USER MANUAL

DMCWin32
Galil Windows API
Tool Kit

MAUNAL REV 2.1

By Galil Motion Control, Inc.

Galil Motion Control, Inc.
3750 Atherton Road
Rocklin, California 95765
Phone: (916) 626-0101
Fax: (916) 626-0102
Internet Address: support@galilmc.com
URL: www.galilmc.com

Rev 1/04

OVERVIEW **5**

PROGRAMMING MODEL 6
Step 1 Register Controller *6*
Step 2 Declare Functions *6*
Step 3 Open Communication *6*
Step 4 Download Program *6*
Step 5 Send Commands *6*
Step 6 Close Communication *6*
VISUAL BASIC 7
Declaration Files *7*
Example 1: Sending Commands in VB *7*
Example 2: Downloading Programs in VB *7*
C/C++ 9
Declaration Files 9
DMCCOM.H 9
DMCWIN.H (C++ only) 9
DMCMLIB.H 9
Linking Your Application with the Galil DLLs 9
Example 1: Sending Commands Using DMCCOM.H *9*
Example 2: Downloading Programs using DMCCOM.H *10*
Example 3: Sending Commands Using the Class Library (C++) *10*

APPLICATION PROGRAMMING INTERFACE (DMCCOM.H) **11**

DLL API LIST FOR DMCCOM.H 11
Communication *11*
Registry *12*
Binary Commands *12*
Data Record *12*
Other *12*
ERROR CODES 13
API FUNCTION DETAILS 14
DMCAddGalilRegistry and DMCAddGalilRegistry2 ..... *14*
DMCArrayDownload *14*
DMCArrayUpload *15*
DMCAssignIPAddress *15*
DMCBinaryCommand *15*
DMCChangeInterruptNotification *16*
DMCClear *16*
DMCClose *16*
DMCCommand *16*
DMCCommand_AsciiToBinary *17*
DMCCommand_BinaryToAscii *17*
DMCCompressFile *17*
DMCCopyDataRecord *18*
DMCDeleteGalilRegistry *18*
DMCDiagnosticsOff *18*
DMCDiagnosticsOn *18*
DMCDownloadFile *18*
DMCDownloadFirmwareFile *19*
DMCDownloadFromBuffer *19*
DMCEditRegistry *19*
DMCEnumGalilRegistry and DMCEnumGalilRegistry2 ..... *19*
DMCError *20*

DMCFastCommand *20*
DMCFile_AsciiToBinary *20*
DMCFile_BinaryToAscii *20*
DMCGetAdditionalResponse *22*
DMCGetAdditionalResponseLen *22*
DMCGetControllerDesc *22*
DMCGetDataRecordByItemId *22*
DMCGetDataRecordConstPointer *23*
DMCGetDataRecordItemOffsetById *23*
DMCGetDataRecordConstPointerArray *23*
DMCGetDataRecordArray *24*
DMCGetDataRecordRevision *24*
DMCGetDataRecordSize *24*
DMCGetGalilRegistryInfo and DMCGetGalilRegistryInfo2 ..... *24*
DMCGetHandle *25*
DMCGetTimeout *25*
DMCGetUnsolicitedResponse *25*
DMCMasterReset *25*
DMCModifyGalilRegistry and DMCModifyGalilRegistry2 ..... *25*
DMCOpen *26*
DMCOpen2 ..... *26*
DMCReadData *26*
DMCReadSpecialConversionFile *26*
DMCRefreshDataRecord *27*
DMCRegisterPnpControllers *27*
DMCReset *27*
DMCSelectController *27*
DMCSendBinaryFile *27*
DMCSendCW2OnClose *28*
DMCSendFile *28*
DMCSetTimeout *28*
DMCStartDeviceDriver *28*
DMCStopDeviceDriver *28*
DMCUploadFile *29*
DMCUploadToBuffer *29*
DMCVersion *29*
DMCWaitForMotionComplete *29*
DMCWriteData *29*

ADVANCED MOTION FUNCTIONS (DMCMLIB) 31

DMCELLIPSE 31
DMCSPLINE 33
DMCSCURVE 36
DMCHELIX 37
DMCGENERALTUNING 39
DMCAUTOTUNING 40

APPLICATION PROGRAMMING TOPICS 41

INTRODUCTION 41
DOWNLOADING PROGRAMS TO THE CONTROLLER 41
CONFIGURING THE GALIL REGISTRY 41
INTERRUPT HANDLING 42
DIAGNOSTICS 43
MANAGING THE TIME-OUT 43
LOW-LEVEL I/O 43

MULTIPLE THREAD APPLICATIONS 43
WAITING FOR MOTION TO COMPLETE 43
BINARY COMMUNICATIONS 44
DATA RECORD ACCESS 45
D *45*
DMC-1700 ....*45*
DMC-1800 ....*45*
Data Record Structure *46*
Data Record API Examples *47*
DMCGetDataRecordByItemId 47
DMCCopyDataRecord 48
Advanced: DMCGetDataRecordConstPointer 49
Advanced: DMCGetDataRecordArray (DMC-1700, 1800) 50
Advanced: QR Command (DMC-14x5/6, 18x2, 2xxx, 3xxx) 51
DISTRIBUTING YOUR APPLICATION 52
WINDOWS 98 SECOND EDITION, ME, 2000, WINDOWS XP 52
WINDOWS NT 4.0 52
MICROSOFT RUN-TIME FILES 52

Overview

This manual describes DMCWin32: the Galil Windows API (Application Programming Interface). It include DLLs (dynamic link libraries) for developing programs on a PC to communicate with a Galil controller as well as sample programs and utilities to help design a software interface. Any Windows programming environment that can interface with DLLs can be used with DMCWin32, including Microsoft Visual Basic, Microsoft Visual C++, and National Instruments LabView.

This section describes the basic requirements for programming in C, C++, and Visual Basic, along with simple programming examples. The remaining sections document each function in the API and provide more advanced program examples. For additional examples, refer to the source code contained in the C, CPP and VB directories located under /DMCWIN.

The DMCWin32 DLL files are used within Windows development environments to create programs that communicate with Galil controllers. Two libraries are included with the tool kit:

1) **DMCCOM.H**: This library contains the basic communication functions.

2) **DMCMLIB.H**: This library contains special advanced motion-related functions.

Using these DLLs, a program can send commands, upload and download programs, and check the status of any controller in a system. Programming with the Galil DLLs is source code compatible across the following Windows environments: 98SE, ME, NT4, 2000, and XP.

Programming Model

No matter what platform is used to develop programs, the following model can be used as a reference. Although there are many functions available, only a small number are needed for most programs.

Step 1 Register Controller

Before communicating with a Galil controller, it must be entered in the Galil registry. This is usually done through the standard Galil software (SmartTerm, WSDK). This registry contains information on each controller in the system including the type of interface and address. PCI and USB controllers use Plug and Play to automatically enter information in the registry and all others have to be entered manually.

Step 2 Declare Functions

Before any of the functions can be used, they must be declared. Galil provides declaration files for Visual Basic, C, and C++. See the details below for each development environment.

Step 3 Open Communication

Start a communication session with the controller using the DMCOpen function. Here, the controller registry number is passed to the function and a handle is returned. Use this handle in all subsequent Galil function calls.

Step 4 Download Program

To download a Galil language program (if one is required) to the controller, use the DMCDownloadFile function. Here, a DMC program that resides on the host computer's hard disk is downloaded into the controller. Once downloaded, the program will run on the controller if the DMC command XQ is sent using the DMCCommand function.

Step 5 Send Commands

To send live commands to the controller, use the DMCCommand function. Most commands can be passed to the controller in this way. The response from the controller is returned as a string. Care should be taken not to send commands that could cause a time out. These include trip points like AMX and AIX and DMC commands that do not produce a response from the controller like DL. These special cases can be handled with other functions like DMCFastCommand or DMCWriteData. See the API details for more information.

Step 6 Close Communication

To end a communication session with the controller use the DMCClose function. This should be done at the end of the program. If DMCClose is not called Windows will not release the handle and associated resources provided in the DMCOpen function. When all the available handles are used Windows will produce an error during the DMCOpen call. During program development the DMCClose function may not be called due to crashes or aborts that cause the PC program to stop abnormally. It may be necessary to reboot the PC to release the handles under these conditions.

Visual Basic

Declaration Files Before using the DLL functions, add the module file included in the \dmcwin\vb directory named **DMCCOM40.BAS**. This module declares the functions, making them available for the VB project. To add this file, select 'Add Module' from the 'Project' menu in VB5/6.

Example 1: Sending Commands in VB

Most commands are sent to the controller with the DMCCommand function. This function allows any Galil command to be sent from VB to the controller. The DMCCommand function will return the response from the controller in a string. Before sending any commands, the DMCCOpen function must be called. This function establishes communication with the controller and is called only once.

The following code illustrates the use of DMCOpen and DMCCommand. Here, the controller is sent the command TPX when a Command Button is pressed. The response is placed in a text box. To use this example, start a new Visual Basic project, place a Text Box and a Command Button on a Form, add the **DMCCOM40.BAS** module, and type the following code:

```
Dim Controller As Integer
Dim hDmc As Long
Dim RC As Long
Dim ResponseLength As Long
Dim Response As String * 256

Private Sub Command1_Click()
   RC = DMCCommand(hDmc, "TPX", Response, ResponseLength)
   Text1.Text = Val(Response)
End Sub

Private Sub Form_Load()
   ResponseLength = 256
   Controller = 1   RC = DMCOpen(Controller, 0, hDmc)
End Sub

Private Sub Form_Unload(Cancel As Integer)
   RC = DMCClose(hDmc)
End Sub
```

Where: 'Controller' is the number for the controller in the registry
'hDmc' is the Windows handle used to identify the controller. It is returned by DMCOpen.
'RC' is the return code for the function
'ResponseLength' is the response string length must be set to the size of the response string
'Response' is the string containing the controller response to the command

Example 2: Downloading Programs in VB

To download a program to the controller, use either DMCDownloadFile or DMCDownloadFromBuffer. Once the file is downloaded to the controller, send the XQ command to start the program. The following example downloads a simple program to the controller and executes it. The code is intended to be executed from within a Visual Basic Module and does not require a Form to execute.

```
Dim Controller As Integer
Dim hDmc As Long
Dim RC As Long
Dim ResponseLength As Long
Dim Response As String * 256
Dim Buffer As String

Private Sub Main()
       ResponseLength = 256
              Controller = 1
              Buffer = "#A;PR1000;BGX;AMX;EN"
```

```
                RC = DMCOpen(Controller, 0, hDmc)
                RC = DMCDownloadFromBuffer(hDmc, Buffer, "")
                RC = DMCCommand(hDmc, "XQ", Response, ResponseLength)
        RC = DMCClose(hDmc)
        End
End Sub
```

Note: See the DLL function descriptions later in this manual for more functions.

C/C++

The DLL functions can be used as included functions or through a class library.

Declaration Files

DMCCOM.H

All Galil communications programs written in C **must** include the **DMCCOM.H** file. This allows a program to access the DLL functions through the declared function calls. The DMCCOM.H header file is located in the \dmcwin\include directory.

DMCWIN.H (C++ only)

C++ programs can use the DMCCOM.H functions or use the class library defined in **DMCWIN.H**.
C++ programs that use the class library need the files DMCWIN.H and DMCWIN.CPP, which contain the class definitions and implementations respectively. These can be found in the \dmcwin\cpp directory.

DMCMLIB.H

These advanced motion functions are not available as a C++ class library and are independent from the DMCCOM functions. To use, include the file **DMCMLIB.H** located in the \dmcwin\include directory.

Linking Your Application with the Galil DLLs

To use the functions in DMCCOM.H, link your application with dmc32.lib. To use the functions in DMCMLIB.H, link your application with DMCMLIB.lib. These library (.lib) files can be found in the \dmcwin\lib directory.

Example 1: Sending Commands Using DMCCOM.H

To initiate communication, declare a variable of type HANDLEDMC (a long integer) and pass the address of that variable in the DMCOpen function. If the DMCOpen function is successful, the variable will contain the handle to the Galil controller which is required for all subsequent function calls. The following simple example program written as a Visual C++ console application tells the controller to move the X axis 1000 encoder counts. Remember to add DMC32.LIB to your project prior to compiling.

```
#include <windows.h>
#include <dmccom.h>

long rc;
HANDLEDMC hDmc;
HWND hWnd;

int main(void)
{
        // Connect to controller number 1
        rc = DMCOpen(1, hWnd, &hDmc);
        if (rc == DMCNOERROR)
        {
          char szBuffer[64];
          // Move the X axis 1000 counts
          rc = DMCCommand(hDmc, "PR1000;BGX;", szBuffer, sizeof(szBuffer));

          // Disconnect from controller number 1 as the last action
          rc = DMCClose(hDmc);
        }
        return 0;
}
```

Example 2: Downloading Programs using DMCCOM.H

The following example downloads a file from the hard drive using DMCDownloadFile() and executes it by sending the command XQ.

Note: Files can also be downloaded from a buffer by using DMCDownloadFromBuffer().

```
long rc;
HANDLEDMC hDmc;
HWND hWnd;
char szBuffer[64];
char filename[] = "c:\\dmcwin\\yourfile.dmc";

int main(void)
{
        // Connect to controller number 1
        rc = DMCOpen(1, hWnd, &hDmc);
        //Download file
        rc = DMCDownloadFile(hDmc, filename, NULL);
        //Start program on controller at label 'A'
        rc = DMCCommand(hDmc, "XQ#A", szBuffer, sizeof(szBuffer));

        //Close communication
        rc = DMCClose(hDmc);

        return 0;
}
```

In this example the label was set to NULL, causing the existing program to be overwritten. If a label is used in place of NULL, the DMCDownloadFile will try to find that label within the program already present on the controller and start the download from there. A "#" will append the program to the existing program.

Note: All existing programs must be halted before an upload or download occurs.

Example 3: Sending Commands Using the Class Library (C++)

Most of the API functions are available in the class library. An object of type CDMCWin is declared which allows access to the class functions. When the class constructor is called, the communication channel is opened. To send commands, use the .command() function. Unlike the function calls using DMCCOM.H, the handle does not need to be passed to all subsequent class calls. Make sure to call the .close() function before ending the program so Windows can release the handle.

This example tells the controller to start jogging the Y axis at 10000 counts per second:

```
CDMCWin controller(1, hWnd, 0);
controller.Command("JG ,10000; BGY", szBuffer, sizeof(sxBuffer));
```

Application Programming Interface (DMCCOM.H)

DLL API List for DMCCOM.H

The following DLL functions are available with the DMCCOM.H header file. For more information see the detailed descriptions later in this manual.

Communication

DMCOpen	Open communications with interrupt support	Page 26
DMCOpen2	Open communications with interrupt support	Page 26
DMCClose	Close communications	Page 16
DMCSendCW2OnClose	Clears most significant bit of unsolicited messages	Page 28
DMCCommand	Send a command	Page 16
DMCFastCommand	Send special commands	Page 20
DMCArrayDownload	Download an array	Page 14
DMCArrayUpload	Upload an array	Page 15
DMCDownloadFile	Download a file from hard disk	Page 18
DMCUploadFile	Upload file to hard disk	Page 29
DMCCompressFile	Compress a Galil language program file	Page 17
DMCDownloadFromBuffer	Download file from buffer	Page 19
DMCUploadToBuffer	Upload file to buffer	Page 29
DMCSendFile	Send a file	Page 28
DMCDownloadFirmwareFile	Download Firmware	Page 19
DMCGetAdditionalResponse	Read long response	Page 22
DMCGetAdditionalResponseLen	Read long response length	Page 22
DMCGetUnsolicitedResponse	Read card messages	Page 25
DMCReadData	Low level read function	Page 26
DMCWriteData	Low level write function	Page 29
DMCClear	Clear FIFO	Page 16
DMCGetTimeout	Return the current timeout	Page 25
DMCSetTimeout	Set timeout	Page 28
DMCWaitForMotionComplete	Wait for motion complete	Page 29
DMCMasterReset	Master reset controller	Page 25
DMCReset	Reset controller	Page 27
DMCVersion	Get firmware version	Page 29

Registry

DMCAddGalilRegistry and DMCAddGalilRegistry2	Add a controller to the registry	Page 14
DMCDeleteGalilRegistry	Delete a controller from registry	Page 18
DMCEditRegistry	Registry Dialog (Requires DMCReg.ocx)	Page 19
DMCEnumGalilRegistry and DMCEnumGalilRegistry2	Read entire registry	Page 19
DMCGetGalilRegistryInfo and DMCGetGalilRegistryInfo2	Read registry for one controller	Page 24
DMCModifyGalilRegistry and DMCModifyGalilRegistry2	Modify a registry entry	Page 25
DMCRegisterPnpControllers	Update registry for PNP controllers (OBSOLETE)	Page 27
DMCGetControllerDesc	Get a description of the controller from the registry	Page 22

Binary Commands

DMCBinaryCommand	Send binary command	Page 15
DMCCommand_AsciiToBinary	Convert ASCII command to binary	Page 17
DMCCommand_BinaryToAscii	Convert a binary command to ASCII	Page 17
DMCFile_AsciiToBinary	Convert file of ASCII commands to binary	Page 20
DMCFile_BinaryToAscii	Convert file of binary commands to ASCII	Page 20
DMCReadSpecialConversionFile	Special conversion file	Page 26
DMCSendBinaryFile	Send a binary file	Page 27

Data Record

DMCRefreshDataRecord	Request a new data record	Page 27
DMCGetDataRecordByItemId	Return part of the data record (preferred method)	Page 22
DMCGetDataRecordItemOffsetById	Get data record item offset	Page 23
DMCGetDataRecordConstPointer	Get pointer to data record	Page 23
DMCGetDataRecordRevision	Get version of record	Page 24
DMCGetDataRecordArray	Return all cached data record (1700, 1800 only)	Page 24
DMCGetDataRecordConstPointerArray	Return a const pointer to all cached data records (1700, 1800 only)	Page 23
DMCCopyDataRecord	Copy data record into a structure	Page 18
DMCGetDataRecordSize	Returns number of bytes in data record	Page 24

Other

DMCChangeInterruptNotification	Notify for interrupt by handle or thread	Page 16
DMCDiagnosticsOff	Stop diagnostics	Page 18
DMCDiagnosticsOn	Start diagnostic file	Page 18
DMCError	Read error message	Page 20
DMCGetHandle	Return the controller handle	Page 25
DMCSelectController	Selection Dialog	Page 27
DMCStartDeviceDriver	Starts Galil device driver (NT4 only)	Page 28
DMCStopDeviceDriver	Stops Galil device driver (NT4 only)	Page 28
DMCAssignIPAddress	Assign the IP Address to an Ethernet controller	Page 15

Error Codes

All functions return an error code. If the function returned DMCNOERROR (0), the function completed successfully. An error code less than 0 is a local error (see the error codes above). An error code greater than 0 is an Win32 API error (32-bit DLLs). These are documented in the Win32 Programming Reference.

0	DMCNOERROR	No error occurred.
	DMCERROR_TIMEOUT	A time-out occurred while waiting for a response from the Galil controller.
-2	DMCERROR_COMMAND	There was an error with the command sent to the Galil controller. The full message depends on the error code returned from the Galil controller.
-3	DMCERROR_CONTROLLER	The Galil controller could not be found in Windows registry.
-4	DMCERROR_FILE	File could not be opened. This error usually occurs when the file name is invalid or the file can not be found.
-5	DMCERROR_DRIVER	Device driver could not be opened, or a read or write error occurred.
-6	DMCERROR_HANDLE	Invalid Galil controller handle. This error will occur if you try to communicate with the Galil controller without first calling DMCOpen.
-7	DMCERROR_HMODULE	Support dynamic link library could not be loaded. This error will occur if DMCBUS16.DLL or DMCBUS32.DLL can not be found for bus controllers and if DMCSER16.DLL or DMCSER32.DLL can not be found for serial controllers.
-8	DMCERROR_MEMORY	Out of memory.
-9	DMCERROR_BUFFERFULL	Response from the controller was larger than the response buffer supplied. The user-supplied buffer for DMCCommand was too small to fit the complete response from the Galil controller. You can call DMCGetAdditionalResponseLen to find out how much data is left to retrieve and then call DMCGetAdditionalResponse to retrieve the additional response data.
-10	DMCERROR_RESPONSEDATA	Response from the controller overflowed the internal additional response buffer. The response from the Galil controller was so large that it overflowed both the user-supplied buffer and the internal additional response buffer. You must increase the size of the user-supplied buffer.
-11	DMCERROR_DMA	Could not communicate with DMA channel.
-12	DMCERROR_ARGUMENT	One or more required arguments to a DMC API function call was NULL.
-13	DMCERROR_DATARECORD	Could not access data record.
-14	DMCERROR_DOWNLOAD	File download failed. The problem is most likely a file that has too many lines or one or more lines which exceed the line length restriction.
-15	DMCERROR_FIRMWARE	Could not update the controller's firmware.
-16	DMCERROR_CONVERSION	Could not convert the DMC command (ASCII to binary or binary to ASCII).
-17	DMCERROR_REGISTRY	Could not access or modify the controller's registry information. You need to log-on to Windows (Windows NT that is) with Administrator privileges.
-18	DMCERROR_RESOURCE	Windows reports a resource conflict with the current hardware configuration. This error could occur if you tried to manually assign resources to a Galil plug-and-play controller.
-19	DMCERROR_BUSY	Could not write to the controller because it is currently busy. You may need to set the time-out value higher because the previous command is taking longer to process by the controller than expected.

-20	DMCERROR_DEVICE_ DISCONNECTED	A Windows plug-and-play controller, such as one which communicates through USB, was disconnected from the system. You must close the handle to the controller (using DMCClose) as soon as possible.
-21	DMCERROR_TIMEING_ERROR	Data is not being transferred to controller fast enough to maintain time synchronization.
-22	DMCERROR_WRITEBUFFER_ TOO_LARGE	The user supplied buffer is too large. Must be < 1024 bytes.
-23	DMCERROR_NO_MODIFY_ PNP_CONTROLLER	Registry modification of PnP controllers is not allowed.
-24	DMCERROR_FUNCTION_ OBSOLETE	This function is obsolete.
-25	DMCERROR_STREAMING_ COMMAND_IN_PROGRESS	A different process is using a streaming command(LS,UL,ED,QD,QU). Try the DMCCommand or DMCWriteData function again later.
-26	DMCERROR_DEVICEDRIVER_ VERSION_TOO_OLD	The device driver needed to communicate with the selected controller is too old for this communication dll.
-27	DMCERROR_STREAMING_CO MMAND_MUST_BE_SOLITARY	Streaming commands (LS, UL, ED, QD, QU) cannot be mixed with other commands on the command line.

API Function Details

The following section provides detailed descriptions of each DMCCOM command. For each command listed, a C function prototype is provided, along with descriptions of each parameter.

Syntax varies for Visual Basic and C++; you can compare the syntax differences by looking at the DMCCOM40.BAS for Visual Basic and DMCWIN.CPP for C++.

DMCAddGalilRegistry and DMCAddGalilRegistry2

```
LONG FAR GALILCALL DMCAddGalilRegistry(PGALILREGISTRY pgalilregistry, PUSHORT pusController);

LONG FAR GALILCALL DMCAddGalilRegistry2(PGALILREGISTRY2 pgalilregistry2, PUSHORT pusController);
```

Add a Galil controller to the Windows registry. The controller number is returned in the argument pusController. The DMCAddGalilRegistry2 function is a replacement for DMCAddGalilRegistry.

pgalilregistry/pgalilregistry2	Pointer to a GALILREGISTRY or GALILREGISTRY2 struct.
pusController	Pointer to an unsigned short that will receive the Galil controller number.

DMCArrayDownload

```
LONG FAR GALILCALL DMCArrayDownload(HANDLEDMC hdmc, PSZ pszArrayName, USHORT usFirstElement,
       USHORT usLastElement, PCHAR pchData, ULONG cbData, PULONG cbBytesWritten);
```

Download an array to the Galil controller. The array must already exist in the controller. Array data can be delimited by a comma or CR (0x0D) or CR/LF (0x0D0A).

Note: The firmware on the controller must be recent enough to support the QD command.

hdmc	Handle to the Galil controller.
pszArrayName	Array name to download to the Galil controller.
usFirstElement	First array element.
usLastElement	Last array element.
pchData	Buffer to write the array data from. Data does not need to be NULL terminated.
cbData	Length of the array data in the buffer.

cbBytesWritten	Number of bytes written.

DMCArrayUpload

```
LONG FAR GALILCALL DMCArrayUpload(HANDLEDMC hdmc, PSZ pszArrayName, USHORT usFirstElement,
       USHORT usLastElement, PCHAR pchData,  ULONG cbData, PULONG pulBytesRead, SHORT fComma);
```

Upload an array from the Galil controller to the PC. The array must exist on the controller. Array data will be delimited by a comma or CR (0x0D) depending of the value of fComma.

Note: The firmware on the controller must be recent enough to support the QU command.

hdmc	Handle to the Galil controller.
pszArrayName	Array name to upload from the Galil controller.
usFirstElement	First array element.
usLastElement	Last array element.
pchData	Buffer to read the array data into. Array data will not be NULL terminated.
cbData	Length of the buffer.
pulBytesRead	Number of bytes read.
fComma	1 comma delimited, 0 carriage return

DMCAssignIPAddress

```
LONG FAR GALILCALL DMCAssignIPAddress(HWND hWnd, PGALILREGISTRY2 pgalilregistry2);
```

Assign an IP Address to an Ethernet controller. The controller must be in BOOTP broadcast mode.

Note: This function is for Ethernet controllers only.

hwnd	The window handle of the calling application. If NULL, the window with the current input focus is used.
szIPAddress	The IP address as a string. Example: "160.35.50.1".

DMCBinaryCommand

```
LONG FAR GALILCALL DMCBinaryCommand(HANDLEDMC hdmc, PBYTE pbCommand, ULONG ulCommandLength,
       PCHAR pchResponse, ULONG cbResponse);
```

Send a DMC command in binary format to the Galil controller. Most commands have a binary equivalent that will be processed faster by the controller.

Note: This function is for Optima Series controllers only.

hdmc	Handle to the Galil controller.
pbCommand	The command to send to the Galil controller in binary format.
ulCommandLength	The length of the command (binary commands are not null-terminated).
pchResponse	Buffer to receive the response data. If the buffer is too small to receive all the response data from the controller, the error code DMCERROR_BUFFERFULL will be returned. The user may get additional response data by calling the function DMCGetAdditionalResponse. The length of the additional response data may ascertained by call the function DMCGetAdditionalResponseLen. If the response data from the controller is too large for the internal additional response buffer, the error code DMCERROR_RESPONSEDATA will be returned.
cbResponse	Length of the buffer.

DMCChangeInterruptNotification

```
LONG FAR GALILCALL DMCChangeInterruptNotification(HANDLEDMC hdmc, LONG lHandle);
```

Change the window handle used in DMCOpen or the thread ID used in DMCOpen2. This value is for notification of interrupts.

hdmc	Handle to the Galil controller.
lHandle	New window handle or thread ID.

DMCClear

```
LONG FAR GALILCALL DMCClear(HANDLEDMC hdmc);
```

Clear the Galil controller FIFOs.

hdmc	Handle to the Galil controller.

DMCClose

```
LONG FAR GALILCALL DMCClose(HANDLEDMC hdmc);
```

Close communications with the Galil controller. Failing to call DMCClose can result in memory leaks.

hdmc	Handle to the Galil controller.

DMCCommand

```
LONG FAR GALILCALL DMCCommand(HANDLEDMC hdmc, PSZ pszCommand, PCHAR chResponse,
       ULONG cbResponse);
```

Send a DMC command in ASCII format to the Galil controller.

Note: This function can only send commands or groups of commands up to 1024 bytes long.

hdmc	Handle to the Galil controller.
pszCommand	The command to send to the Galil controller.
pchResponse	Buffer to receive the response data. If the buffer is too small to receive all the response data from the controller, the error code DMCERROR_BUFFERFULL will be returned. The user may get additional response data by calling the function DMCGetAdditionalResponse. The length of the additional response data may ascertained by call the function DMCGetAdditionalResponseLen. If the response data from the controller is too large for the internal additional response buffer, the error code DMCERROR_RESPONSEDATA will be returned.
cbResponse	Length of the buffer.

DMCCommand_AsciiToBinary

```
LONG FAR GALILCALL DMCCommand_AsciiToBinary(HANDLEDMC hdmc, PSZ pszAsciiCommand,
       ULONG ulAsciiCommandLength, PBYTE pbBinaryResult,  ULONG cbBinaryResult,
       ULONG FAR *pulBinaryResultLength);
```

Convert an ASCII DMC command to a binary DMC command.

Note: This function is for the optima series controllers only.

hdmc	Handle to the Galil controller.
pszAsciiCommand	ASCII DMC command(s) to be converted.
ulAsciiCommandLength	Length of DMC command(s).
pbBinaryResult	Buffer to receive the translated DMC command.
cbBinaryResult	Length of the buffer.
pulBinaryResultLength	Length of the translated DMC command.

DMCCommand_BinaryToAscii

```
LONG FAR GALILCALL DMCCommand_BinaryToAscii(HANDLEDMC hdmc, PBYTE pbBinaryCommand,
       LONG ulBinaryCommandLength, PSZ pszAsciiResult, ULONG cbAsciiResult,
       ULONG FAR *pulAsciiResultLength);
```

Convert a binary DMC command to an ASCII DMC command.

Note: This function is for the optima series controllers only.

Hdmc	Handle to the Galil controller.
pbBinaryCommand	Binary DMC command(s) to be converted.
ulBinaryCommandLength	Length of DMC command(s).
pszAsciiResult	Buffer to receive the translated DMC command.
cbAsciiResult	Length of the buffer.
pulAsciiResultLength	Length of the translated DMC command.

DMCCompressFile

```
LONG FAR GALILCALL DMCCompressFile(PSZ pszInputFileName, PSZ pszOutputFileName,
       USHORT usLineWidth, PUSHORT pusLineCount);
```

Compress a DMC file so that program space in the controller is fully utilized. Lines are put together whenever possible to make more lines available. Leading and trailing spaces are removed as well.

pszInputFileName	The name of the DMC file to compress.
pszOutputFileName	The name of the resulting compressed DMC file.
usLineWidth	The maximum line width. For most controllers, this value is either 40 or 80.
pusLineCount	The line count of the resulting compressed DMC file is returned on output. Output Only.

DMCCopyDataRecord

```
LONG FAR GALILCALL DMCCopyDataRecord(HANDLEDMC hdmc, PVOID pDataRecord);
```

Get a copy of the data record used for fast polling. The data record is only as recent as the last call made to DMCRefreshDataRecord.

hdmc	Handle to the Galil controller.
pDataRecord	A copy of the data record is returned on output. Output Only.

DMCDeleteGalilRegistry

```
LONG FAR GALILCALL DMCDeleteGalilRegistry(SHORT sController);
```

Delete a Galil controller in the Windows registry. If a Plug-and-Play controller (USB or PCI) is deleted, the computer must be rebooted to recover the controller in the registry.

sController	Galil controller number. Use -1 to delete all Galil controllers.

DMCDiagnosticsOff

```
LONG FAR GALILCALL DMCDiagnosticsOff(HANDLEDMC hdmc);
```

Turn off diagnostics log.

hdmc	Handle to the Galil controller.

DMCDiagnosticsOn

```
LONG FAR GALILCALL DMCDiagnosticsOn(HANDLEDMC hdmc, PSZ pszFileName, BOOL fAppend);
```

Turn on diagnostics to log all communications to the controller.

hdmc	Handle to the Galil controller.
pszFileName	File name for the diagnostic file.
fAppend	TRUE if the file will open for append, otherwise FALSE.

DMCDownloadFile

```
LONG FAR GALILCALL DMCDownloadFile(HANDLEDMC hdmc, PSZ pszFileName, PSZ pszLabel);
```

Download a Galil-language application program to the controller from a file on the hard drive.

hdmc	Handle to the Galil controller.
pszFileName	Name of file to download to the Galil controller.
pszLabel	Program label at which to insert the contents of pszFileName. The application program in the controller will be overwritten from this label on. Passing NULL causes the entire application program in the controller to be overwritten.

DMCDownloadFirmwareFile

```
LONG FAR GALILCALL DMCDownloadFirmwareFile(HANDLEDMC hdmc, PSZ pszFileName,
       SHORT fDisplayDialog);
```

Update the controller's firmware. This function will open a binary firmware file and write new firmware to the flash EEPROM of the controller.

Note: This function is for the DMC-1200, DMC-14x5, DMC-1600, DMC-1700, DMC-1800, DMC-18x2, DMC-2000, DMC-2100, DMC-2200, DMC-21x2/3, and DMC-34x5 only.

hdmc	Handle to the Galil controller.
pszFileName	File name to download to the Galil controller.
fDisplayDialog	Display a progress dialog while the firmware file is being downloaded.

DMCDownloadFromBuffer

```
LONG FAR GALILCALL DMCDownloadFromBuffer(HANDLEDMC hdmc, PSZ pszBuffer, PSZ pszLabel);
```

Download a Galil-language application program from a memory buffer to the Galil controller.

hdmc	Handle to the Galil controller.
pszBuffer	String of DMC commands to download to the Galil controller.
pszLabel	Program label at which to insert the contents of pszBuffer. The application program in the controller will be overwritten from this label on. Passing NULL causes the entire application program in the controller to be overwritten.

DMCEditRegistry

```
VOID FAR GALILCALL DMCEditRegistry(HWND hwnd);
```

Edit the Windows registry: add, change, or delete Galil motion controllers. This function requires the Galil ActiveX control DMCReg.ocx to be installed and registered on the PC.

Note: This function invokes a dialog window.

hwnd	The window handle of the calling application. If NULL, the window with the input focus is used.

DMCEnumGalilRegistry and DMCEnumGalilRegistry2

```
LONG FAR GALILCALL DMCEnumGalilRegistry(USHORT FAR* pusCount, PGALILREGISTRY pgalilregistry);

LONG FAR GALILCALL DMCEnumGalilRegistry2(USHORT FAR* pusCount, PGALILREGISTRY2 pgalilregistry2);
```

Enumerate or list all the Galil controllers in the Windows registry. The user needs to make two calls to this function. The first call should have a NULL for the argument pgalilregistry. The number of GALILREGISTRY structs (number of Galil controllers in the Windows registry) will be returned in the argument pusCount. The second call should have sufficient memory allocated for all the GALILREGISTRY structs to be returned and pass the pointer to that memory as the argument pgalilregistry. It is the users responsibility to allocate and free memory to hold the GALILREGISTRY structs. The DMCEnumGalilRegistry2 function is a replacement for DMCEnumGalilRegistry.

pusCount	Pointer to the number of GALILREGISTRY structs returned.

pgalilregistry/pgalilregistry2	Pointer to a GALILREGISTRY or GALILREGISTRY2 struct.

DMCError

```
LONG FAR GALILCALL DMCError(HANDLEDMC hdmc, LONG lError, PCHAR pchMessage, LONG cbMessage);
```

Retrieve the error description for an error code when a Galil API function does not return DMCNOERROR.

hdmc	Handle to the Galil controller.
pchMessage	Buffer to receive the error message text.
cbMessage	Length of the buffer.

DMCFastCommand

```
LONG FAR GALILCALL DMCFastCommand(HANDLEDMC hdmc, PSZ pszCommand);
```

Send a DMC command in ASCII format to the Galil controller and do not wait for a response. Use this function with caution because command responses will not be removed from the controller's output. In some applications it may be necessary to first send the Galil command CW,1 to allow the controller to continue program execution when the controller's output FIFO is full.

Use this function for Galil commands which do not return an acknowledgment from the controller such as providing data for the DL and QD commands.

Note: This function can only send commands or groups of commands up to 1024 bytes long.

hdmc	Handle to the Galil controller.
pszCommand	The command to send to the Galil controller.

DMCFile_AsciiToBinary

```
LONG FAR GALILCALL DMCFile_AsciiToBinary(HANDLEDMC hdmc, PSZ pszInputFileName,
        PSZ pszOutputFileName);
```

Convert a file consisting of ASCII commands to a file consisting of binary commands.

Note: This function is for the optima series controllers only.

hdmc	Not used.
pszInputFileName	File name for the input ASCII file.
pszOutputFileName	File name for the output binary file.

DMCFile_BinaryToAscii

```
LONG FAR GALILCALL DMCFile_BinaryToAscii(HANDLEDMC hdmc, PSZ pszInputFileName,
        PSZ pszOutputFileName);
```

Convert a file consisting of binary commands to a file consisting of ASCII commands.

Note: This function is for the optima series controllers only.

hdmc	Not used.
pszInputFileName	File name for the input binary file.
pszOutputFileName	File name for the output ASCII file.

DMCGetAdditionalResponse

```
LONG FAR GALILCALL DMCGetAdditionalResponse(HANDLEDMC hdmc, PCHAR pchResponse, ULONG cbResponse);
```

Query the Galil controller for more response data. There will be more response data available if the DMCCommand function returned DMCERROR_BUFFERFULL. Once this function is called, the internal additional response buffer is cleared.

hdmc	Handle to the Galil controller.
pchResponse	Buffer to receive the response data.
cbResponse	Length of the buffer.

DMCGetAdditionalResponseLen

```
LONG FAR GALILCALL DMCGetAdditionalResponseLen(HANDLEDMC hdmc, PULONG pulResponseLen);
```

Query the Galil controller for the length of additional response data. There will be more response data available if the DMCCommand function returned DMCERROR_BUFFERFULL.

hdmc	Handle to the Galil controller.
pulResponseLen	Buffer to receive the additional response data length.

DMCGetControllerDesc

```
LONG FAR GALILCALL DMCGetControllerDesc(USHORT usController, PSZ pszControllerDesc,
       ULONG cbControllerDesc);
```

Get a text description of the controller from the registry.

usController	Galil controller number.
pszControllerDesc	Buffer to receive the controller description. Output only.
cbControllerDesc	Length of the buffer.

DMCGetDataRecordByItemId

```
LONG FAR GALILCALL DMCGetDataRecordByItemId(HANDLEDMC hdmc, USHORT usItemId, USHORT usAxisId, PUSHORT
       pusDataType, PLONG plData);
```

Get a data item from the data record. Gets one item from the data record by using a predefined ID (see data record IDs defined in DMCDRC.H). To retrieve data record items by offset instead of ID, use the function DMCGetDataRecord.

hdmc	Handle to the Galil controller.
usItemId	Data record item ID.
usAxisId	Axis ID used for axis data items.
pusDataType	Data type of the data item. The data type of the data item is returned on output. Output Only.
plData	Buffer to receive the data record data. Output only.

DMCGetDataRecordConstPointer

```
LONG FAR GALILCALL DMCGetDataRecordConstPointer(HANDLEDMC hdmc, const char **pchDataRecord);
```

Get a const pointer to the data record. Using this method to access the information in the data record eliminates the copying necessary with DMCCopyDataRecord. Additional const pointers can be created to individual data items by using DMCGetDataRecordItemOfsetById(). These two functions allow a one-time setup. Then, all that is required to access information in the data record is to call DMCRefreshDataRecord() and then dereference the desired pointer into the data record.

hdmc	Handle to the Galil controller.
pchDataRecord	Pointer to a const pointer to the data record.

DMCGetDataRecordItemOffsetById

```
LONG FAR GALILCALL DMCGetDataRecordItemOffsetById(HANDLEDMC hdmc, USHORT usItemId,
       USHORT usAxisId, LPUSHORT pusOffset, LPUSHORT pusDataType );
```

Get the total offset for a data item by data item ID (see data record IDs defined in DMCDRC.H). The returned offset and data type can then be used to extract a value from the data record pointer retrieved using DMCGetDataRecordConstPointer.

hdmc	Handle to the Galil controller.
usItemId	Data record item Id.
usAxisId	Axis Id used for axis data items.
pusOffset	Total offset (number of bytes) from the beginning of the data record to the data item. Output Only.
pusDataType	Data type of the data item. The data type of the data item is returned on output. Output Only.

DMCGetDataRecordConstPointerArray

```
LONG FAR GALILCALL DMCGetDataRecordConstPointerArray(HANDLEDMC hdmc, const char **pchDataRecord,
       LPUSHORT pusNumDataRecords);
```

Get a const pointer to the available data records. This function retrieves all the available cached data records from a PCI or ISA controller. For DMC-1800 controllers, depending on the hardware/software version, the data record may be accessed through 2nd FIFO or Dual Port RAM. The newest version of controller board (Rev. H and above for 0-4 axes board, and Rev. D and above for 5-8 axes board) supports both Dual Port RAM and 2nd FIFO functions. Version 7.0.3.0 of the PCI drivers (Glwdmpci.sys for Windows XP, 2000, ME,and 98SE, and GalilPCI.sys for NT4.0) automatically accesses the data record using the Dual Port RAM on the newer controller versions with firmware version M1 and higher.

Note: Use of this function requires Glwdmpci.sys, Glwdmisa.sys, and GalilPCI.sys driver versions 7.0.0.0 or higher. The cache depth is set by controller properties stored in the Galil Registry. Do not call DMCRefreshDataRecord prior to calling this function.

hdmc	Handle to the Galil controller.
pchDataRecord	Const pointer to an array of data records.
psNumDataRecords	The number of data records contained in the character array.

DMCGetDataRecordArray

```
LONG FAR GALILCALL DMCGetDataRecordArray(HANDLEDMC hdmc, CDMCFullDataRecord **pDataRecordArray,
       LPUSHORT pusNumDataRecords);
```

Get an array of data records. This function retrieves all the available cached data records from a PCI and ISA controller with 2nd FIFO. For DMC1800 controllers, depending on the hardware/software version, the data record may be accessed through 2nd FIFO or Dual Port RAM.

Note: Use of this function requires Glwdmpci.sys, Glwdmisa.sys, and GalilPCI.sys driver versions 7.0.0.0 or higher. The cache depth is set by controller properties stored in the Galil Registry. Do not call DMCRefreshDataRecord prior to calling this function.

hdmc	Handle to the Galil controller.
pchDataRecord	Pointer to an array of CDMCFullDataRecord objects. CDMCFullDataRecord contains members for every possible item in a record. If a member doesn't apply to a given controller (such as axes 5-8 on a four axis controller) then these members must be disregarded.
psNumDataRecords	The number of data records contained in the data record array.

DMCGetDataRecordRevision

```
LONG FAR GALILCALL DMCGetDataRecordRevision(HANDLEDMC hdmc, PUSHORT pusRevision);
```

Get the revision of the data record structure.

hdmc	Handle to the Galil controller.
pusRevision	The revision of the data record structure is returned on output. Output Only.

DMCGetDataRecordSize

```
LONG FAR GALILCALL DMCGetDataRecordSize(HANDLEDMC hdmc, PUSHORT pusRecordSize);
```

Get the size of the data record.

Note: this function is for the DMC-1600, DMC-1700, and DMC-1800 only.

hdmc	Handle to the Galil controller.
pusRecordSize	The size of the data record is returned on output. Output Only.

DMCGetGalilRegistryInfo and DMCGetGalilRegistryInfo2

```
LONG FAR GALILCALL DMCGetGalilRegistryInfo(USHORT usController, PGALILREGISTRY pgalilregistry);

LONG FAR GALILCALL DMCGetGalilRegistryInfo2(USHORT usController, PGALILREGISTRY2 pgalilregistry2);
```

Get Windows registry information for a given Galil controller. The DMCGetGalilRegistryInfo2 function is a replacement for DMCGetGalilRegistryInfo.

usController	Galil controller number.
pgalilregistry/pgalilregistry2	Pointer to a GALILREGISTRY or GALILREGISTRY2 struct.

DMCGetHandle

```
LONG FAR GALILCALL DMCGetHandle(USHORT usController, PHANDLEDMC phdmc);
```

Get the handle for a Galil controller which was already opened using DMCOpen or DMCOpen2. The handle to the Galil controller is returned in the argument phdmc.

usController	A number between 1 and 64. Up to 64 Galil controllers may be addressed per process.
phdmc	Buffer to receive the handle to the Galil controller to be used for all subsequent API calls. Users should declare a variable of type HANDLEDMC and pass the address of the variable to the function.

DMCGetTimeout

```
LONG FAR GALILCALL DMCGetTimeout(HANDLEDMC hdmc, LONG FAR* pTimeout);
```

Get the current time-out value, which determines how long the communications driver will wait for a command response from the controller.

hdmc	Handle to the Galil controller.
pTimeout	Buffer to receive the current time-out value in milliseconds.

DMCGetUnsolicitedResponse

```
LONG FAR GALILCALL DMCGetUnsolicitedResponse(HANDLEDMC hdmc, PCHAR chResponse, ULONG cbResponse);
```

Query the Galil controller for unsolicited responses. These are messages output from programs running in the Galil controller. The most common command to produce messages is MG.

hdmc	Handle to the Galil controller.
pchResponse	Buffer to receive the response data.
cbResponse	Length of the buffer.

DMCMasterReset

```
LONG FAR GALILCALL DMCMasterReset(HANDLEDMC hdmc);
```

Master reset the Galil controller. This returns the controller to its factory default setting, erasing all data saved with BN, BP, and BV. Ensure that important data is backed up before performing a master reset. The master reset may take up to 5 seconds. Make sure the timeout is large enough before performing the master reset.

hdmc	Handle to the Galil controller.

DMCModifyGalilRegistry and DMCModifyGalilRegistry2

```
LONG FAR GALILCALL DMCModifyGalilRegistry(USHORT usController, PGALILREGISTRY pgalilregistry);

LONG FAR GALILCALL DMCModifyGalilRegistry2(USHORT usController, PGALILREGISTRY2 pgalilregistry2);
```

Change a Galil controller in the Windows registry. The DMCModifyGalilRegistry2 function is a replacement for DMCModifyGalilRegistry.

usController	Galil controller number.
pgalilregistry/pgalilregistry2	Pointer to a GALILREGISTRY or GALILREGISTRY2 struct.

DMCOpen

```
LONG FAR GALILCALL DMCOpen(USHORT usController, HWND hwnd, PHANDLEDMC phdmc);
```

Open communications with the Galil controller. The handle to the Galil controller is returned in the argument phdmc. For every DMCOpen, you must issue a DMCClose.

Note: hwnd is not used for controllers which do not support bus interrupts.

usController	A number between 1 and 64. Up to 64 Galil controllers may be addressed per process.
hwnd	The window handle to use for notifying the application program of an interrupt via PostMessage.
phdmc	Buffer to receive the handle to the Galil controller to be used for all subsequent API calls. Users should declare a variable of type HANDLEDMC and pass the address of the variable to the function.

DMCOpen2

```
LONG FAR GALILCALL DMCOpen2(USHORT usController, LONG lThreadID, PHANDLEDMC phdmc);
```

Open communications with the Galil controller with interrupt handling. The handle to the Galil controller is returned in the argument phdmc. For every DMCOpen2, you must issue a DMCClose.

usController	A number between 1 and 64. Up to 64 Galil controllers may be addressed per process.
lThreadID	The thread ID identifies the calling thread and is also used for notifying the application program of an interrupt via PostThreadMessage.
phdmc	Buffer to receive the handle to the Galil controller to be used for all subsequent API calls. Users should declare a variable of type HANDLEDMC and pass the address of the variable to the function.

DMCReadData

```
LONG FAR GALILCALL DMCReadData(HANDLEDMC hdmc, PCHAR pchBuffer, ULONG cbBuffer,
        PULONG pulBytesRead);
```

Low-level I/O function to read data from the Galil controller. The function will read whatever is currently in the controller's output FIFO (bus controller) or communications port input queue (serial controller). The function will read up to cbBuffer characters from the controller. The data placed in the user buffer (pchBuffer) is NOT NULL terminated. The data returned is not guaranteed to be a complete response - you may have to call this function repeatedly to get a complete response.

hdmc	Handle to the Galil controller.
pchBuffer	Buffer to read the data into. Data will not be NULL terminated.
cbBuffer	Length of the buffer.
pulBytesRead	Number of bytes read.

DMCReadSpecialConversionFile

```
LONG FAR GALILCALL DMCReadSpecialConversionFile(HANDLEDMC hdmc, PSZ pszFileName);
```

Read into memory a special BinaryToAscii/AsciiToBinary conversion table.

Note: This function is for the PC based controllers only.

hdmc	Handle to the Galil controller.
pszFileName	File name for the special conversion file.

DMCRefreshDataRecord

```
LONG FAR GALILCALL DMCRefreshDataRecord(HANDLEDMC hdmc, ULONG ulLength);
```

Refresh the data record.

hdmc	Handle to the Galil controller.
ulLength	Refresh size in bytes. Set to 0 unless you do not want a full-buffer refresh.

DMCRegisterPnpControllers

```
LONG FAR GALILCALL DMCRegisterPnpControllers(USHORT* pusCount);
```

OBSOLETE. Update the Windows registry for all Galil plug-and-play controllers. This function may add new controllers to the registry or update existing ones.

pusCount	Pointer to the number of Galil plug-and-play controllers registered (and/or updated).

DMCReset

```
LONG FAR GALILCALL DMCReset(HANDLEDMC hdmc);
```

Reset the Galil controller.

hdmc	Handle to the Galil controller.

DMCSelectController

```
SHORT FAR GALILCALL DMCSelectController(HWND hwnd);
```

Select a Galil motion controller from a list of registered controllers. Returns the selected controller number or -1 if no controller was selected.

Note: This function invokes a dialog window.

hwnd	The window handle of the calling application. If NULL, the window with the input focus is used.

DMCSendBinaryFile

```
LONG FAR GALILCALL DMCSendBinaryFile(HANDLEDMC hdmc, PSZ pszFileName);
```

Send a file consisting of DMC commands in binary format to the Galil controller. Commands are executed immediately.

Note: This function is for the optima series controllers only.

hdmc	Handle to the Galil controller.
pszFileName	File name to send to the Galil controller.

DMCSendCW2OnClose

```
LONG FAR GALILCALL DMCSendCW2OnClose( HANDLEDMC hdmc, BOOL *pbValue );
```

Determines if the controller is sent CW2 command on closing. The CW2 command causes the controller to NOT set the most significant bit of unsolicited messages to 1 (please see the Galil Motion Control command reference). The default behavior is TRUE, which means the CW2 command is sent prior to closing a controller handle.

hdmc	Handle to the Galil controller.
pbValue	Desired Boolean value. On function return pbValue contains the previous value.

DMCSendFile

```
LONG FAR GALILCALL DMCSendFile(HANDLEDMC hdmc, PSZ pszFileName);
```

Send a file consisting of DMC commands in ASCII format to the Galil controller. Commands are executed immediately.

hdmc	Handle to the Galil controller.
pszFileName	File name to send to the Galil controller.

DMCSetTimeout

```
LONG FAR GALILCALL DMCSetTimeout(HANDLEDMC hdmc, LONG lTimeout);
```

Set time-out value. If the time-out value is set to zero, the DLLs will ignore time-out errors. This is useful for sending Galil commands which do not return a response, such as providing records to the DL or QD commands.

hdmc	Handle to the Galil controller.
lTimeout	Time-out value in milliseconds.

DMCStartDeviceDriver

```
LONG FAR GALILCALL DMCStartDeviceDriver(USHORT usController);
```

Start the device driver associated with the given controller. All controller handles must be closed. Use this function to recycle the device driver after making a configuration change through the Windows registry.

Note: For bus controllers on Windows NT4 only.

usController	Galil controller number.

DMCStopDeviceDriver

```
LONG FAR GALILCALL DMCStopDeviceDriver(USHORT usController);
```

Stop the device driver associated with the given controller. All controller handles must be closed. Use this function to recycle the device driver after making a configuration change through the Windows registry.

Note: For bus controllers on Windows NT4 only.

usController	Galil controller number.

DMCUploadFile

```
LONG FAR GALILCALL DMCUploadFile(HANDLEDMC hdmc, PSZ pszFileName);
```

Upload the application program from the Galil controller to a file on the hard disk.

hdmc	Handle to the Galil controller.
pszFileName	Name of PC File to store the application program from the Galil controller in.

DMCUploadToBuffer

```
LONG FAR GALILCALL DMCUploadToBuffer(HANDLEDMC hdmc, PCHAR pchBuffer, ULONG cbBuffer);
```

Upload the application program from the Galil controller to a memory buffer on the PC.

hdmc	Handle to the Galil controller.
pchBuffer	Buffer to store the application program uploaded from the Galil controller.
cbBuffer	Length of the buffer.

DMCVersion

```
LONG FAR GALILCALL DMCVersion(HANDLEDMC hdmc, PCHAR pchVersion, ULONG cbVersion);
```

Get the firmware version that is running on the Galil controller.

hdmc	Handle to the Galil controller.
pchVersion	Buffer to receive the version information.
cbVersion	Length of the buffer.

DMCWaitForMotionComplete

```
LONG FAR GALILCALL DMCWaitForMotionComplete(HANDLEDMC hdmc, PSZ pszAxes, SHORT fDispatchMsgs);
```

Wait for motion complete by creating a thread to query the controller. The function returns when motion is complete.

hdmc	Handle to the Galil controller.
pszAxes	Which axes to wait for: X, Y, Z, W, A, B, C, D, E, F, G, H, S or T. To wait for more than one axis (other than coordinated motion), simply concatenate the axis letters in the string.
fDispatchMsgs	Set to TRUE if you want to get and dispatch Windows messages while waiting for motion complete.

DMCWriteData

```
LONG FAR GALILCALL DMCWriteData(HANDLEDMC hdmc, PCHAR pchBuffer, ULONG cbBuffer,
       PULONG pulBytesWritten);
```

Low-level I/O function to write data to the Galil controller. Data is written to the Galil controller only if it is "ready" to receive it. The function will attempt to write exactly cbBuffer characters to the controller.

hdmc	Handle to the Galil controller.
pchBuffer	Buffer to write the data from. Data does not need to be NULL terminated.
cbBuffer	Length of the data in the buffer. Must be less than 1024 bytes for bus controllers.

pulBytesWritten	Number of bytes written.

Advanced Motion Functions (DMCMLIB)

These functions perform advanced functions that can extend the capabilities of the motion controller.

DMCEllipse

This function allows a controller to perform elliptical motion. When it is called, it will create a text file containing VP commands that define an ellipse. This file can then be sent to the card to perform the motion using DMCSendFile or inserted into a larger sequence of VP and CR commands.

```
LONG GALILCALL DMCEllipse(char *filename, int FirstOffset, int SecondOffset, double a, double b,
       double theta, double delta_theta, double increment, double rotation);
```

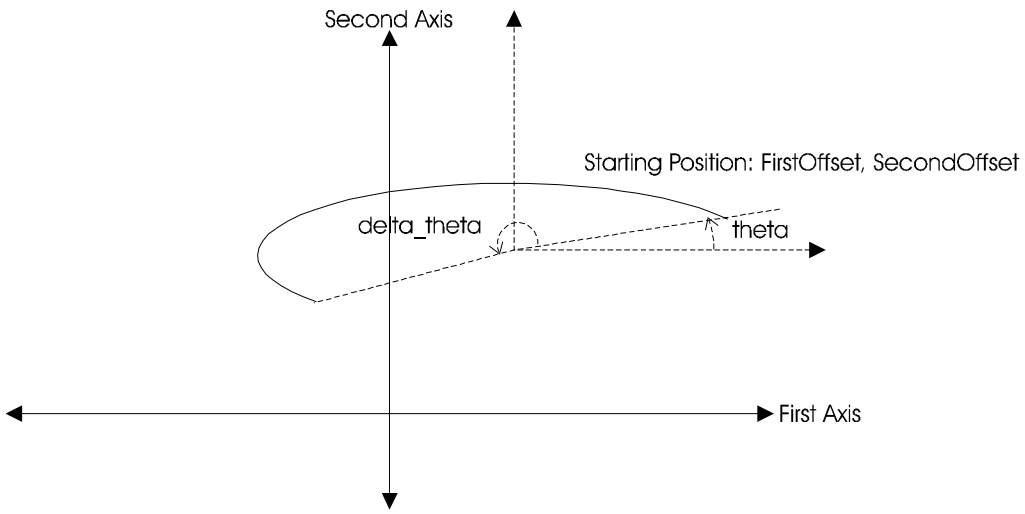
filename	Location used to save the calculated VP commands
FirstOffset	Starting location for the first axis in encoder counts
SecondOffset	Starting location for the second axis in encoder counts
a and *b*	Major and minor axis lengths in encoder counts
theta	Starting angle in degrees
delta_theta	Total angular distance to travel in degrees
increment	Increment advancement of theta used to calculate a new set of points in degrees
rotation	Angular translation of the major axis in degrees

Description

The parameters sent to the DMCEllipse function are similar to those for the CR command. An ellipse is defined by the starting angle, ending angle, and the major and minor axis. This function takes those values and creates an ellipse using VP commands. The end result is a sequence of line segments that produce the curve.

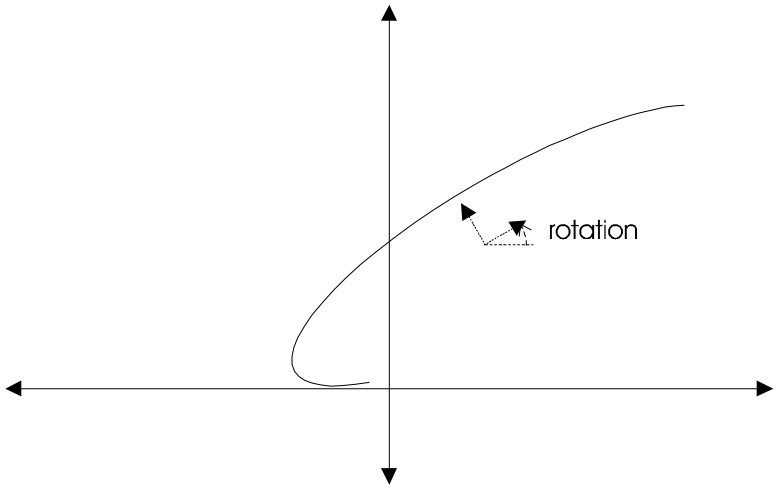
The first point calculated is at the angular position *theta*. The next is at *theta+increment* and so on until the final angle (*theta+delta_theta*) is reached. Obviously, the length of the computed segments will determine how smooth the curve will be. To control the coarseness of the curve use a small increment value. The smaller the increment the more points are created.

FirstOffset and *SecondOffset* determine the position of the first point of the curve.



Ellipse with starting angle of 30 degrees, a delta angle of 190 and an offset starting position.

Using *rotation* the entire ellipse can be translated a number of degrees.



Although it is not clear by the drawing, the starting position is unchanged by the rotation.

Note: The DMCEllipse function can be used to create circles with a very large radius. The Galil CR command is limited to 6000000. To make larger circles make a=b=radius of circle.

Error Codes

If there are no errors the DMCEllipse function returns a 0. Here are the conditions that can generate errors and the code that is returned.

Condition	Code
Bad arguments: delta_theta positive but increment negative and visa versa; delta_theta = 0; increment=0; a or b <=0	-12
Could not open output file	-4

Example
This C program uses the DMCEllipse function to make an ellipse. The ellipse is calculated and put into a file. Then the file is sent to the controller to perform the motion.

```
#include "DMCMLIB.H"

int main(void)
{
        long rc;
        char filename[] = "c:\\ellout.sen";
        HANDLEDMC hDmc;
        HWND hWnd;
        char szBuffer[80];

        //make the output file
        rc = DMCEllipse(filename, 1000, 1500, 2000.0, 800.0, 30.0, 190.0, 5.0, 30.0);

        //send the output file
        rc = DMCOpen(1, hWnd, &hDmc);
        rc = DMCSendFile(hDmc, filename);

        //start motion
        rc = DMCCommand(hDmc, "VE", szBuffer, sizeof(szBuffer));
        rc = DMCCommand(hDmc, "BGS", szBuffer, sizeof(szBuffer));

        rc = DMCClose(hDmc);

        return 0;
}
```

Here the ellipse started with offsets 1000 and 1500, with a major axis of 2000 , minor axis of 800, starting angle of 30, delta angle of 190 and a rotation of 30. To make the motors move the DMCSendFile is used to take the VP commands from the file and put them in the coordinated motion segment buffer inside the controller. The LE sent by DMCCommand tells the controller the sequence is ended and the BGS command starts the motion.

It is not necessary to include DMCCOM.H in the above example because it is already included in the DMCMLIB.H library header file.

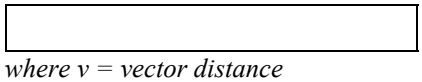
DMCSpline

Use DMCSpline to fit a curve through the points that define a 1-8 axis move to remove infinite accelerations around corners.

```
LONG GALILCALL DMCSpline(char *source_filename, char *output_filename, double delta_vector);
```

source_filename	Input file
output_filename	Output file
delta_vector	Vector distance between spline fit points

Description
DMCSpline fits a curve through the points in the input file using the equation:



where v = vector distance

For each segment in the input file coefficients are calculated for each axis then the points are created as a function of the vector distance. A point is created for every v where v = 0 to the total vector distance with increments of *delta_vector*.

The input file must be in the following form:

```
LI n,n,n,n,n,n,n,n < s
LI n,n,n,n,n,n,n,n < s
LI n,n,n,n,n,n,n,n < s
…

where n = number in the range +/-8,000,000
s = speed of the vector in the range 2-8,000,000
```

Each segment must be at least 2 counts long or an error will occur. If a speed value is not specified do not use the less than sign. A negative speed is ignored. The number of axis is defined by the number of commas in the first line. Make sure the number of commas match on each and every line. The LI (or any non-numeric characters) at the beginning of the line is ignored and can be removed.

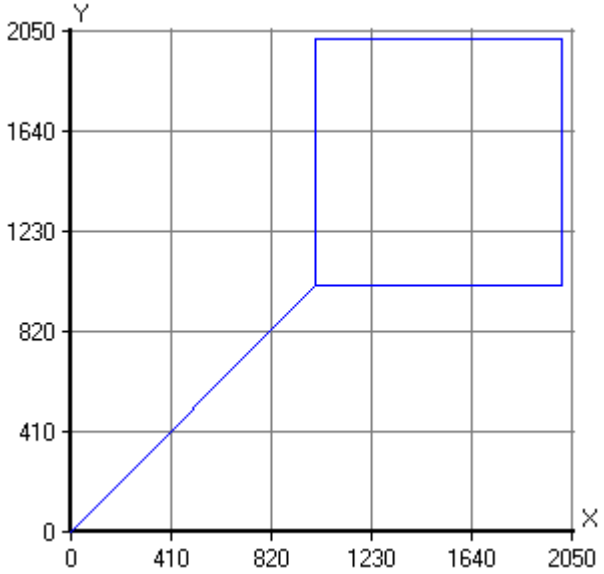
Error Codes
If there are no errors the DMCSpline function returns a 0. Here are the conditions that can generate errors and the code that is returned.

Condition	Code
Axis distance greater than 8 characters long or speed over 12 characters long.	-12
More than 8 axis are specified in input file	-12
Could not open output or input file or input file is empty	-4

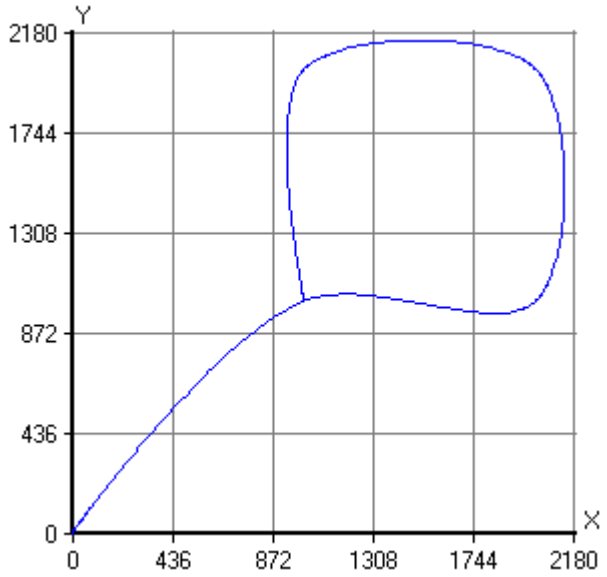
Example
This input file defining a two axis move is used:

```
LI 1000,1000
LI 1000,0
LI 0,1000
LI -1000,0
LI 0,-1000
```

If plotted this is what the original file looks like:



After the spline fit has been run the path looks like this:



Note: A smaller delta_vector will result in a smoother curve but it will also create more LI points in the output file. If the delta_vector is larger than the segment then the no smoothing will occur on that segment.

Here is a sample C program to create a spline motion. This program uses splinein.txt for the input file, splinout.txt for the output and 10 counts for the vector increment.

```
#include "DMCMLIB.H"
int main(void)
{
        long rc;
        char outfile[] = "c:\\splinout.txt";
        char infile[]= "c:\\splinein.txt";
        HANDLEDMC hDmc;
        HWND hWnd;
        char szBuffer[80];

        //make the spline fit
        rc = DMCSpline(infile, outfile, 10.0);

        //send the output file
        rc = DMCOpen(1, hWnd, &hDmc);
        rc = DMCSendFile(hDmc, filename);

        //start motion
        rc = DMCCommand(hDmc, "LE", szBuffer, sizeof(szBuffer));
        rc = DMCCommand(hDmc, "BGS", szBuffer, sizeof(szBuffer));

        rc = DMCClose(hDmc);

        return 0;
}
```

DMCSCurve

This function generates an S-curve motion profile for a given point to point move.

```
ULONG GALILCALL DMCSCurve(char Axis, unsigned long Distance, unsigned long Speed,
       unsigned long Acceleration, unsigned long Jerk, PSZ FileName);
```

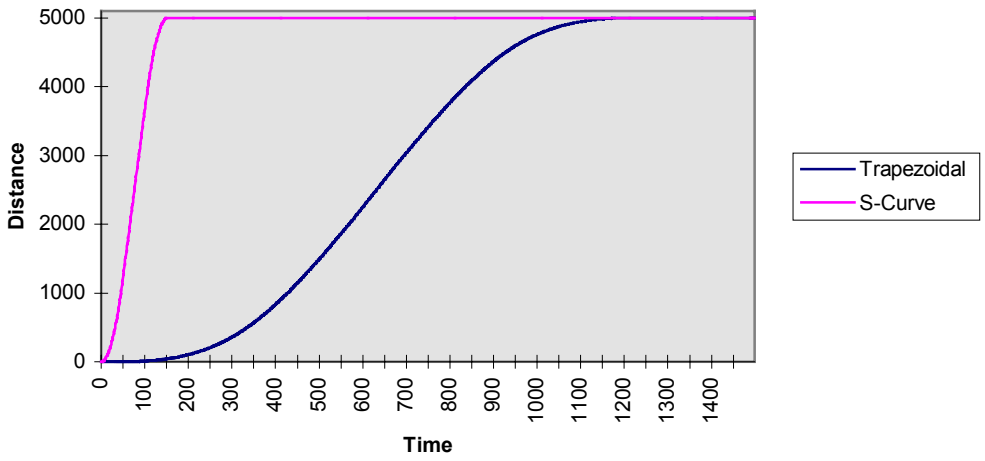
Axis	Axis on which S-curve is to be performed
Distance	Length of move, expressed in encoder counts
Speed	Desired slew speed (encoder counts/sec)
Acceleration	Desired acceleration rate (encoder counts/sec$^{2)}$. The deceleration rate is set to the same value so that the motion profile is symmetrical.
Jerk	Maximum acceptable jerk (encoder counts/sec^3)
FileName	Destination path for DMC output file

Description

This function generates an S-curve motion profile based on a set of motion constraints. If the specified constraints would result in a discontinuous profile (i.e. triangular acceleration or velocity) the function optimizes the parameters so that a true S-curve is generated. The motion profile is generated through a series of contour data (CD) moves.

Note: there is a small amount of rounding error in this function which may cause the actual length of a move to vary from the specified length by a few counts (usually less than 10).

The following figure shows an exaggerated example of an S-curve. The trapezoidal graph is a 5000 count Position Relative move with a slew speed of 25000 and an acceleration of 256000. The S-curve graph is the same move generated by the DMCSCurve function with a limitation on jerk of 10000 counts/sec^3.



Error Codes

If there are no errors, DMCSCurve returns a 0. The following codes are returned in the event of an error:

Condition	Code
Could not open output file	-4
Out of memory	-8
Bad arguments: One or more arguments to a function was NULL or invalid	-12
Invalid Distance parameter	-97
Invalid Speed parameter	-98
Invalid Acceleration parameter	-99

Example
The following C program uses the DMCSCurve function to generate an S-curve, send the resulting output file to the controller and execute it.

```
#include "dmcmlib.h"

void main(void)
{
        long rc;
        char filename[] = "c:\\windows\\desktop\\scurve.dmc";
        HANDLEDMC hDmc;
        HWND hWnd;

// make output file
        rc = DMCSCurve('X',10000,50000,256000,15625000,filename);
// DMCSCurve(char Axis, unsigned long Distance, unsigned long Speed,
// unsigned long Acceleration, unsigned long Jerk, PSZ FileName)

// send output file
        rc = DMCOpen(1, hWnd, &hDmc);
        rc = DMCDownloadFile(hDmc, filename);

// start motion
        rc = DMCCommand(hDmc, "XQ", szBuffer,sizeof(szBuffer));
}
```

In the above example, the X axis is used to generate an S-curve profile. The length of the move is 10000 counts, speed is 50000 counts/sec, acceleration is 256000 counts/sec^2 and jerk is 15625000 counts/sec^3.

After the output file has been generated, it is downloaded to the controller and executed using functions from DMCCOM.H, which has already been included in the DMCMLIB.H header file.

DMCHelix

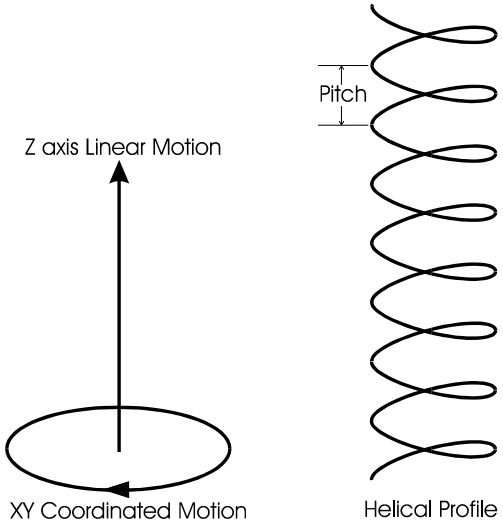
This function generates a helical motion profile. Two axes follow a coordinated circular path while a third axis generates a linear motion perpendicular to the circle.
The function also compensates for offset error (Desired Distance - Actual Distance) which results from round-off error in the GR command.

```
ULONG GALILCALL DMCHelix(char PlaneAxis1, char PlaneAxis2, char TraverseAxis, short Pitch,
        ulong Radius, short StartAngle, long Dist, ulong Speed, ulong Accel, ulong Decel,
        PSZ FileName);
```

PlaneAxis1, PlaneAxis2	Coordinated axes for circular motion
TraverseAxis	Perpendicular axis about which the helix is generated
Pitch	Pitch of helix, expressed in encoder counts traversed per circle. The sign of the pitch determines the direction of the circle: Positive pitch= counter-clockwise, Negative pitch=clockwise
Radius	Radius of circle, expressed in encoder counts
StartAngle	Beginning angle of helix, expressed in degrees
Dist	Desired traverse distance of geared axis
Speed, Accel, Decel	Helical motion parameters. These parameters apply directly to the circular motion, not the traverse motion.
FileName	Destination path for DMC output file

Description
Helical motion is a combination of two types of motion: a circular move in one plane and a linear move perpendicular to the plane of the circle. *PlaneAxis1* and *PlaneAxis2* define the plane of the circle while *TraverseAxis* defines the axis used for the linear move. The *Pitch* parameter relates the circular motion to the linear motion: it is

the number of encoder counts traversed by the *TraverseAxis* per circle. The sign of *Pitch* determines the direction of the circle. For example, in the figure below *Pitch* is a negative number since the circle is drawn in the clockwise direction.



Error Codes
If there are no errors, DMCHelix returns a 0. The following codes are returned in the event of an error:

Condition	Code
Bad arguments: One or more arguments to a function was NULL or invalid	-12
Could not open output file	-4

Example
The following C program uses the DMCHelix function to generate a helix, sends the resulting output file to the controller, and executes it.

```
#include "dmcmlib.h"

void main(void)
{
        long rc;
        char filename[] = "c:\\windows\\desktop\\helix.dmc";
        HANDLEDMC hDmc;
        HWND hWnd;

// make output file
        rc = DMCHelix('X', 'Y', 'Z', -360, 5000, 0, 100000, 50000, 256000, 256000, filename);
// DMCHelix(PlaneAxis1, PlaneAxis2, TraverseAxis, Pitch, Radius, StartAngle, Dist, Speed,
// Accel, Decel, FileName)

// send output file
        rc = DMCOpen(1, hWnd, &hDmc);
        rc = DMCDownloadFile(hDmc, filename);

// start motion
        rc = DMCCommand(hDmc, "XQ", szBuffer,sizeof(szBuffer));

}
```

In the above example, the X and Y axes are used for circular motion while the Z axis is used for linear motion. The pitch of the helix is -360, which means that the Z axis will move forward 360 counts for every XY circle. Since the sign of the pitch is negative, the helix will be generated in the clockwise direction. The radius is 5000, start angle is 0, traversed distance is 100000, speed is 50000, and both acceleration and deceleration are 256000.

DMCGeneralTuning

Use this function to create a DMC program file that can be downloaded and run to automatically tune the servo motor.

```
LONG GALILCALL DMCGeneralTuning(CHAR chAxis, USHORT usStepSize, PSZ pszFileName);
```

chAxis	The axis to be tuned.
UsStepSize	The step size used during the tuning process.
PszFileName	File name use to save the auto tune program.

Description

General tuning is a method used by Galil to automatically tune servo motors. The DMCGeneralTuning function creates a Galil program and stores it in a file. The file must be downloaded and then executed on a Galil controller to perform the tuning.

Once the Galil program is run the servo motor will be stepped back and forth and its position error monitored. Gradually the PID values are increased until the motor becomes unstable. The values are then backed off. The final values should be appropriate for most servo systems. Manual fine tuning may be needed to produce the best system response.

While general tuning is the most flexible tuning method from Galil it may not work on all systems. If this is the case, we recommend using WSDK where other tuning methods are available.

Error Codes

If there are no errors the DMCGeneralTuning function returns a 0. A non zero return value indicates an error occurred. See the header file DMCCOM.H for a definition of the error codes.

Example

This C program creates the automatic tuning file. This file is then downloaded and run on the controller.

```
#include "windows.h"
#include "dmccom.h"
#include "dmcmlib.h"

int main(void)
{
        int rc;
        char filename[] = "c:\\gtune.dmc";
        USHORT stepsize = 300;
        HANDLEDMC hDmc;
        HWND hWnd=0;
        char szBuffer[64];
        char axis='X';

        //Create tuning file
        rc = DMCGeneralTuning(axis, stepsize, filename);

        //Open communications with the controller
        rc = DMCOpen(1, hWnd, &hDmc);

        //Download the file
        rc = DMCDownloadFile(hDmc, filename, NULL);

        //Execute the file
        rc = DMCCommand(hDmc, "XQ", szBuffer, sizeof(szBuffer));

        return 0;
}
```

DMCAutoTuning

Use this function to create a DMC program file that can be downloaded and run to automatically tune the servo motor.

```
LONG GALILCALL DMCAutoTuning(CHAR chAxis, double dPulseMagnitudeVolts, USHORT usPulseDurationMS,
       LPSTR lpstrFileName);
```

chAxis	The axis to be tuned.
dPulseMagnitudeVolts	Pulse magnitude in volts to be sent to the amplifier.
usPulseDurationMS	Pulse duration in milliseconds.
lpstrFileName	File name used to save the auto tune program.

Description

DMCAutoTuning is an alternative to the DMCGeneralTuning function to automatically tune servo motors. The DMCAutoTuning function creates a Galil program and stores it in a file. The file must be downloaded and then executed on a Galil controller to perform the tuning.

The program generated by DMCAutoTuning will send a series of pulses to the amplifier of the specified voltage and duration. These disturbances are used to determine the optimum crossover frequency of the system. After the best crossover frequency is found, the PID values are adjusted for best response at the selected frequency. The final values should be appropriate for most servo systems. Manual fine tuning may be needed to produce the best system response.

While auto tuning is the most flexible tuning method from Galil it may not work on all systems. If this is the case, we recommend using WSDK where other tuning methods are available.

Error Codes

If there are no errors the DMCAutoTuning function returns a 0. A non zero return value indicates an error occurred. See the header file DMCCOM.H for a definition of the error codes.

Example

This C program creates the automatic tuning file. This file is then downloaded and run on the controller.

```
#include "windows.h"
#include "dmccom.h"
#include "dmcmlib.h"

void main(void)
{
       int rc;
       char filename[] = "c:\\atune.dmc";
       double pulsesize = 5.5;
       USHORT pulseduration 20;

       HANDLEDMC hDmc;
       HWND hWnd=0;
       char szBuffer[64];
       char axis='X';

       //Create tuning file
       rc =  DMCAutoTuning(axis, pulsesize, pulseduration, filename)

       //Open communications with the controller
       rc = DMCOpen(1, hWnd, &hDmc);

       //Download the file
       rc = DMCDownloadFile(hDmc, filename, NULL);

       //Execute the file
       rc = DMCCommand(hDmc, "XQ", szBuffer, sizeof(szBuffer));
}
```

Application Programming Topics

Introduction

This section discusses a number of topics of interest to a PC programmer interfacing with a Galil controller. In additions, a number of sample programs are included in the DMCWIN directory under \C\Samples, \CPP\Samples, and \VB\Samples.

Downloading Programs to the Controller

Two API functions are provided to download Galil language programs to the controller: DMCDownloadFile and DMCDownloadFromBuffer. DMCDownloadFromBuffer assumes that each line of one or more commands is separated by a CR/LF, just as if your buffer was a mirror image of a text file. The program label argument can be used to download the Galil language program to a specific label or program line, or may be set to NULL to replace any existing Galil language program.

You may also use the Galil "DL" command to build your own download function. If you do, make sure that you set the time-out (using the API function DMCSetTimeout) to zero as the DL command will not return an acknowledgment from the controller (except in the case of an error) until the end of file character ('\' or Cntl-Z) is sent. Remember to restore the time-out value before exiting your function.

Configuring the Galil Registry

The Galil registry API functions can be used to test if your controller is registered at the beginning of your application. The following code illustrates this:

```
GALILREGISTRY galilregistry;

// Is the controller registered?
if (DMCGetGalilRegistryInfo(1, &galilregistry) == DMCERROR_CONTROLLER)
{
  long rc;
  unsigned short controller;

  // Fill in your controller info, e.g., galilregistry.usAddress = 1000
  // Register the controller
  rc = DMCAddGalilRegistry(&galilregistry, &controller);
}
```

Interrupt Handling

Application programs can receive notification of interrupts from the controller by handling the WM_DMCINTERRUPT message which is defined in DMCCOM.H. The wParam or WORD parameter of the message contains the interrupt status from the controller. The lParam or DWORD parameter is the HANDLEDMC (which was returned from DMCOpen or DMCOpen2), which is useful if you have more than one controller installed which may issue interrupts. The WM_DMCINTERRUPT message is posted to your application using either PostMessage or PostThreadMessage depending on if you set the interrupt notification to a window handle (HWND) or a thread Id (see the text for API functions DMCOpen and DMCOpen2). Interrupt notification can be changed by using the API function DMCChangeInterruptNotification. For standard Windows SDK programming, the WM_DMCINTERRUPT message is retrieved by using GetMessage for HWND or GetThreadMessage for thread Id.

For MFC applications, you will need the following code:

In your .H file, manually add the function prototype for the interrupt handling function to your message map, **after** the Class Wizard comments.

```
//{{AFX_MSG(CMyApp)
afx_msg void OnTimer(UINT nIDEvent);
afx_msg void OnCommand1
afx_msg void OnCommand2
//}}AFX_MSG
afx_msg LONG OnDMCInterrupt(UINT nWP, LONG nLP);
DECLARE_MESSAGE_MAP()
```

In your .CPP file, manually add the ON_MESSAGE macro to your message map, **after** the Class Wizard comments.

```
BEGIN_MESSAGE_MAP(CMyApp, CWinApp)
  //{{AFX_MSG_MAP(CMyApp)
  ON_WM_TIMER()
  ON_COMMAND(IDC_COMMAND1, OnCommand1)
  ON_COMMAND(IDC_COMMAND2, OnCommand2)
  //}}AFX_MSG_MAP
  ON_MESSAGE(WM_DMCINTERRUPT, OnDMCInterrupt)
END_MESSAGE_MAP()
```

In your .CPP file, manually add the interrupt handling function:

```
LONG CMyApp::OnDMCInterrupt(UINT nWP, LONG nLP)
{
  short nStatus = (short)nWP;
  return 0L;
}
```

Diagnostics

There are two API functions DMCDiagnosticsOn and DMCDiagnosticsOff which can be used to provide a detailed trace of communications between your application and the controller. The trace output gets very large very quickly, but it does provide a lot of detail, especially when error conditions arise.

To cut down on the amount of trace being output, call DMCDiagnosticsOn as close to the part of your program you wish to debug as possible. In addition, DMCDiagnosticsOn can be called with a flag to append trace output rather than replace trace output.

Managing the Time-out

The time-out value used by the API functions is often misinterpreted. The time-out value is used by API functions such as DMCCommand to control how long the function will wait before being able to send a command to the controller (controller status set to ready-to-receive) or how long to wait for a complete response (usually a ":" or "?"). Lowering the time-out value does not mean that the DMCCommand or other API functions will respond faster, as most of the time the API functions return long before the time-out has expired. In fact, setting the time-out value too low will cause some functions, such as DMCDownloadFile to fail. For most Galil commands and API functions, the default value of 1000 is usually adequate. Some Galil commands such as master reset (^R^S or the API function DMCMasterReset) may take up to five seconds to complete, so the time-out value may need to be temporarily raised. If you do not care about the response from the controller and/or about time-out conditions, you can set the time-out value to zero.

The time-out value is initialized to the value stored in the Windows Registry database each time you call DMCOpen. You can get the current time-out value with the API function DMCGetTimeout. You can set the current time-out value with the API function DMCSetTimeout.

Note: the lowest you can set the time-out value in the Windows Registry database is 1 ms. The default time-out value is 1000 ms or 1 second.

Low-Level I/O

There are two functions available for performing low-level I/O: DMCWriteData and DMCReadData. They are ideal if you need to send data to the controller which is time critical, such as contour data or linear interpolation segments. The most important point to remember if you use DMCWriteData to send commands to the controller is that you need to periodically call DMCReadData or DMCClear to clear the outbound FIFO or communications buffer. Failure to do so will cause communications and application programs to halt if the outbound FIFO or communications buffer fills up (unless CW,1 is sent). On Galil bus controllers, the outbound FIFO is configured to hold 512 bytes.

Multiple Thread Applications

If you are using multiple threads in your application program and more than one thread makes calls to the Galil DLLs, your program is protected because the API functions are thread safe. The thread safety also allows concurrent communications on different handles.

Waiting for Motion to Complete

A common programming task in communicating with Galil controllers is having to write a wait for motion to

complete function. This is because sending a Galil command such as AMX (wait for the motion on the X axis to complete before continuing) from the PC to the controller will cause communications to/from the controller to stop until the motion is complete. The API function named DMCWaitForMotionComplete does most of the work for you. The function is multi-threaded so as to query the controller in the most efficient manner. The function has an argument named *fDispatchMsgs* that is used to determine whether or not to keep getting and dispatching Windows messages while waiting. For console programs or program threads which have no windows, the argument should be set to FALSE since there is no message loop. For programs or program threads which have windows, the argument should be set to TRUE so your program can continue to process messages while waiting.

Binary Communications

All Galil controllers are communicated with in ASCII format. That is, you send commands in ASCII and the controller responds in ASCII. All controllers but the DMC-1000, 1300, 1500, 1410, 1411, 1412, and 1417 add a binary form of communication: you can send commands in binary format. The controller responds to binary commands the same as ASCII commands: by ASCII. The advantage of sending commands in binary is speed of execution. If the command is already in binary form, it does not need to be decoded by the controller, resulting in higher throughput. Sending contour data is one area where speed is critical and may benefit from sending commands in binary.

Use the API function DMCBinaryCommand to send a command to the controller in binary. Use the API function DMCSendBinaryFile to send a file of binary commands to the controller. In addition, there are API functions to convert commands and files to/from ASCII/binary. Note that converting a command from ASCII to binary in-line or on-the-fly may still result in improved throughput. Following is an example of converting a command from ASCII to binary on-the-fly then sending it to the controller.

```
LONG  rc;
CHAR  szCommand = "PR1000,2000"
CHAR  szResponse[256];
BYTE  BinaryCommand[32];
ULONG BinaryCommandLength;

// Convert a command from ASCII to binary
rc = AsciiCommandToBinaryCommand(szCommand, strlen(szCommand), BinaryCommand,
  sizeof(BinaryCommand), &BinaryCommandLength);

// Send the binary command to the controller and get the response
rc = BinaryCommand(BinaryCommand, BinaryCommandLength, szResponse,
  sizeof(szResponse));
```

Data Record Access

Some Galil controllers can provide a block of data in binary format containing the state of the axes and I/O. This data record should be used instead of conventional commands (TP, TI…) when many data items are required at a high rate (for example to refresh a graphical user interface). The data record can be up to 264 bytes long.

There are four different methods to access the data record: Primary Communication Channel (QR command), Secondary FIFO, DMA, and Dual Port RAM. Regardless of which method is used, the format of the data is the same. The following table shows which controllers support which method(s):

Table 1. Controllers that support data record access.

Controller	Bus	QR Command	Secondary FIFO	Direct Memory Access (DMA)	Dual Port RAM
DMC-1200	PC/104	■			
DMC-14x5/6	Ethernet/Serial	■			
DMC-1600	Compact PCI		■		
DMC-1700	ISA		■	■	
DMC-1800	PCI		■		■
DMC-18x2	PCI	■			
DMC-2xxx	Ethernet/Serial	■			
DMC-3xxx	Ethernet/Serial	■			

DMC-1200, 14x5/6, 18x2, 2xxx, 3xxx

For controllers with only one communication port like the DMC-18x2 and DMC-21x3, the data record is requested by the QR command. This command is issued over the Primary communication channel like any other command, but the response is in a compact binary form rather than ASCII.

DMC-1700

For DMC-1700 controllers, the data record is sent from the controller to the PC using either DMA or the secondary FIFO channel. The command DR is used to set the update rate and select the Data Record Access method (DMA or FIFO). The data record is always available, even if the primary FIFO channel is blocked because a trip-point (such as AM) is pending.

DMC-1800

For DMC-1800 controllers, the data record may be accessed through the secondary FIFO or Dual Port RAM. (Previous board versions supported only the secondary FIFO--Rev. G and below for 1-4 axes board, and Rev. C and below for 5-8 axes board). Version 7.0.3.0 of the PCI drivers (Glwdmpci.sys for Windows XP, 2000, ME, and 98SE, and GalilPCI.sys for NT4.0) automatically accesses the data record using the Dual Port RAM on controllers with firmware version M1 and higher. The command DR is used to set the update rate and DU selects the data record access method (Dual Port RAM or FIFO).

Data Record Structure

Table 2. The data items available in the data record, the constant IDs uses to access them, and their sizes.

Data Item	ID for DMCGetDataRecordByItemId	Data Type
Sample Number (time stamp)	DRIdSampleNumber	unsigned short int
Inputs 1 – 8	DRIdGeneralInput0	unsigned char
Inputs 9 – 16	DRIdGeneralInput1	unsigned char
Inputs 17 – 24	DRIdGeneralInput2	unsigned char
Inputs 25 – 32	DRIdGeneralInput3	unsigned char
Inputs 33 – 40	DRIdGeneralInput4	unsigned char
Inputs 41 – 48	DRIdGeneralInput5	unsigned char
Inputs 49 – 56	DRIdGeneralInput6	unsigned char
Inputs 57 – 64	DRIdGeneralInput7	unsigned char
Inputs 65 – 72	DRIdGeneralInput8	unsigned char
Inputs 73 – 80	DRIdGeneralInput9	unsigned char
Outputs 0 – 8	DRIdGeneralOutput0	unsigned char
Outputs 9 – 16	DRIdGeneralOutput1	unsigned char
Outputs 17 – 24	DRIdGeneralOutput2	unsigned char
Outputs 25 – 32	DRIdGeneralOutput3	unsigned char
Outputs 33 – 40	DRIdGeneralOutput4	unsigned char
Outputs 41 – 48	DRIdGeneralOutput5	unsigned char
Outputs 49 – 56	DRIdGeneralOutput6	unsigned char
Outputs 57 – 64	DRIdGeneralOutput7	unsigned char
Outputs 65 – 72	DRIdGeneralOutput8	unsigned char
Outputs 73 – 80	DRIdGeneralOutput9	unsigned char
Error Code	DRIdErrorCode	unsigned char
Status	DRIdGeneralStatus	unsigned char
Segment Count (S)	DRIdSegmentCountS	unsigned short int
Coordinated Move Status (S)	DRIdCoordinatedMoveStatusS	unsigned short int
Coordinated Move Distance (S)	DRIdCoordinatedMoveDistanceS	long
Segment Count (T)	DRIdSegmentCountT	unsigned short int
Coordinated Move Status (T)	DRIdCoordinatedMoveStatusT	unsigned short int
Coordinated Move Distance (T)	DRIdCoordinatedMoveDistanceT	long
Analog Input 1*	DRIdAnalogInput1	short int
…	…	…
Analog Input 8*	DRIdAnalogInput8	short int

* Since the Analog Input data is contained in the Axis Information section, you can only retrieve data for the analog inputs up to the number of axes on your controller. This is because Analog Input 1 is with the X axis information, Analog Input 2 with the Y axis information, and so on.

Axis Information (Repeated for each Axis)

Status	DRIdAxisStatus	unsigned short int
Switches	DRIdAxisSwitches	unsigned char
Stop Code	DRIdAxisStopCode	unsigned char
Reference Position	DRIdAxisReferencePosition	long
Motor Position	DRIdAxisMotorPosition	long
Position Error	DRIdAxisPositionError	long
Auxiliary Encoder Position	DRIdAxisAuxillaryPosition	long
Velocity	DRIdAxisVelocity	long
Torque	DRIdAxisTorque	short int

Data Record API Examples

Several API functions are provided to use the Data Record Access feature. **DMCRefreshDataRecord** gets a new copy of the data record and places it in system memory. This function must be called each time you wish to update the data record. **DMCGetDataRecordByItemId** gets a specific item from the copy of the data record already in memory (DMCDRC.H defines the data record item IDs). **DMCCopyDataRecord** is used to copy the data record from system memory to a local buffer in your application program.

Advanced users can use **DMCGetDataRecordConstPointer** for even higher-speed access, **DMCGetDataRecordArray** to access many data records stored over time, or the QR command if DMCWin32 is not used or to custom tailor the size of the data record.

DMCGetDataRecordByItemId

Use the DMCGetDataRecordByItemId function to return a single record item as in this Win32 console example:

```
#include "windows.h"
#include "Dmccom.h"
#include "stdio.h"

int main(void)
{
        long rc;
        HANDLEDMC hDmc;
        HWND hWnd=0;
        ULONG length=0;
        USHORT DataType;
        ULONG data;

        //Connect to controller #1
        rc = DMCOpen(1,hWnd, &hDmc);

        if (rc==DMCNOERROR)
        {
                //refresh local copy of data record
                rc = DMCRefreshDataRecord(hDmc, length);

                //get data record data item
                rc = DMCGetDataRecordByItemId(hDmc, DRIdAxisMotorPosition, DRIdAxis1,
                &DataType, &data);

                printf("Position of X is: %d%",(int)data);
        }

        rc = DMCClose(hDmc);

        return 0;
}
```

DMCRefreshDataRecord is used to copy the data record into a buffer in the DLL. All calls to read a record item are done on this local copy. If the data is time sensitive, the buffer should be refreshed just before any call to DMCGetRecordByItemId.

The item returned from DMCGetRecordByItemId is specified by the second parameter passed to the function. In this case, the constant DRIdAxisMotorPosition is passed to get the motor position. The third parameter, DRIdAxis1, specifies the axis. The constants used to define the record items are in the header file Dmcdrc.h and are automatically included by Dmccom.h.

DMCCopyDataRecord

DMCCopyDataRecord can be used to retrieve a separate copy of the current data record (current as of the last call to the function **DMCRefreshDataRecord**) that will not be overwritten by the next call to **DMCRefreshDataRecord**. The structure DMCDATARECORD can be used as template for the data returned. The actual length of the data returned can be determined by using the function **DMCGetDataRecordSize**. You MUST allocate sufficient storage for the entire data record before calling the function **DMCCopyDataRecord**. An example follows:

```
#include "windows.h"
#include "Dmccom.h"
#include "stdio.h"

int main(void)
{
        long rc;
        HANDLEDMC hDmc;
        HWND hWnd=0;
        ULONG length=0;
        USHORT RecordSize;
        DMCDATARECORD MyDataRecord;

        //Connect to controller #1
        rc = DMCOpen(1,hWnd, &hDmc);

        if (rc==DMCNOERROR)
        {
                //refresh local copy of data record
                rc = DMCRefreshDataRecord(hDmc, length);

                //make sure the data record will fit in the structure
                rc = DMCGetDataRecordSize(hDmc, &RecordSize);

                if (sizeof(MyDataRecord) >= RecordSize)
                        //copy data record to the structure
                        rc = DMCCopyDataRecord(hDmc, (PVOID)&MyDataRecord);

                printf("Position of X is:%d%", (int)
                MyDataRecord.AxisInfo[0].MotorPosition);
        }

        rc=DMCClose(hDmc);

        return 0;
}
```

Here the data record is first copied to a local buffer with **DMCRefreshDataRecord** and then placed into our data record structure with **DMCCopyDataRecord**. As with the single item method the data must be refreshed before being read to make sure it is up to date.

Note: you may wish to alter the DMCDATARECORD structure depending on your controller's configuration. See the header file DMCDRC.H for more details.

Advanced: DMCGetDataRecordConstPointer

There is a very slight performance penalty when using **DMCGetDataRecordByItemId**. To gain a little more speed, you could use the API function **DMCGetDataRecordConstPointer** instead.

The following sample code demonstrates how to retrieve the sample number:

```
#include <windows.h>
#include "dmccom.h"
#include <stdio.h>

long rc;
HANDLEDMC hDmc;
HWND hWnd;
ULONG DRlength=0;
USHORT DataType;
LONG oldDRData=0;
const char *pchDataRecord;
unsigned short *pMyLong;
USHORT usOffset;       //Total offset (number of bytes) from the beginning of the data
                       //record to the data item. Output Only.
USHORT usDataType;     //Data type of the data item. The data type of the
                       //data item is returned on output. Output Only.


void main(int argc, char* argv[])
{
        rc=DMCOpen(1, hWnd, &hDmc);  //Connect to controller #1 in Galil registry

        if (rc==DMCNOERROR)
        {
                //Find data record offset
                rc= DMCGetDataRecordItemOffsetById(hDmc, DRIdSampleNumber, DRIdAxis1, &usOffset,
                        &usDataType);

                //Setup constant pointer to data record
                rc= DMCGetDataRecordConstPointer(hDmc, &pchDataRecord);

                //Define data item variables using the offsets
                pMyLong = (unsigned short*)(pchDataRecord + usOffset);

                do
                {
                        rc= DMCRefreshDataRecord(hDmc,DRlength);
                        printf("SAMPLE NUMBER: %d\n", (int)*pMyLong);
                }while(1);

                rc=DMCClose(hDmc); //Close connection
        }
}
```

Advanced: DMCGetDataRecordArray (DMC-1700, 1800)

With Glwdmpci.sys, Glwdmisa.sys, and GalilPCI.sys driver versions 7.0.0.0 and higher, one can use the functions **DMCGetDataRecordArray** or **DMCGetDataRecordConstPointerArray** to obtain pointers to the data records. The number of data records (cache depth) is set in the Galil Registry.

The following example uses the **DMCGetDataRecordArray** to get an array of data records and then retrieves the sample number and X axis motor position from those records.

```
long main(int argc, char* argv[])
{
        long                    rc = 0L;
        int                     i=0,sample,xpos;
        char                    szBuffer[256];
        HANDLEDMC  hDmc;
        const char  *  pointer2record;
        USHORT numberofrecord=0;
        USHORT datarecordsize;
        struct CDMCFullDataRecord * pdatarecordarray;
        USHORT numberofdatarecord=0;

        /* Open a handle to controller number 1 */
        rc = DMCOpen2(5, GetCurrentThreadId(), &hDmc);
        if (rc)
        {
                printf("Could not open controller number 5. RC=%ld\n", rc);
                return rc;
        }

        rc = DMCCommand(hDmc, "DR-1", szBuffer, sizeof(szBuffer));
        Sleep(50);  //give enough time to fill up the data record  catch.
        rc = DMCGetDataRecordSize(hDmc, &datarecordsize);
        if (rc)
        {
                printf("Could not get data record size. RC=%ld\n", rc);
                DMCClose(hDmc);
                return rc;
        }

        rc = DMCGetDataRecordArray(hDmc, &pdatarecordarray, &numberofdatarecord);
        if (rc)
        {
                printf("Could not get data record. RC=%ld\n",rc);
                DMCClose(hDmc);
                return rc;
        }

        while (i<numberofdatarecord){
                sample = pdatarecordarray[i].SampleNumber;
                xpos = pdatarecordarray[i].AxisInfo[0].MotorPosition;
                printf("sample %d  position %d \n",sample,xpos);
                i=i+1;
        }

        /* Close the handle */
        DMCClose(hDmc);
        return rc;
}
```

Advanced: QR Command (DMC-14x5/6, 18x2, 2xxx, 3xxx)

For controllers that do not have a second communication channel, the QR command can be used to retrieve a data record. This command returns, in binary format, a copy of the current data record on demand. There is no need to call the function **DMCRefreshDataRecord**. Use the DMCDATARECORDQR structure (defined in DMCDRC.H) to overlay the binary data returned. It can be used with the function DMCCommand in the following way:

```
#include "windows.h"
#include "Dmccom.h"
#include "stdio.h"

int main(void)
{
        long rc;
        HANDLEDMC hDmc;
        HWND hWnd=0;
        DMCDATARECORDQR MyDataRecordQR;

        //Connect to controller #1
        rc = DMCOpen(1,hWnd, &hDmc);

        if (rc==DMCNOERROR)
        {
                rc = DMCCommand(hDmc, "QR\r", (LPCHAR)&MyDataRecordQR,
                sizeof(MyDataRecordQR));

                printf("X Pos: %d", MyDataRecordQR.DataRecord.AxisInfo[0].MotorPosition);
        }

        rc=DMCClose(hDmc);

        return 0;
}
```

QR is sent using **DMCCommand** and a pointer to the response is returned. This pointer has been cast as a LPCHAR so that the DMCCommand would accept it. The QR command can return a full copy of the data record or a subset depending on the arguments used with it. For more information, consult the Galil Command Reference for your controller.

Note: you may wish to alter the DMCDATARECORDQR structure depending on your controller's firmware and number of axes. See the header file DMCDRC.H for more details.

Distributing Your Application

This section lists the run time files that need to be distributed with software developed with DMCWIN. For additional information, please consult **DISTNOTE.TXT** located within the DMCWIN directory.

Windows 98 Second Edition, ME, 2000, Windows XP

For PCI bus controllers, use GLWDMPCI.SYS. For ISA bus controllers, use GLWDMISA.SYS. For USB, use GLWDMUSB.SYS. These files must reside in the WINDOWS\SYSTEM32\DRIVERS directory.

These device drivers are normally installed using .INF files. You can place GLWDMPCI.INF, GLWDMISA.INF, and GLWDMUSB.INF in the WINDOWS/INF or WINNT/INF directory and the operating system will find the Galil INF files when looking for a driver for the controller.

You should also copy DMC32.DLL and DMCBUS32.DLL (for bus controllers), or DMCSER32.DLL (for serial controllers) in the WINDOWS\SYSTEM or WINNT\SYSTEM32 directory. You will also need to copy and register dmcreg.ocx.

Note: the WINDOWS or WINNT directory may have a different, user-specified name.

Windows NT 4.0

For ISA bus controllers, copy GALIL.SYS to the WINNT\SYSTEM32\DRIVERS directory. For PCI or CompactPCI controllers, copy GALILPCI.SYS to the WINNT\SYSTEM32\DRIVERS directory.

You should also copy DMC32.DLL and DMCBUS32.DLL (for bus controllers), or DMCSER32.DLL (for serial controllers) in the WINNT\SYSTEM32 directory. You will also need to copy and register dmcreg.ocx with the communication dlls.

Note: the WINNT directory may actually have a different, user-specified name.

Microsoft Run-time Files

Make sure the target PC has the MSVCRT.DLL and MFC42.DLL Microsoft run-time files.

`Note`: If your project is built using Visual Basic or Delphi, or a Version of Visual C++ other than 6.0, you will also need to distribute the run-time files for those tools.