micronix USA
PRECISION MOTION SOLUTIONS

MicronixMMCLib DLL Guide

Introduction

This documentation describes the functionality of the Dynamic Link Library (DLL): **MicronixMMCLib.dll**. DLLs provides the standard benefits of shared libraries, such as modularity. A single DLL can be shared by several programs and applications and provide generic interface and functionality of the MMC Devices.

Features

- Serial Port Communication with MMC Devices
  - Connect to / Disconnect from MMC Devices via COM Port
  - Send command to / Receive data from MMC Devices.

Supported MMC Devices

MMC-10 , MMC-100 , MMC-103 , MMC-110 , MMC-200

Current List of Functions

- openComPortMMC(string port)
- closeComPortMMC()
- writeCommandToMMC(string axis, string command)
- pollValues(string axis, string command)
- cleanText(string text)
- statusPoll(string axis)
- positionPoll(string axis)
- velocityPoll(string axis)
- getErrors(string axis)
- waitForStop(string axis, int timeout)
- calcMoveTime(string axis, float moveDistance)

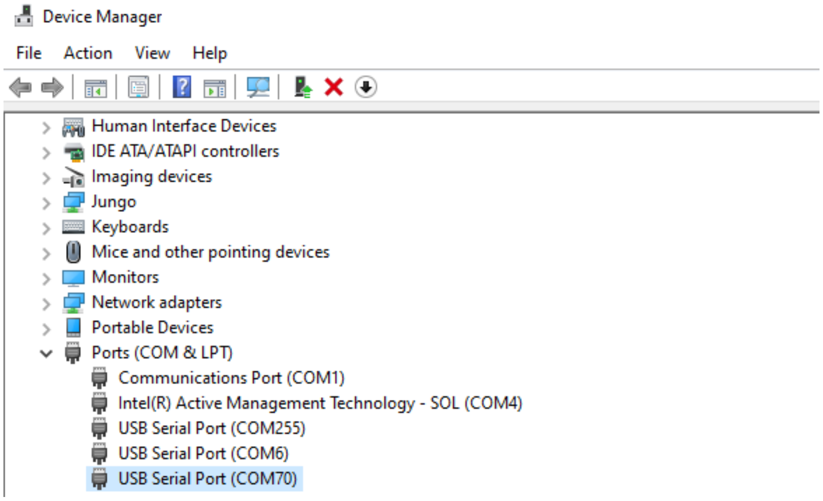
bool openComPortMMC(string *port*)

Function:

Connects to the MMC Device Virtual COM Port through the USB connection.

This function requires user to know the name of the COM port of the MMC Device.

COM port name can be found in Device Manager as “USB Serial Port (COMx)”



- Human Interface Devices
- IDE ATA/ATAPI controllers
- Imaging devices
- Jungo
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Portable Devices
- Ports (COM & LPT)
  - Communications Port (COM1)
  - Intel(R) Active Management Technology - SOL (COM4)
  - USB Serial Port (COM255)
  - USB Serial Port (COM6)
  - USB Serial Port (COM70)

Return Value:

bool output (True or False)

- True is returned when COM port is successfully connected.
- False is returned when COM port fails to connect.
  - Reason for not connecting properly is usually because COM is already accessed.

C Example

```
string port = “COM3”;
bool isOpen = MicronixMMCLibrary.openComPortMMC(port);
if ( isOpen )
{
        Console.WriteLine(“MMC Device on ” + port + “ has been successfully connected.”);
}
else if ( !isOpen )
{
        Console.WriteLine(“COM port is not available.”);
}
```

bool closeComPortMMC( )

Function:

Disconnects the MMC Device opened from the openComPortMMC() function.

Return Value:

bool output (True or False)

- True is returned when COM port is successfully disconnected.
- False is returned when COM port fails to disconnect.
    - Reason for not connecting properly is usually because COM is already disconnected ie. unplugging the USB cable from the MMC Device.

C Example

```
string port = “COM3”;
bool isOpen = MicronixMMCLibrary.openComPortMMC(port);
if ( isOpen)
{
        Console.WriteLine(“MMC Device on ” + port + “ has been successfully connected.”);

        bool isClose = MicronixMMCLibrary.closeComPortMMC();
        if (isClose)
        {
                Console.WriteLine(“MMC Device successfully disconnected”)
        }
}
else if ( !isOpen )
{
        Console.WriteLine(“COM port is not available.”);
}
```

bool writeCommandToMMC(string *axis* , *string command*)

Function:

Writes/Send commands to the MMC Device.

First parameter, string *axis,* is the Axis number of the MMC Device

Second parameter, string *command*, is based on Micronix Motion Control Language.

*Full list of available commands is found in the MMC Device’s reference manuals.

Return Value:

bool output (True or False)

- True is returned when MMC device’s COM port is successfully connected to be able to send command to the device.
- False is returned when COM port is not connected.

C Example

```
string axis = “1”;
string abs_move = “MVA”;                //Send Move to Absolute Position 10mm command
string distance = “10";

string command = abs_move + distance;

MirconixMMCLibrary.writeCommandToMMC(axis, command);
```

string pollValue (string *axis* , string *command*)

Function:

Query data based on the MMC Device’s Axis and its respective parameter to query.

For example, if the user would like to query the current speed of the MMC Device, then the user would query the velocity command (VEL).

Return Value:

string output of the *axis+command* sent

- Consult the MMC Device’s reference manual for full list of commands that can be polled and the expected return value

C Example

```
string axis = “1”;
string command = “VER”       //poll 1VER? which outputs the MMC device’s firmware name
string output = MicronixMMCLibrary.pollValue(axis, command);

ConsoleWriteLine(output);

//Output for 1VER? on an MMC-100 would be “#MMC-100v2.00.00\n\r”
```

```
string position = MicronixMMCLibrary.pollValue(“1” , “POS”);
Console.WriteLine(position);

//Output for 1POS? would be “#[Target Position],[Encoder Position]”
```

string cleanText (string *text*)

Function:

Used in conjunction with pollValue() function where the ‘#’ character, newline character, and return character are omitted from the pollValue() output

For example, the typical output response from pollValue() is “ #**output**/n/r ”. If the user would like just the “**output**” without the unnecessary characters, then the user would input the pollValue() output into the cleanText().

Return Value:

string output of the *text* sent where ‘#’ character, newline character, and return character are removed in *text*

- Consult the MMC Device’s reference manual for full list of commands that can be polled and the expected return value

C Example

```
string axis = “1”;
string command = “VER”        //poll 1VER? which outputs the MMC device’s firmware name
string output = MicronixMMCLibrary.pollValue(axis, command);

ConsoleWriteLine(output);
//Output for 1VER? on an MMC-100 would be “#MMC-100v2.00.00\n\r”

string short_output = MicronixMMCLibrary.cleanText( output );
ConsoleWriteLine(short_output)
//Short_Output should be “MMC-100v2.00.00”
```

```
string position = MicronixMMCLibrary.cleanText(MicronixMMCLibrary.pollValue(“1” , “POS”));
Console.WriteLine(position);
//Output for 1POS? would be “[Target Position],[Encoder Position]”
//Note that ‘#’ , newline character ‘/n’, and return character ‘/n’ omitted
```

bool[] statusPoll (string *axis*)

Function:

Query status byte decimal based on the MMC Device's Axis and performs a binary conversion that converts the status byte decimal into a bool array of 8 binary.

For example, if the user would like to query the status of the MMC Device, then the user would query the status command (STA) using pollValue(axis, "STA") or use the statusPoll() function.

Return Value:

bool[] (a bool array of 8) output of the *axis* sent

- bool[0] : Negative Limit; if true, then negative limit is activated
- bool[1] : Positive Limit; if true, then positive limit is activated
- bool[2] : Program is Running; if true, then an internal program on the MMC Device is running
- bool[3] : In Position; if true, then stage is in idle (FBK0) or in deadband (FBK2 ot FBK3)
- bool[4] : Deceleration; if true, then the stage is in deceleration phase
- bool[5] : Constant Velocity; if true, then the stage is in constant velocity phase.
- bool[6] : Acceleration; if true, then the stage is in acceleration phase
- bool[7] : Error; if true, then an error has occurred and the MMC device should have a red LED indicator on

C Example

```
string axis = “1”;
bool STA[] = MicronixMMCLibrary.statusPoll(axis);

if( STA[0] ) { Console.WriteLine(“Negative Limit Activated”); }
if( STA[1] ) { Console.WriteLine(“Positive Limit Activated”); }
if( STA[2] ) { Console.WriteLine(“PGM is Running”); }
if( STA[3] ) { Console.WriteLine(“Stage is In Position”); }
if( STA[4] ) { Console.WriteLine(“Stage is in Deceleration Phase ”); }
if( STA[5] ) { Console.WriteLine(“Stage is in Constant Velocity”); }
if( STA[6] ) { Console.WriteLine(“Stage is in Acceleration Phase”); }
if( STA[7] ) { Console.WriteLine(“Errors has occurred”); }
```

float positionPoll (string *axis*)

Function:

Query calculated and encoder position based on the MMC Device’s Axis and separates the outputs into a float array of 2 values.

For example, if the user would like to query the status of the MMC Device, then the user would query the position command (POS) using pollValue(axis, “POS”) or use the positionPoll() function.

Return Value:

float[] (a float array of 2) output based on the *axis* sent

- float[0] : Target Position (in mm or degrees)
- float[1] : Encoder Position (in mm or degrees)
    - Note: Encoder position is only valid for closed loop stages and will not output correctly on open loop stages.

C Example

```
string axis = “1”;
float POS[] = MicronixMMCLibrary.positionPoll(axis);

Console.WriteLine(“Target Position is” + POS[0].ToString() );
Console.WriteLine(“Encoder Position is” + POS[1].ToString() );
```

float velocityPoll (string *axis*)

Function:

Query calculated velocity based on the MMC Device's Axis.

For example, if the user would like to query the encoder velocity of the stage in motion, then the user would query the encoder velocity command (VRT) using pollValue(axis, "VRT") or use the velocityPoll() function.

Return Value:

float output based on the *axis* sent

- float : Encoder velocity (in mm/s or degree/second)

C Example

```
string axis = "1";
float VRT = MicronixMMCLibrary.velocityPoll(axis);

Console.WriteLine("Encoder Velocity is" + VRT.ToString() );
```

string getErrors (string *axis*)

Function:

Query error response based on the MMC Device’s Axis.

For example, if the user would like to query the error response, then the user would query the error command (ERR) using pollValue(axis, “ERR”) or use the getErrors() function.

Return Value:

string output based on the *axis* sent

For a full list of error code, please consult the MMC device’s reference manual

C Example

```
string axis = “1”;

string invalid_command = “1AAA”;
MicronixLibrary.writeCommand(invalid_command);
//AAA is not a valid command so an error will be generated

string error_output = MicronixLibrary.getErrors(axis);
Console.WriteLine( error_output );
//error_output should be “ #Error 26 – AAA – Invalid Command/n/r ”
```

bool waitForStop (string *axis* , int *timeout*)

Function:

Waits for axis to stop or reach timeout

- Timeout is in milliseconds
- Timeout of 0 disables the timeout feature

Return Value:

Returns true whenever either the axis has stopped, or timeout has been reached

C Example

```
string axis = “1”;
int timeout = 0;

string move_1 = MVR10;
string move_2 = MVR-10;

MicronixLibrary.writeCommandToMMC(axis, move_1);
while ( MicronixLibrary.waitForStop( axis, timeout ) )
{
        // wait for motion to stop before starting the next move command
}
MicronixLibrary.writeCommandToMMC(axis, move_2);
while( MicronixLibrary.waitForStop( axis, timeout ) ) { //wait for motion to stop }
```

int calcMoveTime (string *axis* , float *moveDistance*)

Function:

Approxiamtes expected move time

- Move time is in seconds
- moveDistance is in mm or degrees

Return Value:

Returns expected move time in milliseconds

C Example

```
string axis = “1”;
float moveDistance = 10;
string command = “MVR”;

string move_1 = command + moveDistance.ToString();
//move_1 = “MVR10”
string move_2 = command + “-” + moveDistance.ToString();
//move_2 = “MVR-10”

int timeout = MicronixLibrary.calcMoveTime(axis, moveDistance) * 1000;
//Assuming VEL2 (2 mm/s) , ACC100 (100 mm/s^2) , and DEC100 (100 mm/s^2),
//with a move distance of 10 mm
//timeout should be at least 7 seconds or 7,000 milliseconds where
//at least 1 second for acceleration, 5 seconds for constant velocity, and 1 second for
//deceleration

MicronixLibrary.writeCommandToMMC(axis, move_1);
while ( MicronixLibrary.waitForStop( axis, timeout ) )
{
        // wait for motion to stop or wait until timeout is reached
}
MicronixLibrary.writeCommandToMMC(axis, move_2);
while( MicronixLibrary.waitForStop( axis, timeout ) ) { //wait for motion to stop }
```