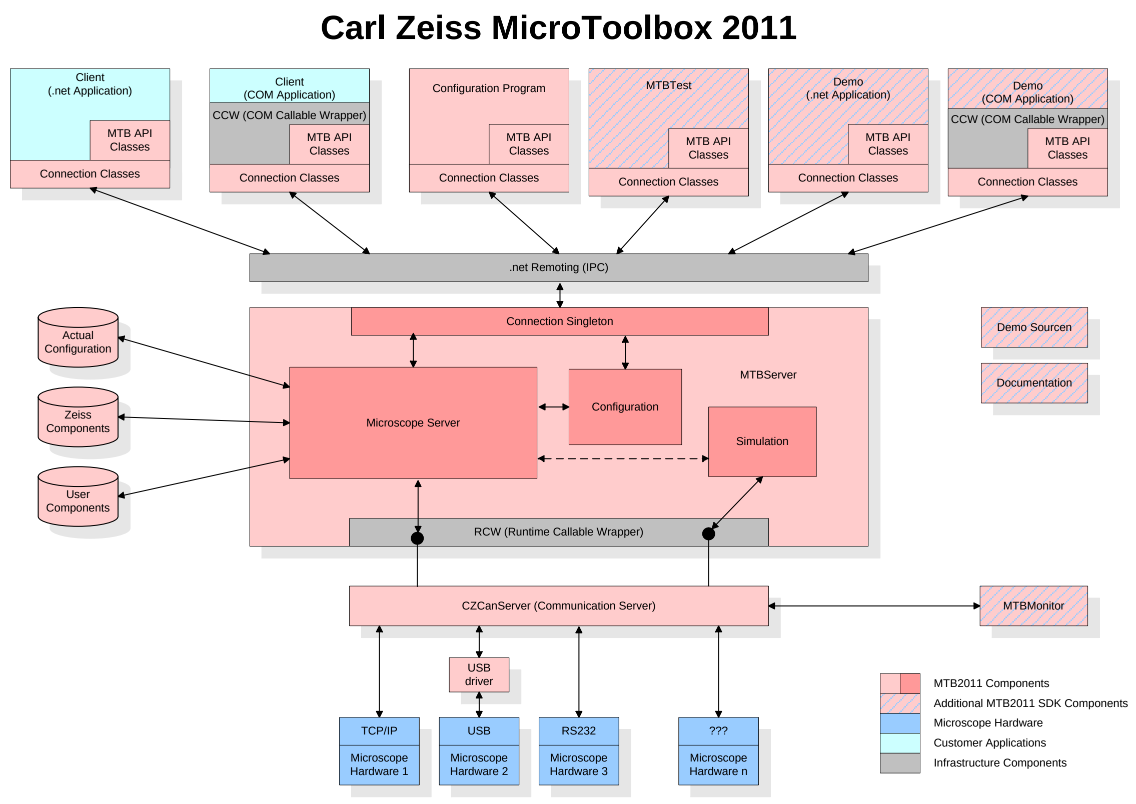
How to start

1 Overview

The Carl Zeiss MicroToolBox 2011 (MTB2011) is a program package, that simplifies and unifies the software control of Carl Zeiss light microscopes and their accessories.

It may be used under the operating systems Windows 7, Windows 8, Windows 8.1 and Windows 10. It is programmed in C# using the .net framework 4.5. Therefore, the simplest and most effective way for using it is also using a .net language, but it is also possible to use the MTB2011 with COM (Component Object Model) interfaces.

1.1 Components of the MTB2011



The central component of the MTB2011 is the MTBServer. It may exist in different versions on a system, each version may be used simultaneously from several clients that were developed for this version. A real hardware device may only be accessed by one MTBServer version. The MTBServer consists of several device specific DLLs: MTBKernelXXX.dll.

Clients may use the server functions defined in the API (Application Programmers Interface), located in the MTBApi.dll. The MTBApi classes are connected with the MTBServer via .net remoting.

The server itself communicates with the hardware using the CzCanSrv.exe. This is another server, that gives a unified access via different interfaces (RS232, USB, TCP-IP) to the hardware. It also may redirect all communication requests to a simulation unit that is integrated in the MTBServer (MTBSimulation.dll). This allows programming, testing and demonstration of software components without having microscope hardware available.

The information which devices and components should be controlled or simulated is held in the configuration file ActiveConfiguration.xml. Information about all available devices and components is stored in the Carl Zeiss delivered MTBComponents_XXX.xml and in the user defined UserComponents.xml. These configuration files are located in the folder: 'C:\Documents and Settings\All Users\Application Data\Carl Zeiss\MTB2011\'

All this configuration information can be conveniently edited with the configuration client MTBConfig.exe.

2 Installation

2.1 SDK/RDK: What is the difference?

The MTB2011 Installation exists in 2 different versions: The Software Developers Kit (SDK) and the Redistribution Kit (RDK).

The RDK is intended to be delivered together with your software. It contains all MTB2011 components, necessary for the end user at runtime.

The SDK is an add on, only for the software development team, which signed the MTB license. It contains the following components: MTB2011 API documentation, simple .net and COM demo programs inclusive source code, a comprehensive test program (MTBTest.exe) showing all MTBApi functions and MTBMonitor, a tool for observing the communication between MTB and hardware.

2.2 Install MTB-Server as service?

Base of the MTB2011 is a server, that has to be running. It may be started in 2 different ways:

1. Automatically as a service: the MTBService is started automatically on booting the Operating System.

2. Manually with MTBProviderConsole: MTBProviderConsole is a small console program, that starts the MTBServer and keeps it running as long as it runs itself.

Installing the MTB as a service is more convenient for the end user. For programmers it is often useful to use MTBProviderConsole, in order to stop all MTB2011 functions and to restart with a 'clean' system.

2.3 MTBDemo source codes

There are 2 demo programs delivered with the MTB2011 SDK: one for native .net applications, programmed in C#, and one for COM applications programmed in C++.

For both demo programs the source code is available. By default it is installed in the directory 'My Documents\Carl Zeiss\MTB2011 (x.x.x.x)\', where x.x.x.x is the version of the MTB.

3 First steps

There are 3 steps necessary for a first demonstration of the MTB2011:

1. Start the MTBServer

2. Create a configuration

3. Start a MTB2011 client

3.1 Start the MTBServer

If you installed the MTBServer as a service (see chapter 2.2), everything is already done. Otherwise start MTBProviderConsole.exe ('MTB Provider Console' Icon on the desktop and in the start menu).

3.2 Create a configuration

In order to create a configuration, start MTBConfig.exe ('Microscope Configuration' Icon on the desktop and in the start menu). The first time you start it, you may get a warning that there is no configuration available. Ignore this. Then create a new configuration, either manually with 'New Configuration' (in the file menu or in the context menu of the configuration tree on the left side), or, if you have already some hardware connected to the computer, with "AutoConfiguration".

In both cases, you should get a configuration entry in the configuration tree on the left side. If you select the components in the tree, alternatives (e.g. other microscope types, other filters…) are displayed in the list box on the upper right side and properties (if there are any) may be set in the window on the lower right side (e.g. ‘Simulate Device’).

If you have a valid configuration, save it either by leaving the program with the ‘OK’ Button or by pressing the ‘Apply’ Button.

3.3 Start a MTB2011 client

A good client for testing the MTB2011 is MTBTest.exe (‘MTB Test’ Icon on the desktop and in the start menu).

After starting the program, press the ‘Connect to MTB’ button on the lower left side. Herewith the client connects to the MTBServer and looks for the configured devices and their components. The components found are listed in the window on the left side.

Then you may press the ‘Build User Interface’ Button. Then a user interface for controlling all components found is shown.

4 Create own programs using the MTB2011

There are 2 sample applications delivered with the SDK: MTBDemoNet, a C# .net application and MTBDemoCOM, a C++ COM application. Please have a look to these applications and their source code. There you will find the schemes, how to connect to the MTB, call MTBAPI functions and getting MTB events.

4.1 Schema for own programs

1. Insert a reference to MTBAPI.dll into your project

2. Create an instance of MTBConnection:

```
        IMTBConnection  m_Connection = new MTBConnection();
```

3. Login to the MTBServer:

```
        string m_MTBID;
        m_Connection.Login("en", out m_MTBID);
```

4. Get access to the root object:

```
        IMTBRoot m_Root = m_Connection.GetRoot(m_MTBID);
```

5. Start monitoring for all components:

```
        m_Root.StartMonitoring(m_MTBID);            // start monitoring
for all components owned by m_Root
```

6. Get access to the components you want to control (refer to MTB Component IDs for a list of available components, their IDs and interfaces), e.g.:

```
        IMTBFocus m_Focus =
(IMTBFocus)m_Root.GetComponent("MTBFocus");
        IMTBReflectorChanger m_ReflectorChanger =
(IMTBReflectorChanger)m_Root.GetComponent("MTBReflectorChanger");
```

7. Work with the components, e.g.:

```
        m_Focus.SetPosition( 1.0, "µm", MTBCmdSetModes.Relative |
MTBCmdSetModes.Synchronous );  // 1 µm relative move, synchronous
        short pos = m_ReflectorChanger.Position;       // get actual
position of the reflector changer
```

```
        ....
```

Please refer to the help file for the available functions, properties and events for each interface. There are a few basic interfaces like IMTBChanger, IMTBContinual and IMTBAxis, that cover most of the needed functions. A good overview of the interfaces you may get from the API Diagram.

8. Logout from the MTBServer:

```
        m_Connection.Logout(m_MTBID);
```

4.2 Some general hints

- always start monitoring (IMTBBase.StartMonitoring) for all components
- don't poll the components for states, positions, settings... . For all changes you may be notified with events.
- most functions (e.g SetPosition, Calibrate,...) that may be time consuming, can be executed either synchronously (MTBCmdSetMode.Synchronous) or asynchronously (MTBCmdSetMode.Default). Therefore the appropriate MTBCmdSetMode parameter can be set in the function.
  Waiting for the end of an asynchronous operation may be conveniently done with IMTBBase.WaitReady().

4.3 Events

At the beginning of your program you should start the event notification mechanism. Then every time your component changes a state or a position, your event handler function is called.

At the end of your program, if you don't need the events any more, you should stop the event notifications.

4.3.1 Start event notifications

Create an EventSink and label it as yours (in this example the IMTBBase events for the reflector changer are sinked):

```
      MTBBaseEventSink m_BaseEventSink = new MTBBaseEventSink();
// create a sink for the IMTBBase Events
      m_BaseEventSink.ClientID = m_MTBId;
// label with the client ID
```

Add event handler functions:

```
      m_BaseEventSink.MTBBusyChangedEvent += new
MTBBaseBusyChangedHandler(BaseEventSink_MTBBusyChangedEvent);
      m_BaseEventSink.MTBErrorEvent  += new
MTBBaseErrorHandler(BaseEventSink_MTBErrorEvent);
```

Please refer to the documentation of the interfaces to find out which events are available.

Advise the server to send the events:

```
      m_BaseEventSink.Advise(m_ReflectorChanger);
```

4.3.2 Event handler

Event handler function for the error event:

```
      void BaseEventSink_MTBErrorEvent(int code, string msg)
      {
          Console.WriteLine("Error {0} occurred: {1}", code, msg);
      }
```

Event handler function for the BusyChanged event:

```
        void BaseEventSink_MTBBusyChangedEvent(bool busy)
        {
            if (busy)
                Console.WriteLine("Reflector turret is moving");
            else
                Console.WriteLine("Reflector turret is settled");
        }
```

Please refer to the interface documentation for the events and their parameters.

4.3.3 Stop event notifications

Unadvise the events:

```
        m_BaseEventSink.Unadvise(m_ReflectorChanger);
```

Remove the event handler functions:

```
        m_BaseEventSink.MTBErrorEvent -= new
MTBBaseErrorHandler(BaseEventSink_MTBErrorEvent);
        m_BaseEventSink.MTBBusyChangedEvent -= new
MTBBaseBusyChangedHandler(BaseEventSink_MTBBusyChangedEvent);
```

Stopping the event notifications is done preferably in the Dispose function of your program. Please refer to the 'dispose pattern' documentation in the MSDN.

4.3.4 Typical events on movements

For motorized movements of a component, you typically will get the following events (in this order):

- BusyChanged(true)
- TargetPositionChanged
- PositionChanged
- ... for longer movements approximately each 100 ms a PositionChanged event is sent
- PositionChanged
- PositionSettled
- BusyChanged(false)

5 In case of errors

Check the following points to isolate the error:

- do you get an exception in your code?
- do you have a correct configuration? Is everything switched on and plugged in?
- does MTBTest run with that configuration?
- can you reproduce the error with MTBTest?

If you contact MikroAPI.support@zeiss.de for further help, then the following would be helpful:

- a technical report, created with the 'Microscope Configuration' program (on the property page of the actual configuration)
- a log file of the communication between microscope and MTB2011. Therefore just start MTBMonitor before/together with your program. All commands are logged and can be saved within the program.
- a MTBServer log file (see below).

5.1 Creating a MTBServer log file

Follow these steps:

1. Stop the MTBServer (either the MTB2011 Service or MTBProviderConsole.exe)

2. If you are using the MTB2011 Service for the server, then please open:
   C:\Program Files\Carl Zeiss\MTB 2011 (x.x.x.x)\MTB Server Console\MTBService.exe.config
   or if you are using the MTBProviderConsole, then please open
   C:\Program Files\Carl Zeiss\MTB 2011 (x.x.x.x)\MTB Server Console\MTBProviderConsole.exe.config
   Both are xml text files.

3. Change
   <add name="MTBTraceSwitch" value="1" />
   to
   <add name="MTBTraceSwitch" value="**4**" />
   and save the file.

4. Then start the MTBServer again.

5. Now try to reproduce the error. The MTBServer is logging now into "MTBServer.log".

6. Don't forget to undo the changes afterwards, otherwise your log file will grow and grow....

6.0 MTB2011 and MTB2004

The MTB2011 is a successor for the MTB2004. Its interfaces are almost compatible, so in most cases a rebuild with the new MTB version should be sufficient.

The further development for the MTB2004 will be reduced to a minimum. Full support for new devices will only be integrated into the MTB2011. So we would strongly recommend to use the MTB2011 for further development.

The MTB2004 1.8.x.x and the MTB2011 may be installed and run together on one PC. But it is not possible to control the same devices simultaneously with both MTB versions. In this case the second MTB, that tries to control this devices will give you an error message.