

MAGTEK COMMUNICATION PROTOCOL (MCP) DRIVER REFERENCE MANUAL

Manual Part Number 99875164-Rev 3

OCTOBER 2004

MAGTEK[®]

REGISTERED TO ISO 9001:2000

1710 Apollo Court
Seal Beach, CA 90740
Phone: (562) 546-6400
FAX: (562) 546-6301
Technical Support: (651) 415-6800
www.magtek.com

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of MagTek, Inc.

MagTek is a registered trademark of MagTek, Inc.

REVISIONS

Rev Number	Date	Notes
1	24 Mar 00	Initial Release
2	21 May 03	Front Matter: added ISO line to logo, changed Tech Support phone number, and replaced 90-day warranty with generic software license agreement.
3	7 Oct 04	Throughout: deleted all references to Windows 95 and parallel port; Sec 2, Operating Systems: added descriptions of Windows 2000, XP, Me; Sec 3: completely revised. Section 6: Deleted.

SOFTWARE LICENSE AGREEMENT

IMPORTANT: YOU SHOULD CAREFULLY READ ALL THE TERMS, CONDITIONS AND RESTRICTIONS OF THIS LICENSE AGREEMENT BEFORE INSTALLING THE SOFTWARE PACKAGE. YOUR INSTALLATION OF THE SOFTWARE PACKAGE PRESUMES YOUR ACCEPTANCE OF THE TERMS, CONDITIONS, AND RESTRICTIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THESE TERMS, CONDITIONS, AND RESTRICTIONS, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ASSOCIATED DOCUMENTATION TO ABOVE ADDRESS ATTENTION: CUSTOMER SUPPORT.

TERMS, CONDITIONS AND RESTRICTIONS

MagTek, Incorporated (the "Licensor") owns and has the right to distribute the described software and documentation, collectively referred to as the "Software".

LICENSE: Licensor grants you (the "Licensee") the right to use the Software in conjunction with MagTek products.

LICENSEE MAY NOT COPY, MODIFY OR TRANSFER THE SOFTWARE IN WHOLE OR IN PART EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. Licensee may not decompile, disassemble or in any other manner attempt to reverse engineer the Software. Licensee shall not tamper with, bypass or alter any security features of the software or attempt to do so.

TRANSFER: Licensee may not transfer the Software or license to the Software to another party without prior written authorization of the Licensor. If Licensee transfers the Software without authorization, all rights granted under this Agreement are automatically terminated.

COPYRIGHT: The Software is copyrighted. Licensee may not copy the Software except for archival purposes or to load for execution purposes. All other copies of the Software are in violation of this Agreement.

TERM: This Agreement is in effect as long as Licensee continues the use of the Software. The Licensor also reserves the right to terminate this Agreement if Licensee fails to comply with any of the terms, conditions or restrictions contained herein. Should Licensor terminate this Agreement due to Licensee's failure to comply, Licensee agrees to return the Software to Licensor. Receipt of returned Software by the Licensor shall mark the termination.

LIMITED WARRANTY: Licensor warrants to the Licensee that the disk(s) or other media on which the Software is recorded to be free from defects in material or workmanship under normal use. THE SOFTWARE IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Because of the diversity of conditions and PC hardware under which the Software may be used, Licensor does not warrant that the Software will meet Licensee specifications or that the operation of the Software will be uninterrupted or free of errors.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE. Licensee's sole remedy in the event of a defect in material or workmanship is expressly limited to replacement of the Software disk(s) if applicable.

GOVERNING LAW: If any provision of this Agreement is found to be unlawful, void or unenforceable, that provision shall be removed from consideration under this Agreement and will not affect the enforceability of any of the remaining provisions. This Agreement shall be governed by the laws of the State of California and shall insure to the benefit of MagTek, Incorporated, its successors or assigns.

ACKNOWLEDGMENT: LICENSEE ACKNOWLEDGES THAT HE HAS READ THIS AGREEMENT, UNDERSTANDS ALL OF ITS TERMS, CONDITIONS AND RESTRICTIONS AND AGREES TO BE BOUND BY THEM. LICENSEE ALSO AGREES THAT THIS AGREEMENT SUPERSEDES ANY AND ALL, VERBAL AND WRITTEN, COMMUNICATIONS BETWEEN LICENSOR AND LICENSEE OR THEIR ASSIGNS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

QUESTIONS REGARDING THIS AGREEMENT SHOULD BE ADDRESSED IN WRITING TO MAGTEK, INCORPORATED, ATTENTION: CUSTOMER SUPPORT, AT THE ABOVE ADDRESS OR E-MAILED TO support@magtek.com

TABLE OF CONTENTS

SECTION 1. OVERVIEW	1
SECTION 2. SYSTEM REQUIREMENTS	3
COMPUTER SYSTEM	3
OPERATING SYSTEM.....	3
SECTION 3. INSTALLATION	5
INSTALLING THE MCP DRIVER.....	5
CHANGING THE MCP DRIVER:.....	17
REMOVING THE MCP DRIVERS.....	30
DEVICE INSTANCE MANAGEMENT	33
Device Instance Overview	33
MCPCFG Utility Overview	33
MCPCFG Command Summary.....	34
Adding a Device Instance from Command Line – RS-232.....	35
Adding a Device Instance from Command Line – USB	35
Adding a Device Instance from Windows Based Application (WINDOWS 2000, XP) – RS-232.....	36
Adding a Device Instance from Windows Based Application (WINDOWS 2000, XP) – USB.....	36
Modifying a Device Instance Properties from Windows Based Application (WINDOWS 2000, XP) – RS-232 and USB.....	36
Removing a Device Instance from Windows Based Application (WINDOWS 2000, XP) – RS232... ..	36
Removing a Device Instance from Windows Based Application (WINDOWS 2000, XP) – USB	36
Removing a Device Instance from a Command Line – RS-232	37
Removing a Device Instance from a Command Line – USB	37
Displaying the List of Device Instances from Command Line.....	37
Displaying Device Instance Properties from Command Line.....	37
Modifying Device Instance Properties from Command Line	38
Modifying Device Instance Properties From Windows Based Application.....	39
Restarting the MCP Driver	39
Stopping the MCP Driver From Command Line.....	39
Windows Configuration Utility.....	40
DEVICE INSTANCE PROPERTIES.....	41
Generic Properties.....	41
Serial Port Properties	43
SECTION 4. TRACE LOG	45
SECTION 5. APPLICATION PROGRAMMABLE INTERFACE	47
TYPICAL OPERATION.....	47
DEVICE CHANNEL LIFE CYCLE.....	48
PROPERTIES.....	48
COMMANDS.....	49
NOTIFICATIONS	50
FUNCTIONS	51
Summary	51
MCPBUS Structure.....	52
McpEnum	56
McpOpen	57
McpClose.....	59
McpReset	60
McpGet.....	61
McpSet	63
McpCall.....	65
McpWait.....	67

CONSTANT DEFINITIONS	70
Operation Attributes.....	70
Property Types	70
Function Return Values.....	70
FUNCTION PROTOTYPES.....	72
INDEX	73

SECTION 1. OVERVIEW

The MCP driver is a kernel-mode driver that provides reliable communications between a user-mode application and an MCP device. It allows user-mode applications to access the features of MCP devices in a uniform way despite the variety of connection interfaces that the devices use. The driver includes a user-mode DLL that provides an easy-to-use interface for user mode applications, simplifying application development.

The driver is a Windows Driver Model (WDM) kernel mode driver (.SYS). It operates under the control of the Windows I/O manager and is configured as an Automatic Start driver. The driver is loaded at boot time by the operating system. The driver accesses the controlled devices through standard serial port support.

SECTION 2. SYSTEM REQUIREMENTS

COMPUTER SYSTEM

The driver requires a 133 MHz Intel Pentium based PC or better.

OPERATING SYSTEM

The driver is compatible with the following operating systems:

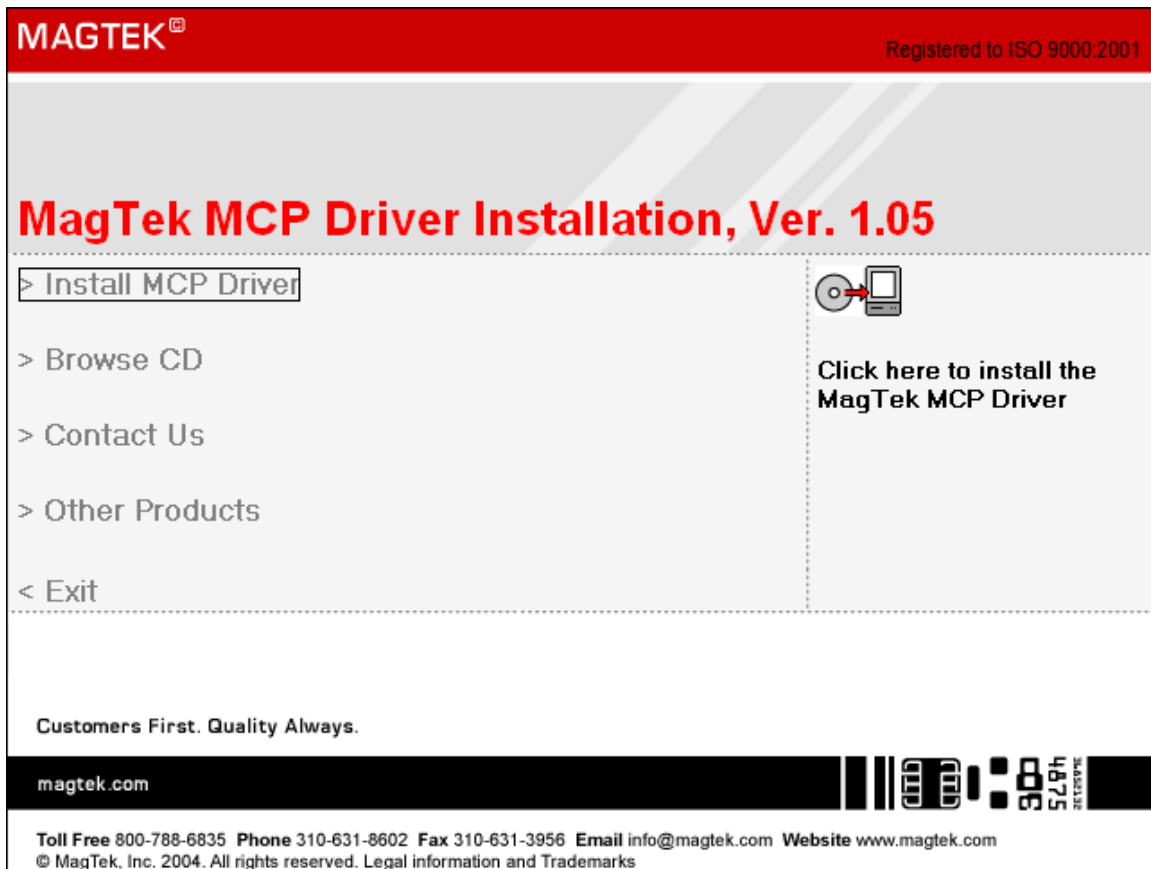
1. Windows 2000, XP
2. Windows NT 4.0
3. Windows 98, Me
4. Communication System

The MCP driver uses the standard operating system drivers for device communication. Under Windows 98 and Me, the driver uses the VCOMM client API in order to communicate through the serial ports. Under Windows NT, the MCP driver uses the standard serial port drivers. The MCP driver does not access the serial resources directly. This requires that the standard drivers for those communication ports provided by Windows are started and properly configured.

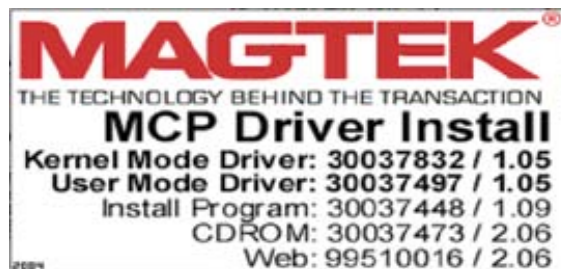
SECTION 3. INSTALLATION

INSTALLING THE MCP DRIVER

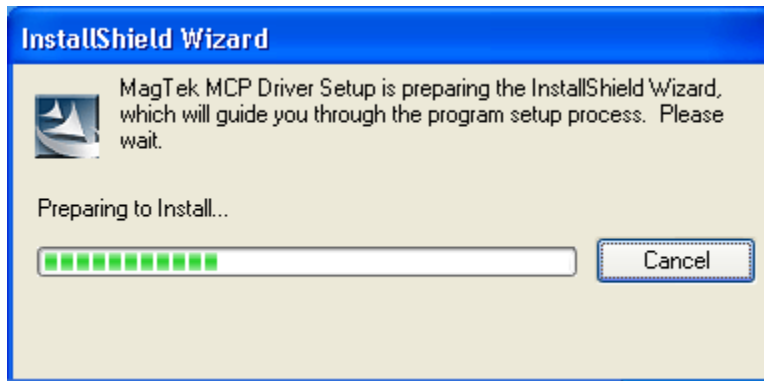
Insert the disk, P/N 30037473. If the program does not automatically start, navigate to the CD and run the AutoRun.exe program. A menu should shortly appear:



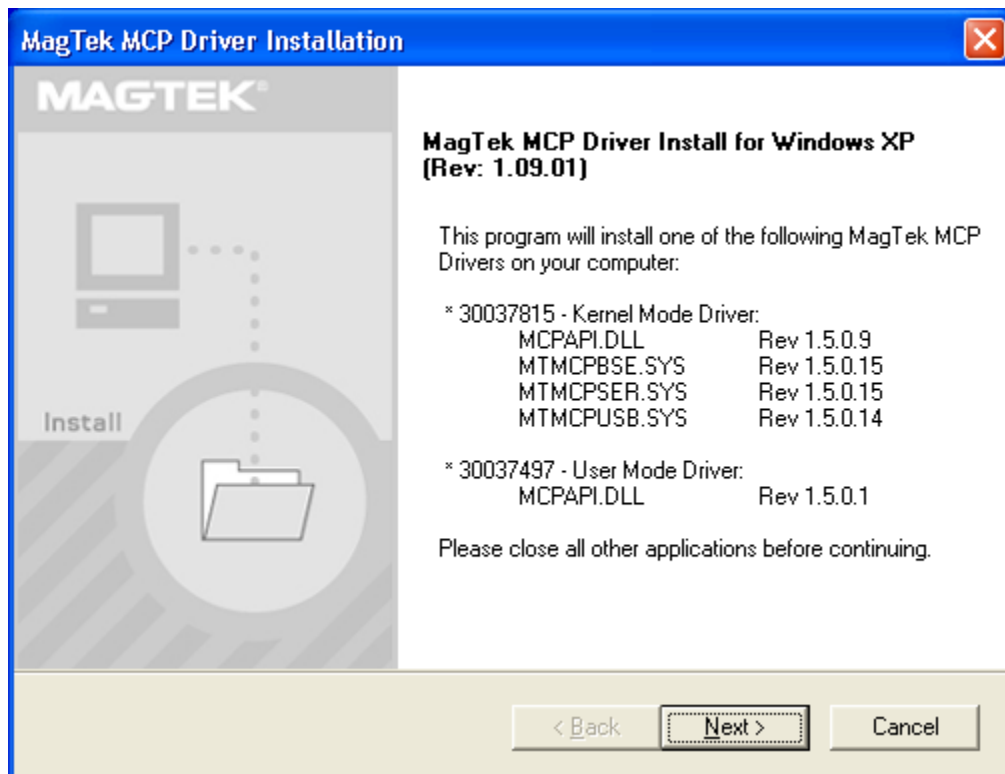
Click "Install/Change MCP Driver". Shortly, the MCP Installation program will start running. You will see a splash screen:



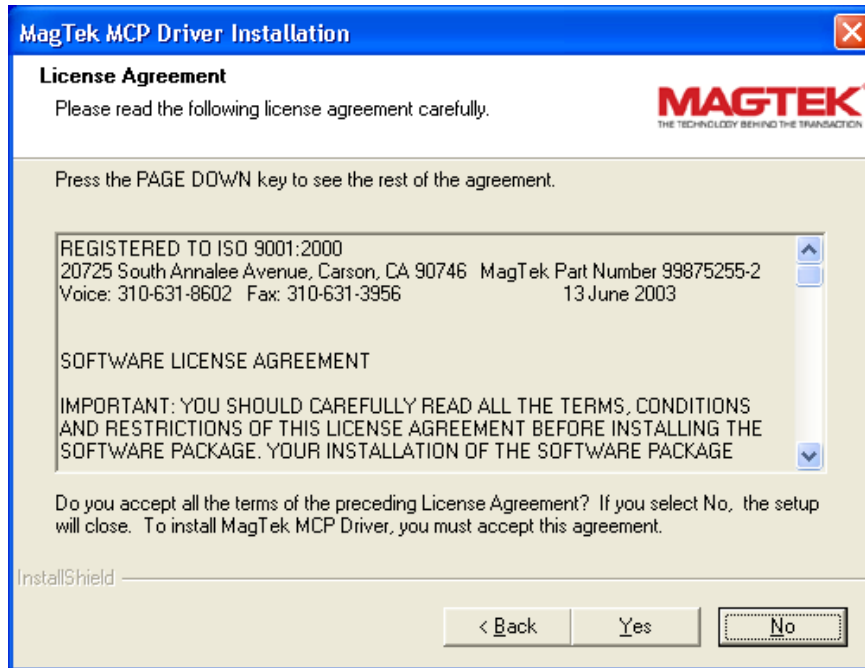
Next, this window will appear.



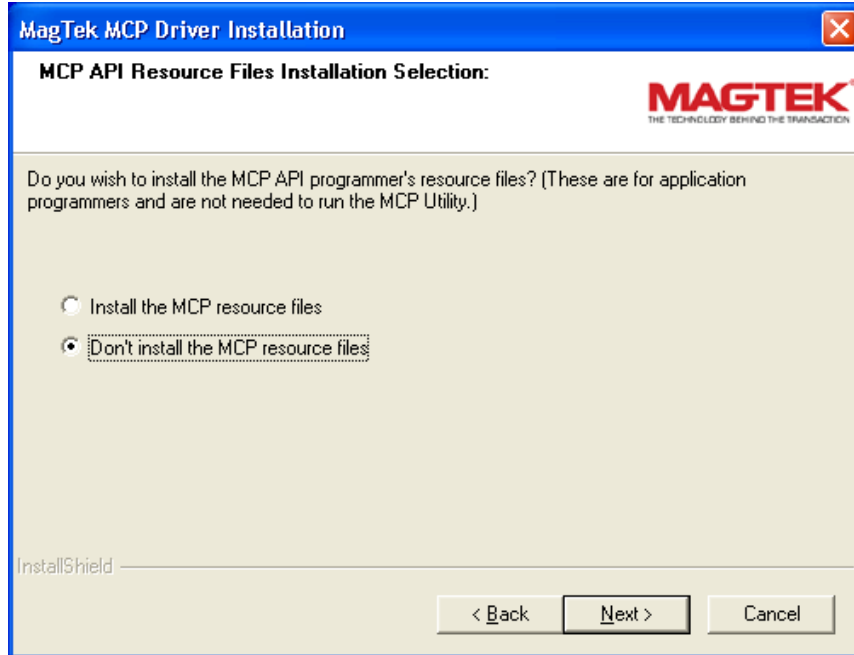
After this, the MCP Installation welcome screen will appear. Click the “Next >” button.



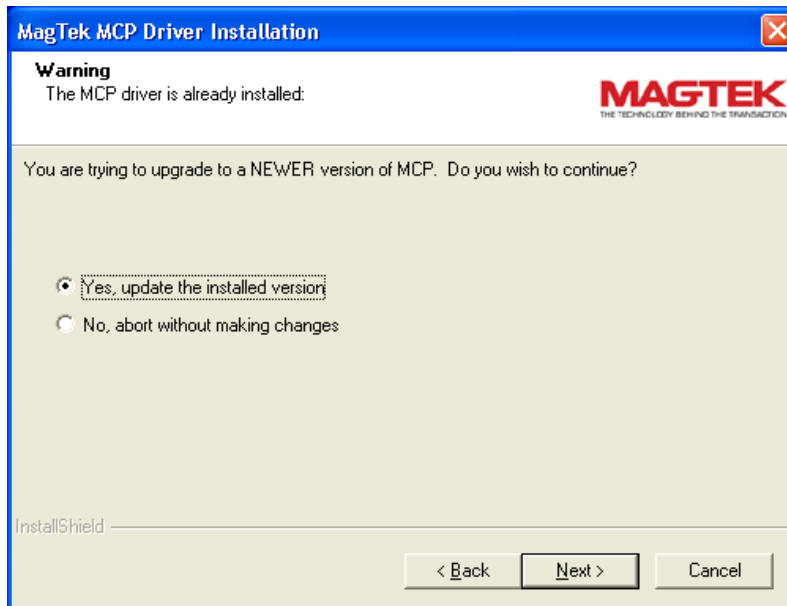
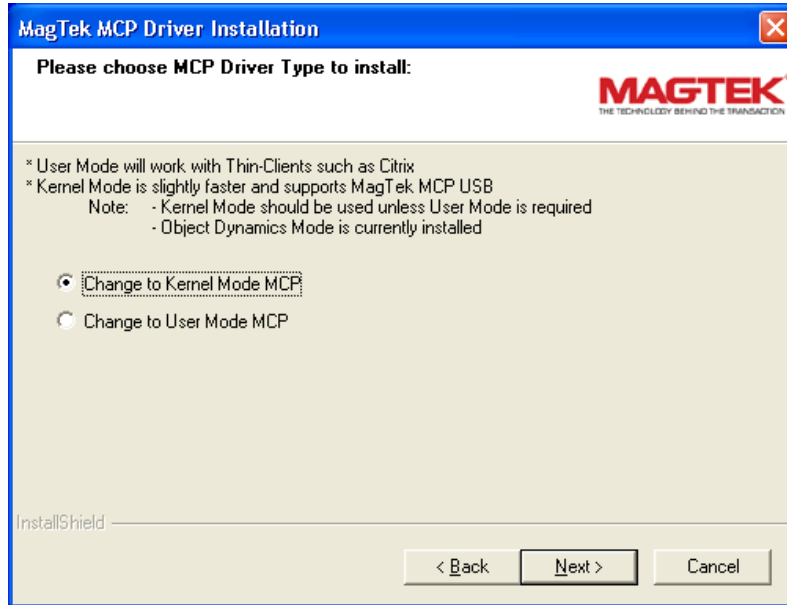
Read the End User Licensing Agreement, and then click the “Next >” button if you accept it.



Choose whether you wish to install MCP resources. These are only recommended if you are developing MCP applications and are not required to use the MCP Drivers.



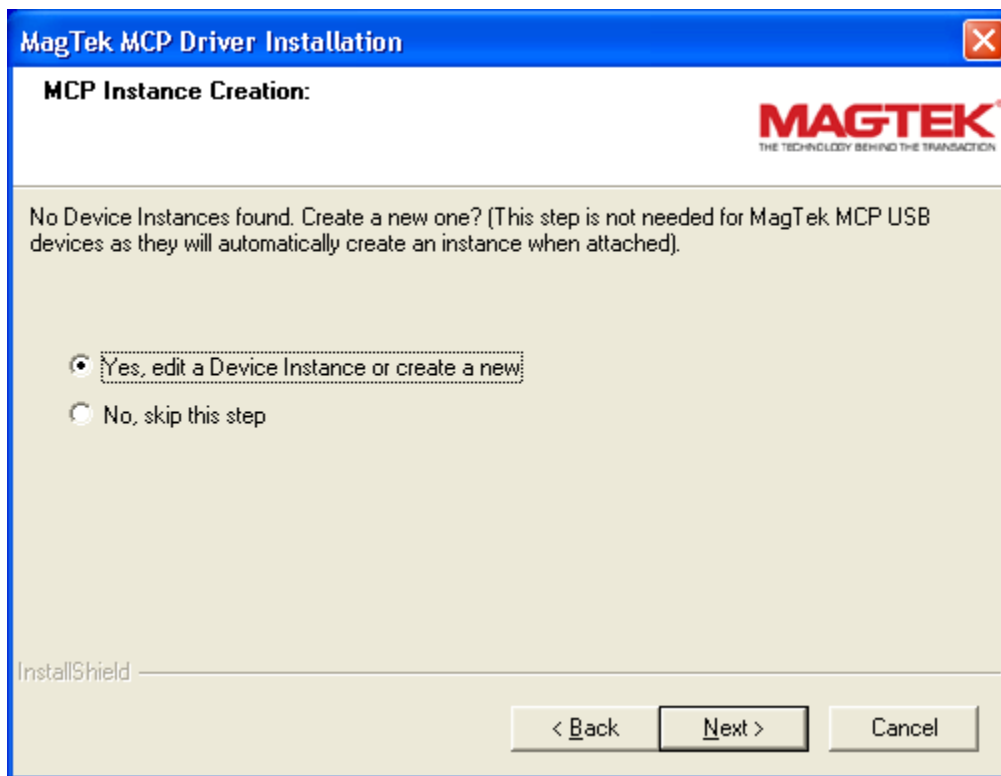
Choose the type of MCP Driver to install. On Win 98 and ME machines, this option will not be given as they only support the Kernel Mode MCP Driver. On Win NT, 2K and XP machines, you can choose either the Kernel Mode MCP Driver, which will be the proper choice for most machines or the User Mode MCP Driver, which is needed only on Client/Server systems such as CITRIX. If you do not need the User Mode MCP, it is highly recommended that you use the default Kernel Mode MCP as it is faster than the User Mode MCP.



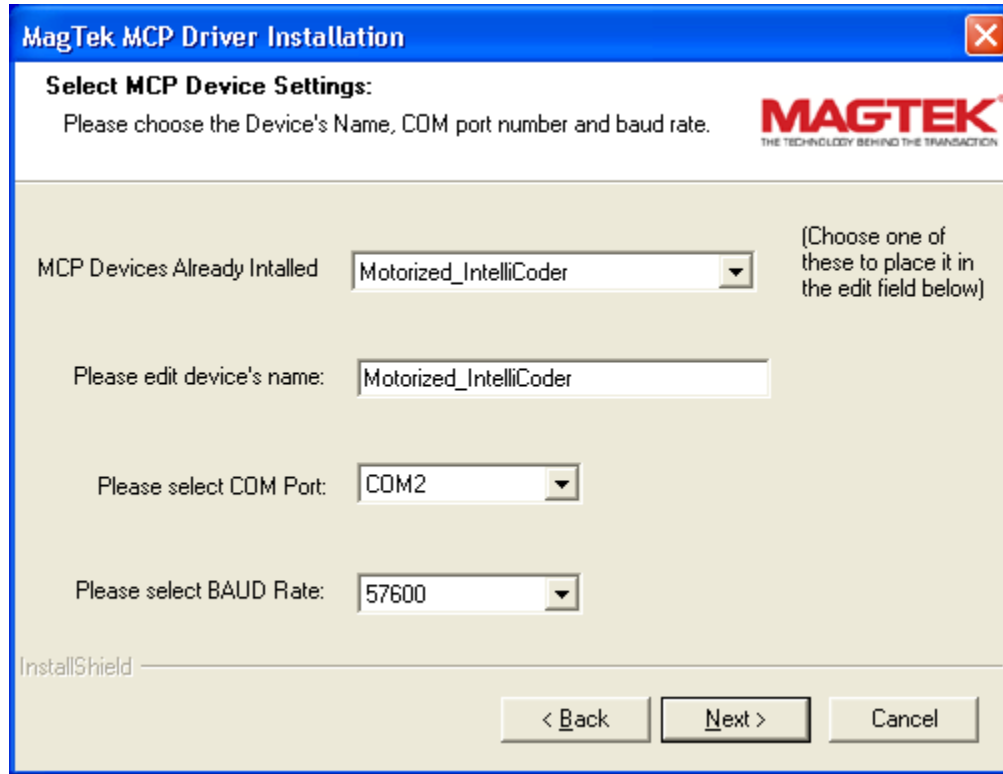
If you wish to create an Instance at this time, make sure the “Yes, edit a Device...” selection is active (as shown below) and then click “Next >”. If you will create an Instance later (by going to the Add/Remove Programs in the Control Panel [see below]) then choose “No, skip this step” and then click “Next >”.

Note

If you will be installing an USB MCP device (such as the IntelliStripe 380 USB”), you do not need to create an Instance here. The computer will create an Instance when the device is first plugged in. It should not be plugged into the computer, however, until this installation is complete and the computer has rebooted.



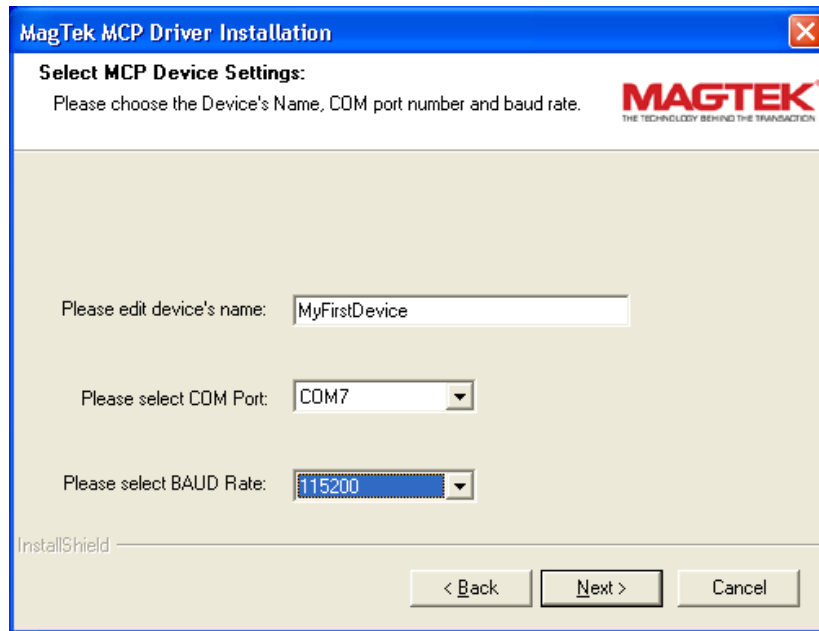
If you do choose to create an Instance, the windows below will appear. Edit the information as desired and click “Next >”.



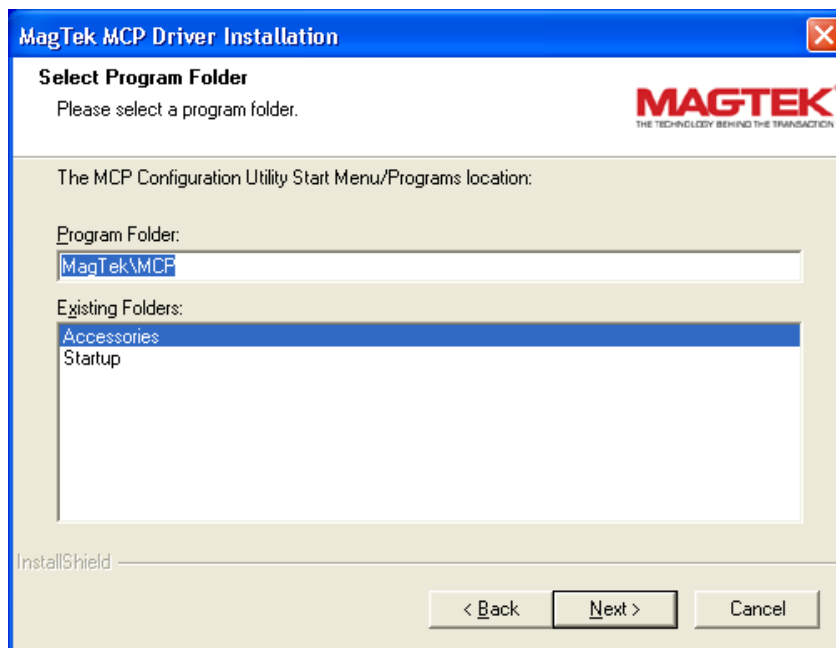
The image shows a Windows-style dialog box titled "MagTek MCP Driver Installation". The window has a blue title bar with a close button (X) in the top right corner. Below the title bar, the text "Select MCP Device Settings:" is followed by the instruction "Please choose the Device's Name, COM port number and baud rate." The MagTEK logo, with the tagline "THE TECHNOLOGY BEHIND THE TRANSACTION", is positioned in the top right of the main content area. The main content area is a light beige color and contains four rows of controls: 1. "MCP Devices Already Installed" with a dropdown menu showing "Motorized_IntelliCoder" and a note "(Choose one of these to place it in the edit field below)". 2. "Please edit device's name:" with a text input field containing "Motorized_IntelliCoder". 3. "Please select COM Port:" with a dropdown menu showing "COM2". 4. "Please select BAUD Rate:" with a dropdown menu showing "57600". At the bottom left, the text "InstallShield" is visible. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

If upgrading from a previous MCP Driver installation, a list of currently installed Instances will appear in the box “MCP Devices Already Installed”.

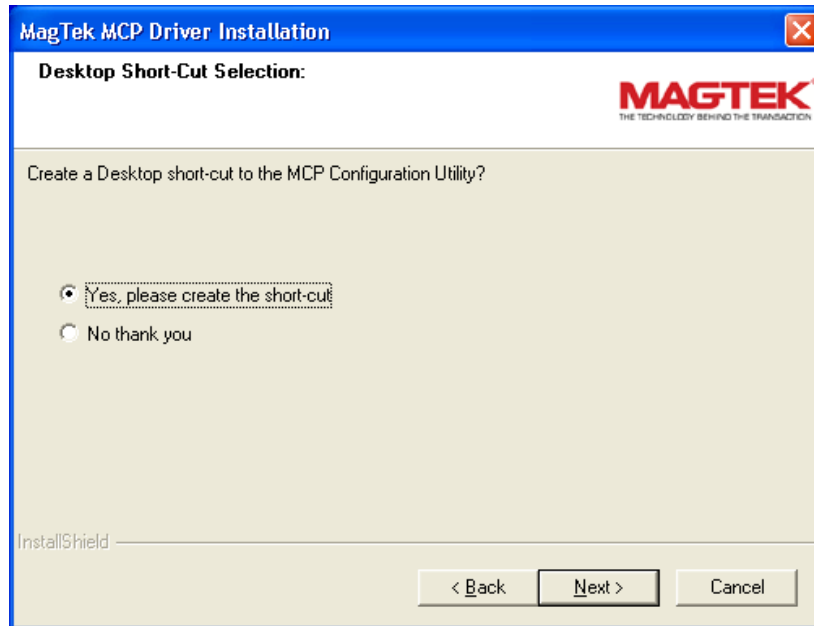
The following is an example window of a new installation with the different fields edited.



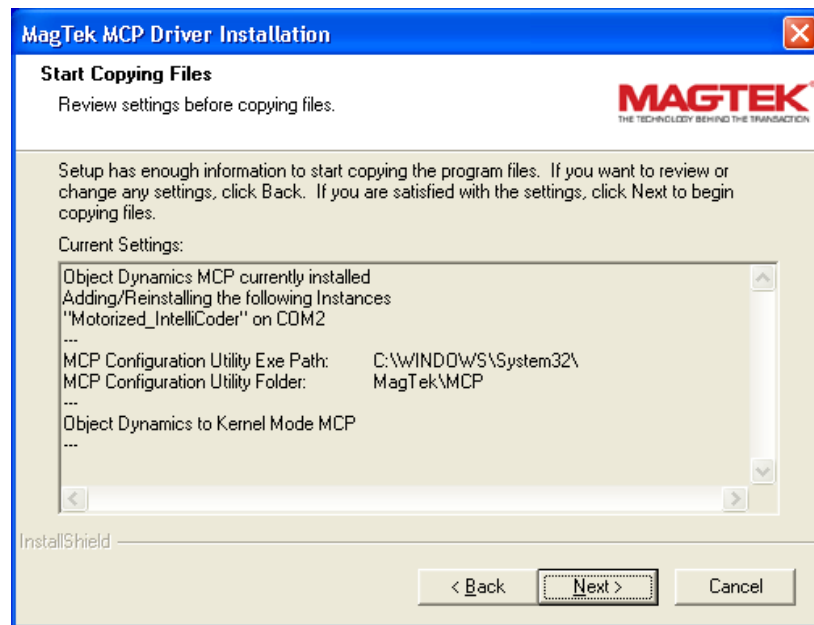
Select the name and location where you would like the short cut to the MCP Configuration program in the Start Menu. This utility is only installed if the Kernel Mode MCP Driver is selected. If you have chosen the User Mode MCP, you will not be shown this window.



In the same manner, if you have chosen the Kernel Mode MCP, you can choose if you want the computer to create a shortcut icon on the computer’s desktop to the MCP Configuration Program.

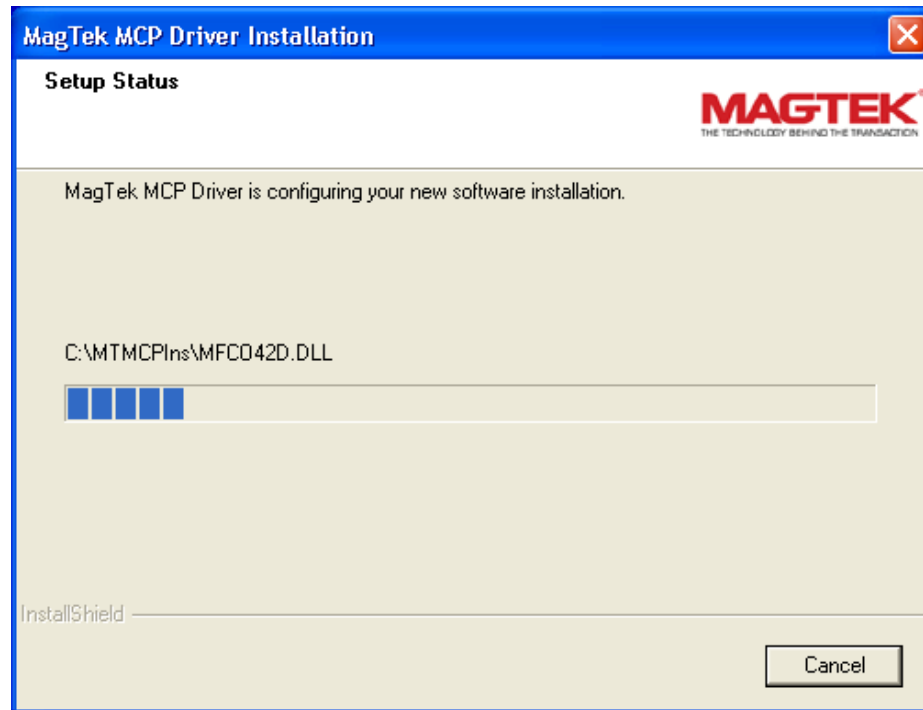
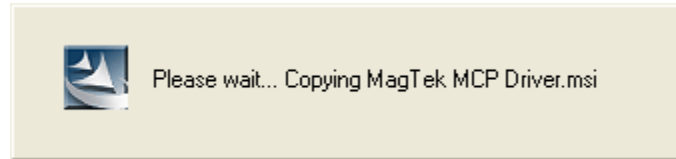
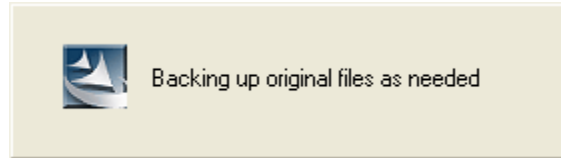


Finally, you will see this window, which summarizes your chosen installation. If all is correct, then click “Next >”. Otherwise, chose “Back” to go back and make changes, or “Cancel” to exit this installation without installing anything.

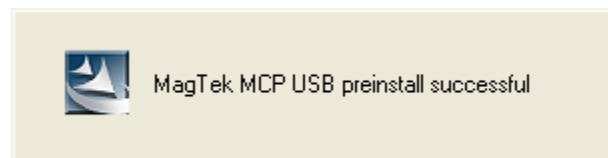


MagTek Communications Protocol, Driver Reference Manual

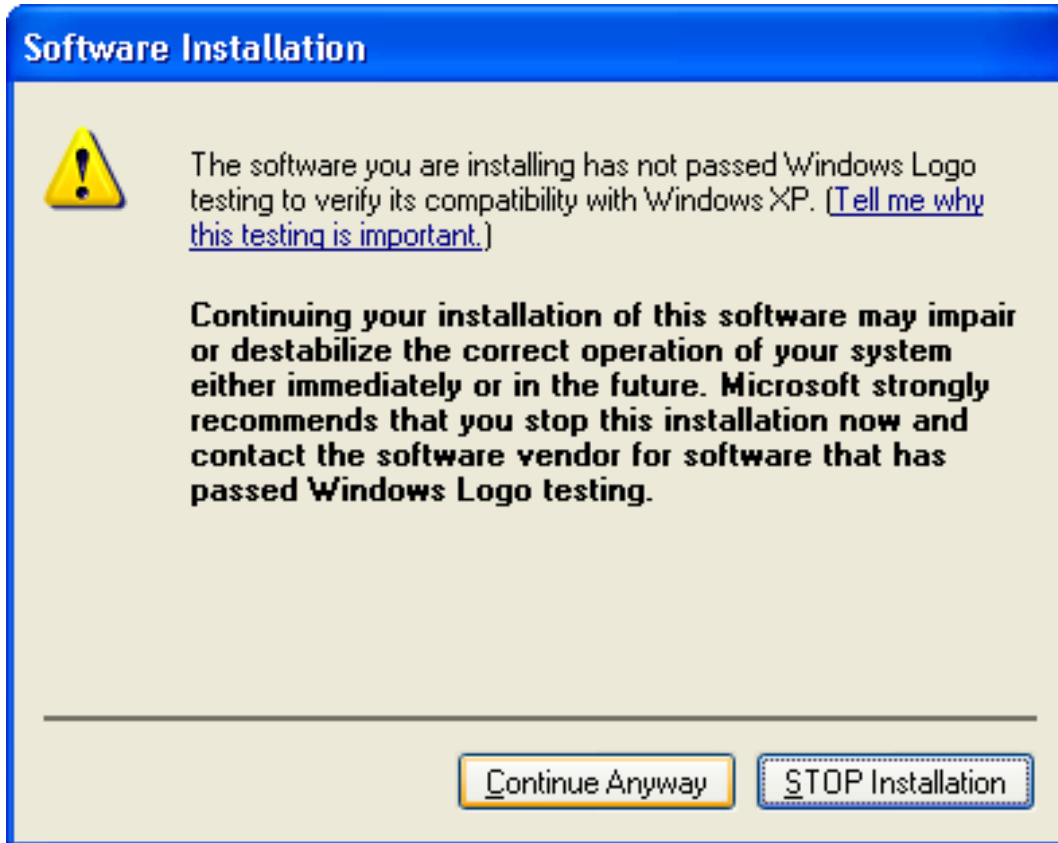
During the installation, you will see the following windows (not all the windows you will see are shown here but this is a representative sample).



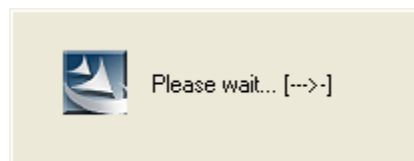
If you have chosen the Kernel Mode MCP install, you will see this window indicating that the computer is ready to accept MCP USB devices (after the reboot).



The following screen (on XP only) will appear because the program is not Windows Logo compliant. Click the “Continue Anyway” button to continue.

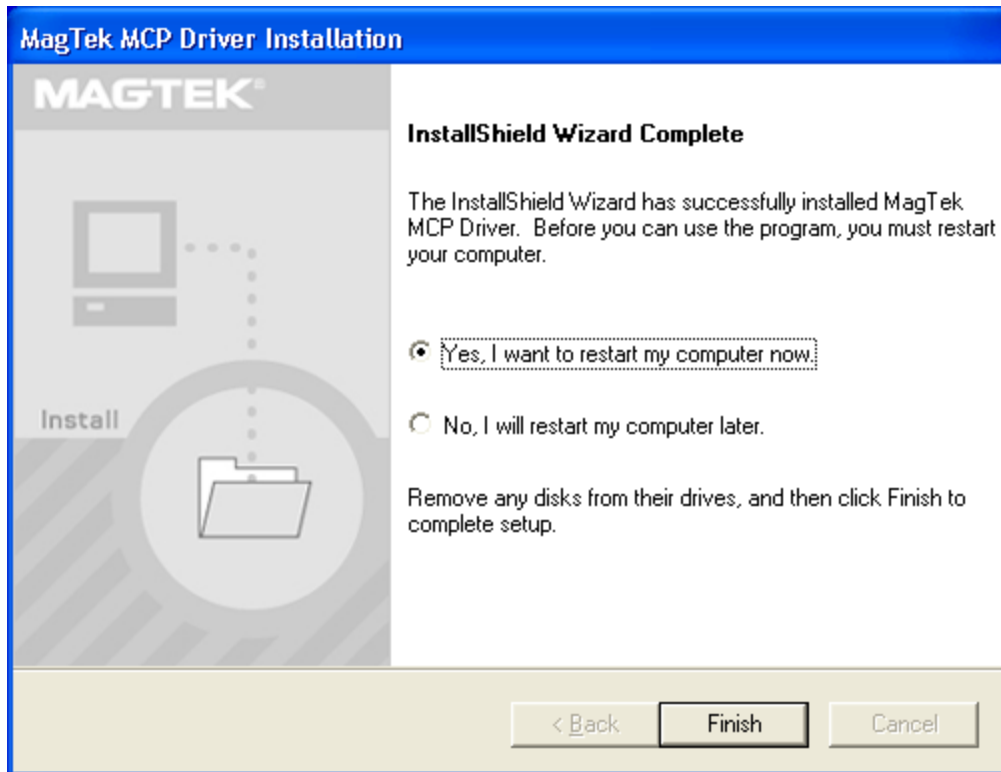


This window is shown while the computer sets up the new Instance.



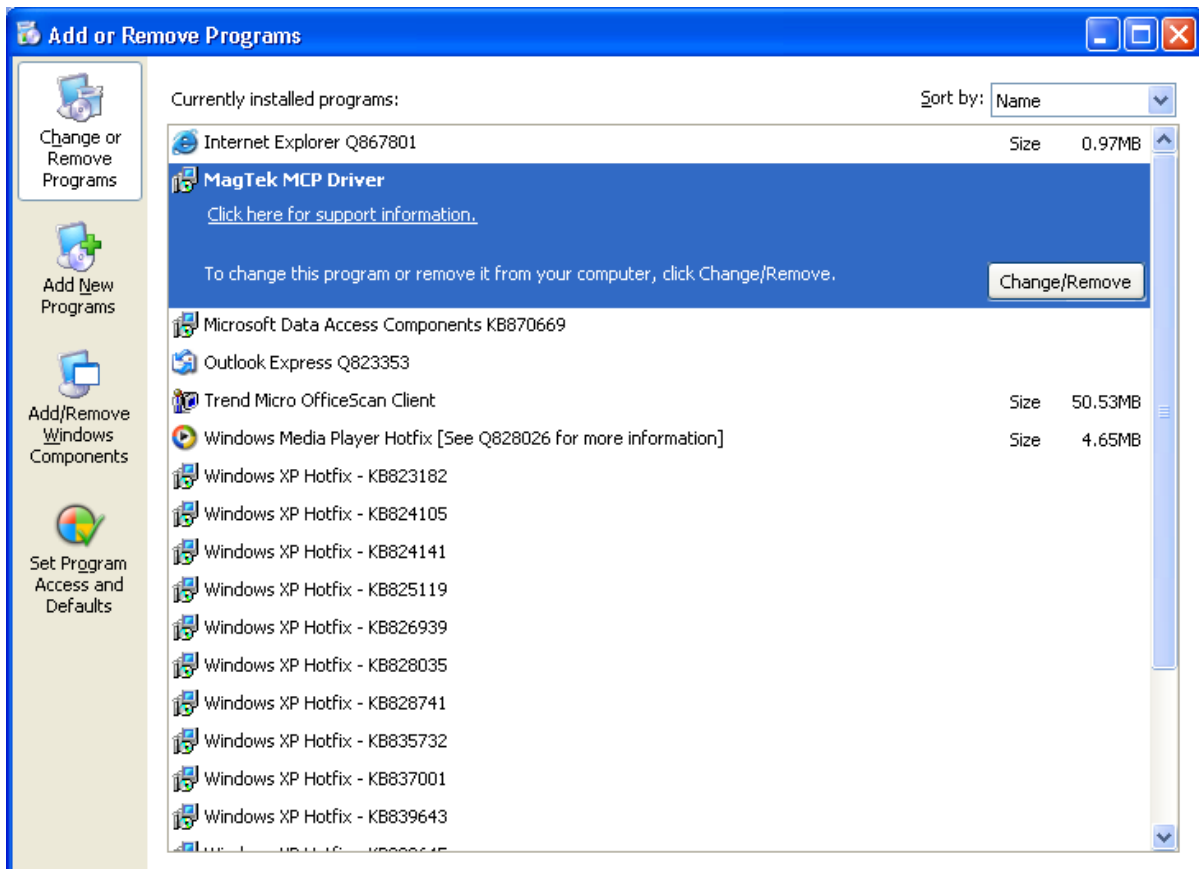
MagTek Communications Protocol, Driver Reference Manual

Make sure “Yes, I want to restart...” is selected and then click “Finish” to reboot and complete the installation. The Kernel Mode driver will not load until the computer reboots. The User Mode driver is available without rebooting, but it is recommended that you reboot at this time if you are changing from Kernel Mode to User Mode.

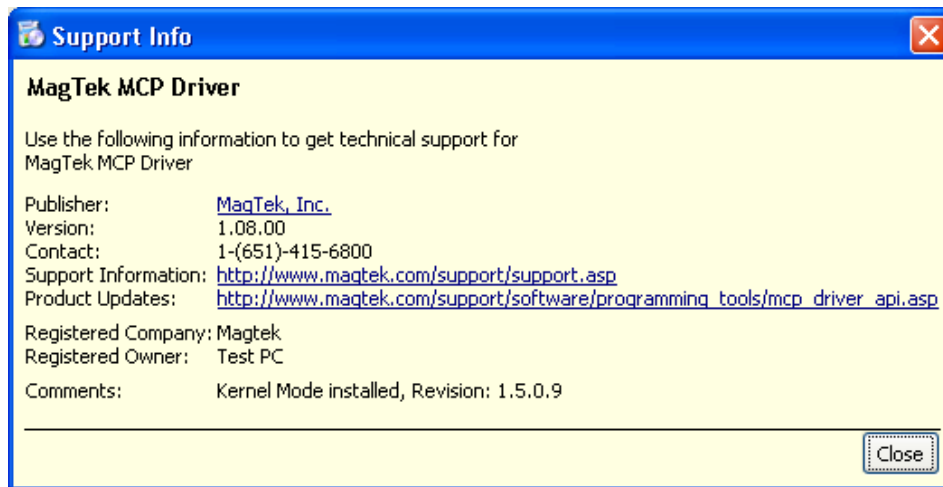


CHANGING THE MCP DRIVER:

Go to the Control Panel and select Add/Remove programs (Start/Settings/Control Panel is Win98/ME/NT/2K and Start/Control Panel in WinXP). Select the MagTek MCP Driver entry by clicking on it once.



To see the current type of MCP Driver installed, and to get support information, click on the “[Click here for support information](#)”. You will see the following window:



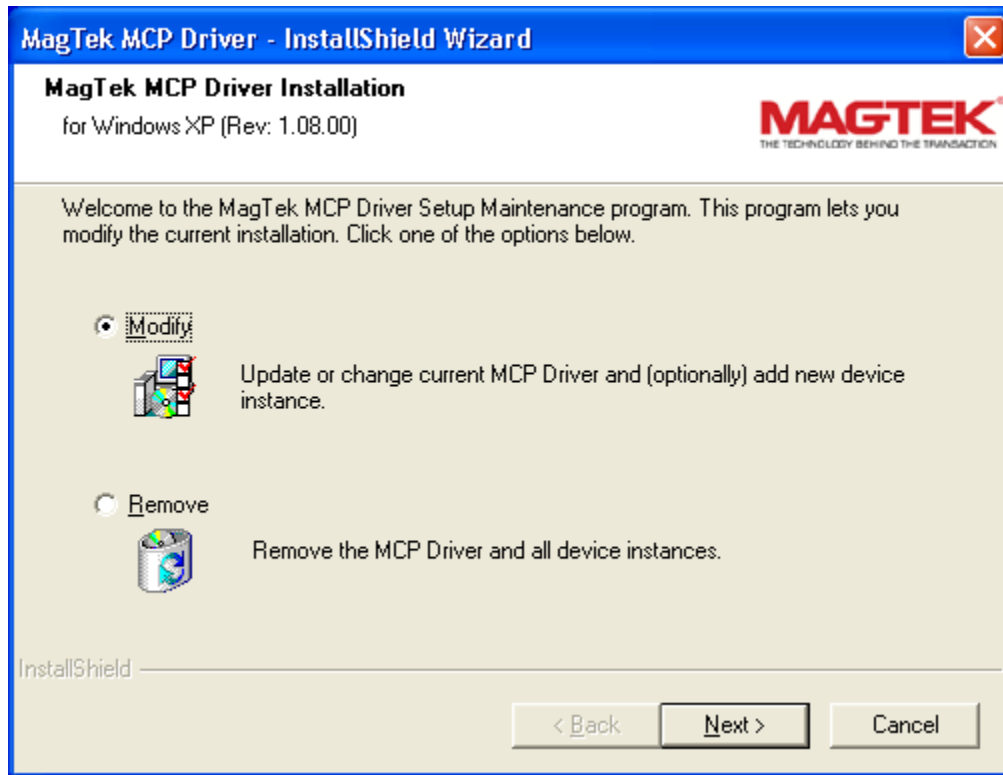
In this case, we have the Kernel Mode MCP Driver installed and it is version 1.5.0.9. Click “Close” to dismiss this window.

Click on the Change/Remove button and you will see the same opening windows as when the install was first run.

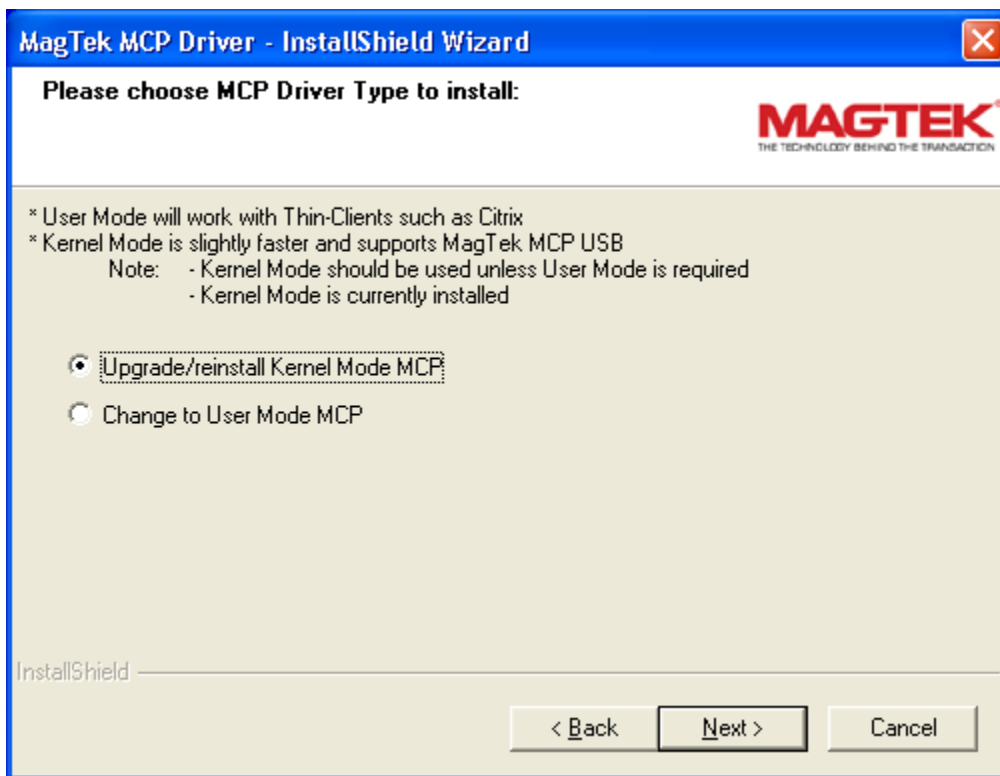
When the opening windows go away, you will see the following.



Change the selection to “Modify” and then click “Next >”.

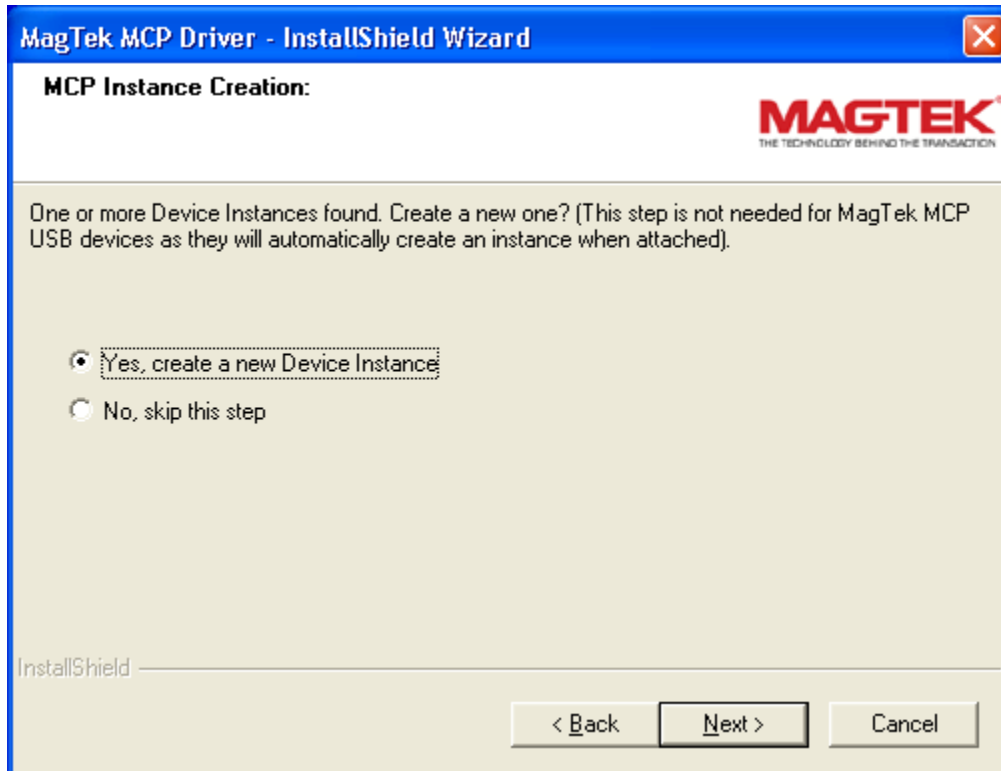


If this is a Win NT/2K or XP machine, you will be given the choice to keep the current MCP Driver or change it to the other type.



Choose the desired MCP type and then click “Next >”.

You can also create a new Instance or modify a current installed one. To create or modify an Instance, make sure the “Yes...” selection is active and then click “Next >”. Otherwise, choose “No...” and then click “Next >”.



If there are any previous Instances, they will appear in the pulldown menu “MCP Devices Already Installed”. If you choose an Instance out of this list, it will replace all the text in the “Please edit device’s name:” field. Edit the Instances settings as desired. If you enter an Instance name of an existing Instance, the existing Instance will be over written. If you enter a name that does not exist, a new Instance will be created.

MagTek MCP Driver - InstallShield Wizard

Select MCP Device Settings:
Please choose the Device's Name, COM port number and baud rate.

MCP Devices Already Installed: MyFirstDevice (Choose one of these to place it in the edit field below)

Please edit device's name: MyFirstDevice

Please select COM Port: COM7

Please select BAUD Rate: 115200

InstallShield

< Back Next > Cancel

The following shows the Instance was modified so a new Instance will be created.

MagTek MCP Driver - InstallShield Wizard

Select MCP Device Settings:
Please choose the Device's Name, COM port number and baud rate.

MAGTEK
THE TECHNOLOGY BEHIND THE TRANSACTION

MCP Devices Already Intalled: MyFirstDevice (Choose one of these to place it in the edit field below)

Please edit device's name: MySecondDevice

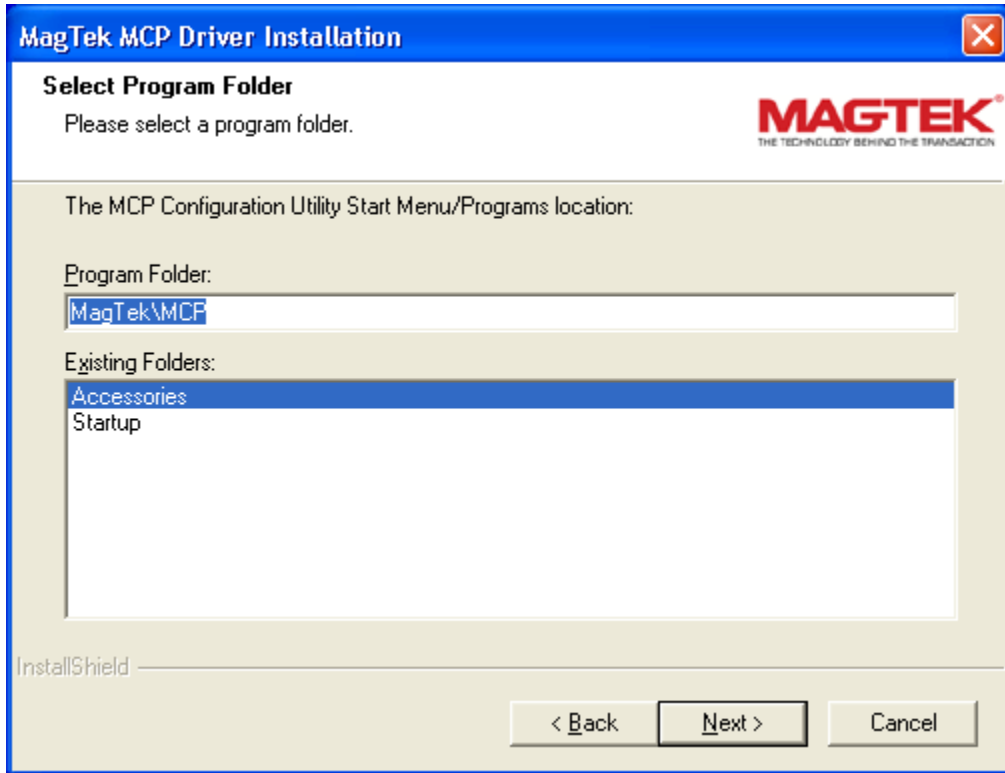
Please select COM Port: COM1

Please select BAUD Rate: 28800

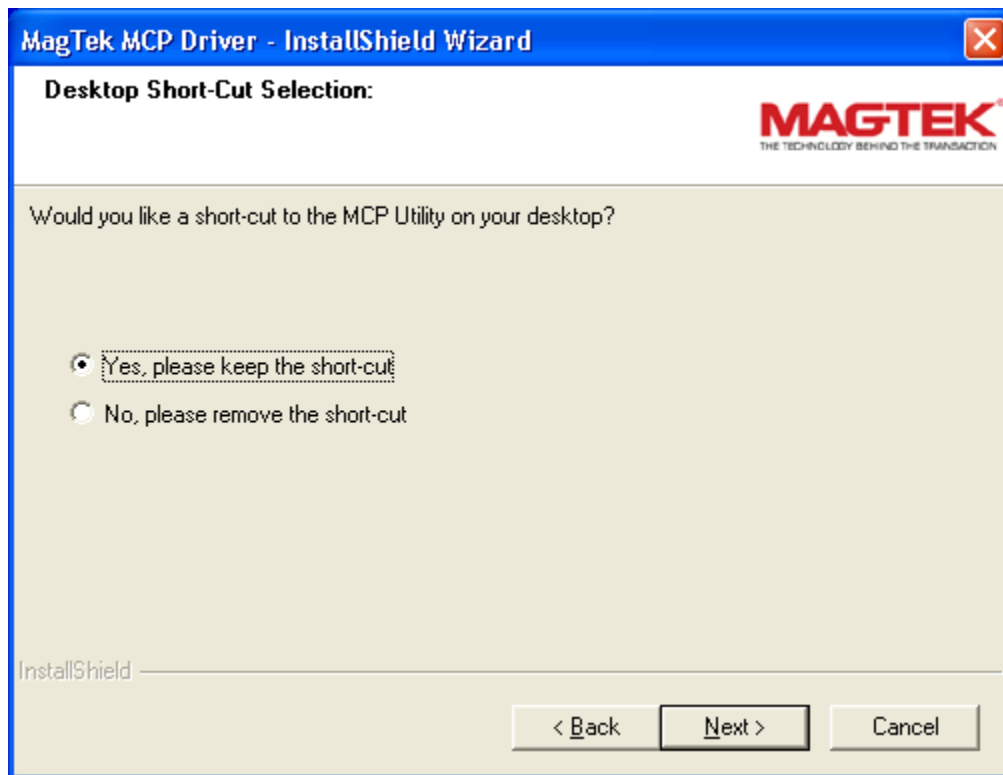
InstallShield

< Back Next > Cancel

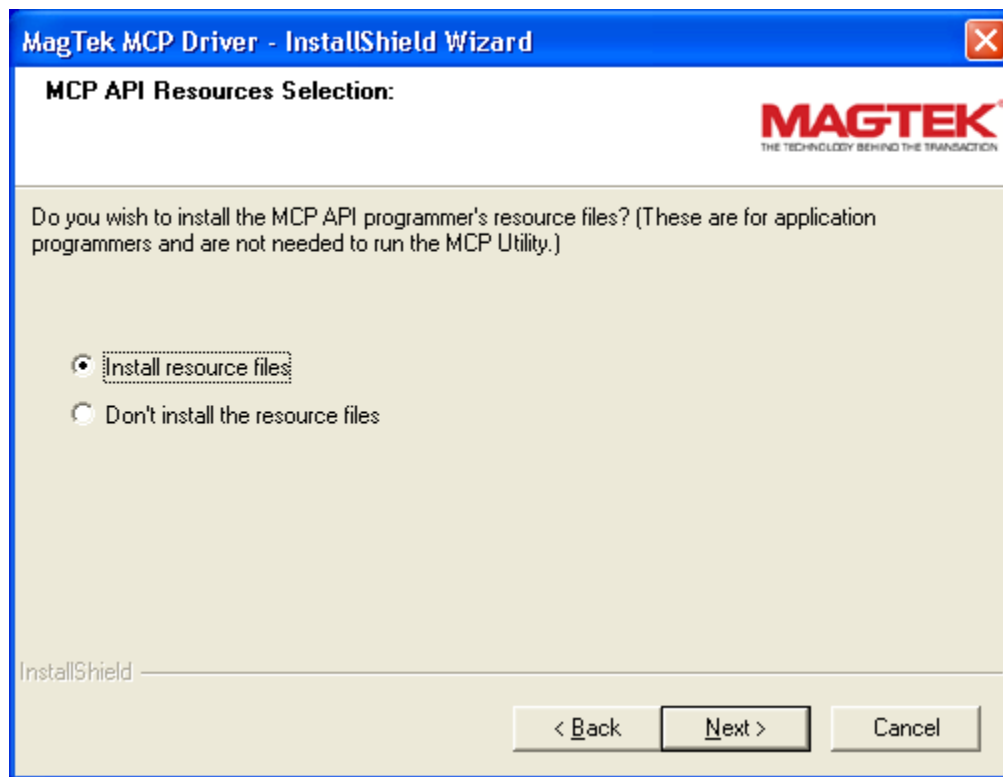
Select the remaining options as desired and then allow the computer to reboot. You will then have the modified options.



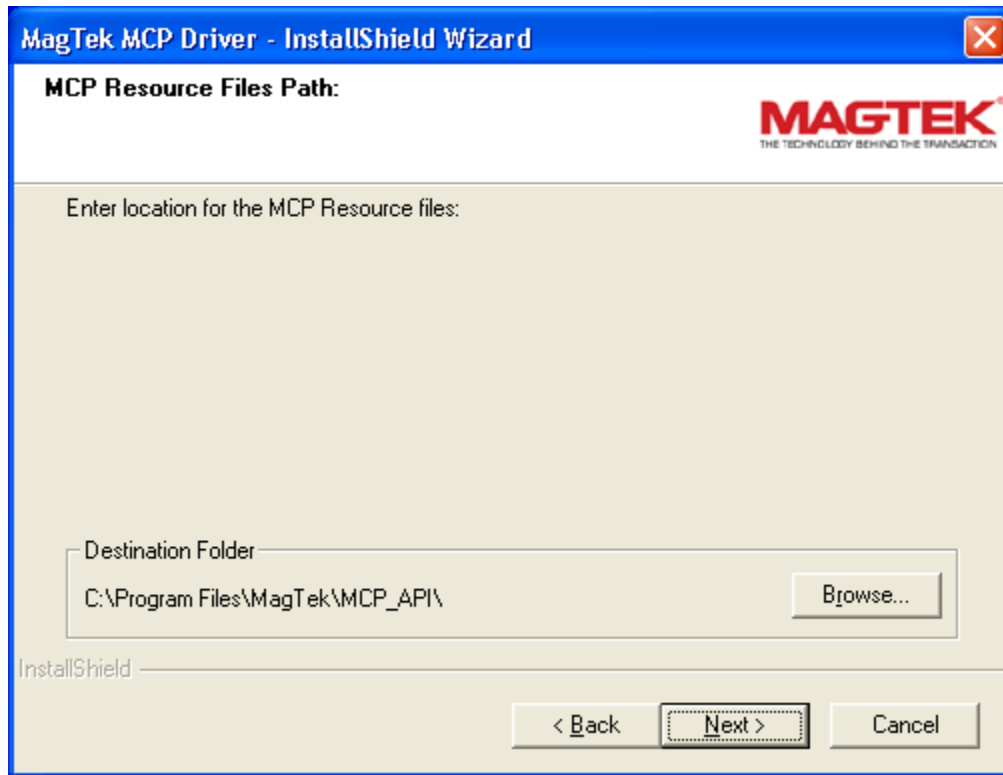
The next choice is to keep or remove the short-cut



If you choose to install the MCP resources, the following will appear:

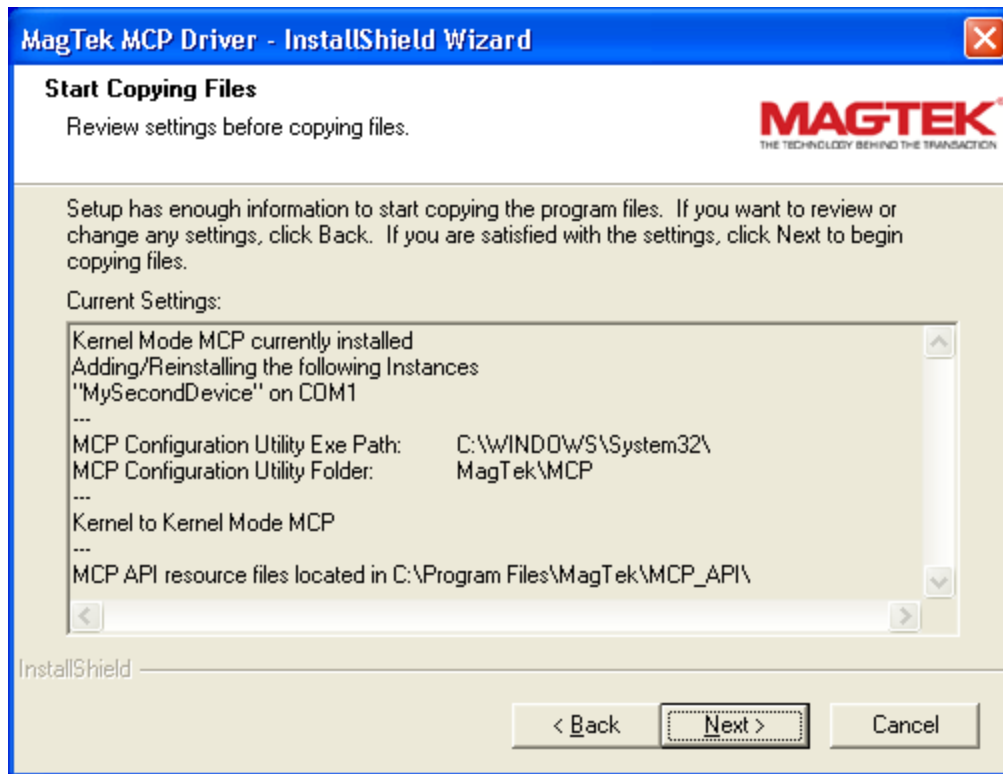


You will then be asked where you would like to save them.

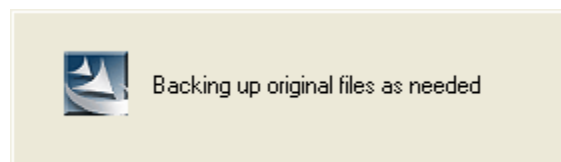


When complete click Next.

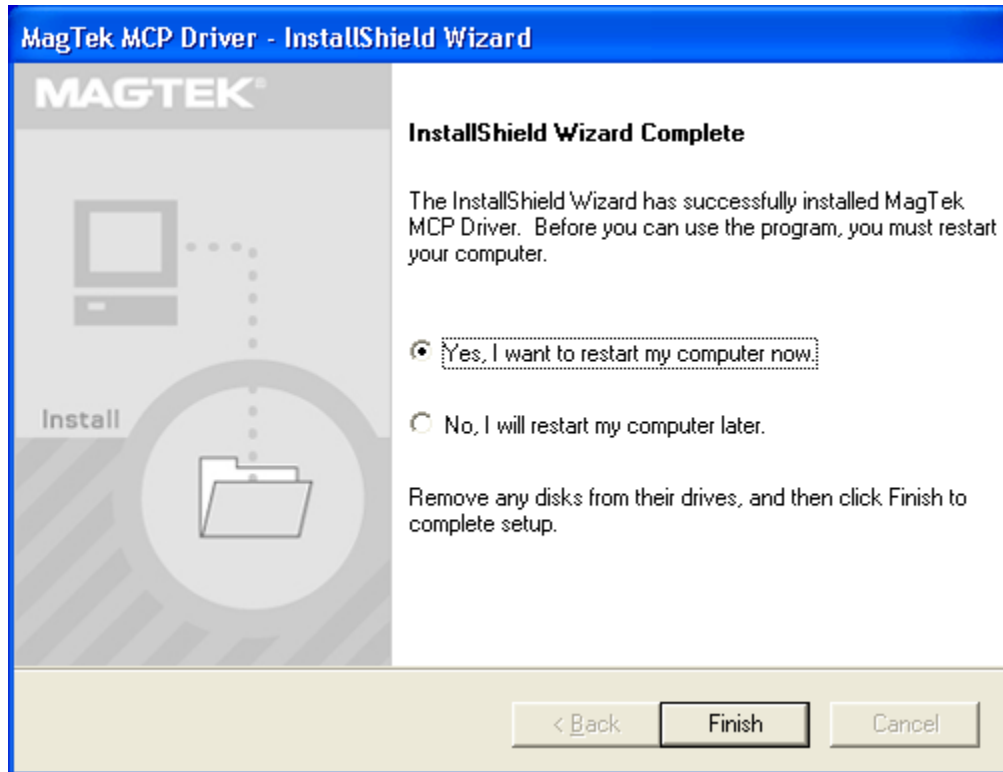
Review the settings and either click Back or Next as described below.



The following window will appear while the files are backed up.



When backup is complete, the following will appear:



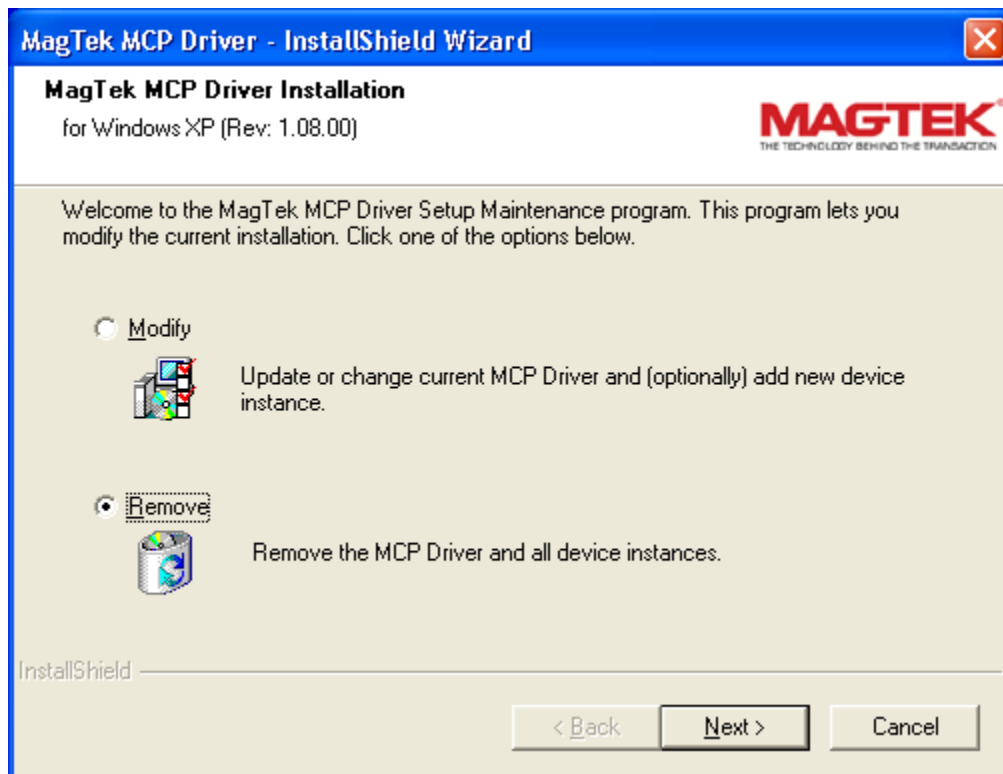
To complete, click Yes and Finish.

REMOVING THE MCP DRIVERS

Note

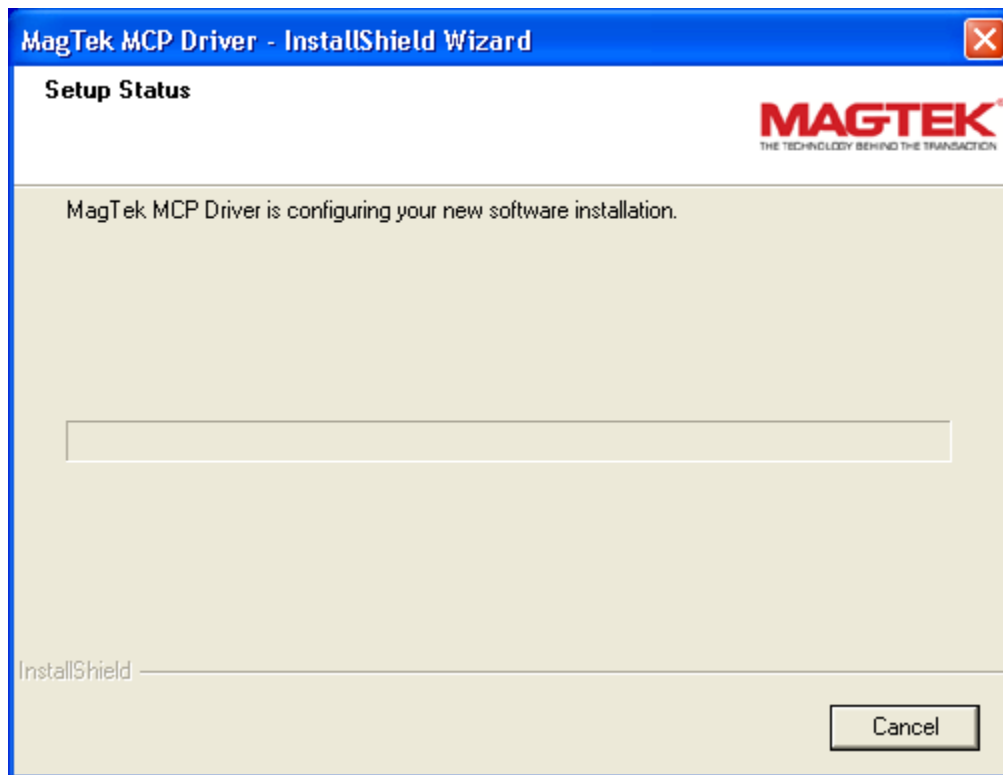
You cannot remove individual Instances with this option (you must use the MCP Configuration utility). The Remove function will remove the MCP Driver and ALL Instances.

Choose the MCP Drivers entry from the Control Panel's Add/Remove Programs.

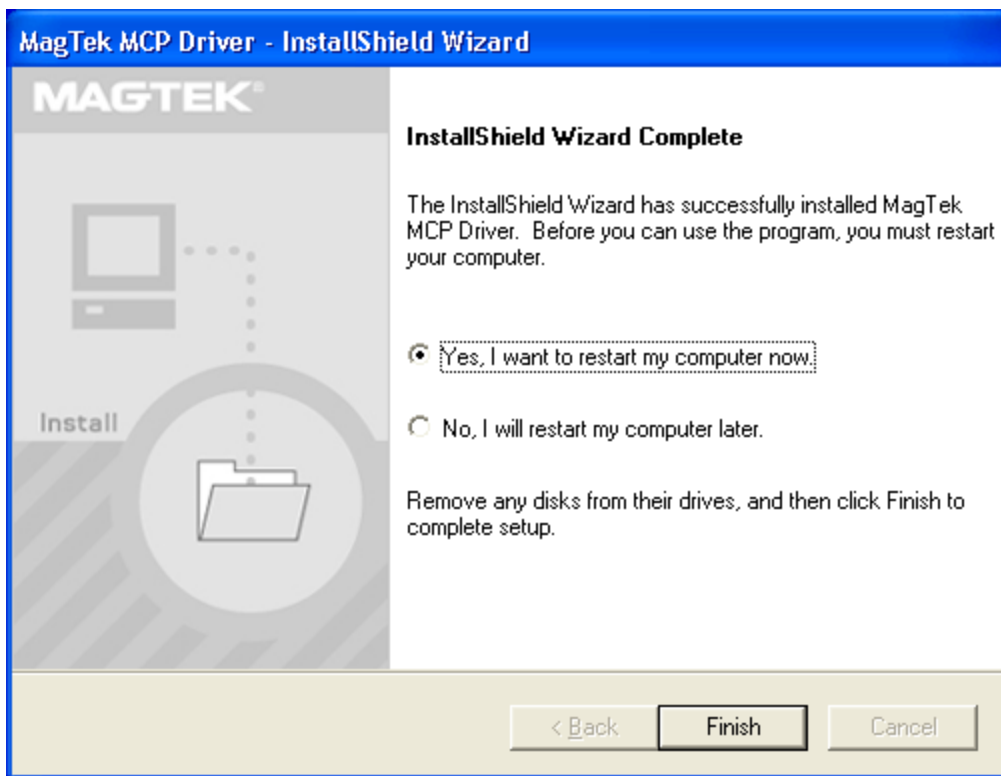


Choose “Remove” and then click “Next >”.

You will see a series of windows.



You must let the computer reboot to complete the removal of the MCP Driver.



DEVICE INSTANCE MANAGEMENT

Device Instance Overview

Before the MCP driver can communicate to a device, a device instance needs to be added. This can be done as part of the driver installation routine. A device instance is a logical connection between the MCP driver and an MCP device attached to a communication port on the computer. Creating a device instance assigns a name to a device instance and establishes properties that are used by the driver to communicate with a device.

MCPCFG Utility Overview

The MCPCFG Utility supports several installation-related commands, intended for use only by the installation script. These commands are listed only in the command summary later in this section, but are not described in detail. The utility is installed with the MCP driver. It is copied by the installer into the Windows directory. MCPCFG is a command-line utility, which can be used by an operator to enter commands at the Windows MS-DOS prompt, from a batch file, or from an application. The utility has only limited error checking capabilities so it must be used with care. For example, the entered baud rate parameter's value is not checked for validity.

Note that changes made using MCPCFG do not take effect until the driver is restarted. On Windows NT, the driver can be restarted without rebooting by using MCPCFG. On Windows 95 and 98, the system must be rebooted for the changes to take effect.

MCPCFG Command Summary

The following shows a summary of all commands supported by MCPCFG.EXE:

Command	Description
Device Instance Management	
mcpcfg a <name> <pname1>=<val1> [<pname2>=<val2> ...]	Add device
mcpcfg aq <name> <pname1>=<val1> [<pname2>=<val2> ...]	Add device (quiet)
mcpcfg r <name>	Remove device
mcpcfg rq <name>	Remove device (quiet)
mcpcfg l	Display devices
mcpcfg s <name> <pname1>=<val1> [<pname2>=<val2> ...]	Change settings
mcpcfg sq <name> <pname1>=<val1> [<pname2>=<val2> ...]	Change settings (quiet)
mcpcfg p <name>	Display device settings
Installer Support Commands	
mcpcfg restart	Stop (if started) and re-start MCP drivers (NT only)
mcpcfg stop	Stop (if started) MCP drivers (NT only)
mcpcfg osver	Determine OS version.
mcpcfg reboot	Display the standard "Reboot Needed" dialog box, re-boot if user answers "Yes"
mcpcfg msg <message text>	Display a message box; wait for user to click OK.
mcpcfg wait <command line>	Start a GUI-based Win32 process and wait for it to terminate.

Adding a Device Instance from Command Line – RS-232

Use the following command to add a new device instance:

```
mcpcfg a <name> Port.Name=<port> [Transport=<xpt> [<more settings>]]
```

where:

- <name>** is a name chosen for the new device instance. This name must contain only characters that are valid for a file name and should have no ‘\’ characters. The name must be unique, i.e.; no other MCP device should be using it. The new device will be visible under this name to applications using the MCP API.
- <port>** specifies the connection port to which the device is attached. It should specify a valid serial port. If the port name is not one of the pre-defined system port names (COM1..COM4), a Transport setting must also be specified (see below). If one of the standard names is used, MCPCFG will automatically assign the transport type (“Serial” for COMx)
- <xpt>** specifies the transport type to be used for the device. This setting is optional if a standard connection port is used. If specified, it should be “Serial”.
- <more settings>** other device settings may be specified at the time the device instance is created. Any number of additional settings may be specified. The syntax for these is the same as for the `mcpcfg s` command (see below). If other device settings are not specified, the default property values will be used. These defaults are listed in the properties sections.

For Windows NT Administrator privilege is required to execute this command.

Example:

```
mcpcfg a IntelliStripe Port.Name=COM1
```

The informational and diagnostic messages displayed by this command can be suppressed by using the “quiet” version:

```
mcpcfg aq <name> ...
```

Adding a Device Instance from Command Line – USB

Device Instances are automatically added. If the device has a serial number, the instance name is DeviceName.SerialNumber.

**Adding a Device Instance from Windows Based Application (WINDOWS 2000, XP)
– RS-232**

- 1) Close all applications currently using the MCP Driver
- 2) Type a descriptive name on the text box.
- 3) Make appropriate changes to settings
- 4) Click the add button
- 5) Click OK

**Adding a Device Instance from Windows Based Application (WINDOWS 2000, XP)
– USB**

Device Instances are automatically added. If the device has a serial number, the instance name is DeviceName.Serial.Number

**Modifying a Device Instance Properties from Windows Based Application
(WINDOWS 2000, XP) – RS-232 and USB**

- 1) Close all applications currently using the MCP Driver
- 2) Select a device instance from the list box
- 3) Make appropriate changes to settings
- 4) Click OK.

**Removing a Device Instance from Windows Based Application (WINDOWS 2000,
XP) – RS232**

- 1) Close all applications currently using the MCP Driver
- 2) Select a device instance from the list box
- 3) Click on remove to delete the device
- 4) Click OK.

**Removing a Device Instance from Windows Based Application (WINDOWS 2000,
XP) – USB**

The USB device instances are automatically removed when the device is physically removed from the system. Please stop the device by clicking on the “Safely Remove Hardware” Icon on the task bar first before physically removing it from the USB port.

Removing a Device Instance from a Command Line – RS-232

Use the following command syntax to remove an existing device instance.

```
mcpcfg r <name>
```

<name> is a name of the device instance to be removed. This name was specified with the mcpcfg a command when the device was added.

For Windows NT Administrator privilege is required to execute this command.

Example:

```
mcpcfg r IntelliStripe
```

The informational and diagnostic messages displayed by this command can be suppressed by using the “quiet” version:

```
mcpcfg rq <name>
```

Removing a Device Instance from a Command Line – USB

The USB device instances are automatically removed when the device is physically removed from the system. Please stop the device by clicking on the “Safely Remove Hardware” Icon on the task bar first before physically removing it from the USB port.

Displaying the List of Device Instances from Command Line

The following command displays the names of all configured device instances:

```
mcpcfg l
```

For Windows NT this command may be used without Administrator privilege.

Displaying Device Instance Properties from Command Line

To display the current settings for a device instance use the following command:

```
mcpcfg p <name>
```

where:

<name> is a name of the device instance. This name was specified with the mcpcfg a command when the device was added. To see a list of existing device instance names, use mcpcfg l.

The list displayed contains all properties that can be set for a device instance, whether or not they have been explicitly set by the operator. The values of the properties that have not been explicitly set are displayed as <default>. This means that the driver will use its built-in default value for these properties. Since the actual default values are not known to MCPCFG, it could not display them. Refer to the device instance properties section of this document for the built-in default values of all properties.

The `mcpcfg p` command does not require administrator privilege to be executed on NT.

Example:

```
mcpcfg p IntelliStripe
```

Modifying Device Instance Properties from Command Line

Modifying a Device Instance Properties:

- 1) Close all applications currently using the MCP Driver
- 2) Select a device instance from the list box
- 3) Make appropriate changes to settings
- 4) Click OK.

To change one or more properties use the following command:

```
mcpcfg s <name> <prp_name_1>=<value_1> [<prp_name_2>=<value_2> ...]
```

where:

<name> is a name of the device instance.

<prp_name_x> is the name of the property to be set.

<value_x> is the new value for the property. If this string is empty, the property is set to the driver's built-in default value. Note that setting the `Port.Name` or the `Transport` property to "default" is not allowed.

The whole string `<prp_name_x>=<value_x>` should contain no spaces. If the value is a string and it must contain spaces, enclose the whole `<prp_name_x>=<value_x>` expression in double quotes, e.g.: `"TraceFilePath=c:\dir with spaces\logfile.txt"`.

Any number of `<prp_name_x>=<value_x>` pairs may be specified with this command (subject to limitations of the Windows command-line interpreter).

For Windows NT Administrator privilege is required to execute the `mcpcfg s` command.

Example:

```
mcpcfg s IntelliStripe Port.BaudRate=9600 Port.TraceEnabled=1
```

The informational and diagnostic messages displayed by this command can be suppressed by using the “quiet” version:

```
mcpcfg sq <name> <prp_name_1>=<value_1> [ <prp_name_2>=<value_2> ... ]
```

Modifying Device Instance Properties From Windows Based Application

Adding a Device Instance:

- 1) Close all applications currently using the MCP Driver
- 2) Type a descriptive name on the text box.
- 3) Make appropriate changes to settings
- 4) Click the add button
- 5) Click OK

Removing a device:

- 1) Close all applications currently using the MCP Driver
- 2) Select a device instance from the list box
- 3) Click on remove to delete the device
- 4) Click OK.

Modifying a Device Instance Properties:

- 1) Close all applications currently using the MCP Driver
- 2) Select a device instance from the list box
- 3) Make appropriate changes to settings
- 4) Click OK.

Restarting the MCP Driver

On Windows NT, MCPCFG can be used to restart the MCP driver, so that changes made using the commands described above can take effect without rebooting the system. On Windows 95 and 98, the system must be rebooted for the changes to take effect. No application should have the MCP driver opened at the time this command is executed. This command stops the MCP device driver and then restarts it. If the MCP driver was not started, no error message is displayed and the driver is started.

To restart the MCP driver, use the following command:

```
mcpcfg restart
```

Administrator privilege is required to execute this command.

Stopping the MCP Driver From Command Line

On Windows NT, MCPCFG can be used to stop the MCP driver. No application should have the MCP driver opened at the time this command is executed.

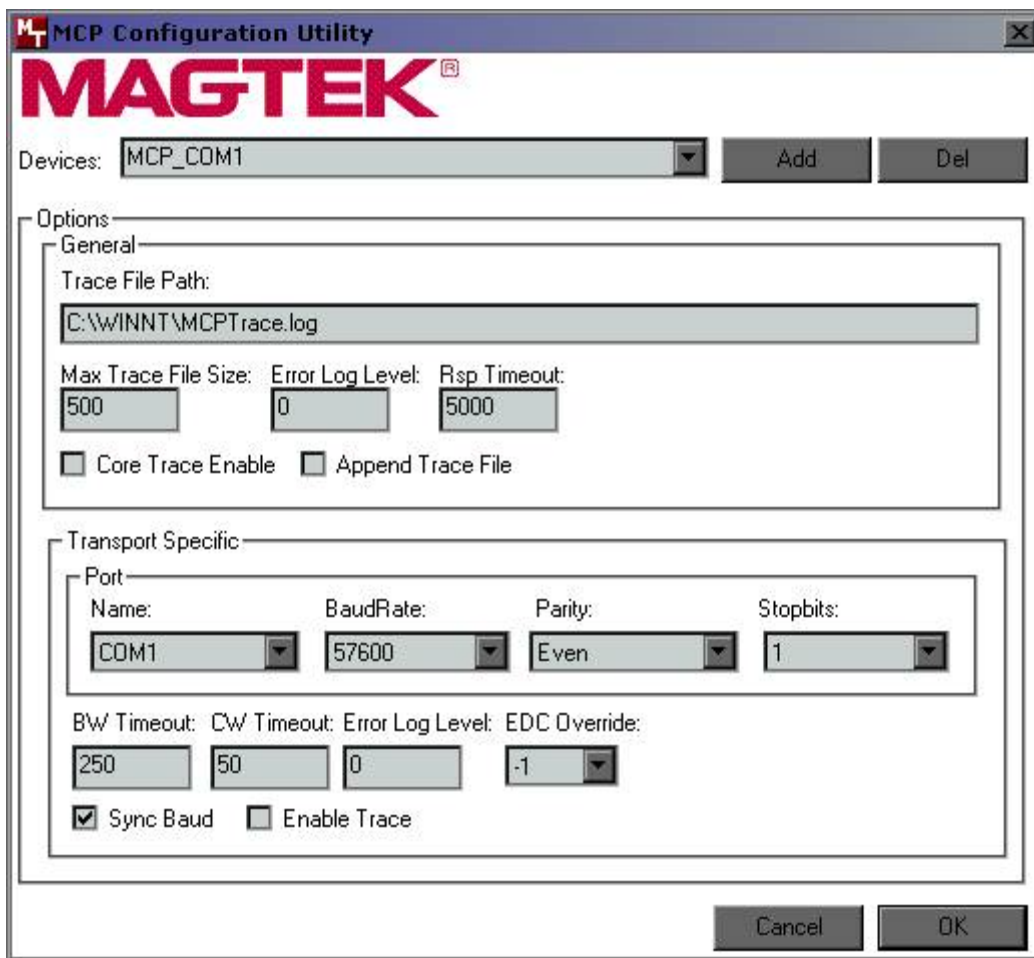
To stop the MCP device driver, use:

```
mcpcfg stop
```

Administrator privilege is required to execute this command.

Windows Configuration Utility

This utility can create, display, modify and remove device Instances, however, this only works under Windows 2000 and XP operating systems.



DEVICE INSTANCE PROPERTIES

Generic Properties

These properties are used regardless of what communication transport type is used. They are not transport specific.

Property Name	Description
TraceFilePath	<p>Specifies the path and file name of the trace file. Both application message and transport frame trace data are placed in this file if tracing when enabled. If the path does not exist or the file name is invalid, tracing is disabled.</p> <p style="text-align: center;">Default file is "MCPTRACE.LOG" Default path is the Windows path.</p>
MaxTraceFileSize	<p>Maximum size (in K-bytes) of the trace file. When this size is reached, no new entries are logged. Note that 1K = 1024 bytes. Due to a limitation in Windows 95 and 98 disk cache management, it is recommended that the trace file size be limited to 1MB for each 16MB of installed memory. Exceeding the recommended limits may cause Windows to run out of memory and/or communications errors to occur.</p> <p style="text-align: center;">Default is 500 (K bytes)</p>
AppendTraceFile	<p>If tracing is enabled, the trace file will be cleared when the port is opened if this value is FALSE (0). If TRUE (1), new trace entries are appended to the existing contents of the trace file.</p> <p style="text-align: center;">Default is FALSE (no append).</p>
Core.TraceEnabled	<p>Enable or disable application message trace. This type of tracing logs only the application message portion of transported blocks. The message frames and transported blocks that do not contain application messages will not be logged. Another property, the transport dependent Port.TraceEnabled property allows all block content to be logged. Note that enabling this property and the Port.TraceEnabled property at the same time causes duplicate information to be logged.</p> <p style="text-align: center;">0 – FALSE – trace disabled (Default) 1 – TRUE – trace enabled</p>
Core.ErrorLogLevel	<p>Error log level. Specifies the level of the MCP session exceptions to be logged into system event log: warnings, errors, etc. (This property is not related to the message trace utility). This is only used for debugging the driver and should normally be kept at its default setting. Can be one of the following values:</p> <p style="text-align: center;">0 – no logging (Default) 1 – log errors only 2 – log everything</p>

MagTek Communications Protocol, Driver Reference Manual

Property Name	Description
Core.RspTimeout	<p>Message Response Timeout (specified in milliseconds). This is the maximum time required for the device to process an application command request and return a response. If this time expires, the request will be aborted. This property should normally be kept at its default value. This value should only be changed if a device is compatible with other values.</p> <p style="text-align: center;">Default is 5000 ms (0 is not allowed).</p>

Serial Port Properties

These properties are serial port specific.

Value	Notes
Port.Name	Serial Port Name to which the device is connected. Can be one of the standard values COM1, COM2, COM3, COM4. There is no default value. This property must be specified when a device instance is added. If one of the standard values is used, the Transport property is automatically set to Serial. If not, the Transport property must be set to Serial manually.
Transport	For a serial port this property is always set to Serial. It is automatically set if one of the standard values is used for the Port.Name property while adding a device instance. If not, it needs to be set manually.
Port.EDCTypeOverride	EDC type that the driver should use for all frames sent to or received by the MCP device. Upon protocol initialization, the driver retrieves the supported EDC types from the MCP device. If this property is not -1, the specified EDC type is used instead of the EDC type specified by the device. If EDCTypeOverride is -1, the driver uses the best EDC type supported by the device (CRC first, LRC second, No EDC last). Can be one of the following values: <ul style="list-style-type: none"> -1 - Use the best EDC type supported by the MCP device. (Default) 0 - No EDC 1 - 16-bit CRC (Cyclic Redundancy Check) 2 - 8-bit LRC (Longitudinal Redundancy Check)
Port.TraceEnabled	Enable or disable transport block trace. This type of tracing logs all transported blocks. Another property, the transport generic Core.TraceEnabled property allows only the application message portion of blocks to be logged. Note that enabling this property and the Core.TraceEnabled property at the same time causes duplicate information to be logged. <ul style="list-style-type: none"> 0 – FALSE – trace disabled (Default) 1 – TRUE – trace enabled
Port.ErrorLogLevel	Error log level. Specifies the level of the MCP session exceptions to be logged into system event log: warnings, errors, etc. (This property is not related to the message trace utility). This is only used for debugging the driver and should normally be kept at its default setting. Can be one of the following values: <ul style="list-style-type: none"> 0 – no logging (Default) 1 – log errors only 2 – log everything

MagTek Communications Protocol, Driver Reference Manual

Value	Notes
Port.BlockWaitTimeout	Block Wait Timeout (specified in milliseconds). Maximum time allowed for a device to confirm it received a frame from the host. This property should normally be kept at its default value. This value should only be changed if a device is compatible with other values. Default is 250 ms (0 is not allowed).
Port.CharacterWaitTimeout	Character Wait Timeout (specified in milliseconds). Maximum time allowed between characters when receiving a frame from a device. This property should normally be kept at its default value. This value should only be changed if a device is compatible with other values. Default is 50 ms (0 is not allowed).
Port.BaudRate	Serial port baud rate. Valid values are 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 and 115200. The default value is 9600. Note that baud sync only is intended to work on baud rates 9600 – 115200. Devices normally support only a subset of these values. See the devices technical reference manual to determine which values are supported.
Port.Parity	Serial port data parity. This property should normally be kept at its default value. This value should only be changed if a device is compatible with other values. N None E Even (Default) O Odd S Space M Mark Devices normally support only a subset of these values. See the devices technical reference manual to determine which values are supported.
Port.StopBits	Number of stop bits. This property should normally be kept at its default value. This value should only be changed if a device is compatible with other values. 1 1 stop bit (Default) 2 2 stop bits Devices normally support only a subset of these values. See the devices technical reference manual to determine which values are supported.
Port.BaudSync	Specifies whether the driver should submit a synchronization signal to the device or not. This property should normally be kept at its default value. This value should only be changed if a device is compatible with other values. 0 FALSE – do not synchronize 1 TRUE – synchronize w/device (Default) Devices normally support only a subset of these values. See the devices technical reference manual to determine which values are supported.

SECTION 4. TRACE LOG

The trace log is a file that logs the communications between the driver and a device. The trace log is used for helping application and device developers during the device development phase. It can be used for diagnosing problems with existing applications. The log can capture complete blocks transmitted and received by the driver or just the application message portion of blocks. The trace file is a text file and can be read using a standard text editor.

The trace log utility can be enabled and configured using device instance properties. Details of these properties are in the device instance properties section of this document. These properties control enabling/disabling logging of the entire block, enabling/disabling logging of only the application message portion of the block, the name and path of the trace log file, the size of the trace log file and clearing of the trace log file. The properties are `Port.TraceEnabled`, `Core.TraceEnabled`, `TraceFilePath`, `MaxTraceFileSize` and `AppendTraceFile` respectively.

Each line in the trace file has the following format:

```
<datetime stamp> <origin> <direction> <data> C R L F
```

where each field is described below:

<datetime stamp>:

The datetime stamp format is as follows:

```
MM/dd/YYYY  
hh:mm:ss:ms
```

Under Windows 98, Me, time is calculated at GMT.

Under Windows 2000, NT and XP time is the local time.

<origin>:

Origin of entry: MSG – application message portion of block
 FRM – entire block including frame

<direction>:

Direction of data with respect to the host: In – incoming
 Out - outgoing

<data>:

Data logged in hexadecimal format.

^C _R ^L _F:

Carriage Return / Line Feed

The following is an example of a trace entry for an application message request and response (Device application Get Model Number property):

MagTek Communications Protocol, Driver Reference Manual

```
08\10\2004 11:44:48:267 MSG Out 00 00 00 00 02 00
08\10\2004 11:44:48:267 FRM Out 01 00 10 00 06 17 00 00 00 00 02 00 24 73
08\10\2004 11:44:48:282 MSG In 40 00 00 00 02 00 49 6E 74 65 6C 6C 69 53 74
72 69 70 65 20 33 38 30 00
08\10\2004 11:44:48:282 FRM In 00 01 11 00 18 08 40 00 00 00 02 00 49 6E 74
65 6C 6C 69 53 74 72 69 70 65 20 33 38 30 00 65 F7
08\10\2004 11:44:48:313 MSG Out 00 00 00 00 02 01
08\10\2004 11:44:48:313 FRM Out 01 00 13 00 06 14 00 00 00 00 02 01 5D 1F
08\10\2004 11:44:48:313 MSG In 40 00 00 00 02 01 31 36 30 35 31 33 33 32 43
30 32 00
08\10\2004 11:44:48:313 FRM In 00 01 12 00 12 01 40 00 00 00 02 01 31 36 30
35 31 33 33 32 43 30 32 00 85 EC
```

SECTION 5. APPLICATION PROGRAMMABLE INTERFACE

The application programmable interface (API) is presented through a 32 bit, Win32 DLL **MCPAPI.DLL**. This dll is installed during driver installation. The API gives a Windows application a set of easy to use “C” style functions that it can use to communicate with the MCP driver/device. This dll can be used by a variety of development platforms. For example, the dll can be used by Microsoft Visual Basic 6.0 and Microsoft Visual C++.

TYPICAL OPERATION

The host application can enumerate (get the name of) the available devices using the *McpEnum* function. The function returns the names of all the device Instances configured on the host computer. The results obtained through that function can be used in the subsequent *McpOpen* calls.

To establish a communication channel with the desired device, the host application must first open the device by calling the *McpOpen* function. If the operation is successful, a handle is returned which should be used as *hDevice* parameter on subsequent API calls.

At any time after the device has been opened, the host application can re-establish, or reset the channel by calling *McpReset*. This will abort any outstanding requests and will restore the communication channel to its initial state.

After the device has been opened, command requests can be sent to the device by calling the *McpCall* function.

At some point the device could send an unsolicited notification, which can be retrieved by the application using the *McpWait* function.

The application also may get the current values of the driver or device properties through *McpGet* calls. For modifying the values of those properties the application can use an *MCPSet* call.

When the device is no longer needed, the communication channel with a given device can be closed by calling the *McpClose* function.

DEVICE CHANNEL LIFE CYCLE

Devices are identified at the API level by names. The host application enumerates (gets the name of) the available devices using the *McpEnum* function. On every subsequent call the function returns a name of an MCP device connected and configured on the host computer. By calling this function in a loop, an application can enumerate all devices.

The host application may use a hard-coded device name, a name obtained from the registry or other configuration data, or it may ask the user to select a device from the list returned by *McpEnum*. The application can establish a communication channel with the desired device, by calling *McpOpen* with the device name (the device name can be any of the names returned by the *McpEnum* function). *McpOpen* returns a device handle which the application uses on all subsequent MCP function calls regarding this device.

Any application on the host computer may open any device. There is no limitation of the number of the devices that one application can open but a device can be opened by no more than one application at a time.

After opening the device, the host application can reestablish, or reset the channel by calling *McpReset*. This function cancels any outstanding commands and restores the communication channel to its initial state. The result of reset is the same as closing and re-opening the device, except that the device handle remains the same. Using the *McpReset* operation is preferable to closing and reopening the device.

PROPERTIES

Each device application can have its own set of properties. Generic commands can be supported by devices that allow these properties to be retrieved and set. Each driver device instance also has its own set of properties. The MCPAPI interface provides two functions, *McpGet* and *McpSet*, which give an application the ability to get and set properties. These functions are general purpose so that additional properties may be easily added in the future without modifying the interface.

The MCP property interface allows the device application properties to be accessed through their numeric property ID and Application ID. Device applications identify properties only by numeric IDs as opposed to being specified by name due to their limited amount of memory and processing power.

The host application can access the driver device instance properties through the same set of commands. These properties can be specified by name only as opposed to numeric IDs. See the device instance properties section of this document for more details on these properties. Only active-time driver properties can be modified by the application.

COMMANDS

Host applications access device functions by communicating with a specific device application running on the device. Each device application is identified by a numeric application ID which is unique for the device. Every device application defines the command set that it understands and can process. Commands are identified by a numeric command ID and may contain additional data needed to process the command. The command ID is unique for the given application.

The device responds to commands within short, finite amount of time as defined by the *Core.RspTimeout* for the device. The response contains a result code that indicates whether the command was completed successfully, and if not, what was the reason for failure. The response may also contain additional data to be returned to the host application. If the device response is not received in the *Core.RspTimeout* time period, the MCP driver will fail the command request.

MCP driver handles the process of issuing a command request to the device and receiving the device response, so from the viewpoint of the host application, the act of sending the command and receiving the response is a single action. To execute a command, the host application uses the API *McpCall* function.

NOTIFICATIONS

MCP devices can send unsolicited messages – notifications, in case an external event occurred (e.g. user swiped a card, pushed a button, etc.) or a device application has changed its state (e.g. printer out of paper, command completed, etc.). Unlike command responses, notifications can be issued by the device application at any time.

Notifications contain the identification of the device application that issued them (application ID), the command ID, and a result code. It may also contain additional data about the event that occurred.

The host application waits for notifications by calling *McpWait*. The MCP driver can block the application until a notification arrives from the device. When the driver receives a notification from the device, it completes the pending *McpWait* request providing the notification information received from the device.

If a notification comes while the host application is not waiting for it, the MCP driver buffers the notification until the application calls *McpWait*. The driver does not impose a limit on the number of the notifications that can wait at the same time.

The *McpWait* function uses a timeout (*dwTimeout*) value to indicate how it is to return to the host application.

Value	Meaning
-1	Return with the next buffered notification or wait indefinitely for the next notification to arrive if there are no notification buffered.
=0	Return immediately with the next buffered notification or an error if no notifications are currently buffered. Do not wait for a notification to arrive if no notifications are queued.
>0	Return with the next buffered notification or wait the specified number of milliseconds for the next notification to arrive. Returns with an error if no notifications arrive within the allotted time.

The function also can wait/check for new notifications, discard all previous notifications, or accept notifications that happened before the time of issuing the *McpWait* call. The device does not expect a response or any specific action from the host application in response to a notification message.

FUNCTIONS

Summary

The following section describes each function of the API. A short description of the purpose for each function is provided along with detailed descriptions of input and output parameters and return values. Refer to the constant definitions section of this document for function return value, property type and operation attribute constants.

The MCP API comprises the following functions:

Function	Description
McpEnum	Enumerate available MCP devices on the host computer.
McpOpen	Establish a communication channel with an MCP device.
McpClose	Close a communication channel with an MCP device.
McpReset	Reset the communication channel.
McpGet	Retrieve the value of a property from an MCP device, embedded application, or MCP driver.
McpSet	Modify the value of a property from an MCP device application or MCP driver device instance.
McpCall	Execute a command defined by an embedded application in an MCP device.
McpWait	Wait for a notification message from an MCP device.

MCPBUS Structure

The **MCPBUS** structure represents a common structure (bus) used to pass arguments to the MCP API functions. Not all members of this structure are used for every function; refer to each function description below to determine how the members are used.

```
typedef struct _MCPBUS
{
    DWORD    dwOperAttribute    ; // operation attribute
    DWORD    dwApplicationID    ; // application ID
    DWORD    dwCommandID       ; // command ID
    DWORD    dwResultCode      ; // command/notification result code
    LPSTR    lpszPropertyName  ; // pointer to property name
    DWORD    dwPropertyID      ; // property ID
    DWORD    dwPropertyType    ; // property type
    LPVOID   lpInBuffer        ; // pointer to input buffer
    DWORD    dwInBufferLen     ; // length of data in input buffer
    LPVOID   lpOutBuffer       ; // pointer to output buffer
    DWORD    dwOutBufferSize   ; // output buffer size
    DWORD    dwResponseLen     ; // length of response in output buffer
    DWORD    dwTimeout         ; // notification timeout value
    DWORD    dwContext         ; // device enumeration context
} MCPBUS, *PMCPBUS;
```

Members

dwOperAttribute

Operation Attribute. Some MCP API functions require this member to be set prior to invoking the function to distinguish how the operation is processed. The following table defines the attributes and for which functions they apply.

Function	Code	Meaning
McpWait	MCP_ATTR_NONE	Do not discard any queued notifications before responding.
	MCP_ATTR_WAIT_NEW	Discard any notifications that may have been queued before returning a response.
McpGet, McpSet	MCP_ATTR_NONE	MCP Device specific property
	MCP_ATTR_PROP_DRIVER	MCP Driver Device instance specific property

dwApplicationID

Application ID. This number identifies which application within the device the operation is directed. Each device has its own set of embedded applications and associated IDs. Refer to the device's Command Reference Manual for the available Application IDs and their function.

dwCommandID

Command ID. This number identifies which specific command to perform within the device application specified by the *dwApplicationID* member. Each device application has its own set of commands and associated IDs. Refer to the device's Command Reference Manual for the available Application/Command IDs and their function.

dwResultCode

Command Result Code. This value indicates whether the command was completed successfully, and if not, what was the reason for failure. It is used also by some notifications. Each device application / command has its own set of result codes. Refer to the device's Command Reference Manual for the possible result codes.

lpszPropertyName

Property Name. Properties can be identified by name or by numeric ID. When obtaining (*McpGet*) or modifying (*McpSet*) property values, this field contains a pointer to a null terminated string of 8-bit ANSI characters representing the name of the property being obtained or modified. Either this member or *dwPropertyID* is used to identify the property being obtained or modified. A driver device instance property can only be accessed by name. A device application property can only be accessed by property ID. *LpszPropertyName* should be set to null when accessing device application properties. If both *lpszPropertyName* and *dwPropertyID* are specified, the value specified by this member takes precedent.

dwPropertyID

Property ID. Device applications identify properties by numeric ID. When obtaining (*McpGet*) or modifying (*McpSet*) property values, this field contains the ID of the property being obtained or modified. Either this member or *lpszPropertyName* is used to identify the property being obtained or modified. If both *lpszPropertyName* and *dwPropertyID* are specified, the value specified by *lpszPropertyName* takes precedent. Therefore, *LpszPropertyName* should be set to null when accessing device application properties.

dwPropertyType

Property Type. When obtaining (*McpGet*) or modifying (*McpSet*) property values from the driver or device, this field specifies the data type of the property value identified by either *dwPropertyID* or *lpszPropertyName*. The lower four bits of the type specify the property type; the upper four bits are reserved for future extensions (e.g. to specify that the property is identified by name and not by numeric ID). The following types are defined: (upper 4 bits are used specially)

Code	Meaning
MCP_TYPE_NONE	No type.
MCP_TYPE_DWORD	32-bit unsigned integer
MCP_TYPE_STRING	Null terminated string of 8-bit ANSI characters.
MCP_TYPE_BOOLEAN	Boolean value (True or False)
MCP_TYPE_BINARY	Binary data

lpInBuffer

Pointer To Input (Request) Buffer. If a command contains data in addition to the Application and Command IDs, the host application places it in this buffer. The length of the data in this buffer is indicated by *dwInBufferLen*. Set to null if there is no additional data associated with this command. Refer to the device's Command Reference Manual for the definition of the content of this buffer. If this parameter is not null but *dwInBufferLen* is zero, the data will not be transmitted.

dwInBufferLen

Length of data (in bytes) contained in the input buffer pointed to by *lpInBuffer*. Set value to zero if there is no additional data associated with the command. This value will always be interpreted as zero if *lpInBuffer* is null.

lpOutBuffer

Pointer To Output (Response) Buffer. If there is data being returned in the response to a command, the MCP driver places the data in the output buffer pointed to by this member value. The size of the buffer pointed to by this value must be specified in the *dwOutBufferSize* member so that the MCP driver does not overrun the callers buffer. The length of the response is indicated by the value in *dwResponseLen*. Refer to the device's Command Reference Manual for the definition of the content of the buffer.

An error will be reported if this value is null and the device responds with data or if the amount of data returned by the device exceeded the length of this buffer (*dwOutBufferSize*).

dwOutBufferSize

Size of Output Buffer. The MCP driver needs to know the length (in bytes) of the callers Output Buffer (pointed to by *lpOutBuffer*) to avoid overrunning the buffer in the event that the response from the device exceeds the length of the Output Buffer.

The caller should always allocate an Output Buffer (pointed to by *lpOutBuffer*) and set this member to the length of the buffer with the expectation that data could be returned.

dwResponseLen

Length of data (in bytes) contained in the output buffer pointed to by *lpOutBuffer*. This value is returned upon the completion of a function call to indicate the length of the response contained in the output buffer. If the value is zero, there is no data associated with the response.

dwContext

Device Enumeration Context. When enumerating MCP devices with the *McpEnum* function, this member is used to indicate whether to restart the enumeration or continue with the next device. Set this value to zero initially to indicate restart enumeration with the first device. This value should be left alone on subsequent calls to *McpEnum* until the function returns indicating there are no more devices.

dwTimeout

Notification Wait Timeout. Used by the *McpWait* function to inform the MCP driver how long to wait for a notification event to occur before returning from the function. There is also a flag in the *dwOperAttribute* member which can be used to discard any and all notifications prior to processing the *McpWait* function.

Value	Meaning
-1	Return with the next buffered notification or wait indefinitely for the next notification to arrive if there are no notification buffered.
=0	Return immediately with the next buffered notification or an error if no notifications are currently buffered. Do not wait for a notification to arrive if no notifications are queued.
>0	Return with the next buffered notification or wait the specified number of milliseconds for the next notification to arrive. Return with an error if no notifications arrive within the allotted time.

Remarks

The MCPBUS is used as an input parameter to many of the MCP API functions. The MCP driver modifies only those members explicitly defined by the individual functions. For example, the API will not modify the *lpOutBuffer* or *dwOutBufferSize* members (the buffer that *lpOutBuffer* points to will of course be modified); therefore, the caller may set these values once and not need to reset them prior to invoking subsequent MCP API functions. The only structure members modified by an MCP API function are those explicitly identified as output parameters.

McpEnum

This function is used to enumerate (get the names of) all of the MCP compliant devices configured and connected on the host computer.

```
DWORD McpEnum( PMCPBUS pMCPBUS );
```

Parameters

pMCPBUS

Pointer to the MCPBUS structure used by the MCP API function to perform its operation.

The following structure members are used as input parameters.

Member	Meaning
dwContext	0 on first call (restart enumeration); value left unmodified on subsequent invocations to get the next device name.
lpOutBuffer	Pointer to storage buffer to receive the next device name.
dwOutBufferSize	Size of buffer pointed to by <i>lpOutBuffer</i>

The following structure members are used as output parameters.

Member	Meaning
dwContext	Enumeration Context, used in subsequent calls to <i>McpEnum</i> (do not modify)
lpOutBuffer	Buffer pointed to by this member contains the device name (null terminated)
dwResponseLen	Length of device name (including terminating null) in output buffer

Return Value

The returned value indicates the outcome of the enumeration function.

Value	Meaning
MCP_ST_OK	Success. The next device name is located in the callers Output Buffer pointed to by <i>lpOutBuffer</i> .
MCP_ST_NOT_FOUND	No more devices found or device not found
MCP_ST_OVERFLOW	The Output Buffer is not large enough to hold the returned value
MCP_ST_FAILED	Other error.

Remarks

The device name returned in the callers Output Buffer can be used in a subsequent *McpOpen* call. The first time this function is called, the *Context* member is set to zero indicating to restart the enumeration process. On subsequent calls, the *Context* member should be left unchanged so that the next device name can be returned. By calling this function in a loop, a host application can enumerate all devices.

McpOpen

This function is used to open the communication channel and establish communications with an MCP device. Use *McpClose* to close the channel.

```
DWORD McpOpen( LPSTR lpDeviceName, PHANDLE hpDevice );
```

Parameters

lpDeviceName

Pointer to a null terminated string which specifies the logical device name of the MCP device to which communications is to be established. This name could be a hard coded device name, a name obtained from the registry, or other configuration data, or more likely, a name returned from the *McpEnum* function.

hpDevice

Pointer to storage area to receive the open handle to the specified device (communications resource). This handle is used on subsequent calls to functions requiring communications with the MCP device. If the function fails, the value of *hpDevice* is not modified.

Return Value

The returned value indicates the outcome of the open function.

Value	Meaning
MCP_ST_OK	Success. The channel has been established.
MCP_ST_NOT_FOUND	Unknown device or device not found
MCP_ST_ACCESS_DENIED	Device is already open
MCP_ST_BAD_ACCESS	<i>lpDeviceName</i> or <i>hpDevice</i> parameter is an invalid pointer
MCP_ST_FAILED	Other error

Remarks

The host application uses *McpOpen* to open a handle to a communications resource. If the specified resource is currently being used, this function fails. Any thread of the process can use the handle returned by *McpOpen* to identify the resource in any of the functions that access the resource.

Some devices automatically synchronize to communication parameters used by the host. For example, some serial devices automatically synchronize to the baud rate the host is using. This synchronization occurs during the call to *McpOpen*. Once a device is synchronized, it will not

try to synchronize to new communication parameters until it is reset or power cycled. Therefore, if a device has been synchronized to a hosts communication parameters and the device is to be resynchronized using different communication parameters then the device should be reset or power cycled before trying to open the device with *McpOpen*. For example, if a serial port device has synchronized to a device instance with the baud rate property set to 9600 and the host now desires to Use *McpOpen* to synchronize the device to a device instance with the baud rate property set to 19200, then the device will have to be reset or power cycled first or else the *McpOpen* function will fail.

McpClose

This function is used to close a communication channel to an MCP device which was opened with the *McpOpen* function.

```
DWORD McpClose( HANDLE hDevice );
```

Parameters

hDevice

Handle to the specified device (communications resource) that is to be closed. This handle was returned from the *McpOpen* function.

Return Value

The returned value indicates the outcome of the close function.

Value	Meaning
MCP_ST_OK	Success. The channel has been closed.
MCP_ST_INVALID	Invalid device handle <i>hDevice</i>
MCP_ST_FAILED	Other error

Remarks

When the channel is closed, all pending requests are aborted.

To reset the channel, the *McpReset* function should be used instead of closing and reopening the channel.

McpReset

This function is used to reset the channel and reestablish communications with an MCP device. The result of reset is the same as closing and reopening the channel except the device handle remains the same.

```
DWORD McpReset( HANDLE hDevice );
```

Parameters

hDevice

Handle to the specified device (communications resource) that is to be reset. This handle was returned from the *McpOpen* function.

Return Value

The returned value indicates the outcome of the reset function.

Value	Meaning
MCP_ST_OK	Success. The channel has been reset.
MCP_ST_INVALID	Invalid device handle <i>hDevice</i>
MCP_ST_FAILED	Other error

Remarks

After opening a device, the host application can re-establish, or reset the channel by calling *McpReset*. This function cancels any outstanding commands and restores the communication channel to its initial state. Using this function is more desirable than closing and reopening the channel.

McpGet

This function is used to get the value of a property from the MCP driver MCP device application. Properties can be accessed through their property ID or property name.

```
DWORD McpGet( HANDLE hDevice, PMCPBUS pMCPBUS );
```

Parameters

hDevice

Handle to the specified device (communications resource) to get the property. This handle was returned from the *McpOpen* function.

pMCPBUS

Pointer to the MCPBUS structure used by the MCP API function to perform this operation. The following structure members are used as input parameters.

Member	Meaning
dwOperAttribute	MCP_ATTR_NONE - get an MCP device property MCP_ATTR_PROP_DRIVER - get an MCP driver property
dwApplicationID	Application ID (not used if MCP_ATTR_PROP_DRIVER is set)
lpSzPropertyName	Pointer to null terminated property name. NULL if using a Property ID
dwPropertyID	Property ID. Used only if <i>lpSzPropertyName</i> is NULL and MCP_ATTR_NONE is set in <i>dwOperAttribute</i> .
dwPropertyType	Property type. Used for type verification.
lpOutBuffer	Pointer to storage buffer to receive the property value
dwOutBufferSize	Size (in bytes) of storage buffer pointed to by <i>lpOutBuffer</i>

The following structure members are used as output parameters.

Member	Meaning
lpOutBuffer	The buffer pointed to by this member contains the returned property value
dwResponseLen	Length (in bytes) of property value returned in the buffer pointed to by <i>lpOutBuffer</i>

Return Value

The returned value indicates the outcome of the get property function.

Value	Meaning
MCP_ST_OK	Success. The property value is located in the output buffer pointed to by <i>lpOutBuffer</i> .
MCP_ST_NOT_FOUND	The property could not be found
MCP_ST_INVALID	Invalid device handle <i>hDevice</i>
MCP_ST_REFUSE	The data type does not match the expected data type.
MCP_ST_OVERFLOW	The output buffer is not large enough to hold the returned value
MCP_ST_FAILED	Other error.

Remarks

The *dwOperAttribute* member is used to indicate whether the property is being accessed from the MCP driver (MCP_ATTR_PROP_DRIVER) or MCP device applications (MCP_ATTR_PROP_DEVICE). One or the other attributes must be indicated.

The MCP device applications identify properties only by numeric IDs due to the limited amount of memory and processing power. *lpzPropertyName* must be set to null when accessing device application properties. Refer to the device's Command Reference Manual for the available property Ids, application Ids and their function.

The MCP driver properties can be specified by name only. See Device instance properties section for a list of these.

McpSet

This function is used to set the value of a property in the MCP driver, MCP device or MCP device application. Properties can be accessed through their property ID or property name.

```
DWORD McpSet( HANDLE hDevice, PMCPBUS pMCPBUS );
```

Parameters

hDevice

Handle to the specified device (communications resource) for the property being set. This handle was returned from the *McpOpen* function.

pMCPBUS

Pointer to the MCPBUS structure used by the MCP API function to perform this operation. The following structure members are used as input parameters.

Member	Meaning
dwOperAttribute	MCP_ATTR_NONE - set an MCP device property MCP_ATTR_PROP_DRIVER - set an MCP driver property
dwApplicationID	Application ID (not used if MCP_ATTR_PROP_DRIVER is set)
lpszPropertyName	Pointer to null terminated property name. NULL if using a Property ID
dwPropertyID	Property ID. Used only if <i>lpszPropertyName</i> is NULL and MCP_ATTR_NONE is set in <i>dwOperAttribute</i> ..
dwPropertyType	Property type. Used for type verification. Never use MCP_TYPE_NONE.
lpInBuffer	Pointer to storage buffer containing the property value to be set
dwInBufferLen	Length (in bytes) of property value contained in the buffer pointed to by <i>lpInBuffer</i>

There are no structure members used as output parameters.

Return Value

The returned value indicates the outcome of the enumeration function.

Value	Meaning
MCP_ST_OK	Success. The property value has been set
MCP_ST_NOT_FOUND	The property could not be found
MCP_ST_INVALID	Invalid device handle <i>hDevice</i>
MCP_ST_REFUSE	The data type does not match the expected data type.
MCP_ST_OVERFLOW	The property value is too large
MCP_ST_OUT_OF_RANGE	The property value is not within the range of the allowed values
MCP_ST_BAD_ACCESS	There has been an attempt to set a "Read Only" property
MCP_ST_FAILED	Other error.

Remarks

The *dwOperAttribute* member is used to indicate whether the property is being accessed from the MCP driver (MCP_ATTR_PROP_DRIVER) or MCP device applications (MCP_ATTR_NONE).

The MCP device applications identify properties only by numeric IDs due to the limited amount of memory and processing power. *lpszPropertyName* must be set to null when accessing device application properties. Refer to the device's Command Reference Manual for the available property IDs and their function.

The MCP driver properties can be specified by name only. See Device instance properties section for a list of these.

McpCall

This function is used to execute an MCP command request. Commands can be issued to the MCP device or MCP device application embedded within the MCP device.

```
DWORD McpCall( HANDLE hDevice, PMCPBUS pMCPBUS );
```

Parameters

hDevice

Handle to the specified device (communications resource) that is to execute the command. This handle was returned from the *McpOpen* function.

pMCPBUS

Pointer to the MCPBUS structure used by the MCP API function to perform this operation. The following structure members are used as input parameters.

Member	Meaning
dwApplicationID	Device Application ID.
dwCommandID	Command Code. Specific command to be executed by the application.
lpInBuffer	Pointer to storage buffer containing addition data associated with the command.
dwInBufferLen	Length (in bytes) of data contained in the Input Buffer pointed to by <i>lpInBuffer</i>
lpOutBuffer	Pointer to storage buffer to receive the response from the application
dwOutBufferSize	Size (in bytes) of output storage buffer pointed to by <i>lpOutBuffer</i>

The following members are used as output parameters.

Member	Meaning
dwResultCode	Result Code. Refer to the device's Command Reference for possible values.
lpOutBuffer	Pointer to storage buffer containing the response from the application
dwResponseLen	Length (in bytes) of the response returned in the buffer pointed to by <i>lpOutBuffer</i>

Return Value

The returned value indicates the outcome of the call function.

Value	Meaning
MCP_ST_OK	Success. The command was sent and response received.
MCP_ST_OVERFLOW	The Output Buffer is not large enough to hold the response
MCP_ST_FAILED	Other error.

Note

The return value of this function does not necessarily indicate the outcome of the command; it simply indicates whether the request was sent to the device and a response received successfully. To determine the outcome of the specific command sent to the application, the `dwResultCode` structure member needs to be interrogated.

Remarks

Host applications access device functions by communicating with specific device applications running on the device. Each device application is identified by a numeric application ID that is unique for the device. Every device application defines the command set that it understands and can process. Each command is identified by a numeric command ID and may contain additional data needed to process the command. The command ID is unique for the given application.

The device responds to commands within short, finite amount of time. The response contains a `dwResultCode` that indicates whether the command was completed successfully, and if not, what was the reason for failure. The response may also contain additional data to be returned to the host application.

Refer to MCP device's Command Reference Manual to determine the allowable Application IDs, Command IDs, and possible Result Codes.

McpWait

This function is used to get the next notification which the device has sent. When notifications are received from a device, they are queued for subsequent retrieval by the host application using this function.

```
DWORD McpWait( HANDLE hDevice, PMCPBUS pMCPBUS );
```

Parameters

hDevice

Handle to the specified device (communications resource) for notifications. This handle was returned from the *McpOpen* function.

pMCPBUS

Pointer to the MCPBUS structure used by the MCP API function to perform this operation. The following structure members are used as input parameters.

Member	Meaning
dwOperAttribute	MCP_ATTR_NONE - do not discard buffered notifications MCP_ATTR_WAIT_NEW - discard all buffered notifications; wait for the next new notification from the device
dwTimeout	Timeout value for notification to arrive: -1 Return immediately with next notification if one is buffered or wait indefinitely for the next one to arrive from the device. =0 Return immediately with next buffered notification or error if no notifications are buffered. Note: MCP_ATTR_WAIT_NEW should not be set if specifying this value for the timeout. >0 Return immediately with the next notification if one is buffered or wait this number of milliseconds for the next one to arrive.
lpOutBuffer	Pointer to storage buffer to receive notification information
dwOutBufferSize	Size (in bytes) of storage buffer pointed to by <i>lpOutBuffer</i>

The following structure members are used as output parameters.

Member	Meaning
dwApplicationID	ID of the device application generating the notification
dwCommandID	ID of the notification (application specific)
dwResultCode	Result Code.
lpOutBuffer	The buffer pointed to by this member contains the returned notification data
dwResponseLen	Length (in bytes) of notification data returned in buffer pointed to by <i>lpOutBuffer</i>

Return Value

Value	Meaning
MCP_ST_OK	Success. Notification data is in the Output Buffer.
MCP_ST_NOT_FOUND	No notifications found (valid only if dwTimeout was zero)
MCP_ST_TIMEOUT	No notification arrived within the specific dwTimeout period

Remarks

This function can be used to retrieve the next notification message which the driver has received and buffered from device. How the driver is returned from this function call depends on the *dwTimeout* member:

- 1) If one just wants to poll the driver to determine whether there are any notifications in the queue, 0 would be specified so that the function would return immediately with the next notification if one was queued or with an error code indicating that no notifications were buffered.
- 2) If one wanted to retrieve the next queued notification or wait (for no longer than a finite period of time) until the next notification arrives, a positive non-zero value would be specified.
- 3) If one wanted to retrieve the next queued notification or wait indefinitely until the next notification arrives, a "-1" value would be specified.

If the caller wishes to discard all queued notifications and only wait for the next new notification message, the MCP_ATTR_WAIT_NEW flag should be set in the *dwOperAttribute* prior to calling the *McpWait* function.

If this function returns successfully, the *dwApplicationID* indicates the device application which issued the notification, *dwCommandID* indicates the specific condition which caused the notification and *dwResultCode* indicates the result code. Additional data may be returned in the Output Buffer. Refer to the device's Command Reference Manual for the available Application IDs and their defined notifications.

CONSTANT DEFINITIONS

Operation Attributes

```
#define MCP_ATTR_NONE          0    // No attributes specified
#define MCP_ATTR_WAIT_NEW      1    // Discard any notifications that may
                                   // have been queued before returning
                                   // a response.
#define MCP_ATTR_PROP_DRIVER   2    // MCP Driver specific property
```

Property Types

```
#define MCP_TYPE_NONE          0    // No type.
#define MCP_TYPE_DWORD         1    // 32-bit unsigned integer
#define MCP_TYPE_STRING         2    // Null terminated string of 8-bit
                                   // ANSI characters.
#define MCP_TYPE_BOOLEAN       3    // Boolean value (True=1 or False=0)
#define MCP_TYPE_BINARY        4    // Binary data
```

Function Return Values

```
#define MCP_ST_OK              0    // successful operation

// resource manipulation
#define MCP_ST_ALLOC           20   // can't alloc (for variable size alloc/pool)
#define MCP_ST_NO_ROOM        21   // can't alloc fixed sz entry in fixed sz pool
#define MCP_ST_OVERFLOW       22   // value, buffer, whatever may overflow
#define MCP_ST_UNDERFLOW     23   // value, buffer, counter, whatever may
                                   // underflow
#define MCP_ST_EMPTY          24   // queue/smith is empty - no elements
#define MCP_ST_FULL           25   // queue/smith is full
#define MCP_ST_EOF            26   // end of file

// valchk
#define MCP_ST_INVALID        30   // value not valid in operation context
#define MCP_ST_BAD_VALUE     31   // value not valid in module's context
#define MCP_ST_OUT_OF_RANGE  32   // value out of range
#define MCP_ST_NULL_PTR      33   // NULL ptr is not valid in operation context
#define MCP_ST_BAD_SYNTAX    34   // text/formatted data syntax error
#define MCP_ST_BAD_NAME     35   // name invalid in module's context

// severe errors
#define MCP_ST_UNEXPECTED    40   // unexpected condition of parameter or
                                   // external object
```

Section 5. Application Programmable Interface

```
#define MCP_ST_PANIC          41    // unexpected condition of self or internal
                                // object
#define MCP_ST_DEADLOCK      42    // deadlock detected
#define MCP_ST_STACK_OVERFLOW 43    // stack overflow detected / no enough stack

// failures */
#define MCP_ST_REFUSE        45    // oper rejected voluntarily (inappropriate
                                // state)
#define MCP_ST_NO_ACTION     46    // oper requires no action (eg close closed
                                // file)
#define MCP_ST_FAILED        47    // operation failed / not successful

// state check
#define MCP_ST_NOT_INITED    50    // object is not initialized
#define MCP_ST_NOT_ACTIVE    51    // oper allowed only when object is active
#define MCP_ST_NOT_OPEN      52    // oper allowed only when object is open
#define MCP_ST_NOT_CONNECTED 53    // oper allowed only when object is
                                // connected
#define MCP_ST_NOT_CONSTRUCTED 54  // object construction not completed

// i/o system
#define MCP_ST_IOERR         60    // I/O error - file system / peripherals error
#define MCP_ST_BAD_CHKSUM    61    // wrong chksum

// set-related
#define MCP_ST_NOT_FOUND     62    // requested item not found
#define MCP_ST_DUPLICATE     63    // duplicated item is not allowed

// critical section / privileges
#define MCP_ST_BUSY          70    // in critical section (dynamic, short period)
#define MCP_ST_ACCESS_DENIED 71    // temporary lack of access (dynamic, long
                                // period)
#define MCP_ST_PRIVILEGE     72    // privilege violation (static)
#define MCP_ST_SCOPE_VIOLATION 73  // life, validity, etc. scope rules violated
#define MCP_ST_BAD_ACCESS    74    // access type not appropriate / not possible

// operation status
#define MCP_ST_PENDING       80    // operation is pending
#define MCP_ST_TIMEOUT       81    // timeout has expired
#define MCP_ST_CANCELED      82    // operation canceled on user's request
#define MCP_ST_CANCELLED     MCP_ST_CANCELED
#define MCP_ST_ABORTED       83    // op. aborted on system/module's request
#define MCP_ST_RESET         84    // object has been orderly reset
```

```
#define MCP_ST_CLEANUP          85    // object is about to be destroyed, now in
                                   // cleanup
#define MCP_ST_OVERRIDE        86    // operation/entity was overridden
#define MCP_ST_POSTPONE        87    // postpone operation: inappropriate state

// API
#define MCP_ST_CANT_BIND        90    // can't bind / resolve name, handle, etc.
#define MCP_ST_API_ERROR        91    // invalid function request, etc.
#define MCP_ST_WRONG_VERSION    92    // incompatible version of data/code detected
#define MCP_ST_NOT_IMPLEMENTED  93    // feature not implemented
#define MCP_ST_NOT_SUPPORTED    94    // feature not supported

// object model & environment
#define MCP_ST_BAD_OID          100   // invalid object id
#define MCP_ST_BAD_MESSAGE      101   // object does not handle this message
```

FUNCTION PROTOTYPES

```
DWORD WINAPI McpEnum (MCPBUS *pMCPBUS);
DWORD WINAPI McpOpen (LPSTR lpDeviceName, HANDLE *hpDevice);
DWORD WINAPI McpClose (HANDLE hDevice);
DWORD WINAPI McpReset (HANDLE hDevice);
DWORD WINAPI McpGet (HANDLE hDevice, MCPBUS *pMCPBUS);
DWORD WINAPI McpSet (HANDLE hDevice, MCPBUS *pMCPBUS);
DWORD WINAPI McpCall (HANDLE hDevice, MCPBUS *pMCPBUS);
DWORD WINAPI McpWait (HANDLE hDevice, MCPBUS *pMCPBUS);
```

INDEX

A

- Adding a Device Instance from Command Line – RS-232.....34
- Adding a Device Instance from Command Line – USB.....34
- Adding a Device Instance From Windows Based Application (WIN NT, 2000, XP) – RS-232..35
- Adding a Device Instance from Windows Based Application (WINDOWS 2000, XP) – USB ..35
- Application Programmable Interface.....47

C

- Commands.....49
- Communication System Requirements.....3
- Computer System Requirements.....3
- Constant Definitions.....70

D

- Device Channel Life Cycle.....48
- Device Instance Properties.....40
- Displaying Device Instance Properties from Command Line.....36
- Displaying the List of Device Instances from Command Line.....36
- DLL (Direct Library Link).....1

E

- End User Licensing Agreement.....7

F

- Function Prototypes.....72
- Function Return Values.....70
- Functions, API.....51

G

- Generic Properties.....40

I

- Installation.....5
- Installing The MCP Driver.....5

K

- Kernel Mode Driver (.SYS).....1
- Kernel Mode MCP Driver.....9

L

- Licensing Agreement.....7

M

- MCP driver (MagTek Communication Protocol) 1
- MCP resources.....8
- MCP Resources Installation.....8
- MCPBUS Structure.....52
- McpCall.....65
- MCPCFG Command Summary.....33
- McpClose.....59
- McpEnum.....56
- McpGet.....61
- McpOpen.....57
- McpReset.....60
- McpSet.....63
- McpWait.....67
- Modifying a Device Instance Properties from Windows Based Application (WINDOWS 2000, XP) – RS-232 and USB.....35
- Modifying Device Instance Properties From Command Line.....37
- Modifying Device Instance Properties From Windows Based Application.....38

N

- Notifications.....50

O

- Operating System Requirements.....3
- Operation Attributes.....70

P

- Properties, Device Application.....48
- Property Types.....70

R

- Removing a Device Instance from a Command Line – RS-232.....36
- Removing a Device Instance from a Command Line – USB.....36
- Removing a Device Instance from Windows Based Application (WINDOWS 2000, XP) – RS232.....35
- Removing a Device Instance from Windows Based Application (WINDOWS 2000, XP) – USB.....35
- Restarting the MCP Driver.....38

MagTek Communications Protocol, Driver Reference Manual

S

Serial Port Properties42
Stopping the MCP Driver From Command Line
.....38
System Requirements3

T

Trace Log45
Typical Operation, API47

U

User Mode MCP Driver 9

W

Windows 2000 3
Windows 98, 3
Windows Configuration Utility 39
Windows Driver Model (WDM) 1
Windows Me 3
Windows NT 4.0..... 3
Windows XP 3