following parameters.</td>
</tr>
<tr>
<td>3</td>
<td>Float Start</td>
<td>This parameter defines the first register of floating-point data. All requests with register values greater than or equal to this value will be considered floating-point data requests. This parameter is only used if the Float Flag is enabled.</td>
</tr>
<tr>
<td>4</td>
<td>Float Offset</td>
<td>This parameter defines the start register for floating-point data in the internal database. This parameter is only used if the Float Flag is enabled.</td>
</tr>
<tr>
<td>5</td>
<td>Protocol</td>
<td>This parameter specifies the MODBUS protocol to be used on the port. Valid protocols are: 0 = MODBUS RTU and 1 = MODBUS ASCII.</td>
</tr>
<tr>
<td>6</td>
<td>Baud Rate</td>
<td>This is the baud rate to be used on the port. Enter the baud rate as a value. For example, to select 19K baud, enter 19200. Valid entries are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 38400, 576, and 115.</td>
</tr>
<tr>
<td>7</td>
<td>Parity</td>
<td>This is the parity code to be used for the port. Values are None, Odd, Even.</td>
</tr>
<tr>
<td>8</td>
<td>Data Bits</td>
<td>This parameter sets the number of data bits for each word used by the protocol. Valid entries for this field are 5 through 8.</td>
</tr>
<tr>
<td>9</td>
<td>Stop Bits</td>
<td>This parameter sets the number of stop bits to be used with each data value sent. Valid entries are 1 and 2.</td>
</tr>
<tr>
<td>10</td>
<td>RTS On</td>
<td>This parameter sets the number of milliseconds to delay after RTS is asserted before the data will be transmitted. Valid values are in the range of 0 to 65535 milliseconds.</td>
</tr>
<tr>
<td>11</td>
<td>RTS Off</td>
<td>This parameter sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal will be set low. Valid values are in the range of 0 to 65535.</td>
</tr>
<tr>
<td>12</td>
<td>Minimum Response Time</td>
<td>This parameter specifies the minimum number of milliseconds to delay before responding to a request message. This pre-send delay is applied before the RTS on time. This may be required when communicating with slow devices.</td>
</tr>
<tr>
<td>13</td>
<td>Use CTS Line</td>
<td>This parameter specifies if the CTS modem control line is to be used. If the parameter is set to 0, the CTS line will not be monitored. If the parameter is set to 1, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire).</td>
</tr>
<tr>
<td>14</td>
<td>Slave ID</td>
<td>This parameter defines the virtual MODBUS slave address for the internal database. All requests received by the port with this address are processed by the module. Verify that each device has a unique address on a network. Valid range for this parameter is 1 to 255 (247 on some networks).</td>
</tr>
<tr>
<td>15</td>
<td>Bit in Offset</td>
<td>This parameter specifies the offset address in the internal MODBUS database to use with network requests for MODBUS Function 2 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.</td>
</tr>
</tbody>
</table>
### Offset | Content | Description
--- | --- | ---
16 | Word in Offset | This parameter specifies the offset address in the internal MODBUS database to use with network request for MODBUS function 4 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.

17 | Out in Offset | This parameter specifies the offset address in the internal MODBUS database to use with network requests for MODBUS function 1, 5, or 15 commands. For example, if the value is set to 100, an address request of 0 will correspond to register 100 in the database.

18 | Holding Reg Offset | This parameter specifies the offset address in the internal MODBUS database to use with network requests for MODBUS function 3, 6, or 16 commands. For example, if a value of 50 is entered, a request for address 0 will correspond to the register 50 in the database.

19 | Command Count | This parameter specifies the number of commands to be processed by the MODBUS master port.

20 | Minimum Command Delay | This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries.

21 | Command Error Pointer | This parameter sets the address in the internal MODBUS database where the command error will be placed. If the value is set to -1, the data will not be transferred to the database. The valid range of values for this parameter is -1 to 4999.

22 | Response Timeout | This parameter represents the message response timeout period in 1-millisecond increments. This is the time that a port configured as a master will wait before re-transmitting a command if no response is received from the addressed slave. The value is set depending upon the communication network used and the expected response time of the slowest device on the network.

23 | Retry Count | This parameter specifies the number of times a command will be retried if it fails. If the master port does not receive a response after the last retry, the slave devices communication will be suspended on the port for Error Delay Counter scans.

24 | Error Delay Counter | This parameter specifies the number of polls to skip on the slave before trying to re-establish communications. After the slave fails to respond, the master will skip commands to be sent to the slave the number of times entered in this parameter.

25 | Spare |

26 | Spare |

27 | Spare |

28 | Spare |

29 | Spare |

### 7.3.2 MCM Port x Commands

<table>
<thead>
<tr>
<th>Offset</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 7</td>
<td>Command #1</td>
<td>This set of registers contains the parameters for the first command in the master command list. Refer to the data object section of the documentation.</td>
</tr>
<tr>
<td>8 to 15</td>
<td>Command #2</td>
<td>Command #2 data set</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>782 to 789</td>
<td>Command #100</td>
<td>Command #100 data set</td>
</tr>
</tbody>
</table>
### 7.4 Modbus Error and Status Data Area Addresses

Modbus error and status data are stored in registers based on port number. Starting register addresses are shown in the following table.

<table>
<thead>
<tr>
<th>Modbus Port</th>
<th>Starting Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4400</td>
</tr>
<tr>
<td>1</td>
<td>4800</td>
</tr>
<tr>
<td>2</td>
<td>5200</td>
</tr>
<tr>
<td>3</td>
<td>5600</td>
</tr>
</tbody>
</table>

*Note:* None of the addresses are available in the Modbus address range. In order to view them via a Modbus request, they must be moved into the 0 to 3999 address range using the Data Map section of the configuration file. For additional information on how to move data within the gateway's internal database, see Moving Data.

#### 7.4.1 Modbus Ports: Error and Status

The serial port (Modbus Master/Slave) Error and Status Data areas are discussed in this section.

The data area is initialized with zeros whenever the gateway is restarted. This occurs during a cold-start (power-on), reset (reset push-button pressed) or a warm-boot operation (commanded from a debug menu or after downloading a new configuration). The addresses listed are for Port 0 only; but the format is the same for each port. The start address for each port is given in the previous section, Modbus Error and Status Data Area Addresses (page 109).

<table>
<thead>
<tr>
<th>Example Internal Database Address</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4400</td>
<td>0</td>
<td>Number of Command Requests</td>
</tr>
<tr>
<td>4401</td>
<td>1</td>
<td>Number of Command Responses</td>
</tr>
<tr>
<td>4402</td>
<td>2</td>
<td>Number of Command Errors</td>
</tr>
<tr>
<td>4403</td>
<td>3</td>
<td>Number of Requests</td>
</tr>
<tr>
<td>4404</td>
<td>4</td>
<td>Number of Responses</td>
</tr>
<tr>
<td>4405</td>
<td>5</td>
<td>Number of Errors Sent</td>
</tr>
<tr>
<td>4406</td>
<td>6</td>
<td>Number of Errors Received</td>
</tr>
<tr>
<td>4407</td>
<td>7</td>
<td>Configuration Error Code</td>
</tr>
<tr>
<td>4408</td>
<td>8</td>
<td>Current Error/Index</td>
</tr>
<tr>
<td>4409</td>
<td>9</td>
<td>Last Error/Index</td>
</tr>
</tbody>
</table>

Refer to the following Error Codes (page 112) section to interpret the status/error codes present in the data area.
7.4.2 Master Port: Command List Errors

The individual command errors for each Master port are returned to the address locations specified in the following table. Each port can have up to 100 commands configured. Each configured command will use one word of these data areas to store a value representing the execution status from the most recent command execution attempt.

<table>
<thead>
<tr>
<th>Modbus Port</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4410 to 4509</td>
</tr>
<tr>
<td>1</td>
<td>4810 to 4909</td>
</tr>
<tr>
<td>2</td>
<td>5210 to 5309</td>
</tr>
<tr>
<td>3</td>
<td>5610 to 5709</td>
</tr>
</tbody>
</table>

The first word in the defined register location contains the status/error code for the first command in the port’s Command List. Successive words in the Command Error List are associated with corresponding commands in the list.

Refer to Error Codes (page 112) to interpret the status/error codes present in this data area.

Port 0 Command Error List Layout

The addresses listed are for Port 0 only; but the format is the same for each port. The start address for each port is given in the previous section, Master Port: Command List Errors (page 110).

<table>
<thead>
<tr>
<th>Internal Database Address (Example)</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4410</td>
<td>0</td>
<td>Command #0 Error Status</td>
</tr>
<tr>
<td>4411</td>
<td>1</td>
<td>Command #1 Error Status</td>
</tr>
<tr>
<td>4412</td>
<td>2</td>
<td>Command #2 Error Status</td>
</tr>
<tr>
<td>4413</td>
<td>3</td>
<td>Command #3 Error Status</td>
</tr>
<tr>
<td>4414</td>
<td>4</td>
<td>Command #4 Error Status</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4507</td>
<td>97</td>
<td>Command #97 Error Status</td>
</tr>
<tr>
<td>4508</td>
<td>98</td>
<td>Command #98 Error Status</td>
</tr>
<tr>
<td>4509</td>
<td>99</td>
<td>Command #99 Error Status</td>
</tr>
</tbody>
</table>

Note that the values in the Command Error List tables are initialized to zero (0) at power-up, cold boot, and warm boot. If a command executes successfully, the value in the associated register will remain at zero (0), indicating no command error was detected. Any non-zero value in this table indicates the corresponding command experienced an error. The Error Code (page 112) shown will provide valuable troubleshooting information.
The data in this table is dynamic. It is updated each time a command is executed. Therefore, if the command fails once and succeeds on the next attempt, the Error Code from the previously failed attempt will be replaced with zero and lost. Error Codes are not archived in the gateway's database. To see if the port has experienced an error since the most recent restart and what the most recently occurring error was, if any, you can check the Last Error/Index (page 109).

### 7.4.3 Master Port: Modbus Slave List Status

The slave status list contains the current poll status of each slave device on a Master port. Slaves attached to a Master port can have one of three states.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The slave has not defined in the command list for the Master port and will not be polled from the Command List.</td>
</tr>
<tr>
<td>1</td>
<td>The slave is configured to be polled by the Master port and the most recent communication attempt was successful.</td>
</tr>
<tr>
<td>2</td>
<td>The Master port has failed to communicate with the slave device. Communication with the slave is suspended for a user defined period based on the scanning of the command list.</td>
</tr>
</tbody>
</table>

Slaves are defined to the system when the gateway loads the Master Command List during start-up and initialization. Each slave defined will be set to a state value of 1 in this initial step. If the Master port fails to communicate with a slave device (timeout expired on a command, retries failed), the Master will set the state of the slave to a value of 2 in this status table. This suspends communication with the slave device for a user-specified Error Delay Count.

When the Master first suspends polling of a particular slave, it creates a Error Delay Counter for this slave address and set the value in that counter equal to the Error Delay Counter parameter in the configuration file. Then, each time a command in the list is scanned that has the address of a suspended slave, the delay counter value for that slave will be decremented. When the value reaches zero, the slave state will be set to 1. This will re-enable polling of the slave.

The first word in the defined register locations contains the status code for slave node address 1. Each successive word in the list is associated with the next node in sequence, up to slave node address 255.

The individual Slave List Status for each Modbus port are returned to the address locations specified in the following table.

<table>
<thead>
<tr>
<th>Modbus Port</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4510 to 4764</td>
</tr>
<tr>
<td>1</td>
<td>4910 to 5164</td>
</tr>
<tr>
<td>2</td>
<td>5310 to 5564</td>
</tr>
<tr>
<td>3</td>
<td>5710 to 5965=4</td>
</tr>
</tbody>
</table>
Port 0 Slave List Status Layout

The addresses listed are for Port 0 only; but the format is the same for each port. The start address for each port is given in the previous section, Master Port: Modbus Slave List Status. (page 111)

<table>
<thead>
<tr>
<th>Internal Database Address (Example)</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4510</td>
<td>0</td>
<td>Slave #1 Status</td>
</tr>
<tr>
<td>4511</td>
<td>1</td>
<td>Slave #2 Status</td>
</tr>
<tr>
<td>4512</td>
<td>2</td>
<td>Slave #3 Status</td>
</tr>
<tr>
<td>4513</td>
<td>3</td>
<td>Slave #4 Status</td>
</tr>
<tr>
<td>4514</td>
<td>4</td>
<td>Slave #5 Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the values in the Slave List Status tables are initialized to zero (0) at power-up, cold boot and during warm boot.

7.5 Error Codes

These are error codes that are part of the Modbus protocol or are extended codes unique to this gateway.

7.5.1 Modbus Error Codes

These error codes are generated or returned on both the Master and slave ports. These codes are the standard Modbus errors.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Illegal Function</td>
</tr>
<tr>
<td>2</td>
<td>Illegal Data Address</td>
</tr>
<tr>
<td>3</td>
<td>Illegal Data Value</td>
</tr>
<tr>
<td>4</td>
<td>Failure in Associated Device</td>
</tr>
<tr>
<td>5</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>6</td>
<td>Busy, Rejected Message</td>
</tr>
</tbody>
</table>

7.5.2 gateway Communication Error Codes

These gateway-specific error codes are also returned from the command polling process and stored in the Command Error List memory area.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>CTS modem control line not set before transmit</td>
</tr>
<tr>
<td>-2</td>
<td>Timeout while transmitting message</td>
</tr>
<tr>
<td>-11</td>
<td>Timeout waiting for response after request</td>
</tr>
<tr>
<td>253</td>
<td>Incorrect slave address in response</td>
</tr>
<tr>
<td>254</td>
<td>Incorrect function code in response</td>
</tr>
<tr>
<td>255</td>
<td>Invalid CRC/LRC value in response</td>
</tr>
</tbody>
</table>
7.5.3 Command List Error Codes

These command-specific error codes are detected during initial command list loading at power-up or gateway reset and are stored in the Command Error List memory region.

<table>
<thead>
<tr>
<th>CODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-41</td>
<td>Invalid enable code</td>
</tr>
<tr>
<td>-42</td>
<td>Internal address &gt; maximum address</td>
</tr>
<tr>
<td>-43</td>
<td>Invalid node address (&lt;0 or &gt; 255)</td>
</tr>
<tr>
<td>-44</td>
<td>Count parameter set to 0</td>
</tr>
<tr>
<td>-45</td>
<td>Invalid function code</td>
</tr>
<tr>
<td>-46</td>
<td>All parameters set to 0</td>
</tr>
<tr>
<td>-47</td>
<td>All parameters set to -1</td>
</tr>
</tbody>
</table>

7.5.4 Modbus Configuration Error Word

Modbus Configuration Error Word errors are stored in protocol-specific registers. The following table lists the Port/Register address configuration.

<table>
<thead>
<tr>
<th>Modbus Port</th>
<th>Configuration Error Word Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4407</td>
</tr>
<tr>
<td>1</td>
<td>4807</td>
</tr>
<tr>
<td>2</td>
<td>5207</td>
</tr>
<tr>
<td>3</td>
<td>5607</td>
</tr>
</tbody>
</table>

A register containing a code indicates a problem with the configuration. The following table lists the codes, a description of the problem, and parameters to correct the error condition within the configuration file.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0001</td>
<td>Invalid Enabled parameter (Yes or No)</td>
</tr>
<tr>
<td>1</td>
<td>0x0002</td>
<td>Invalid RS-Interface parameter (0 to 2)</td>
</tr>
<tr>
<td>2</td>
<td>0x0004</td>
<td>Invalid Type (Master or Slave)</td>
</tr>
<tr>
<td>3</td>
<td>0x0008</td>
<td>Invalid Protocol (RTU or ASCII)</td>
</tr>
<tr>
<td>4</td>
<td>0x0010</td>
<td>Invalid Baud Rate</td>
</tr>
<tr>
<td>5</td>
<td>0x0020</td>
<td>Invalid Parity (None, Odd, Even)</td>
</tr>
<tr>
<td>6</td>
<td>0x0040</td>
<td>Invalid Data Bits (7 or 8 bits)</td>
</tr>
<tr>
<td>7</td>
<td>0x0080</td>
<td>Invalid Stop Bits (1 or 2)</td>
</tr>
<tr>
<td>8</td>
<td>0x0100</td>
<td>Invalid Use CTS Line (Yes or No)</td>
</tr>
<tr>
<td>9</td>
<td>0x0200</td>
<td>Retry Count Invalid (0 to 10)</td>
</tr>
<tr>
<td>10</td>
<td>0x0400</td>
<td>Invalid Floating Point Data:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Float Flag not Yes or No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Float Start less than 0 or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Float Offset is Invalid</td>
</tr>
<tr>
<td>11</td>
<td>0x0800</td>
<td>Invalid Internal Slave ID (1 to 255) (Slave Only)</td>
</tr>
</tbody>
</table>
### 7.6 LED Indicators

LED indicators provide a means of monitoring the operation of the system and individual ports. There are extremely useful for troubleshooting. The gateway provides LEDs to help monitor each port. In addition, system configuration errors, application errors, and fault indications are all indicated by LEDs, providing alerts to possible problems.

#### 7.6.1 Common gateway LEDs

<table>
<thead>
<tr>
<th>LED</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Off</td>
<td>Power is not connected to the power terminals or source is insufficient to properly power the gateway (800mA at 24vdc minimum required)</td>
</tr>
<tr>
<td></td>
<td>Green Solid</td>
<td>Power is connected to the power terminals. Verify that the other LEDs for operational and functional status come on briefly after power-up (check for burned-out LEDs).</td>
</tr>
<tr>
<td>Fault</td>
<td>Off</td>
<td>Normal operation.</td>
</tr>
<tr>
<td></td>
<td>Red Solid</td>
<td>A critical error has occurred. Program executable has failed or has been user-terminated and is no longer running. Press Reset p/b or cycle power to clear error. If not, use the Debug procedures described later in this manual.</td>
</tr>
<tr>
<td>Cfg</td>
<td>Off</td>
<td>Normal operation.</td>
</tr>
<tr>
<td></td>
<td>Amber Solid</td>
<td>The unit is in configuration mode. The configuration file is currently being downloaded or, after power-up, is being read, the unit is implementing the configuration values, and initializing the hardware. This will occur during power cycle, or after pressing the reset button. It also occurs after a cold/warm boot command is received.</td>
</tr>
<tr>
<td>Err</td>
<td>Off</td>
<td>Normal operation.</td>
</tr>
<tr>
<td></td>
<td>Flashing</td>
<td>An error condition has been detected and is occurring on one of the application serial ports. Check configuration and troubleshoot for communication errors.</td>
</tr>
<tr>
<td></td>
<td>Solid Red</td>
<td>This error flag is cleared at the start of each command attempt (master/client) or on each receipt of data (slave/adapter/server); so, if this condition exists, it indicates a large number of errors are occurring in the application (due to bad configuration) or on one or more ports (network communication failures).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0x1000</td>
<td>Invalid Entry for Register Offset Data (Slave Only)</td>
</tr>
<tr>
<td>13</td>
<td>0x2000</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>0x4000</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>0x8000</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### 7.6.2 LEDs for Serial Ports

ProLinx gateways may have as many as five (5) serial ports. Each of these serial ports has two LEDs indicating status.

<table>
<thead>
<tr>
<th>LED</th>
<th>Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug - ACT</td>
<td>Off</td>
<td>No activity on the port.</td>
</tr>
<tr>
<td>Port 0 - ACT</td>
<td>Green</td>
<td>The port is actively transmitting or receiving data</td>
</tr>
<tr>
<td>Port 1 - ACT</td>
<td>Flash</td>
<td></td>
</tr>
<tr>
<td>Port 2 - ACT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port 3 - ACT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debug - ERR</td>
<td>Off</td>
<td>Normal state. When off and Port Active led is indicating</td>
</tr>
<tr>
<td>Port 0 - ERR</td>
<td></td>
<td>activity, there are no communication errors</td>
</tr>
<tr>
<td>Port 1 - ERR</td>
<td>RED</td>
<td>Activity on this LED indicates communication errors occurring.</td>
</tr>
<tr>
<td>Port 2 - ERR</td>
<td>On Solid or Flashing</td>
<td>Activity on this LED indicates communication errors occurring. To determine the exact error, connect the Debug terminal to the Debug port and use the built-in Diagnostic Menus.</td>
</tr>
</tbody>
</table>
8 DOS 6 XL Reference Manual

The DOS 6 XL Reference Manual makes reference to compilers other than Digital Mars C++ or Borland Compilers. The PLX-ADM and ADMNET modules only support Digital Mars C++ and Borland C/C++ Compiler Version 5.02. References to other compilers should be ignored.
9 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.

Originator
A client that establishes a connection path to a target.

Output Image
Table of output data sent to nodes on the network.

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
10 Support, Service & Warranty

In This Chapter

- Contacting Technical Support ................................................................. 123
- Warranty Information ................................................................................ 124

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.
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](http://www.prosoft-technology.com/warranty).

Documentation is subject to change without notice.
# Index

## A
- ADM API • 41
- ADM API Architecture • 41
- ADM API Files • 43
- ADM API Functions • 45
- ADM Functional Blocks • 41
  - adm_prot.c • 43
  - adm_prot.h • 43
- ADM_RegisterProtocol • 49
- ADM_RegisterUserFunc • 51
- All ProLinx® Products • 2
- API • 119
- API Libraries • 39
- Application Development Function Library - ADM API • 45

## B
- Backplane • 119
- BIOS • 119
- Building an Existing Borland C++ 5.02 ADM Project • 27
- Building an Existing Digital Mars C++ 8.49 ADM Project • 17
- Byte • 119

## C
- Cable Connections • 12
- Calling Convention • 39
- CIP • 119
- Clock Functions • 84
- Command List Error Codes • 113
- Common gateway LEDs • 114
- Configuration Data • 106
- Configuring Borland C++ 5.02 • 27
- Configuring Digital Mars C++ 8.49 • 17
- Connecting Power to the Unit • 11
- Connection • 119
- Console Port Functions • 88
- Consumer • 119
- Contacting Technical Support • 123
- Controller • 119
- Core Functions • 47
- Creating a New Borland C++ 5.02 ADM Project • 29
- Creating a New Digital Mars C++ 8.49 ADM Project • 19
- Debugging Strategies • 37
- Development Tools • 41
- DLL • 120
- DOS 6 XL Reference Manual • 7, 117
- Downloading Files to the Module • 34
- Downloading the Sample Program • 17, 27
- Embedded I/O • 120
- Error Codes • 109, 110, 112
- ExplicitMsg • 120
- gateway Communication Error Codes • 112
- General Specifications • 103
- Hardware • 37
- Hardware Specifications and Equipment Ratings • 104
- Header File • 40
- HSC • 120
- Important Installation Instructions • 2
- Input Image • 120
- Introduction • 7
- LED Functions • 90
- LED Indicators • 114
- LEDs for Serial Ports • 115
- Library • 120
- LIMITED WARRANTY • 124
- Linked Library • 120
- Local I/O • 120
- Long • 120
- Master Functional Specifications • 104
- Master Port
  - Command List Errors • 110
    - Modbus Slave List Status • 111, 112
- MCM Database Definition • 106
- MCM Port x Commands • 108
- MCM Port x Configuration • 106
- MCM_BytesInReceiveBuffer • 100
- MCM_BytesInTransmitBuffer • 99
- MCM_ClockCheck • 86
- MCM_ClockGetHandle • 84
- MCM_ClockGetValue • 87
- MCM_ClockStart • 85
- MCM(DBBitChanged • 78
- MCM_DBByteChanged • 79
- MCM_DBChanged • 80
- MCM_DBClearBit • 60
- MCM_DBDoubleChanged • 83
- MCM_DBFloatChanged • 82
- MCM_DBGetBit • 58
- Database • 42
- Database Functions • 58
- DB9 to Mini-DIN Adaptor (Cable 09) • 15
- Debug / Status Port • 41
MCM4_DBGetByte • 61
MCM_DBGetBytes • 71
MCM_DBGetDFloat • 69
MCM_DBGetFloat • 67
MCM_DBGetIntPtr • 77
MCM_DBGetLong • 65
MCM_DBGetString • 75
MCM_DBGetWord • 63
MCM_DBGetWords • 73
MCM_DBLongChanged • 81
MCM_DBSetBit • 59
MCM_DBSetByte • 62
MCM_DBSetBytes • 72
MCM_DBSetDFloat • 70
MCM_DBSetFloat • 68
MCM_DBSetLong • 66
MCM_DBSetString • 76
MCM_DBSetWord • 64
MCM_DBSetWords • 74
MCM_FlushReceiveBuffer • 102
MCM_FlushTransmitBuffer • 101
MCM_GetAsciiString • 97
MCM_GetByte • 96
MCM_GetCTS • 95
MCM_GetDataString • 98
MCM_GetKey • 89
MCM_InstallConsole • 54
MCM_InstallDatabase • 53
MCM_LED_Set • 90
MCM_Open • 47
MCM_RegisterProtocol • 48
MCM_Run • 56
MCM_Send • 88
MCM_SendBytes • 91
MCM_SendBytesDirect • 92
MCM_Shutdown • 94
MCM_Startup • 93
MCM4 ADM.C • 42
MCMADM.H • 42
Modbus Configuration Error Word • 113
Modbus Error and Status Data Area Addresses • 109
Modbus Error Codes • 112
Modbus Ports
  Error and Status • 109, 111
  Module • 120
Mounting the gateway on the DIN-rail • 11
Multithreading Considerations • 40
Mutex • 121
MVI Suite • 121
MVI46 • 121
MVI56 • 121
MVI69 • 121
MVI71 • 121
MVI94 • 121
Operating System • 7
Originator • 121
Output Image • 121
Package Contents • 9
Pinouts • 2, 12, 15
Port 0 Command Error List Layout • 110
Port 0 Slave List Status Layout • 112
Ports • 105
Preparing the PLX-MCM4 Module • 9
Producer • 121
Product Specifications • 103
Programming the Module • 37
PTO • 121
PTQ Suite • 121
Reference • 103
RS-232 • 12
  Modern Connection • 12
  Null Modern Connection (Hardware Handshaking) • 13
  Null Modern Connection (No Hardware Handshaking) • 13
RS-232 Configuration/Debug Port • 14
RS-422 • 15
RS-485 • 14
RS-485 and RS-422 Tip • 15
RS-485 Programming Note • 37
Sample Code • 40
Scanner • 122
Serial (Mini DIN 8) • 105
Serial Communications • 42
Serial Port Functions • 91
Setting Port 0 Configuration Jumpers • 10
Setting Up Your Compiler • 17
Setting Up Your Development Environment • 17
Side-nect • 122
Slave Functional Specifications • 104
Software • 38
Support, Service & Warranty • 123
Target • 122
Theory of Operation • 41
Thread • 122
Understanding the ADM API • 39
Word • 122
Your Feedback Please • 3