

3509, route de Baziège  
31670 LABÈGE  
France  
Tél. : 33 (0)5 61 39 96 18  
Fax : 33 (0)5 61 39 17 58  
[www.midi-ingenierie.com](http://www.midi-ingenierie.com)

**midi ingenierie**

# **.Net components user guide**



Date : 20/05/2014

Reference : dotnet\_v4\_um\_en.pdf

Ref. MI : ELE1940895.doc

Release : 4

Author : E.LOPEZ

<http://www.midi-ingenierie.com>





## Sommaire

<b>1. Installation and implementation .....</b>	<b>3</b>
1.1. Installation .....	3
1.2. Implementation with C# 2010 Visualstudio .....	4
1.2.1. Create an application .....	4
1.2.2. Reference the Midi-ingenierie components in the solution .....	4
1.2.2.1. Add the Midi-ingenierie components in the toolbox .....	4
1.2.2.2. Verify the right Midi-ingenierie assembly references in the solution .....	6
1.2.3. Create Midi-ingénierie components instances .....	7
1.2.4. Culture and language .....	8
1.2.4.1. On design time .....	8
1.2.4.2. On run time .....	8
<b>2. Components generalities .....</b>	<b>9</b>
2.1. Basic principles .....	9
2.2. The MiCm communication component .....	9
2.3. The xMacxx axis control components .....	10
2.4. Communication component double-session application example .....	11
<b>3. The communication component .....</b>	<b>12</b>
3.1. Generalities .....	12
3.2. Communication pipe et pathName .....	12
3.3. The module address system .....	12
3.3.1. Basic principles : .....	12
3.3.2. Using the 16 bits address : .....	13
3.3.3. BroadcastMessage : .....	14
3.3.4. Multi-thread access : .....	14
3.4. Properties .....	15
3.5. Methods .....	21
3.5.1. Communication driver methods with "standard" access .....	21
3.5.2. Communication driver methods with "expert" access .....	28
3.5.3. The command prefix in the expert methods .....	29
3.6. Events .....	30
3.6.1. Writting property events .....	30
3.6.2. Using method events .....	31
<b>4. Axis components .....</b>	<b>32</b>
4.1. Generalities .....	32
4.2. Component management inherents properties .....	32
4.3. Propretries .....	34
4.4. Methods .....	35
<b>5. Annexe A – Communication components default index .....</b>	<b>36</b>
5.1. Annexe A1 – Communication defaults .....	36
5.2. Annexe A2 – Property writting defaults .....	36
<b>6. Annexe B - Special protocols use .....</b>	<b>37</b>
6.1. Mac type module management protocol .....	37
6.1.1. Local Mac converter use : .....	37
6.1.2. Global Mac converter use : .....	37
6.1.3. The Mac modules command converter : .....	38
6.2. The Console protocol .....	40
6.2.1. Protocole console local use: .....	40
6.2.2. Protocole console global use: .....	40
6.3. The F30xx secateurs management .....	41
6.3.1. F30xx protocol local use : .....	41

# 1. Installation and implementation

Midi-ingenierie supplies **.Net** components to facilitate Midi-ingenierie axis control application developpement.

Components will be used with the **2.0 version** (and later) .Net framework:

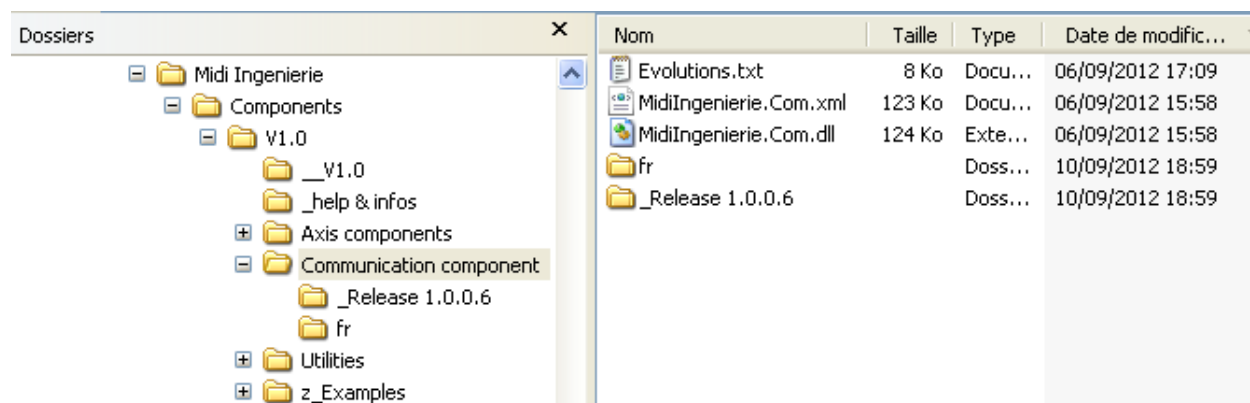
- the communication component belongs to the **MidiIngenierie.Com.dll** assembly.
- the axis control components belong to the **MidiIngenierie.Axe.dll** assembly.

## 1.1. Installation

The Midi ingenierie components are available on the <http://www.midi-ingenierie.com/> site.

They are belonging to a Vx,x package which contains:

- the components assemblies themselves.
- the all available documentation.
- some axis control and configuration tools.
- some examples for different developpement platforms (Visual Basic, C#, Labview...).



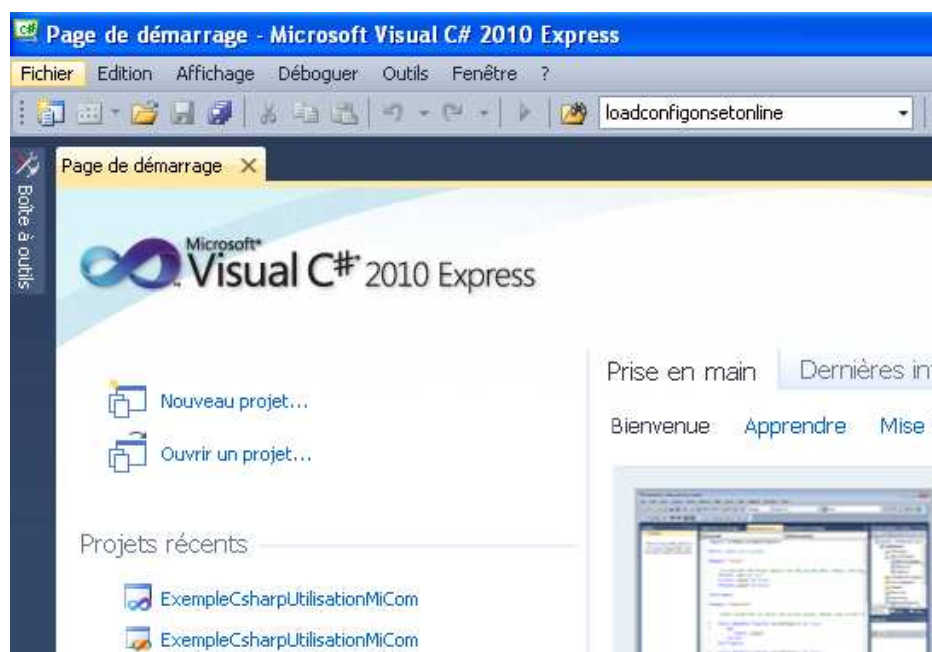
The setup program supplies a default installation in the **Midi ingenierie** directory just under the main c: directory.

The user could change for any other directory like **Program files** or anywhere...



## 1.2. Implementation with C# 2010 Visualstudio

### 1.2.1. Create an application

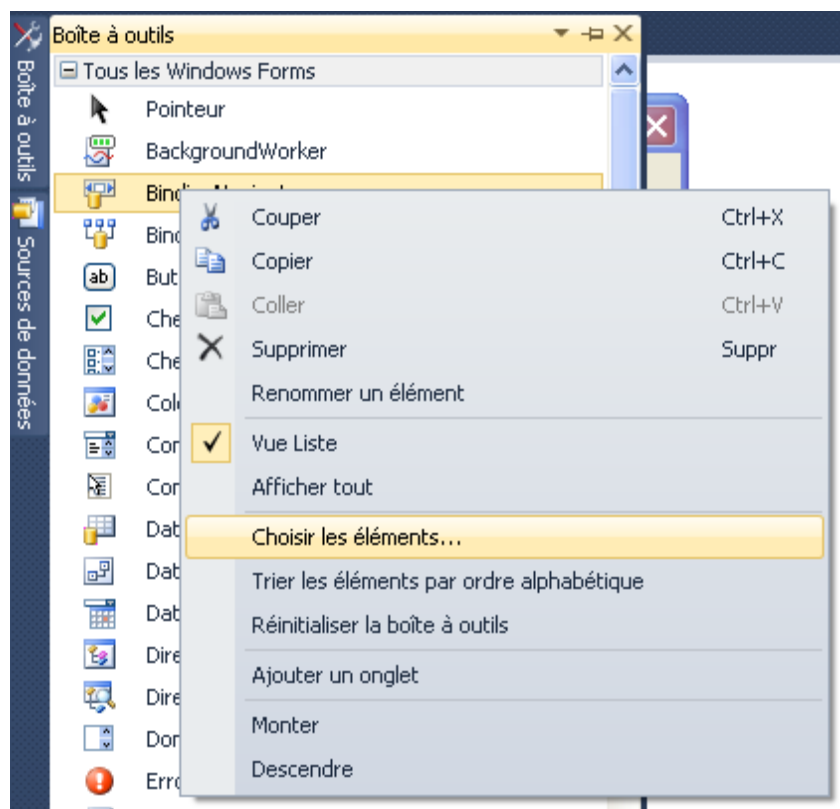


Open a new project  
**Application Windows Forms.**

Save your solution  
**MyApplication.sln** in the suitable directory.

### 1.2.2. Reference the Midi-ingenierie components in the solution

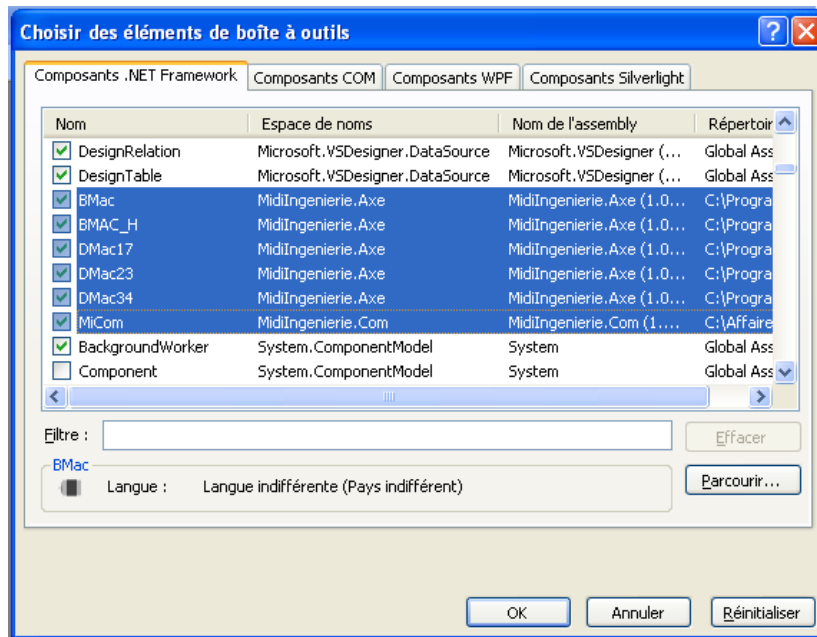
#### 1.2.2.1. Add the Midi-ingenierie components in the toolbox



Right click on any toolbox control, then click the

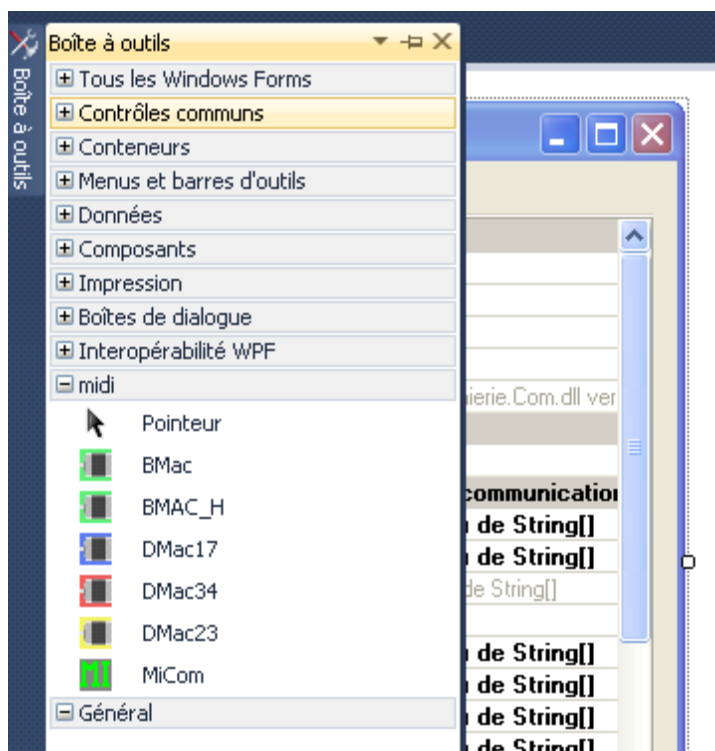
**Choose the éléments...**

tab. The components selection panel let you add all Midi-ingenierie components.



Using the **Search** button, **open** the **MidiIngenierie.Axe** and

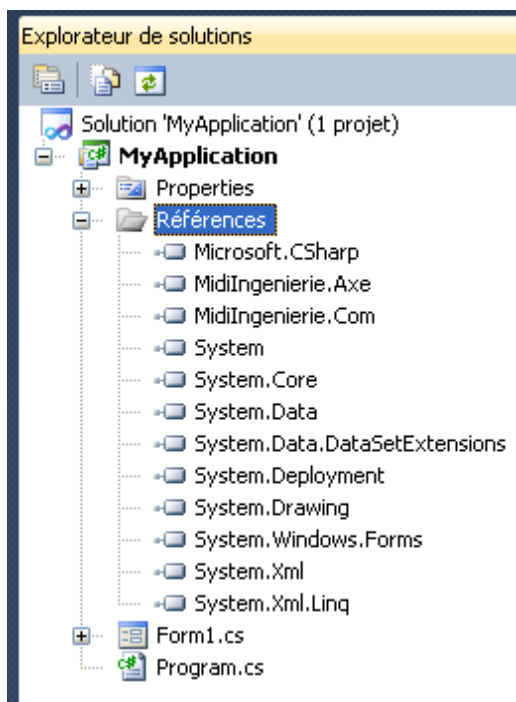
**MidiIngenierie.Com** assemblies to add the available components.



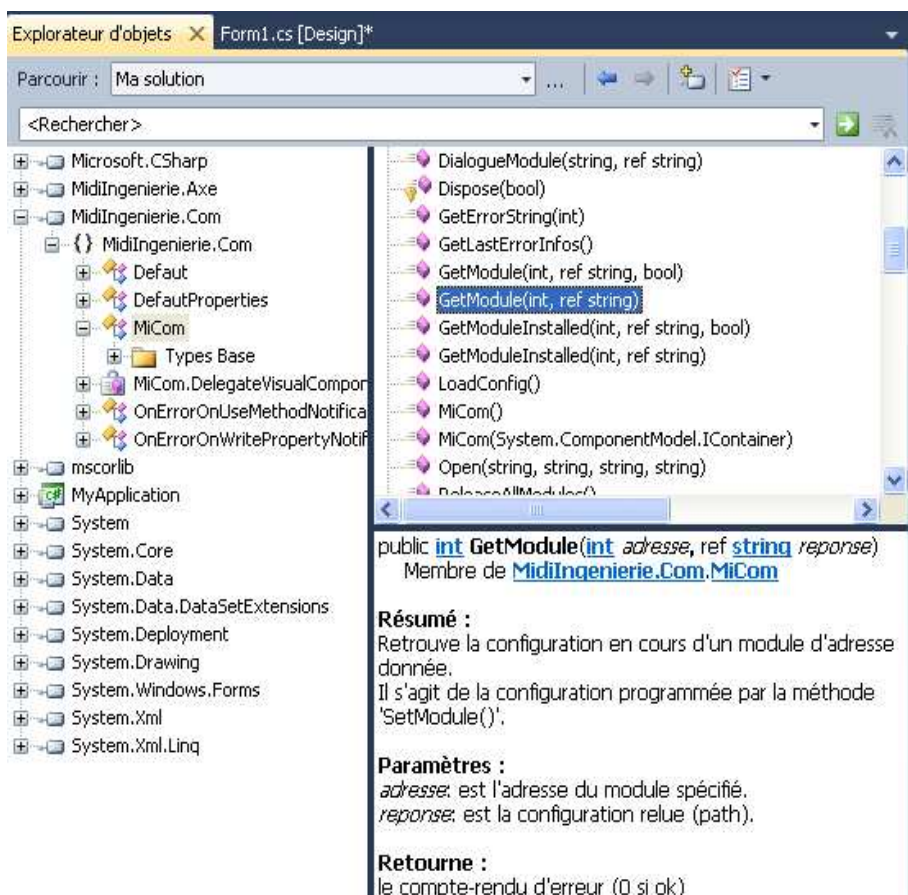
The toolbox contains now the different Midi-ingénierie available components.



### 1.2.2.2. Verify the right Midi-ingenierie assembly references in the solution



In the project **Références** index, verify the **MidiIngenierie.Com.dll** assembly reference displaying the communication component, then verify the **MidiIngenierie.Axe.dll** assembly reference displaying the axis components.



In the project **Références** index, you can double-click a referenced object to access the object explorer.

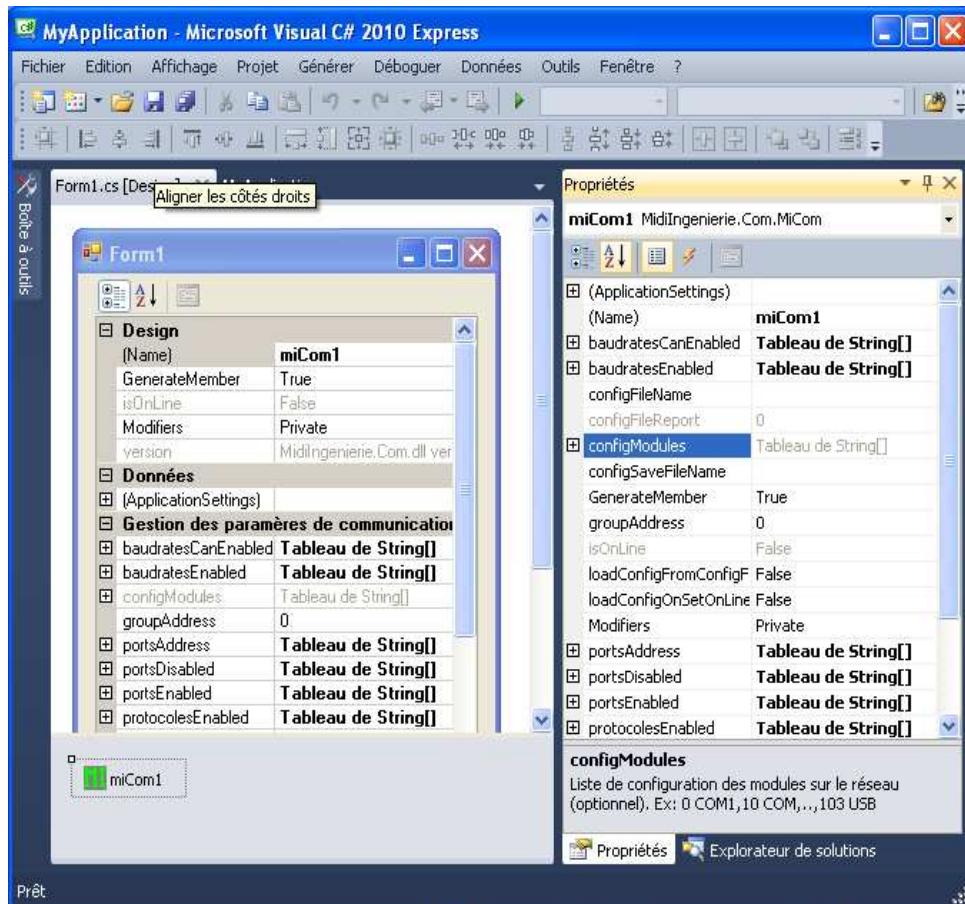
The object explorer expose all the available component classes, properties and methods.



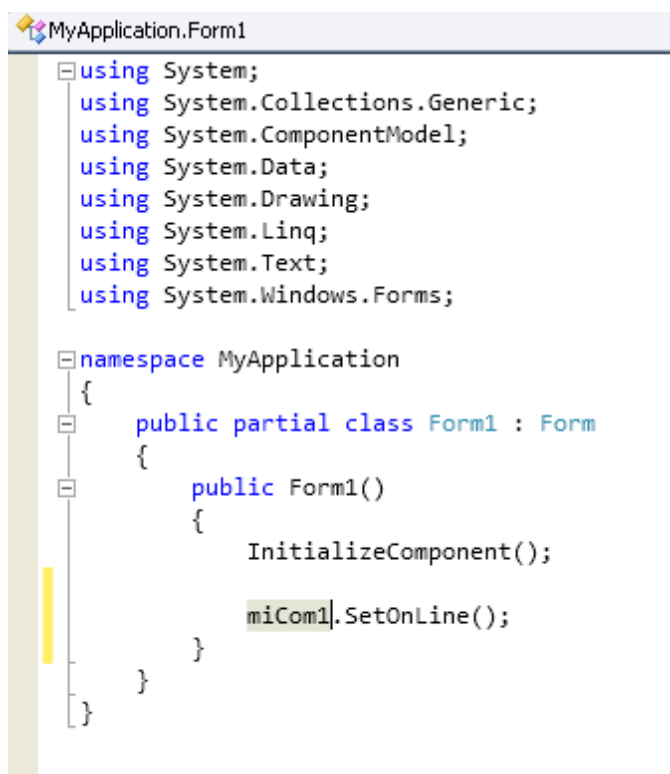


### 1.2.3. Create Midi-ingénierie components instances

When components stay in the toolbox, components instances can be drag in a form like any other classic control (button...)



The created objects properties can be displayed in the involved **Properties** window.



#### Attention:

When you create a communication component instance on a WindowsForm, you need call the **setOnLine()** method directly in the form constructor class after the **InitializeComponent()** method designer automatically call.

The network communication with possibly present modules is systematically initially disabled to not inadequately interfere during design time.

The **setOnLine()** method call will enable run time communication.



## 1.2.4. Culture and language

Culture is automatically managed using Midi-ingenierie components.

### 1.2.4.1. On design time

This concern:

- the component properties window comments
- the components explorer comments \*
- the automatic code writing comments \*

\* These comments are supplied in **.xml** files associated to the assembly **.dll** file.

The english language default file is supplied in the assembly directory.

The french language file is supplied in the **/fr** subdirectory.

**Attention:** when no **.xml** file is present, comments are not available.

when no **/fr.xml** file is present, comments are supplied in english default language.

### 1.2.4.2. On run time

This concern:

- The component managed error messages (example: the communication error messages)
- the component properties window comments, when used in a **propertygrid** control.

On run time, the multilingual messages management doesn't need any associated file.





## 2. Components generalities

### 2.1. Basic principles

#### first principle

**Every module (axis controller or servomotor) gets its own configurable address.**

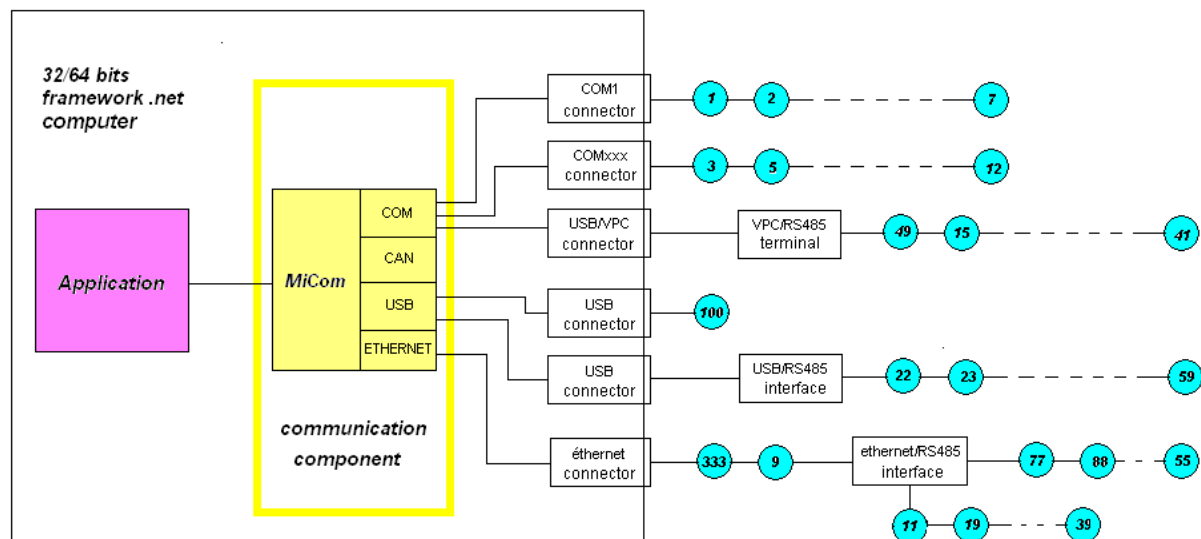
(0 as factory default setting)

#### second principle

**The application can connect every network module with a unique address, regardless the network topology.**

(COM, CAN, Usb or ethernet link)

### 2.2. The MiCm communication component



It supplies the Midi-ingenierie basic communication with the 32 and 64 bits Windows XP PC.



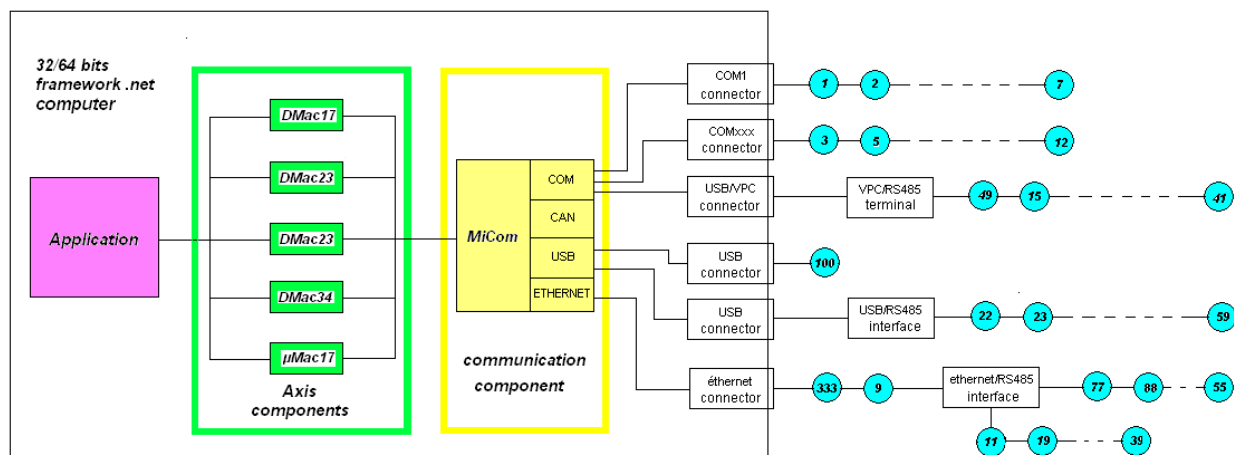
It works with the different today communication links:

- port Com link
- virtual port Com link
- Usb/Hid link
- ethernet/Tcp/IP link
- Can link

The user has no charge with the protocol line management and can ignore all about network topology. He only needs to know module addresses.

However, a module direct access through the communication component needs to use the Midi-ingénierie specific module language (command/reply). The user can refer to the **Module user guide**. All user guides are available connecting the <http://www.midi-ingénierie.com/> site.

## 2.3. The xMacxx axis control components



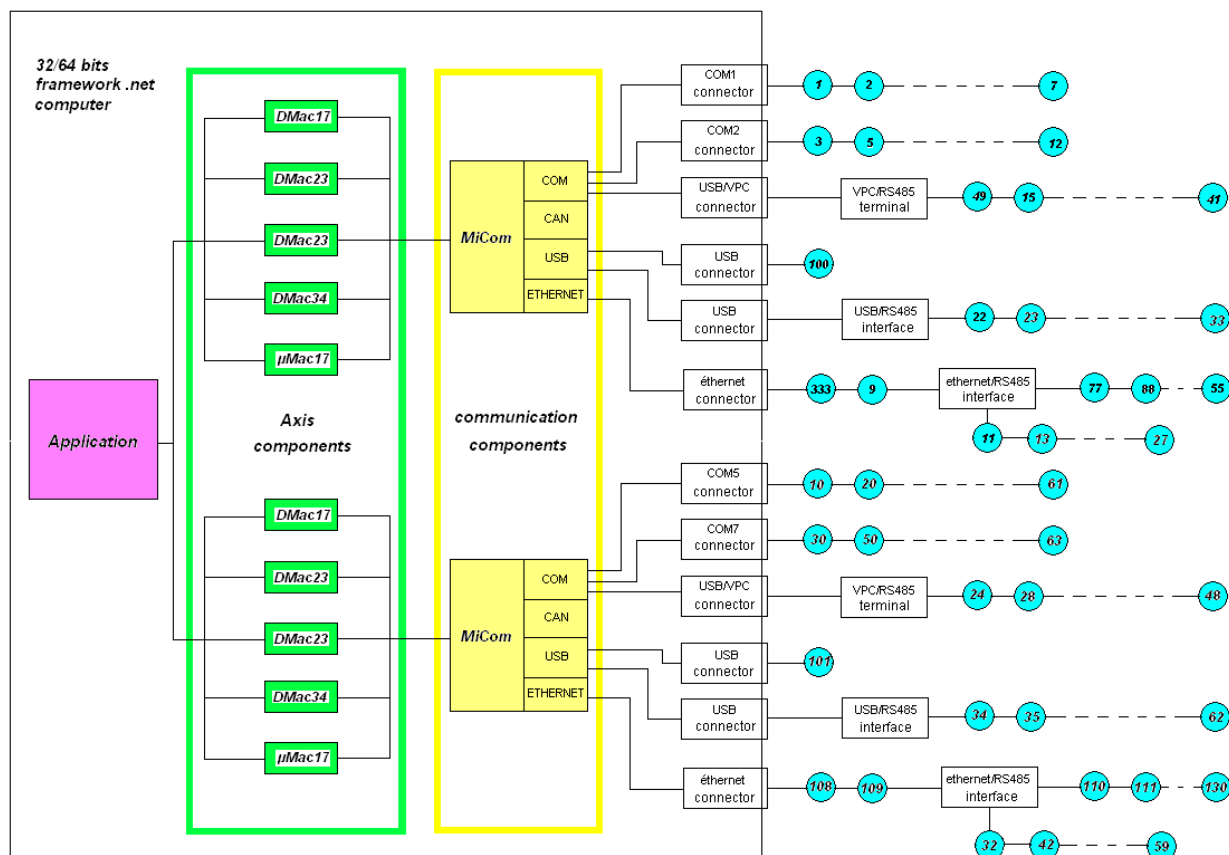
Every component supplies an axis type control. That means control is performed using component properties and methods regardless module language and communication line protocol.

An axis component implementation needs, at least, one communication component instance implementation.

As many axis components as available axis types:

- μMAC17
- DMAC17
- DMAC23
- DMAC34
- BMAC
- BMAC\_H

## 2.4. Communication component double-session application example



Two communication component instances topology example



## 3. The communication component

### 3.1. Generalities

The **MiCom** component expose a properties and methods collection to manage an heterogeneous communication network system. The main argument is the physical module **address**.

**Address** is a tied module argument. It's actually a **physical address** (module micro-switch) or a **programmable address** (saved in non volatil module memory) with 0 factory initial value.

**The unique address principle : only one module for one network address .**

**In a such system, the application only needs to know the module address regardless any other communication parameters (communication pipe, baudrate, protocol ...).**

Thereby, from the application view, the module address is unique.  
The application could manage a **0 to 65535** address panel.

### 3.2. Communication pipe et pathName

We name the **communication pipe** as being the communication entity bound with a **pathName**.

The **pathName** is the unique communication pipe system entity.

The **pipe** is a physically point to point or multipoint link :

- **point to point** : one module on the pipe like direct Usb(hid) or ethernet link.
- **multi-points** : any modules on the same pipe:
  - RS485 line on port COM, virtual port COM, Usb(hid)/RS485 or ethernet/RS485 interface with a maximum of 64 modules.
  - Can line with a maximum of 127 modules.

With a port Com, the pathName looks like "COMx" x is from 1 to 255

With a port Can, the pathName looks like "CANx" x is from 1 to ....

With a Usb or ethernet port, the pathName is more sofisticate, w'll use the generic terms "USB" et "ETH"

**Two different applications could'nt work on the same communication pipe.**

### 3.3. The module address system

#### 3.3.1. Basic principles :

##### The physical address :

- every Midi-ingenierie module gets its own "physical" address that allows to locate it on a network regardless the link (RS485, Usb, CAN, ethernet). It can be a hard address (switchs) or a soft one (Eeprom).

##### Module types :

- we can find 2 module types :
- the **old modules** with a link:
  - RS485 with a 6 bits physical address ( 0 to 63)
  - CAN with a 7 bits physical address ( 1 to 127 )
- the **new modules** with a 16 bits physical address ( 0 to 65535)



### Compatibility :

- the compatibility will be ensured so that old and new modules could work together all over the network.

### Link types :

- point to point:** "direct" Usb(hid) or ethernet , the module is new 16 bits address type.
- multipoint:** port COM, Usb(Virtual port Com), Usb(hid)/RS485 or ethernet/RS485 interface, port CAN, the module can be new 16 bits or old 6/7 bits address type.

### Dialog address :

- the dialog address starts every module command message regardless the link.
- the module reply message starts with the same dialog address such as old module transmissions:
  - 6bits from 0 to 63 with RS485, Usb and ethernet
  - 7bits from 0 to 127 with CAN
- the dialog address is so :
  - dialog address ( @<sub>6bits</sub> ) = physical address ( @<sub>16bits</sub> ) [modulo 64] with RS485, Usb, ethernet
  - dialog address ( @<sub>7bits</sub> ) = physical address ( @<sub>16bits</sub> ) [modulo 128] with CAN
- The communication driver performs the link between physical address, communication port and dialog address.**

### With the new address type modules :

- there is a new slot in the reply to the @<sub>6bits</sub>RV identity request message returning the 16bits module address.
- The new address type module will reply to the @<sub>6bits</sub>RV command in case of:
  - @<sub>6bits</sub> = @<sub>16bits</sub> [ modulo 64] with RS485
  - @<sub>7bits</sub> = @<sub>16bits</sub> [ modulo 128] with CAN
  - regardless @<sub>6bits</sub> with point to point Usb and ethernet

## **3.3.2. Using the 16 bits address :**

**. unique module address from application vue .**

(that is : one module using one address)

**. application address + group address = module adresse + port address .**

with:

- application address** is the application used address.
- module address** is the physical module address .
- group address** [optional] allows to different application instances to work with different modules on the same machine or on different machines with an ethernet link.
- port address** [optional] allows different modules with identical physical address to work on different communication ports. (COMx CANx USB ETH)

**group address** and **port address** are defined at communication component level and tied with communication ports.

**Group adresse** example:

With a first application instance, a **0000h** (default value) group address is given to the ethernet port. So the application will use **8203h** to address the **8203h** ethernet physical address module.

With a second application instance, a **1000h** group address is given to the ethernet port. So the application will identically use **8203h** to address the **9203h** Ethernet physical address module.



**Port adresse** example:

A **0000h** (default value) port address is given to the COM1 port.  
So the application will use **0000h** to address the **00h** COM1 physical address module.

A **3000h** port address is given to the Usb port.  
So the same application instance will use **3000h** to address the **0000h** Usb physical address module.

### 3.3.3. BroadcastMessage :

When an application transmits a 16bits address message with a -1 value, the message is transmitted without address on the all opened communication pipes regardless any result. (first one communication needs succeed while a pipe would be opened)

A better way is using **DialogueAllModules()** specific broadcast message method.

### 3.3.4. Multi-thread access :

As using list structure management, the communication component doesn't actually allow simultaneous properties and methods access with different threads of a **one application instance**.

So, simultaneous access will be **sequentially** processed, that is a thread access is automatically delayed as long as an other thread access is in progress.

How manage **a real component multi-thread acces** ?

You must, for that, **give every thread a different communication component instance**.

So, every instance gets its own ressources (list ...) and every thread access in part time (Windows OS management) to te component properties and methods regardless any other thread.

**So, for example, we could perform simultaneous communications on different communication pipes.**



### 3.4. Properties

```
public string version { get; }
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Component soft version property.

Return the version number x.x.x.x before the build date jj.mm.aaaa

```
public bool isOnLine { get; }
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Network connection state property.

return 'true' when the connection is on.

```
public string[] portsComDetected { get; }
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Founded serial COM ports sorted list property.

Return "COMx" type strings.

Example: "COM1","COM3", ... , "COM15", "COM255" .

```
public string[] portsCanDetected { get; }
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Founded CAN ports sorted list property.

Return "CANx" type strings.

Example: "CAN1","CAN2" .

```
public string[] portsUsbDetected { get; }
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Founded Midi-ingénierie specific Usb chanals sorted list property.

Return Usb/hid path type strings .

usb/hid path example : hid#vid\_10c4.pid\_ea80....

#### Exemple:

"hid#vid\_05a4&pid\_9862&.....",.....,"hid#vid\_062a&pid\_0000&....."

```
public string[] portsEthDetected { get; }
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Founded Midi-ingénierie specific ethernet chanals sorted list property.

Return IPAddress.portaddress type strings .

Example: "192.168.0.77.10001", ... , "192.168.0.125.10001" .





```
public int nbrPortsComDetected { get; }  
Membre de MidiIngenierie.Com.MiCom
```

**Résumé :**

Founded COM port count property.

```
public int nbrPortsCanDetected { get; }  
Membre de MidiIngenierie.Com.MiCom
```

**Résumé :**

Founded CAN port count property.

Return -1 when there isn't any CAN system on this machine.

```
public int nbrPortsUsbDetected { get; }  
Membre de MidiIngenierie.Com.MiCom
```

**Résumé :**

Founded Midi-ingénierie specific Usb chanals count property.

Return -1 when there isn't any USB system on this machine.

```
public int nbrPortsEthDetected { get; }  
Membre de MidiIngenierie.Com.MiCom
```

**Résumé :**

Founded Midi-ingénierie specific ethernet chanals count property.

Return -1 when there isn't any ethernet system on this machine.

```
public string[] baudratesEnabled { set; get; }  
Membre de MidiIngenierie.Com.MiCom
```

**Résumé :**

Read/write enabled RS485 baudrates list property.

(bits/s).

Example: "9600", ... , "38400" .

Warning: when the list is empty, there is no active RS485 connection.

```
public string[] baudratesCanEnabled { set; get; }  
Membre de MidiIngenierie.Com.MiCom
```

**Résumé :**

Read/write enabled CAN baudrates list property.

(kbits/s).

Example: "1000" .

Warning: when the list is empty, there is no active CAN connection.

```
public string[] protocolesEnabled { set; get; }  
Membre de MidiIngenierie.Com.MiCom
```

**Résumé :**

Read/write enabled protocols list property.

Example: "XON" when using Xon/Xoff protocol

"CON" when using console protocol

Warning: when the list is empty, there is no active connection.



```
public string\[\] typesModuleEnabled { set; get; }  
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Read/write enabled module types list property.

Example: "xMac", ... , "Mac" .

Enabled modules list :

xMac for the µMac, DMac, BMac and RMac modules types.

Mac for Mac modules types.

F30xx for secateurs.

Warning: when the list is empty, module type xMac is default value.

```
public string\[\] portsEnabled { set; get; }  
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Read/write enabled ports list property.

Ports are named with their complet path or with a generic path.

Complet path examples: "COM1" "CAN1" .

Generic path examples:: "COM" "CAN" "USB" "ETH".

Usb et ethernet ports will be only named with their generic path.

Using property example: "COM1", "ETH" .

Warning: when the list is empty, all ports are enabled except those belong to the 'portsDisabled' property.

```
public string\[\] portsDisabled { set; get; }  
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Read/write disabled ports list property.

Ports are named with their complet path or with a generic path.

Complet path examples: "COM1" "CAN1" .

Generic path examples:: "COM" "CAN" "USB" "ETH".

Usb et ethernet ports will be only named with their generic path.

Using property example: "CAN", "USB" .

Warning: an empty list has no effect.

```
public string\[\] portsAddress { set; get; }  
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Read/write port adress list property.

The port adress is so that: user addr + [group addr] = port addr + module addr .

With a such adress, an application can connect to modules with same physical address on different communication (ports) canals.

Ports are named with their complet path or with a generic path.

Complet path examples: "COM1" "CAN1" .

Generic path examples:: "COM" "CAN" "USB" "ETH".

Usb et ethernet ports will be only named with their generic path.

Using property example: "COM1 100", "ETH 1000" .

default value: 0.



```
public int groupAddress { set; get; }
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Read/write group address property.

The group address is so that: user addr + group addr = [port addr] + module addr .

With a such address, two instances of a same application can connect to modules on a common communication (port) canal.

Only usb and ethernet links can be share by different applications, COMx and CANx can't physically be share.

default value: 0.

```
public string[] configModules { get; }
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Read modules configuration list property.

Every list element give a configuration module arguments.

These are arguments address and [path] (optionnal) that are the SetModule() method arguments used to set the module connection.

Example: "0 COM", "2 COM4", ... , "1600 ETH" .

When using the 'SetOnLine()' method, all the configuration list modules are installed, ie a module connection set is tried with the above arguments.

This list is globally created when using the 'LoadConfig()' method that downloads the configuration file of which name is the 'configFileName' property value.

This global download is automatically done when using the 'SetOnLine()' method and the 'loadConfigOnSetOnLine' property was written 'true'.

The list can be update to, item by item, when using the 'SetModule()' method.

The list is globally saved when using the 'SaveConfig()' method which create the configuration file of which name is the 'configSaveFileName' property value.

The modules configuration list is limited to 255 items.

```
public string[] modulesInstalled { get; }
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Installed (detected or also connected) modules list property.

Every list element give the installed module arguments.

These are the real arguments address,path,baudrate,protocol and type used for the connection.

Example: "0 COM1 38400 XON DMAC23", ... , "1600 ETH 115200 XON DMAC34" .

#### Remarks:

- a module can belong the configuration list and not belong the installed module list when this one could not be installed(ex: the module is not present). Conversely, when the module is installed, it necessary belongs the module configuration list.

```
public string configFileName { set; get; }
Membre de MidiIngenierie.Com.MiCom
```

#### Résumé :

Read/write configuration file name property.



```
public string configSaveFileName { set; get; }
```

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Read/write save configuration file name property.

The path can be identically the configuration file path (see the 'configFileName' property).

**Remarks:**

- configuration and saving filename could be identic.

```
public bool loadConfigOnSetOnLine { set; get; }
```

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Boolean property which enable the automatically start of the configuration file download when using the 'SetOnLine()' method.

The configuration file path is the 'configFileName' property value.

```
public bool loadConfigFromConfigFile { set; get; }
```

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Boolean property which enable the immediatly start of the configuration file download.

Writting the 'true' value start the download.

The property value is automatically reset to the 'false' value (push button action).

The configuration file path is the 'configFileName' property value.

**Remarks:**

- identic to the LoadConfig() method use

```
public bool saveConfigToConfigFile { set; get; }
```

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Boolean property which enable the immediatly start of the save configuration file upload.

Writting the 'true' value start the upload.

The property value is automatically reset to the 'false' value (push button action).

The save configuration file path is the 'configSaveFileName' property value.

**Remarks:**

- identic to the SaveConfig() method use

```
public int configFileReport { get; }
```

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Configuration file error rubric number property.

```
public int visualComponentsRefreshPeriod { set; get; }
```

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Visual components global refresh period property.



**public string loggingEnable**{ set; get; }  
Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Logging mode selection property.

Component access and communications will be traced in the application directory

"MidilIngenierie.Com.log" file.

The file management is the user responsibility (delete, copy, rename...).

Available parameter values:

- ALL" or "all" all evenements are traced.
- OnUsingComponent" or "OnU" component properties et methods access are traced.
- OnAllCommunication" or "OnA" all communications are traced.
- OnCommunicationError" or "OnC" only on error communications are traced.
- OFF" or "off" no evenement is traced.



## 3.5. Methods

### 3.5.1. Communication driver methods with "standard" access

*These methods allow the user to manage module collection only with the address knowledge, regardless any network topology expertise.*

*Arguments between [ ] are optional.*

```
public int SetModule(int address [, string configModule])
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Add or replace a given address and configuration module to the network configuration element.  
When the network connection is set ('isOnLine' property value is true), the module is installed (try to connect).

When the module is already in the list, it is first removed like using the ReleaseModule() method.

There is a maximum of 256 configured/installed modules.

#### Paramètres :

*address*: is the module address.

*configModule*: is the configuration argument (path).

#### Retourne :

The error report number:

- 0 when all is right .
- Default.CONFIG\_MODULE\_UNEXPECTED when the configModule argument is unexpected.
- Default.MODULE\_IS\_NOT\_DETECTED when the module is not connected.
- Default.MODULE\_IS\_NOT\_INSTALLED when the configuration list is full (256 maximum items).

```
public int GetModule(int address, ref string rep [, bool errorEventEnabled])
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

**Get the pending given address module configuration.**

This is the configuration programmed when using the 'SetModule()' method.

#### Paramètres :

*address*: is the module address.

*rep*: is the retrieve configuration (path).

*errorEventEnabled*: false when the OnErrorOnUseMethod process is disabled.

#### Retourne :

The error report number (0 when all is right)

#### Example:

*rep = "COM"*

```
public int GetModuleInstalled(int address, ref string rep [, bool errorEventEnabled])
```

Membre de [MidiIngenierie.Com.MiCom](#)

#### Résumé :

Get the given address module intallation arguments.

These are the path,baudrate,protocol and type real arguments used when setting the connection.


**Paramètres :**

*address*: is the module address.

*rep*: is the retrieve configuration (path baudrate protocole type).

*errorEventEnabled*: false when the OnErrorOnUseMethod process is disabled.

**Retourne :**

The error report number (0 when all is right)

**Example:**

*rep* = "COM3 38400 XON DMAC23"

**public int ReleaseModule(int address)**

Membre de [MidiIngenierie.Com.MiCom](#)

**Résumé :**

Remove a given address module from the network configuration element.  
 the module is disconnected.

**Paramètres :**

*address*: is the module address.

**Retourne :**

allways 0 (all is right)

**public int ReleaseAllModules()**

Membre de [MidiIngenierie.Com.MiCom](#)

**Résumé :**

Remove the all network modules configuration.  
 All the modules are disconnected.

**Retourne :**

allways 0 (all is right)

**public int SetOnLine()**

Membre de [MidiIngenierie.Com.MiCom](#)

**Résumé :**

Enable the network connection.  
 You must enable the network connection first in your application before to try a first communication.  
 All forward installed modules are uninstalled.  
 All modules belonging the network configuration element will be installed (trying a first connection)  
 The 'isOnLine' property get the 'true' value.  
 When the 'loadConfigOnSetOnLine' property is on, the configuration file is first downloaded.

**Retourne :**

error report number (0 si ok)

**public int SetOffLine()**

Membre de [MidiIngenierie.Com.MiCom](#)

**Résumé :**

Disable the network connection.  
 The network configuration element is preserved.  
 All forward installed modules are uninstalled, but they could be installed again with the next use of the SetOnLine() method.  
 The 'isOnLine' property get the 'false' value.





## Retourne :

allways 0

```
public int DialogueModule([int address], string cmd, ref string rep [, ref int status] [, bool
errorEventEnabled])
```

Membre de [MidiIngenierie.Com](http://MidiIngenierie.Com).MiCom

## Résumé :

Dialog sequence (command/reply) with a given address module.

The 'cmd' argument contains the address just forward the interpretable module command.

The 're' argument return the module reply when there's one.

The 'status' argument return module state informations when the command succeed.

(see the Module user manual to get all the details about the interpretable module command/reply system).

## Paramètres :

*address*: is the module address.

*cmd*: is the command to the module with first the module address in decimal. When there's no address, the command is a global command, it is transmitted to the all already detected modules.

*rep*: is the module reply when there's one.

*status*: is the module status when the command succeed.

*errorEventEnabled*: false when the OnErrorOnUseMethod process is locally disabled.

## Retourne :

The error report number (0 when all is right)

## Remarks:

- when the address argument gets a -1 value, the command message is transmitted without address notion (**BroadcastMessage**) and regardless any result allover the opened communication pipes. (It would be better to use the **DialogueAllModules()** method)

- when the address argument gets a value superior to 65635 (0FFFF in hexa) the Msb address part specifies a local special protocol use :

> address from 10000h to 1FFFFh when automatically using the Mac protocol (syntax)

> address from 20000h to 2FFFFh when automatically using Console protocol

> address from 30000h to 3FFFFh when automatically using secateurs F30xx protocol

```
public int DialogueAllModules(string cmd)
```

Membre de [MidiIngenierie.Com](http://MidiIngenierie.Com).MiCom

## Résumé :

Transmission of a global command to the all already detected modules.

Indeed, this method enable to manage the movements synchronisation commands, direct movements with the "SYNC TOP" command and interpolated movements with the "SYNC INTERPOL" command.

WARNING: the real time synchronisation could only be done on a common communication canal !

## Paramètres :

*cmd*: is the command that is transmitted to the all already detected modules.

## Retourne :

allways 0 (all is right)

```
public int LoadConfig()
```

Membre de [MidiIngenierie.Com](http://MidiIngenierie.Com).MiCom

## Résumé :

Network configuration download from the configuration file.

The file path is the 'configFileName' property value.

All forward installed modules are removed.



**Retourne :**

0 when the download succeed.

When the file is wrong, the wrong rubric number is the 'configFileReport' property value.

**Remarks:**

- after configuration upload, modules will be finally installed when **SetOnLine()** method will be invoke, that is the communication could start.
- configuration upload can also result from writting a **loadConfigFromConfigFile** property **true** value or can automatically result from a **SetOnLine()** method use when the **loadConfigOnSetOnLine** property has got a **true** value.

**public int SaveConfig()**

Membre de [MidiIngenierie.Com](http://MidiIngenierie.Com).[MiCom](http://MiCom)

**Résumé :**

**Network saving configuration upload from the saving configuration file.**

The file path is the 'configSaveFileName' property value.

**Retourne :**

0 when the upload succeed.

**Remarks:**

- configuration saving can also result from writting a **saveConfigToConfigFile** property **true** value.

**public string GetErrorString(int num\_err)**

Membre de [MidiIngenierie.Com](http://MidiIngenierie.Com).[MiCom](http://MiCom)

**Résumé :**

**Get an explicit message corresponding to the given error number.**

**Paramètres :**

*num\_err*: is the given error number.

**Retourne :**

the explicit message.

**Example:**

"reception timeout default"

**public string[] GetLastErrorInfos()**

Membre de [MidiIngenierie.Com](http://MidiIngenierie.Com).[MiCom](http://MiCom)

**Résumé :**

Get a string array with the last appened error informations.

**Retourne :**

1st information: the signed integer error number.

2nd information: an explicit error type message.

3rd information: the method name which caused the error.

4th information: the concerned module address.

5th information: The main method argument.



**public int StartVisualComponentsRefresh()**

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Start the periodic refresh process for the all refresh service added visual components.  
There's no start when there's no added component.

**Retourne :**

The added components number.

**public int StopVisualComponentsRefresh()**

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Stop the periodic refresh process for the all refresh service added visual components.

**Retourne :**

The added components number.

**public AddVisualComponentsRefresh([MidilIngenierie.Com.MiCom.DelegateVisualComponentsRefresh](#) process)** **int**

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Add a visual component to the periodic refresh service.///

**Paramètres :**

*process*: is the periodically performed delegate process which refresh period is the visualComponentsRefreshPeriod property value.

**Retourne :**

The added components number.

**public RemoveVisualComponentsRefresh([MidilIngenierie.Com.MiCom.DelegateVisualComponentsRefresh](#) process)** **int**

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Remove a visual component from the periodic refresh service.///

**Paramètres :**

*process*: is the concerned delegate process.

**Retourne :**

The added components number.

**Remarks:**

- when there no specified process, all delegate references are deleted ant the refresh process stop.

**public int UploadFile(string fileName [, bool errorEventEnabled])**

Membre de [MidilIngenierie.Com.MiCom](#)

**Résumé :**

Specified named file Upload to the modules.



The file is first read to consider eventually upload directives.  
Each command line is transmitted to the command forward address module(s).

**Paramètres :**

*fileName*: is the upload filename.

*errorEventEnabled*: false when the OnErrorOnUseMethod process is locally disabled.

**Retourne :**

0 when all is right.

**public int StopUploadFile()**

Membre de [MidiIngenierie.Com](http://MidiIngenierie.Com).MiCom

**Résumé :**

File uploading forcestop.

WARNING: the forcestop will succeed only if the calling thread differ from the file upload thread.

**Retourne :**

always 0



## Setting a module:

- the module is scanned with the optional **path** argument and according to the **baudratesEnabled** and **protocolesEnabled** properties.
- a correctly installed module will be added to the **modulesInstalled** property, that gives the installed module list.  
( an failure primarily results when the module is not physically present on the network )
- [**path**] is the [optional] argument that gives the specific or generic pipe.  
(specific pipe examples: COM1 CAN1 )  
(generic pipe examples: Com Can Usb Eth )
- when the **path** argument is present, the module is installed only when the path complies **portsEnabled** and **portsDisabled** properties.
- when the **path** module argument is the one already installed module path (ex: RS485 line), the configuration **baudrate et protocole** will stay the first installed module configuration. So on a multi-module pipe, all modules get the same baudrate et protocol arguments.
- the scan stop when the first module with a correct address is found.

## Visual component refresh management:

- before to start the periodic refresh process, the application must write all the desired visual components to the refresh service .
- for every concerned visual component, the application so provides a delegate (using the **AddVisualComponentsRefresh()** method) that is the method to be processed when the refresh period arrives.
- the delegate prototype looks like **public void DelegateVisualComponentsRefresh()**
- the **visualComponentsRefreshPeriod** property gets an expected value with 1/10s unit.
- the **StartVisualComponentsRefresh()** method globally starts the process.
- **Attention:** every delegate is processed every refresh period, that is all delegates will be processed during the period. The user gets the responsibility for setting a suitable period according to the delegates number and without damaged response time application. (Example: with 100ms period and 10 delegates, a refresh process will arrived every 10ms)



### 3.5.2. Communication driver methods with "expert" access

*These methods allow the user to direct access the communication driver. You need a complete network topology expertise to manage all the path, baudrate, protocol and address module arguments.*

```
public int Open(string path, string baudrate, string protocol, string prefix)
Membre de MidIngenierie.Com.MiCom
```

**Résumé :**

Communication pipe opening expert method.

**Paramètres :**

*path*: is the communication pipe path to open.

*baudrate*: is the RS485 communication baudrate.

*protocol*: is the RS485 communication protocol.

*prefix*: is an argument to manage communication pipe specific commands and/or to prefix the Dialogue() method 'cmd' argument.

**Retourne :**

The opened communication pipe handler (0 or negative when it isn't opened)

```
public int Dialogue(int hCom, string cmd, ref string rep, ref int status)
Membre de MidIngenierie.Com.MiCom
```

**Résumé :**

Communication pipe dialog sequence expert method (command/reply).

The 'hCom' argument is the 'Open()' method returned handler.

The 'cmd' argument is a module interpretable command.

Just forward the command is the destination module address.

When there's no address, the command is a global command to all the communication pipe modules.

The 'rep' argument return the addressed module reply when there's one.

The 'status' argument return module state informations when the command succeed.

(see the Module user manual to get all the details about the interpretable module command/reply system).

(see the MI\_v0\_AN03\_fr.pdf application note to get details about status informations).

**Paramètres :**

*hCom*: is the communication pipe handler.

*cmd*: is the command to the module. When existing, the Open() method 'prefix' argument automatically precedes the command.

*rep*: is the module reply when there's one.

*status*: is the module status when the command succeed.

**Retourne :**

The error report number (0 when all is right)

```
public int Close(int hCom)
Membre de MidIngenierie.Com.MiCom
```

**Résumé :**

Communication pipe closing expert method.

The 'hCom' argument is the 'Open()' method returned handler.

**Paramètres :**

*hCom*: is the communication pipe handler.

**Retourne :**

The error report number (0 when all is right)



### 3.5.3. The command prefix in the expert methods

This is a string with a preceding and following **./.**  character.

When using, the **prefix** directly precedes the module command in the **Dialogue()** method command **cmd** argument.

The prefix can also be programmed in the **Open()** method command **prefix** argument. In this case, it is automatically included before the **cmd** argument when using the **Dialogue()** method.

#### Example :

```
Dialogue(address, "/Mac23Move_to 123456", ref rep, ref status);
```

The prefix allows to configure special communication pipe parameters or to inform driver to use, by example, a special protocol.

#### **/Mac23/**

Module commands are automatically send with the **Mac23** module command syntax.

#### **/Mac34/**

Module commands are automatically send with the **Mac34** module command syntax.





## 3.6. Events

*Next events allow the communication errors management especially when there is a component property writting access error.*

### 3.6.1. Writting property events

**public event** [System.EventHandler<OnErrorOnWritePropertyNotificationArgs>](#)  
**OnErrorOnWriteProperty**  
Membre de [MidiIngenierie.Com.MiCom](#)

**Résumé :**

Writing property error event.

**public string** Ev1 { set; get; }  
Membre de [MidiIngenierie.Com.OnErrorOnWritePropertyNotificationArgs](#)

**Résumé :**

Propertyname string.

**public string** Ev2 { set; get; }  
Membre de [MidiIngenierie.Com.OnErrorOnWritePropertyNotificationArgs](#)

**Résumé :**

Error cause informations string.

**public string** Ev3 { set; get; }  
Membre de [MidiIngenierie.Com.OnErrorOnWritePropertyNotificationArgs](#)

**Résumé :**

Error avoid advices string.

**public int** Ev4 { set; get; }  
Membre de [MidiIngenierie.Com.OnErrorOnWritePropertyNotificationArgs](#)

**Résumé :**

Property number..

**public int** Ev5 { set; get; }  
Membre de [MidiIngenierie.Com.OnErrorOnWritePropertyNotificationArgs](#)

**Résumé :**

Always zero.



### 3.6.2. Using method events

**public event [System.EventHandler<OnErrorOnUseMethodNotificationArgs>](#) OnErrorOnUseMethod**  
Membre de [MidiIngenierie.Com.MiCom](#)

**Résumé :**

Using method error event.

**public [string](#) Ev1 { set; get; }**

Membre de [MidiIngenierie.Com.OnErrorOnUseMethodNotificationArgs](#)

**Résumé :**

Methodname string.

**public [string](#) Ev2 { set; get; }**

Membre de [MidiIngenierie.Com.OnErrorOnUseMethodNotificationArgs](#)

**Résumé :**

Error cause informations string.

**public [string](#) Ev3 { set; get; }**

Membre de [MidiIngenierie.Com.OnErrorOnUseMethodNotificationArgs](#)

**Résumé :**

Main argument method string.

**public [int](#) Ev4 { set; get; }**

Membre de [MidiIngenierie.Com.OnErrorOnUseMethodNotificationArgs](#)

**Résumé :**

Error number.

**public [int](#) Ev5 { set; get; }**

Membre de [MidiIngenierie.Com.OnErrorOnUseMethodNotificationArgs](#)

**Résumé :**

Method address argument (-1 when address argument doesn't exist or when the message is global).



## 4. Axis components

### 4.1. Generalities

Every axis component reflects a Midi-ingenierie axis type, that is the properties and methods collection reflect all the module variables and commands as listed in the proper axis type **user guide** .  
All user guides are available connecting the <http://www.midi-ingenierie.com/> site.

Component class available axis type

**DMac17 DMAC17**  
**DMac23 DMAC23-1 DMAC23-2**  
**DMac34 DMAC34-1 DMAC34-2**  
**BMac BMAC**  
**BMac\_H BMAC\_H**

### 4.2. Component management inherents properties

These are the proper specific dotNet component management properties which get no correspondent axis variable.

```
public MidiIngenierie.Com.MiCom _Dialog { set; get; }
```

Membre de [MidiIngenierie.Axe.MIAxe](#)

#### Résumé :

Classe pour la gestion de la communication avec les modules conçus par la société Midi-Ingénierie

*Get or set the communication component reference that physically routes commands and replies to the module.*

```
public int Address { set; get; }
```

Membre de [MidiIngenierie.Axe.MIAxe](#)

#### Résumé :

*Get or set the axis communication address.  
The \_Dialog property is previously settled.*

```
public bool AxeOnLine { get; }
```

Membre de [MidiIngenierie.Axe.MIAxe](#)

#### Résumé :

*Get the detected module state that is a communication component information.  
The \_Dialog property is previously settled and references the communication component.*

```
public string Type { set; get; }
```

Membre de [MidiIngenierie.Axe.MIAxe](#)

#### Résumé :

*Get or set the planned target axis type.*



```
public string Type_real { get; }
Membre de MidilIngenierie.Axe.MIAxe
```

#### Résumé :

*Get the real detected axis type.*

**Attention:** *when this property differs the **Type[design]** property, it could be any unwanted behavior.*

```
public string FileName { get; }
Membre de MidilIngenierie.Axe.MIAxe
```

*Get the used **MidilIngenierie.Axe.dll** complet assembly filename.*

```
public string Version { get; }
Membre de MidilIngenierie.Axe.MIAxe
```

*Get the used **MidilIngenierie.Axe.dll** assembly version cod.*

```
public string Identity { get; }
Membre de MidilIngenierie.Axe.MIAxe
```

#### Résumé :

*Get the complet identity module when it has been detected. (AxeOnLine is true)*

```
public string Label { set; get; }
Membre de MidilIngenierie.Axe.MIAxe
```

#### Résumé :

*Get or set a specific information that identifies the module in the application context.*

```
public string Path { get; }
Membre de MidilIngenierie.Axe.MIAxe
```

#### Résumé :

*Get the communication pipe that is installed by the communication component when the module has been detected. (AxeOnLine True)*



## 4.3. Properties

These properties reflect the all module variables and some module commands as listed in the proper axis type **user guide**.

All user guides are available connecting the <http://www.midi-ingenierie.com/> site.

Examples:

```
public string High_Speed { set; get; }
Membre de MidilIngenierie.Axe.MIAxe
```

Résumé :

*corresponding variable: **#HIGH\_SPEED** or **#HSP** high speed parameter ( $10^2$  tour/mn)*

```
public string Low_Speed { set; get; }
Membre de MidilIngenierie.Axe.MIAxe
```

Résumé :

*corresponding variable: **#LOW\_SPEED** or **#LSP** low speed parameter ( $10^2$  tour/mn)*

```
public string Error { get; }
Membre de MidilIngenierie.Axe.MIAxe
```

Résumé :

*corresponding variable: **#ERROR** or **#ERR** binary value*

```
public string Move_To { set; get; }
Membre de MidilIngenierie.Axe.MIAxe
```

Résumé :

moveTo

*corresponding command: **MOVE\_TO** or **MTO** position parameter ( $10^4$  tour)*

You'll find, in the user guide, explanations for every variable and command including units and limit values.



## 4.4. Methods

These methods reflect some module commands as listed in the proper axis type **user guide** .  
 All user guides are available connecting the <http://www.midi-ingenierie.com/> site..

Examples:

```
public int Move_Interpol(int moveInterpolArg)
Membre de MidiIngenierie.Axe.MIAxe
```

**Résumé :**

**Paramètres :**

**Retourne :**

*corresponding command: **MOVE\_INTERPOL** or **MIN** displacement parameter ( $10^4$  tour)*

```
public int Stop(MidiIngenierie.Axe.MIAxe.HaltStopOptions WhatToDo)
Membre de MidiIngenierie.Axe.MIAxe
```

**Résumé :**

Provoque l'émission de la commande STOP pour le module concerné : Le module s'arrête en passant de la vitesse courante à une vitesse nulle en un temps défini proportionnellement au paramètre #DECEL\_TIME

**Paramètres :**

*WhatToDo*: Précise si l'arrêt concerne le mouvement ou la séquence

**Retourne :**

compte\_rendu positionné par la couche dialogue

*corresponding command: **STOP** or **STO***

You'll find, in the user guide, explanations for every command including units and limit values.



## 5. Annexe A – Communication components default index

### 5.1. Annexe A1 – Communication defaults

-5	str_ON_PARAM_BAUD	The communication port can't be open - Wrong 'baudrate' argument.
-4	str_ALREADY_USED	The communication port can't be open - The port is already used with an other application.
-3	str_ALREADY_OPENED	The communication port is already opened in this application.
-2	str_ON_OPEN	The communication port can't be open - The port with the provided path doesn't exist ...
-1	str_ON_PATH	The communication port can't be open - Wrong 'path' argument.
31	str_NOT_OPENED	Communication default - The port is not opened.
32	str_ON_HANDLE	Communication default - The provided handle is out of limits.
33	str_MESSAGE_TO_LONG	Communication default - The transmit message is too long.
34	str_MESSAGE_EMPTY	Communication default - The transmit message is empty.
35	str_IO_EXCEPTION	Communication default - An 'IO exception' has arrived.
90	str_DEVICE_DISCONNECTED	Communication default - The module has (or had) disconnected.
91	str_VCP_DISCONNECTED	Communication default - The communication virtual port (VCP) has disconnected.
100	str_EMISS_IMPOSSIBLE	Communication default - The transmit process can't succeed.
101	str_TO_EMISS	Communication default - Transmission timeout.
120	str_ON_EMISS	Communication default - Transmit process default.
200	str_REC_IMPOSSIBLE	Communication default - The receipt process can't succeed.
201	str_TO_REC_ACK	Communication default - Reception timeout (ACK).
202	str_TO_REC_ETX	Communication default - Reception timeout (ETX).
203	str_TO_REC	Communication default - Reception timeout.
210	str_ON_REC_ACK	Communication default - Protocol default, the 'ACK' character hasn't arrived.
211	str_ON_REC_XOFF	Communication default - Protocol default, the 'XOFF' character hasn't arrived.
212	str_ON_REC_XERR	Communication default - The module couldn't interpret the message, the 'XERR' character has arrived.
213	str_ON_REC_NAK	Communication default - Protocol default, the 'NAK' character has arrived.
214	str_ON_REC_ETX	Communication default - Protocol default, the 'ETX' character hasn't arrived.
220	str_ON_REC	Communication default - Reception process default.
230	str_ON_REC_LONG	Communication default - Wrong received message length.
231	str_ON_REC_CHKS	Communication default - Wrong checksum in the received message.
1000	str_CONFIG_MODULE_UNEXPECTED	Using methode default - The module configuration is wrong.
1001	str_MODULE_IS_NOT_CONFIGURATED	Using methode default - The module is not configured.
1002	str_MODULE_IS_NOT_INSTALLED	Using methode default - The module is not installed.
1003	str_MODULE_IS_NOT_DETECTED	Using methode default - The module is not detected.
1004	str_ADRESSE_MODULE_UNEXPECTED	Using methode default - Wrong 'address' argument.
1005	str_UNEXPECTED_HANDLE	Using methode default - Trying to use an unknown driver type.
1006	str_CONFIG_FILE_IS_NOT_DEFINE	Using methode default - The configuration file name is not defined.
1007	str_CONFIG_FILE_IS_NOT_EXISTING	Using methode default - The configuration file doesn't exist.
1008	str_CONFIG_FILE_IS_ON_ERROR	Using methode default - The configuration file is wrong.
1009	str_SAVE_CONFIG_FILE_IS_NOT_DEFINE	Using methode default - The save configuration file doesn't exist.
1010	str_SAVE_CONFIG_FILE_IS_NOT_CREATED	Using methode default - The save configuration file can't be created.
1011	str_FILE_IS_NOT_DEFINE	Using methode default - The file name is not defined.
1012	str_FILE_IS_NOT_EXISTING	Using methode default - The file doesn't exist.
1013	str_FILE_IS_ON_ERROR	Using methode default - The file is wrong.
	str_DEFAULT	Not identified default.

### 5.2. Annexe A2 – Property writting defaults

<i>Properties</i>	<i>représentative number</i>	
baudratesEnabled	N_baudratesEnabled	1
baudratesCanEnabled	N_baudratesCanEnabled	2
protocolesEnabled	N_protocolesEnabled	3
typesModuleEnabled	N_typesModuleEnabled	4
portsEnabled	N_portsEnabled	5
portsDisabled	N_portsDisabled	6
portsAddress	N_portsAddress	7
groupAddress	N_groupAddress	8
configModules	N_configModules	9
configFileName	N_configFileName	10
configSaveFileName	N_configSaveFileName	11
loadConfigFromConfigFile	N_loadConfigFromConfigFile1	12
loadConfigFromConfigFile	N_loadConfigFromConfigFile2	13
saveConfigToConfigFile	N_saveConfigToConfigFile	14
visualComponentsRefreshPeriod	N_visualComponentsRefreshPeriod	15





## 6. Annexe B - Special protocols use

### 6.1. Mac type module management protocol

The **Mac** type modules have got a language syntax quite different from today **xMac** modules ( $\mu$ Mac, DMac, BMac et RMac)

The communication component can execute the transmit and receive language conversion. So, **Mac** modules can be managed simply using **DMac** language with limits due to intrinsic own **Mac** module capacities. To allow easy program migration, the old driver Simpa language syntax is also still available.

Three possibilities to access module :

- with the **expert** language that module can directly interpret without using the converter.  
**Notice pour le contrôle MAC23 V24 et MAC34 V24 en mode expert**  
ref : **mac23\_34\_ex\_v5\_um\_fr.pdf**
- with the **Simpa** language syntax using the converter.  
**Manuel d'utilisation du Module Mac23**  
ref : **mac23\_v9\_um\_fr.pdf**
- with the **DMac** language syntax using the converter.  
**See included document.**

#### 6.1.1. Local Mac converter use :

**The converter is enable during the only ongoing communication.**

- using the **Dialogue(hCom,adr/cmd,rep,status)** expert method .

In this case the **cmd** method argument should be prefixed with :

- /Mac23/ for Mac23 modules
- /Mac34/ for Mac34 modules

Example (address 5 Mac module access) :

```
hcom = Open("COM3", " ", " ")
Dialogue(hcom, "/Mac23/05Move_to 12345", ref rep, ref status)
Close(hcom)
```

- using the **DialogueModule(adr,cmd,rep,...)** method. In t

In this case the **adr** argument should added with the **10000h** offset

Example (address 5 Mac module access) :

```
DialogueModule(10005h, "Move_to 12345", ref rep, ....)
```

#### 6.1.2. Global Mac converter use :

**The converter is autommatically enable during the module scanning.**

- writting **"Mac"** item in the **typesModuleEnabled** property.

In this case at least one of the next items should be writte into the property list :

- "Mac" pour une sélection génériq
- "Mac23"
- "Mac34"

Example (address 5 Mac module access) :

```
typesModuleEnabled = Array["xMac","Mac"]
DialogueModule(5, "Move_to 12345", ref rep, ....) (witout adr offset)
```



### 6.1.3. The Mac modules command converter :

#### Commands :

#### DMac

**sad** param  
(set\_address)

**sba** param  
(set\_baudrate)

**mre** ---  
(module\_reset) **all**

**pow(er)** **off**  
**on**

**ref(erence)** **on**  
**off**

**#pos(ition) := 0**

**hen** **on**  
(hard\_ends) **off**  
**pos**  
**neg**

**sen** **on**  
(soft\_ends) **off**

**#pen := param**  
(positive\_end)  
**#nen := param**  
(negative\_end)

**#tra := param**  
(torque\_ratio)

**#lsp := param**  
(low\_speed)

**#hsp := param**  
(high\_speed)  
**#ati := param**  
(accel\_time)

**syn(chro)** **on**  
**top**

**Sto(p)**

**hal(t)**

**mto** param  
(move\_to)

#### Simpa

**am** param

**mr**  
**mrz**

**gr**  
**gs**

**mza**  
**mzi**

**di**

**mbr**  
**mn**  
**mbr**  
**mbr**

**mbs**  
**mn**

**bp** param

**bn** param

**gi** param

**wl** param

**wx** param1 param2

**ws**  
**sy**

**ge**

**gs**

**ga** param

#### Mac

@ @0800000000xx

@ @1B0011223301 > 9600  
@ @1B0011223302 > 19200  
@ @1B0011223303 > 38400  
@ @010000000000  
@ @000000000000

@ @1A0000000000  
@ @190000000000

@ @070000000001  
@ @070000000000

@ @0B0000000000

@ @030000000001  
@ @030000000000  
@ @030000000001  
@ @030000000001

@ @030000000008  
@ @030000000000

@ @0900BBBBBBBB

@ @0A00BBBBBBBB

@ @0C00xxxxxxCC

@ @0E00xxxxVV00

@ @0D00xxxxVVVV  
@ @0F00xxxxxxNN

@ @100000000000  
@ @110000000000

@ @170000000000

@ @180000000000

@ @1300PPPPPPPP



**mt0** 0  
(move\_to)

**msh** param  
(move\_speed)

**mon** param1 param2  
(move\_on)

**gh**

**gf** param

**gt** param1 param2

@ @140000000000

@ @150000SSVVVV

@ @1600DDDDVVVV

### Requests:

#### DMac

**rv**

**read #pos**(ition)

**read #sta**(tus)

**read #lsp**  
(read #low\_speed)

**read #hsp**  
(read #high\_speed)

**read #ati**  
(read #accel\_time)

**read #tra**  
(read #torque\_ratio)

**read #pen**  
(read #positive\_end)

**read #nen**  
(read #negative\_end)

**read #out**(put)  
(busy)

**read #inp**(ut)  
(butée+, butée-, reference)

**read #svo**  
(supply\_voltage)

**read #cte**  
(cpu\_temperature 0,1°C)

**read #tem**(perature)  
(°C)

#### Simpa

**rv**

**qp**

**qx**

**ql**

**ql**

**ql**

**ql**

**qb**

**qb**

**qa**

**qa**

**qa**

**qa**

**qa**

#### Mac

@ @29xx00000000  
@ @2Axx00000000  
@ @240000000000

@ @200000000000

@ @210000000000

@ @250000000000

@ @250000000000

@ @260000000000

@ @260000000000

@ @220000000000

@ @230000000000

@ @270000000000

@ @270000000000

@ @270000000000

@ @270000000000

@ @270000000000



## 6.2. The Console protocol

This is an unsophisticated protocol eventually used by any Midi ingenierie modules.  
You'll find description in the document :

**Familles ...Mac & ....Simpa, Note d'application, Liaison calculateur : protocoles et syntaxe**

ref : *mi\_v0\_an03\_fr.pdf*

### 6.2.1. Protocole console local use:

- using the **Open("path","baudrate","CON")**, **Dialogue()** et **Close()** expert methods.

**In this case the protocol is always enabled with the giving path.**

The **protocole** argument of the **Open()** method should be :

- **"Console"** only the 3 first characters are significant

Example (console protocol address 5 module access) :

```
Open("COM3", " ", "console")
```

```
Dialogue(hcom, "05Move_to 123456", ref rep, ref status)
```

```
Close(hcom)
```

- using the **DialogueModule(adr,cmd,rep,...)** method.

In this case the **adr** argument should added with the **20000h** offset and **the protocol is enabled for the communication path used by the module.**

Example (console protocol address 5 module access) :

```
DialogueModule(20005h, "Move_to 12345", ref rep, ....)
```

### 6.2.2. Protocole console global use:

**The protocol is autommatically enable during the module scanning.**

- writting **"Console"** item in the **protocolesEnabled** property.

In this case the next item should be written into the property list:

- **"CON"**

Example (console protocol address 5 module access) :

```
protocolesEnabled = Array["Xon","Con"]
```

```
DialogueModule(5, "Move_to 12345", ref rep, ....) (witout adr offset)
```



## 6.3. The F30xx secateurs management

This protocol is compatible with the Console protocol. However a specific Infaco secateurs syntax is used. With this link type, the user should know the secateurs syntax , no converter is available.

You'll find description of protocol and syntax in the document :

**Supplément au manuel de l'ensemble Sécateur F3010**

ref :

**Only local use is available.**

### 6.3.1. F30xx protocol local use :

- using the **Open("path","38400","CON")** , **Dialogue()** et **Close()** expert methods.

**In this case the protocol is always enabled with the giving path.**

The **baudrate** and **protocole** arguments of the **Open()** method should be :

- **"38400"** for **baudrate** argument
- **"Console"** for **protocole** argument, only the 3 first characters are significant

Example (secateur module access) :

```
Open("COM3", "38400", "console")
Dialogue(hcom, "R76", ref rep, ref status)
Close(hcom)
```

- using the **DialogueModule(adr,cmd,rep,...)** method.

In this case the **adr** argument should added with the **30000h** offset and **the protocol is enabled for the communication path used by the module.**

Example (address 5 secateur module access) :

```
DialogueModule(30005h, "R76", ref rep, ....)
```

*Remark : the secateurs is a module without address management, so it have to be lonesome connected on a pipe. The address argument is just needed for the communication component.*