User Guide

MTCom

Atlas Copco Industrial Technique AB

9836 6568 01
Software release 1.0.3.1
2011-02
Edition 1.2

Contents

Introduction

MTCom is a communication server used for communicating with MicroTorque devices over USB. It provides a unified programming interface and the possibility for several applications to communicate with one single device at the same time. MTCom runs as a windows service and supports 16 simultaneous devices to be connected to a computer at any one time.

MTCom is backwards compatible with G4Com and its DLL can be used as a drop in replacement for applications that use the old G4Com API. A program that previously used G4Com can be made to work with MTCom without the need to recompile it. To make this work the MTCom.dll will need to be named g4com.dll and replace the existing g4com.dll that was installed with the application.

Changes from G4Com

MTCom was written as a replacement for G4Com and contains several improvements to increase stability of the communication and to make it easier for applications to communicate with MicroTorque devices. The main highlights of MTCom are listed below.

- Backwards compatible with G4Com. Can work as a drop in replacement for G4Com.
- Runs as a windows service so that communication will not be disturbed if the system tray application is closed.
- General improvements in stability and error handling.
- Summary protocol is marked with a sequence number so that an application can easier identify results of different joints.
- Support for ACTA-MT4 trace protocol.
- New API provides more control and better integration with C#.

Revision history

Edition	Date	Comment
1.0	2010-09-16	First revision
1.1	2010-09-20	Updated for version 1.0.2.2
1.2	2011-02-16	Updated for version 1.0.3.1

Installation

Before installation

Before installing MTCom the user should make sure that any instances of G4Com are shutdown. MTCom and G4Com can not run simultaneously on one computer since they both try to connect to the same USB devices. G4Com does not need to be uninstalled but if it is started when the MTCom service is already running it will not be able to see any devices.

Installing MTCom

Installation of MTCom is performed by the supplied installation package. By default MTCom will be installed in "`C:\Program files\Atlas Copco Tools AB\MTCom`".

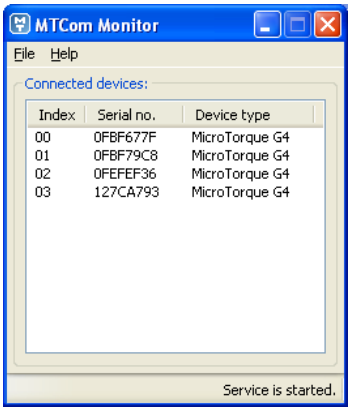
The following files will be placed in the installation folder:

- **MTComSvc.exe** – The MTCom server that runs as a windows service.
- **MTCom.dll** – The MTCom API DLL that provides the communication interface for all applications that communicate with MicroTorque devices. This file can be renamed to g4com.dll and placed in the folder of an application that previously used G4Com.
- **MTComMonitor.exe** – Monitoring application that lists connected devices and provides functionality to start and stop the MTCom service.

When the installation is complete the MTCom service and monitoring application is automatically started by the installtion process. MTCom Monitor will reside in Windows system tray. The MTCom Monitor main window can be opened by clicking the icon in the system tray. The main window of MTCom will display a list of connected devices similar to G4Com.

Starting and stopping MTCom

The MTCom service can be started and stopped from within the MTCom Monitor application. The service should however never be stopped unless the intention is to use G4Com for a time. Stopping the service will stop any ongoing communication with MicroTorque devices.



To stop the service, choose "File -> Stop service" from the menu.

To start the service, choose "File -> Start service" from the menu.

The current status of the MTCom service is displayed in the status bar of the main window.

Troubleshooting

Problem	Possible causes
Unable to start the MTCom service.	The USB driver was not installed correctly. Connect a MicroTorque device and verify that the driver is installed then try starting the service again.
No devices are visible in MTCom Monitor even though they are connected to the computer.	Make sure that there isn't an instance of G4Com running on the computer. MTCom and G4Com can not run at the same time.
Unable to start or stop the MTCom service from within MTCom Monitor.	For MTCom Monitor to be able to start and stop services it needs to run with administrative rights. The remaining functionality of the program will work for all users.

USB communication

Channels

The MTCom server can communicate with up to 16 MicroTorque devices at the same time. For each device connected it will separate the communication into 16 channels on which custom applications can communicate. Some of these channels are reserved for special purposes like protocol output and trace data, others are for general use.

Channel	Purpose
0	General purpose.
1	General purpose.
2	General purpose.
3	General purpose.
4	General purpose.
5	General purpose.
6	General purpose.
7	General purpose.
8	General purpose.
9	General purpose.
10	Reserved, do not use for custom applications.
11	Reserved, do not use for custom applications.
12	Reserved, do not use for custom applications.
13	Summary data protocol output.
14	Summary data protocol output.
15	Trace data protocol output.

Data format

All datasets on general purpose channels have the following structure:

STX	Channel no.	Data	Checksum High	Checksum Low	ETX

Field	Description
STX	ASCII control character 0x02
Channel no.	Channel number data was sent on. ASCII hex character (0-9, A-F)
Data.	The actual data of the dataset.
Checksum High	High nibble of the checksum. ASCII hex character (0-9, A-F)
Checksum Low	Low nibble of the checksum. ASCII hex character (0-9, A-F)
ETX	ASCII control character 0x03

The checksum is calculated as an 8-bit sum (without carry) of the channel no. and data fields.

Example:

```
STX   0    P    G    0    7    2    E    ETX
0x02 0x30 0x50 0x47 0x30 0x37 0x32 0x45 0x03
```

In this example the checksum is the sum of (0x30 + 0x50 + 0x47 + 0x30 + 0x37) = 0x12E. If the checksum is higher than 0xFF only the 8 least significant bits are used. In the example above we use 0x2E. This checksum is then written as an ASCII representation in the dataset, characters 2 and E.

Programming interface

MTCom API functions

The following functions are exported from MTCom.dll. These functions are further explained later in this document.

Function	Short description
MT_Init	Initializes the MTCom DLL. Must be called before any other function.
MT_GetVersion	Returns the version string of MTCom service.
MT_GetDllVersion	Returns the version string of MTCom.dll
MT_GetDeviceList	Returns a semicolon-separated list of connected devices.
MT_GetDeviceInfo	Returns the connection status and device type of a device.
MT_Open	Opens a connection to a device.
MT_Close	Closes a connection to a device.
MT_Clear	Clears any existing datasets of a given device and channel.
MT_WriteSet	Sends a dataset to a connected device.
MT_ReadSet	Retrieves a dataset from a connected device.
MT_GetSummary	Retrieves the summary of the last completed joint.
MT_GetNoTraceChannels	Returns the number of trace channels present on the device.
MT_GetTraceInfo	Returns information of a specified device and trace channel.
MT_GetTracePoints	Returns trace points of a specified device and trace channel.
MT_GetOutput	Returns the state of the internal outputs of a given device.
MT_GetLastError	Returns the error code of the last function that failed.

All functions listed above return **`TRUE`** if successful and **`FALSE`** on failure unless otherwise specified. If a function fails MT_GetLastError will return a more detailed error code.

MT_Init

This function initializes the MTCom DLL and must be called before any other funcions of the MTCom API are used.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_Init(void);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_Init")]
public static extern bool MT_Init();
```

MT_GetVersion

This function is used to retrieve the version number of the running MTCom service. If the function call succeeds **`oVersion`** will contain a null-terminated string of the version number.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_GetVersion(char * oVersion);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_GetVersion")]
public static extern bool MT_GetVersion(
    [MarshalAs(UnmanagedType.LPStr)] StringBuilder version);
```

MT_GetDllVersion

This function is used to retrieve the version number of the MTCom DLL. If the function call succeeds **`oVersion`** will contain a null-terminated string of the version number.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_GetDllVersion(char * oVersion);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_GetDllVersion")]
public static extern bool MT_GetDllVersion(
    [MarshalAs(UnmanagedType.LPStr)] StringBuilder version);
```

MT_GetDeviceList

This function is used to retrieve the serial numbers of the current connected MicroTorque devices. If the function call succeeds **`oDevices`** will contain a semicolon separated list of serial numbers and **`oDeviceCount`** will contain the number of devices connected.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_GetDeviceList(char * oDevices,
    int * oDeviceCount);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_GetDeviceList")]
public static extern bool MT_GetDeviceList(
    [MarshalAs(UnmanagedType.LPStr)] StringBuilder devices,
    out int deviceCount);
```

MT_GetDeviceInfo

This function is used to retrieve information of a device with a given serial number. If the function call succeeds **`oDeviceStatus`** will hold the connection status and **`oDeviceType`** will the type of device that is connected. Refer to the following tables for a list of valid values of both parameters.

Device Status	Description
0	Device with given serial number is not connected.
1	Device is present on the USB but connection to it has not been established yet.
2	Device is connected and MTCom has initiated handshake.
3	Device is ready.

Device Type	Description
0	MTCom has not yet identified the device.
1	'Microtest MC'.
2	'MicroTorque Controller G4'.
3	'MicroTorque ACTA MT4'
4	'MTF400 Basic'
5	'MTF400 Advanced'

C++ declaration:

```
enum MT_DeviceStatus
{
    MT_DEVICE_NOT_FOUND = 0,
    MT_DEVICE_PRESENT,
    MT_DEVICE_CONNECTED,
    MT_DEVICE_READY
};

enum MT_DeviceType
{
    MT_DEVICE_UNKNOWN = 0,
    MT_DEVICE_MICROTEST_MC,
    MT_DEVICE_MICROTORQUE_G4,
    MT_DEVICE_ACTA_MT4,
    MT_DEVICE_MTF400_BASIC,
    MT_DEVICE_MTF400_ADVANCED
};

extern "C" BOOL __stdcall MT_GetDeviceInfo(const char * serial,
```

C# declaration:

```
public enum DeviceStatus : int
{
    MT_DEVICE_NOT_FOUND = 0,
    MT_DEVICE_PRESENT,
    MT_DEVICE_CONNECTED,
    MT_DEVICE_READY
}

public enum DeviceType : int
{
    MT_DEVICE_UNKNOWN = 0,
    MT_DEVICE_MICROTEST_MC,
    MT_DEVICE_MICROTORQUE_G4,
    MT_DEVICE_ACTA_MT4,
    MT_DEVICE_MTF400_BASIC,
    MT_DEVICE_MTF400_ADVANCED
}

[DllImport("MTCom.dll", EntryPoint = "MT_GetDeviceInfo")]
public static extern bool MT_GetDeviceInfo(
    [MarshalAs(UnmanagedType.LPStr)] string serial,
    [MarshalAs(UnmanagedType.I4)] out DeviceStatus deviceStatus,
```

MT_Open

This function opens a connection to a device with a given serial number and returns the handle to the opened device. If it could not open a connection to the device it will return **INVALID_HANDLE_VALUE**.

(The reserved parameter is reserved for future use and should be set to 0).

C++ declaration:

```
#define INVALID_HANDLE_VALUE ((HANDLE)(LONG_PTR)-1)

extern ”C” HANDLE __stdcall MT_Open(const char *serial, int reserved);
```

C# declaration:

```
public readonly static IntPtr INVALID_HANDLE_VALUE = new IntPtr(-1);

[DllImport("MTCom.dll", EntryPoint = "MT_Open")]
public static extern IntPtr MT_Open(
    [MarshalAs(UnmanagedType.LPStr)] string serial, int reserved);
```

MT_Close

This function closes an open connection to a device, it does not return anything.

C++ declaration:

```
extern ”C” void __stdcall MT_Close(HANDLE * client);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_Close")]
public static extern IntPtr MT_Close(ref IntPtr client);
```

MT_Clear

This function clears any previously received datasets on a specified channel. It can be used to clear summary data and summary sequence number for a device if called with channel number 13 or 14.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_Clear(HANDLE client, int channel);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_Clear")]
public static extern bool MT_Clear(IntPtr client, int channel);
```

MT_WriteSet

This function writes a dataset to a device. The application can chose to build entire datasets including control characters, channel number and checksum and pass it to **`MT_WriteSet`** or it can chose to only send the command and let MTCom construct the remaining parts of the dataset.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_WriteSet(HANDLE client, int channel,
    const char * data, int size);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_WriteSet")]
public static extern bool MT_WriteSet(IntPtr client, int channel,
    [MarshalAs(UnmanagedType.LPArray)] byte[] dataset, int noBytes);
```

MT_ReadSet

This funcion reads a dataset from a device. The function will only indicate failure (return **`FALSE`**) if an actual error occurred. If it timed out waiting for a dataset it will return **`TRUE`** and set **`oBytesRead`** to zero.

The byte array passed as **`oData`** must be greater or equal to 4kb in size.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_ReadSet(HANDLE client, int channel,
    char * oData, int *oBytesRead, int timeout);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_ReadSet")]
public static extern bool MT_ReadSet(IntPtr client, int channel,
    [MarshalAs(UnmanagedType.LPArray)] [Out] byte[] buffer,
    [MarshalAs(UnmanagedType.I4)] out int noBytes, int timeOut);
```

MT_GetSummary

This function retrieves the last summary (result of the last joint) as received by MTCom. Summary protocol data is automatically sent on channel 13 and 14 by the MicroTorque device when a joint is completed. Since this function simply returns the last summary as received by MTCom, an application that relies on this function to retrieve summary data should keep track of the sequence number to make sure that the correct summary was retrieved. See the example in the next chapter.

The byte array passed as **`oSummary`** must be equal or greater to 1kb in size.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_GetSummary(HANDLE client, char *oSummary,
    int *oNoBytes, int *oSequenceNo);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_GetSummary")]
public static extern bool MT_GetSummary(IntPtr client,
    [MarshalAs(UnmanagedType.LPArray)] [Out] byte[] summary,
    [MarshalAs(UnmanagedType.I4)] out int noBytes,
    [MarshalAs(UnmanagedType.I4)] out int sequenceNo);
```

MT_GetNoTraceChannels

This function is used to get the maximum number of trace channels of a MicroTorque device.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_GetNoTraceChannels(HANDLE client,
    int * oNoTraceChannels);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_GetNoTraceChannels")]
public static extern bool MT_GetNoTraceChannels(IntPtr client,
    [MarshalAs(UnmanagedType.I4)] out int noTraceChannels);
```

MT_GetTraceInfo

This function is used to get information about the trace of the last joint that was buffered in MTCom. It will retrieve the number of points in the trace, its sample rate (in samples/sec) and the torque unit used.

C++ declaration:

```
enum MT_Unit
{
    MTU_mNm = 0,
    MTU_cNm,
    MTU_Nm,
    MTU_mN,
    MTU_N,
    MTU_kN,
    MTU_inlbf,
    MTU_lbf,
    MTU_inozf,
    MTU_gcm,
    MTU_kgm,
    MTU_ftlbf,
    MTU_ozf,
    MTU_kgf,
    MTU_gf,
    MTU_INVALID = -1,
};

extern ”C” BOOL __stdcall MT_GetTraceInfo(HANDLE client, int traceChannel,
    int * oNoPoints, int * oSampleRate, int * oTorqueUnit);
```

C# declaration:

```
public enum Unit : int
{
    MTU_mNm = 0,
    MTU_cNm,
    MTU_Nm,
    MTU_mN,
    MTU_N,
    MTU_kN,
    MTU_inlbf,
    MTU_lbf,
    MTU_inozf,
    MTU_gcm,
    MTU_kgm,
    MTU_ftlbf,
    MTU_ozf,
    MTU_kgf,
    MTU_gf,
    MTU_INVALID = -1,
}

[DllImport("MTCom.dll", EntryPoint = "MT_GetTraceInfo")]
public static extern bool MT_GetTraceInfo(IntPtr client, int traceChannel,
    [MarshalAs(UnmanagedType.I4)] out int noPoints,
    [MarshalAs(UnmanagedType.I4)] out int sampleRate,
    [MarshalAs(UnmanagedType.I4)] out int torqueUnit);
```

MT_GetTracePoints

This function retrieves the actual trace points of the currently buffered trace in MTCom. **`MT_GetTraceInfo`** should be called first to get the number of points available. The function can be used to retrieve the entire trace or just parts of it depending on the parameters passed to it.

The variable **`traceChannel`** indicates which channel MTCom should retrieve trace data for. Usually this is channel 0 (which is the driver in the case of G4 or the first sensor in the case of ACTA-MT4).

The array of **`MT_TracePoint`** structures passed as **`oTracePoints`** should be able to hold the number of points passed in the variable **`maxPoints`**.

The integer **`startPoint`** indicates the point into the trace of where the MTCom should begin copying the trace data.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_GetTracePoints(HANDLE client,
    int traceChannel, struct MT_TracePoint * oTracePoints,
    int startPoint, int maxPoints, int * oPointsReceived);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_GetTracePoints")]
public static extern bool MT_GetTracePoints(IntPtr client,
    int traceChannel,
    [MarshalAs(UnmanagedType.LPArray)] [Out] TracePoint[] points,
    int startPoint, int maxPoints,
    [MarshalAs(UnmanagedType.I4)] out int noPoints);
```

MT_GetOutput

This returns the state of the internal outputs of a device. The retrieved value is a bit mask and has different meaning depending on which type of device that is queried.

Output	MT-G4 & MTF400	ACTA-MT4
Bit 0	Current state of OK signal.	Last measurement was High.
Bit 1	Current state of BUSY signal.	Last measurement was Low.
Bit 2	Not used.	Last measurement was OK.
Bit 3	Not used.	Reserved for future use.

C++ declaration:

```
extern ”C” BOOL __stdcall MT_GetOutput(HANDLE client, int *oOutput);
```

C# declaration:

```
[DllImport("MTCom.dll", EntryPoint = "MT_GetOutput")]
public static extern bool MT_GetOutput(IntPtr client,
    [MarshalAs(UnmanagedType.I4)] out int output);
```

MT_GetLastError

This function returns the error code of the last function that failed. It can be called after any MTCom API function failed to get more detailed information about what went wrong.

C++ declaration:

```
enum MT_ErrorCode
{
    MT_OK = 0,
    MT_ERR_INVALID_VERSION = -1,
    MT_ERR_INVALID_PARAMETER = -2,
    MT_ERR_INVALID_DATA = -3,
    MT_ERR_INVALID_CHECKSUM = -4,
    MT_ERR_CONNECTION_ERROR = -5,
    MT_ERR_INVALID_MESSAGE_ID = -6,
    MT_ERR_MUTEX_TIMEOUT = -7,
    MT_ERR_PIPE_READ_FAILED = -8,
    MT_ERR_PIPE_WRITE_FAILED = -9,
    MT_ERR_USB_READ_FAILED = -10,
    MT_ERR_USB_WRITE_FAILED = -11,
};

extern ”C” int __stdcall MT_GetLastError(void);
```

C# declaration:

```
public enum ErrorCode : int
{
    MT_OK = 0,
    MT_ERR_INVALID_VERSION = -1,
    MT_ERR_INVALID_PARAMETER = -2,
    MT_ERR_INVALID_DATA = -3,
    MT_ERR_INVALID_CHECKSUM = -4,
    MT_ERR_CONNECTION_ERROR = -5,
    MT_ERR_INVALID_MESSAGE_ID = -6,
    MT_ERR_MUTEX_TIMEOUT = -7,
    MT_ERR_PIPE_READ_FAILED = -8,
    MT_ERR_PIPE_WRITE_FAILED = -9,
    MT_ERR_USB_READ_FAILED = -10,
    MT_ERR_USB_WRITE_FAILED = -11,
}

[DllImport("MTCom.dll", EntryPoint = "MT_GetLastError")]
public static extern ErrorCode MT_GetLastError();
```

G4Com API (deprecated)

The MTCom DLL is backwards compatible with the G4Com DLL and can be used as a drop in replacement for applications that use the old G4Com API. **These functions are however deprecated and should not be used in newly developed software.** They are included here for reference only.

This section is not yet complete. Refer to the MicroTorque G4 Controller communication protocol, user guide for a list of available functions.

MTCom API examples

MTCom can be used from any language capable of calling external DLL functions. The examples in this document are written in C#.

Initiating communication

This is an example on how to initialize MTCom and how to open a connection to a MicroTorque device. It is written in C# using the MTCom interface class that can be found in this document.

```
// Initialize MTCom DLL
if (MTCom.Init())
{
    string serialNo = "127CA793";

    // Open connection to a device using a known serial number.
    IntPtr client = MTCom.Open(serial, 0);

    if (client != MTCom.INVALID_HANDLE_VALUE)
    {
        // Success, we are now connected to the device.

        // ...

        // Close connection to the device when finished with it.
        MTCom.Close(client);
    }
    else
    {
        // Display error code.
        Console.Write("MTCom.Open failed w/err: {0}.",
            MTCom.GetLastError());
    }
}
```

Writing and reading datasets

This is an example on how to write and read datasets to and from a MicroTorque device. It is written in C# using the MTCom interface class that can be found in this document.

```
// Request the version number of a MicroTorque device using channel 5
byte[] request = System.Text.Encoding.ASCII.GetBytes("IV”);
bool success = MTCom.WriteSet(client, 5, request, request.Length);

if (success)
{
    byte[] reply = new byte[4096];

    // Read back the reply with a one second timeout.
    success = MTCom.ReadSet(client, 5, reply, out noBytes, 1000);

    if (success)
    {
        if (noBytes > 0)
        {
            // We have a valid dataset, display it.
            System.Console.WriteLine("Reply: {0}”,
               System.Text.Encoding.ASCII.GetString(reply, 0, noBytes));
        }
        else
        {
            // Timed out.
        }
    }
}

if {!success)
{
    // Display error code.
    Console.Write("Operation failed w/err: {0}.", MTCom.GetLastError());
}
```

Retrieving summary protocol data

The following code snippet provides an example on how to retrieve a summary of a completed joint. It is written in C# and uses the MTCom interface class that can be found later in this document.

```
// The application is initialized and a connection to a device is opened.

// ...

// Clear the summary data and reset sequence number to zero.
MTCom.Clear(client, 13);
int previousSequenceNo = 0;

while (applicationIsRunning)
{
    // Driver is started and the application checks that the
    // BUSY signal goes from HIGH to LOW.
    success = StartDriverAndWait(client, CHANNEL, 5000);

    // Retrieve summary for the joint
    if (success)
    {
        int sequenceNo, noBytes;
        byte[] summary = new byte[1024];

        for (int retries = 0; retries < 10; retries++)
        {
            // Attempt to retrieve a summary of the completed joint.
            if (MTCom.GetSummary(client, summary, out noBytes,
                    out sequenceNo))
            {
                if (sequenceNo != previousSequenceNo)
                {
                    // Summary received, display it.
                    System.Console.WriteLine("Summary: {0}”,
                        System.Text.Encoding.ASCII.GetString(
                            reply, 0, noBytes));

                    // Save sequence number for next time around.
                    previousSequenceNo = sequenceNo;
                }
            }

            System.Threading.Thread.Sleep(50);
        }
    }
}
```

Retrieving trace data

The following code snippet provides an example on how to retrieve trace data after a joint has been completed. It is written in C# and uses the MTCom interface class that can be found later in this document.

```
// The application is initialized and a connection to a device is opened.

// ...

// The application detects that a joint is completed by checking
// that the BUSY signal goes from HIGH to LOW.

// ...

// Retrieve the details of the buffered trace.
int noPoints, sampleRate, torqueUnit;
success = MTCom.GetTraceInfo(handle, 0, out noPoints, out sampleRate,
                             out torqueUnit);

if (success && noPoints > 0)
{
    // Retrieve the data points of the trace.
    MTCom.TracePoint[] points = new MTCom.TracePoint[noPoints];
    int maxPoints = noPoints;

    success = MTCom.GetTracePoints(handle, 0, points, 0, maxPoints,
                                   out noPoints);

    if (success)
    {
        // Success, display number of trace points received.
        Console.WriteLine("{0} number of points received.", noPoints);
    }
}

if {!success)
{
    // Display error code.
    Console.Write("Operation failed w/err: {0}.", MTCom.GetLastError());
}
```

MTCom interface class

This is a complete listing of the MTCom DLL interface class which can be used in custom C# applications.

MTCom.cs

```
namespace MicroTorque
{
    public class MTCom
    {
        public const int SUMMARY_CHANNEL_1 = 13;
        public const int SUMMARY_CHANNEL_2 = 14;
        public const int TRACE_CHANNEL = 15;

        public enum Unit : int
        {
            MTU_mNm = 0,
            MTU_cNm,
            MTU_Nm,
            MTU_mN,
            MTU_N,
            MTU_kN,
            MTU_inlbf,
            MTU_lbf,
            MTU_inozf,
            MTU_gcm,
            MTU_kgm,
            MTU_ftlbf,
            MTU_ozf,
            MTU_kgf,
            MTU_gf,

            MTU_INVALID = -1,
        }

        public enum DeviceStatus : int
        {
            MT_DEVICE_NOT_FOUND = 0,
            MT_DEVICE_PRESENT,
            MT_DEVICE_CONNECTED,
            MT_DEVICE_READY
        }

        public enum DeviceType : int
        {
            MT_DEVICE_UNKNOWN = 0,
            MT_DEVICE_MICROTEST_MC,
            MT_DEVICE_MT_G4,
            MT_DEVICE_ACTA_MT4,
            MT_DEVICE_MTF400_BASIC,
            MT_DEVICE_MTF400_ADVANCED
        }
```

MTCom.cs continued:

```
        public enum ErrorCode : int
        {
            MT_OK = 0,
            MT_ERR_INVALID_VERSION = -1,
            MT_ERR_INVALID_PARAMETER = -2,
            MT_ERR_INVALID_DATA = -3,
            MT_ERR_INVALID_CHECKSUM = -4,
            MT_ERR_CONNECTION_ERROR = -5,
            MT_ERR_INVALID_MESSAGE_ID = -6,
            MT_ERR_MUTEX_TIMEOUT = -7,
            MT_ERR_PIPE_READ_FAILED = -8,
            MT_ERR_PIPE_WRITE_FAILED = -9,
            MT_ERR_USB_READ_FAILED = -10,
            MT_ERR_USB_WRITE_FAILED = -11,
        }

        [StructLayout(LayoutKind.Sequential, Pack = 1, Size = 16)]
        public struct TracePoint
        {
            [MarshalAs(UnmanagedType.R8)]
            public double torque;
            [MarshalAs(UnmanagedType.R8)]
            public double angle;
        }

        public readonly static IntPtr INVALID_HANDLE_VALUE =
            new IntPtr(-1);

        [DllImport("MTCom.dll", EntryPoint = "MT_Init")]
        public static extern bool Init();

        [DllImport("MTCom.dll", EntryPoint = "MT_GetDeviceList")]
        public static extern bool GetDeviceList(
            [MarshalAs(UnmanagedType.LPStr)] StringBuilder devices,
            [MarshalAs(UnmanagedType.I4)] out int numDevices);

        [DllImport("MTCom.dll", EntryPoint = "MT_GetDeviceInfo")]
        public static extern bool GetDeviceInfo(
            [MarshalAs(UnmanagedType.LPStr)] string serial,
            [MarshalAs(UnmanagedType.I4)] out DeviceStatus deviceStatus,
            [MarshalAs(UnmanagedType.I4)] out DeviceType deviceType);

        [DllImport("MTCom.dll", EntryPoint = "MT_Open")]
        public static extern IntPtr Open(
            [MarshalAs(UnmanagedType.LPStr)] string serial,
            int reserved);

        [DllImport("MTCom.dll", EntryPoint = "MT_Close")]
        public static extern void Close(ref IntPtr client);

        [DllImport("MTCom.dll", EntryPoint = "MT_Clear")]
        public static extern bool Clear(IntPtr client, int channel);

        [DllImport("MTCom.dll", EntryPoint = "MT_WriteSet")]
        public static extern bool WriteSet(IntPtr client, int channel,
            [MarshalAs(UnmanagedType.LPArray)] byte[] dataset,
            int noBytes);
```

MTCom.cs continued:

```
        [DllImport("MTCom.dll", EntryPoint = "MT_ReadSet")]
        public static extern bool ReadSet(IntPtr client, int channel,
            [MarshalAs(UnmanagedType.LPArray)] [Out] byte[] buffer,
            [MarshalAs(UnmanagedType.I4)] out int noBytes,
            int timeOut);

        [DllImport("MTCom.dll", EntryPoint = "MT_GetSummary")]
        public static extern bool GetSummary(IntPtr client,
            [MarshalAs(UnmanagedType.LPArray)] [Out] byte[] summary,
            [MarshalAs(UnmanagedType.I4)] out int noBytes,
            [MarshalAs(UnmanagedType.I4)] out int sequenceNo);

        [DllImport("MTCom.dll", EntryPoint = "MT_GetNoTraceChannels")]
        public static extern bool GetNoTraceChannels(IntPtr client,
            [MarshalAs(UnmanagedType.I4)] out int noTraceChannels);

        [DllImport("MTCom.dll", EntryPoint = "MT_GetTraceInfo")]
        public static extern bool GetTraceInfo(IntPtr client,
            int traceChannel,
            [MarshalAs(UnmanagedType.I4)] out int noPoints,
            [MarshalAs(UnmanagedType.I4)] out int sampleRate,
            [MarshalAs(UnmanagedType.I4)] out int torqueUnit);

        [DllImport("MTCom.dll", EntryPoint = "MT_GetTracePoints")]
        public static extern bool GetTracePoints(IntPtr client,
            int traceChannel,
            [MarshalAs(UnmanagedType.LPArray)] [Out] TracePoint[] points,
            int startPoint, int maxPoints,
            [MarshalAs(UnmanagedType.I4)] out int noPoints);

        [DllImport("MTCom.dll", EntryPoint = "MT_GetOutput")]
        public static extern bool GetOutput(IntPtr client,
            [MarshalAs(UnmanagedType.I4)] out int output);

        [DllImport("MTCom.dll", EntryPoint = "MT_GetLastError")]
        public static extern ErrorCode GetLastError();
    }
}
```

MTCom utility functions

These functions are used in some of the previous examples. They are written for a MicroTorque G4 controller and together will provide a way of starting the driver and waiting for a joint to complete.

```
/// <summary>
/// This function will send a command to a MicroTorque device and wait
/// for an acknowledge.
/// </summary>
static bool SendAndWaitForAck(IntPtr client, int channel, string command,
                              long timeout)
{
    byte[] dataSet = System.Text.Encoding.ASCII.GetBytes(command);
    bool success, ackReceived = false;

    if (timeout < 0)
        return false;

    success = MTCom.WriteSet(client, channel, dataSet, dataSet.Length);

    if (success)
    {
        int noBytes;
        byte[] reply = new byte[4096];

        success = MTCom.ReadSet(client, channel, reply, out noBytes,
                                (int)timeout);

        if (success && noBytes > 0 && reply[2] == 'A')
            ackReceived = true;
    }

    return ackReceived;
}

/// <summary>
/// This function will start a G4 driver and wait for the joint to
/// be completed.
/// </summary>
static bool StartDriverAndWait(IntPtr client, int channel, long timeout)
{
    System.Diagnostics.Stopwatch stopWatch =
    new System.Diagnostics.Stopwatch();
    bool success, jointCompleted = false;

    stopWatch.Start();

    // Reset the driver to clear any errors of previous joint
    success = SendAndWaitForAck(client, channel, "G0",
                  timeout - stopWatch.ElapsedMilliseconds);

    // Start the driver
    if (success)
    {
        success = SendAndWaitForAck(client, channel, "G1",
                      timeout - stopWatch.ElapsedMilliseconds);
    }
```

Utility functions continued:

```
    // Wait for BUSY signal to go HIGH
    while (success && stopWatch.ElapsedMilliseconds < timeout)
    {
        int output;
        success = MTCom.GetOutput(client, out output);

        if (success && (output & 0x02) != 0)
            break;

        System.Threading.Thread.Sleep(10);
    }

    // Wait for BUSY signal to go LOW
    while (success && stopWatch.ElapsedMilliseconds < timeout)
    {
        int output;
        success = MTCom.GetOutput(client, out output);

        if (success && (output & 0x02) == 0)
        {
            jointCompleted = true;
            break;
        }

        System.Threading.Thread.Sleep(10);
    }

    return jointCompleted;
}
```

9836 6568 01
Software release 1.0.3.1
2011-02
Edition 1.2

www.atlascopco.com