

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

midi ingenierie

Guide d'utilisation des composants .NET



Date : 20/05/2014

Référence : dotnet_v4_um_fr.pdf

Réf. MI : ELE1940894.doc

Révision : 4

Auteur : E.LOPEZ

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





Sommaire

1. Installation et mise en oeuvre	4
1.1. Installation	4
1.2. Mise en oeuvre avec Visualstudio C# 2010	5
1.2.1. Créer une application	5
1.2.2. Référencer les composants Midi-ingénierie dans la solution	5
1.2.2.1. Ajouter les composants Midi-ingénierie dans la boîte à outils	5
1.2.2.2. Vérifier le référencement des assembly midi ingénierie dans la solution	7
1.2.3. Créer des instances des composants Midi-ingénierie	8
1.2.4. Culture et langage	9
1.2.4.1. En mode développement (Design time)	9
1.2.4.2. En mode exécution (Run time)	9
2. Généralités sur les composants	10
2.1. Les principes de base	10
2.2. Le composant de communication MiCom	10
2.3. Les composants de contrôle d'axe xMacxx	11
2.4. Exemple d'application avec double session du composant de communication	12
3. Le composant de communication	13
3.1. Généralités	13
3.2. Canal (ou ligne) de communication et pathName	13
3.3. Le système d'adressage des modules	14
3.3.1. Les principes de base :	14
3.3.2. Utilisation de l'adresse 16 bits :	15
3.3.3. BroadcastMessage (messages à destination générale) :	15
3.3.4. Accès multi-thread :	16
3.4. Les propriétés	17
3.5. Les méthodes	23
3.5.1. Méthodes d'accès 'standard' au driver de communication	23
3.5.2. Les méthodes d'accès "expert" au driver de communication	30
3.5.3. Le préfixe de commande dans les méthodes expert	31
3.6. Les événements	32
3.6.1. Évènement sur écriture d'une propriété	32
3.6.2. Évènement sur utilisation d'une méthode	33
4. Les composants d'axe	34
4.1. Généralités	34
4.2. Propriétés inhérentes à la gestion de composant	34
4.3. Les propriétés	36
4.4. Les méthodes	37
5. Annexe A – Répertoire des défauts du composant de communication	38
5.1. Annexe A1 – Défauts de communication	38
5.2. Annexe A2 - Défauts en écriture de propriété	38
6. Annexe B - Utilisation de protocoles spécifiques	39
6.1. Le protocole de gestion des modules de type Mac	39
6.1.1. Utilisation locale du convertisseur Mac :	39
6.1.2. Utilisation globale du convertisseur Mac :	39
6.1.3. Le convertisseur de commandes pour modules Mac :	40
6.2. Le protocole Console	42
6.2.1. Utilisation locale du protocole console:	42
6.2.2. Utilisation globale du protocole console :	42
6.3. Le protocole de gestion des sécateurs de type F30xx	43
6.3.1. Utilisation locale du protocole F30xx:	43

Version 1.0 19/07/2012 document d'origine

Version 2 27/09/2012 ajout des chapitres Installation et Utilisation sous Visual Studio C# 2010



Version 3 02/01/2013 précise que les listes des propriétés **portsXxxDetected** sont ordonnées

Version 4 20/05/2014 compilé avec **version 2.0** du **.NET Framework** (3.5 en version précédente)



1. Installation et mise en oeuvre

Midi-ingénierie met à disposition des programmeurs des composants **.Net** pour faciliter le développement d'applications dédiées à l'utilisation des contrôleurs d'axe et des servo-moteurs proposés par la société.

Les composants sont utilisables avec la **version 2.0** (et suivantes) du framework .Net :

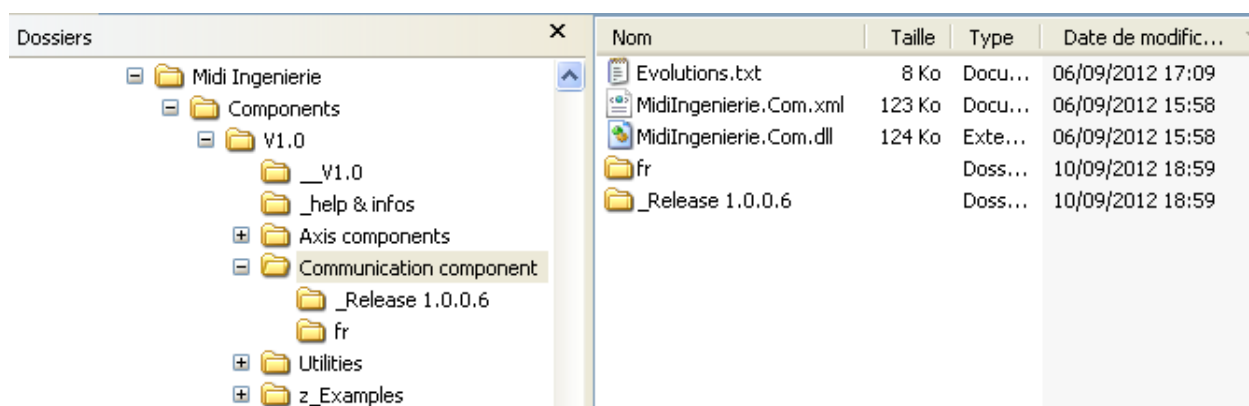
- le composant de communication est disponible sous l'assembly **MidiIngenierie.Com.dll**.
- les composants de contrôle d'axe sont disponibles sous l'assembly **MidiIngenierie.Axe.dll**.

1.1. Installation

Les composants Midi ingénierie sont disponibles sur le site <http://www.midi-ingenierie.com/>.

Ils sont fournis sous forme d'un package repéré par son numéro de version Vx,x qui comprend :

- les assembly respectifs des composants.
- la documentation disponible.
- des utilitaires pour la configuration et le contrôle des modules d'axe.
- des exemples adaptés aux différentes plateformes de développement (Visual Basic, C#, Labview...).

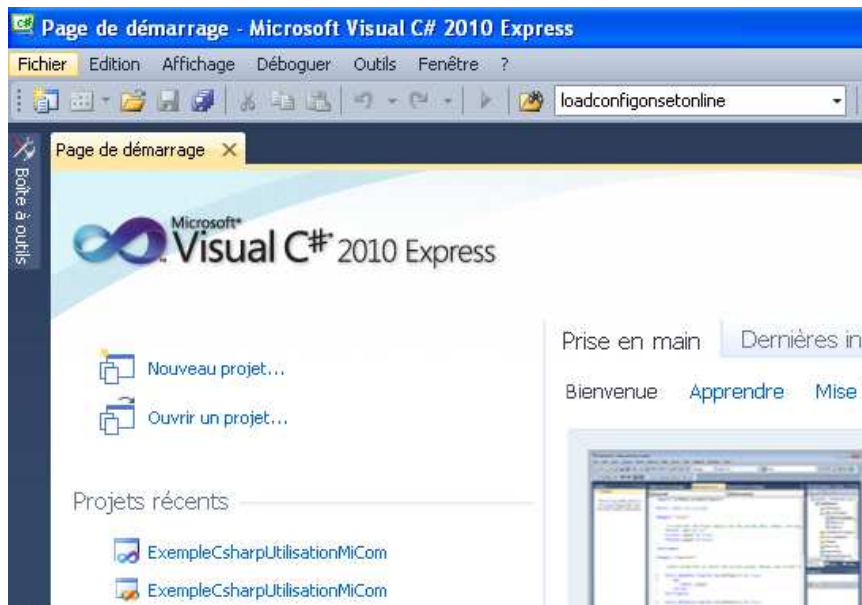


L'installateur propose une installation à défaut dans le répertoire **Midi ingenierie** directement sous la racine principale. L'utilisateur pourra réorienter vers un répertoire de son choix du type **Program files** ou autre ...



1.2. Mise en oeuvre avec Visualstudio C# 2010

1.2.1. Créer une application

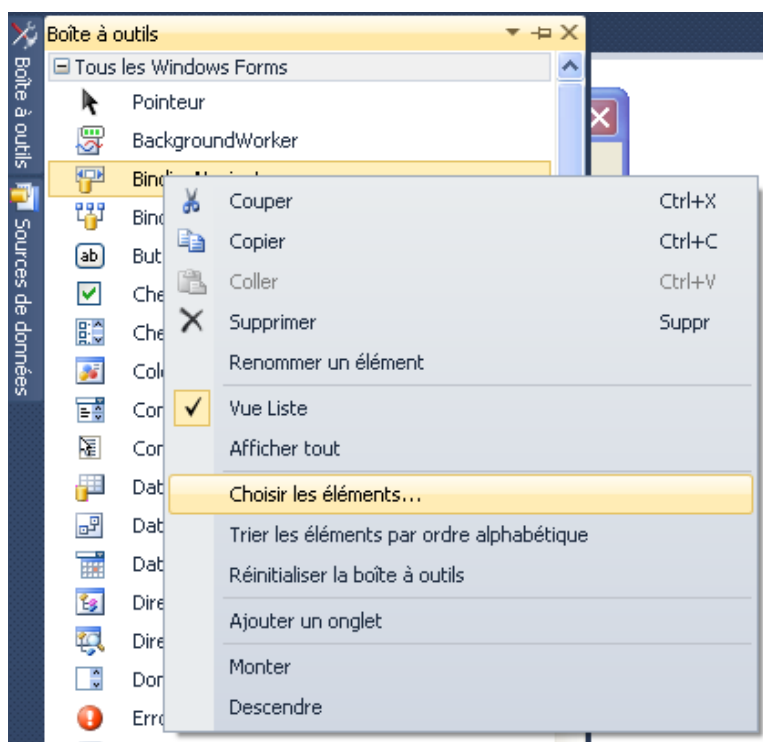


Ouvrir un nouveau projet **Application Windows Forms**.

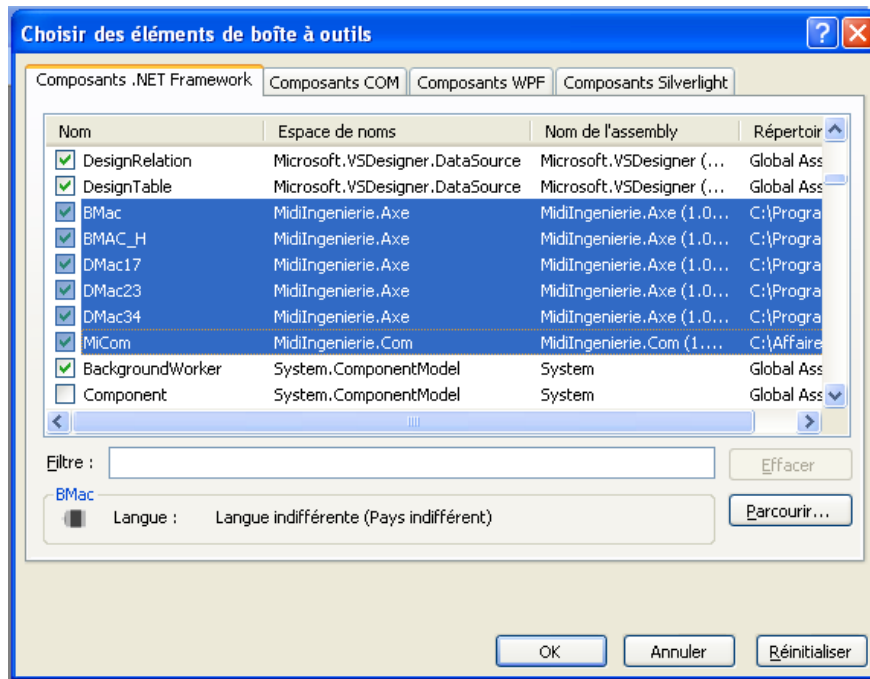
Enregistrer votre solution **MyApplication.sln** dans le répertoire désiré.

1.2.2. Référencer les composants Midi-ingénierie dans la solution

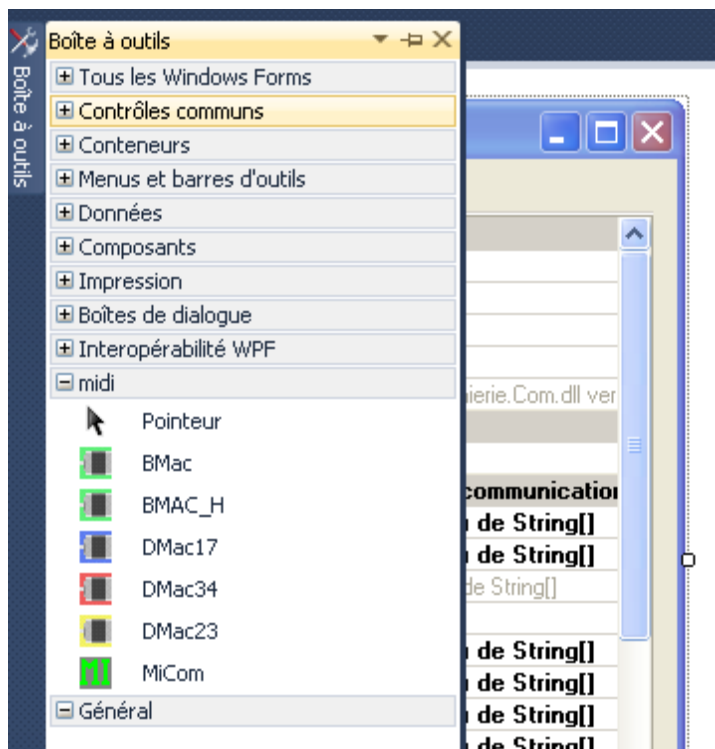
1.2.2.1. Ajouter les composants Midi-ingénierie dans la boîte à outils



en cliquant à droite sur un contrôle quelconque de la boîte à outils, puis en cliquant sur l'onglet **Choisir les éléments...** on accède à l'écran permettant d'ajouter les composants Midi ingénierie.



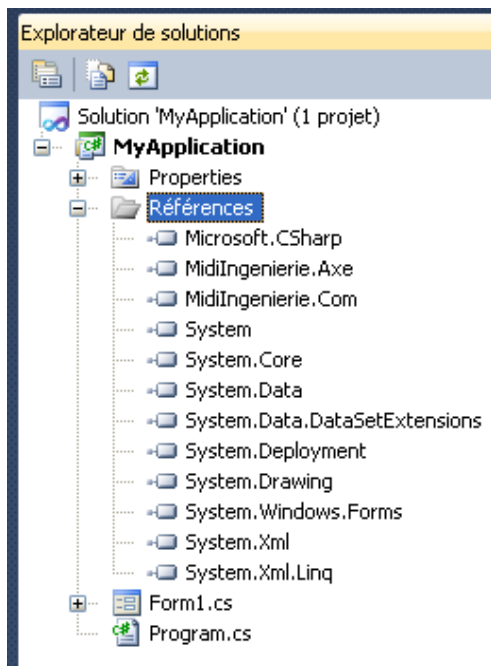
En utilisant le bouton **Parcourir...**, ouvrir les assembly MidiIngenierie.Axe et MidiIngenierie.Com pour ajouter les composants disponibles.



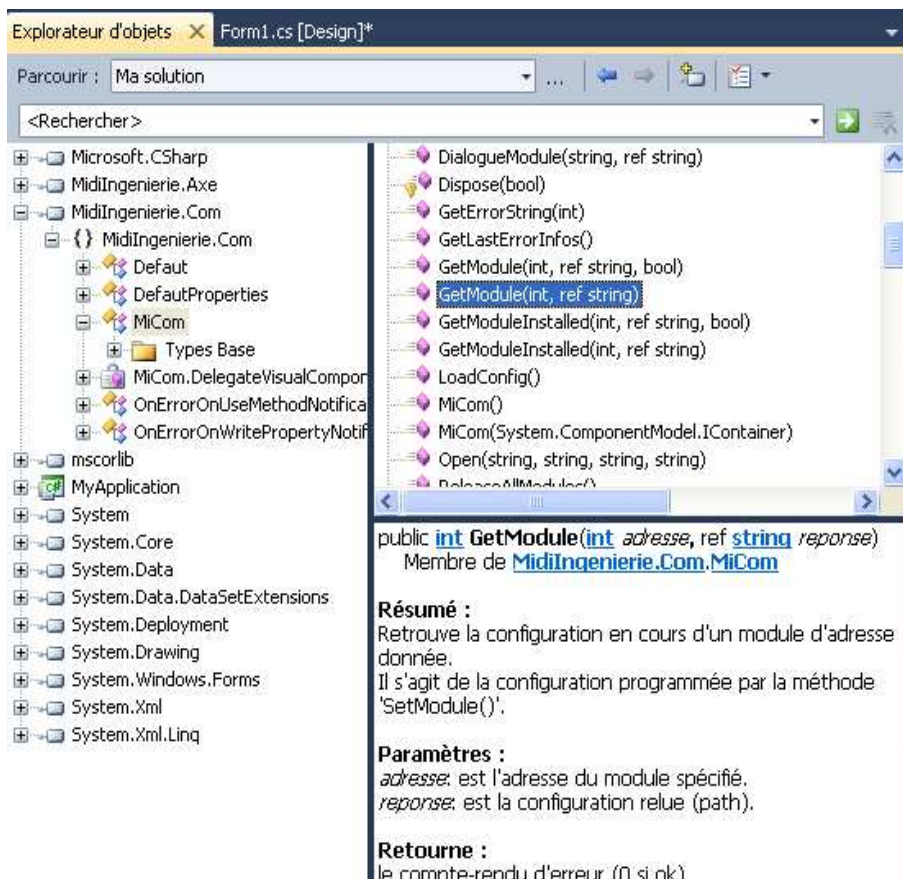
Désormais, la boîte à outil contient les différents composants Midi-ingénierie disponibles



1.2.2.2. Vérifier le référencement des assembly midi ingénierie dans la solution



Dans la rubrique **Références** de votre projet vérifier les références sur l'assembly **MidiIngenierie.Com.dll** qui expose le composant de communication ainsi que sur l'assembly **MidiIngenierie.Axe.dll** qui expose les composants d'axes.

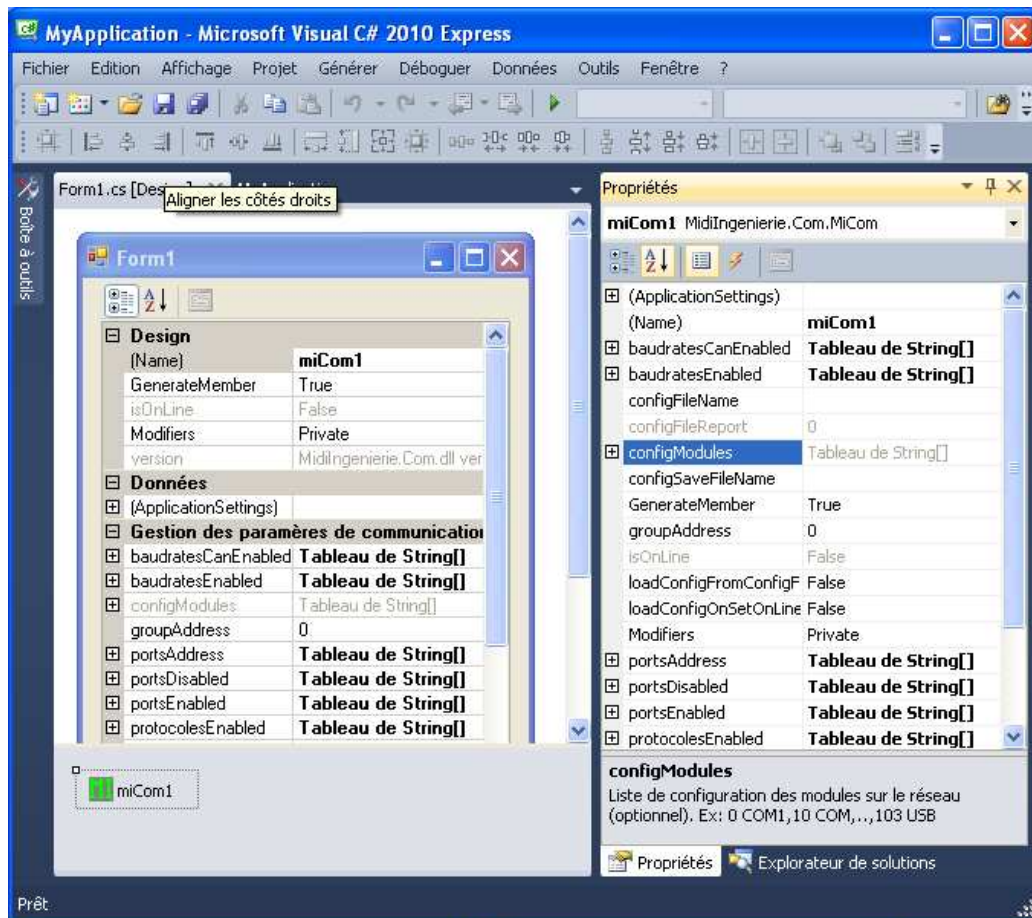


En double-cliquant sur un objet référencé dans la rubrique référence de l'explorateur de solutions on accède à l'explorateur d'objets qui permet d'exposer l'ensemble des propriétés et méthodes des composants disponibles.

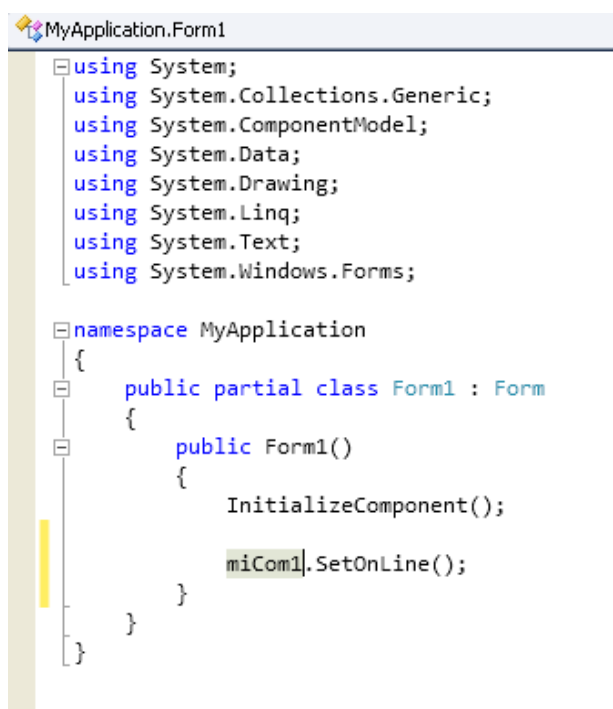


1.2.3. Créer des instances des composants Midi-ingénierie

Une fois les composants placés dans la boîte à outil, des instances de ces composants peuvent être glissées dans une fenêtre au même titre qu'un bouton ou tout autre contrôle classique.



Les propriétés des objets ainsi créés peuvent se visualiser dans la fenêtre **Propriétés**.



Attention:

Lorsque vous créez une instance du composant de communication sur une WindowsForm, il convient d'introduire un appel à la méthode **SetOnLine()** directement dans le constructeur de la fenêtre après l'appel automatique du designer à la méthode **InitializeComponent()**.

La communication avec d'éventuels modules présents sur le réseau est systématiquement verrouillée à défaut pour ne pas interférer de manière inadéquate en mode design (design time). L'appel à la méthode **SetOnLine()** permet de déverrouiller (autoriser) cette communication en mode exécution (run time).



1.2.4. Culture et langage

La culture est automatiquement gérée avec l'utilisation des composants Midi-ingénierie.

1.2.4.1. En mode développement (Design time)

Cela concerne:

- les commentaires de la grille des propriétés du composant
- les commentaires de l'explorateur de composants *
- les commentaires de saisie de code *

* Ces commentaires sont fournis sous forme de fichiers **.xml** associés au fichier **.dll** de l'assembly.

Le fichier à défaut (langue anglaise) est fourni dans le répertoire même de l'assembly.

Le fichier en langue française est fourni dans le sous répertoire **/fr**

Attention: lorsqu'aucun fichier **.xml** n'est présent, les commentaires ne sont pas disponibles .
lorsque le fichier **/fr.xml** n'est pas présent, les commentaires sont, à défaut, en langue anglaise .

1.2.4.2. En mode exécution (Run time)

Cela concerne:

- les messages de défaut directement gérés par le composant (exemple: les défauts de communication)
- les commentaires de la grille des propriétés du composant utilisé sous forme d'un contrôle **propertygrid**

Aucun fichier associé n'est nécessaire à la gestion multilingue de ces messages.



2. Généralités sur les composants

2.1. Les principes de base

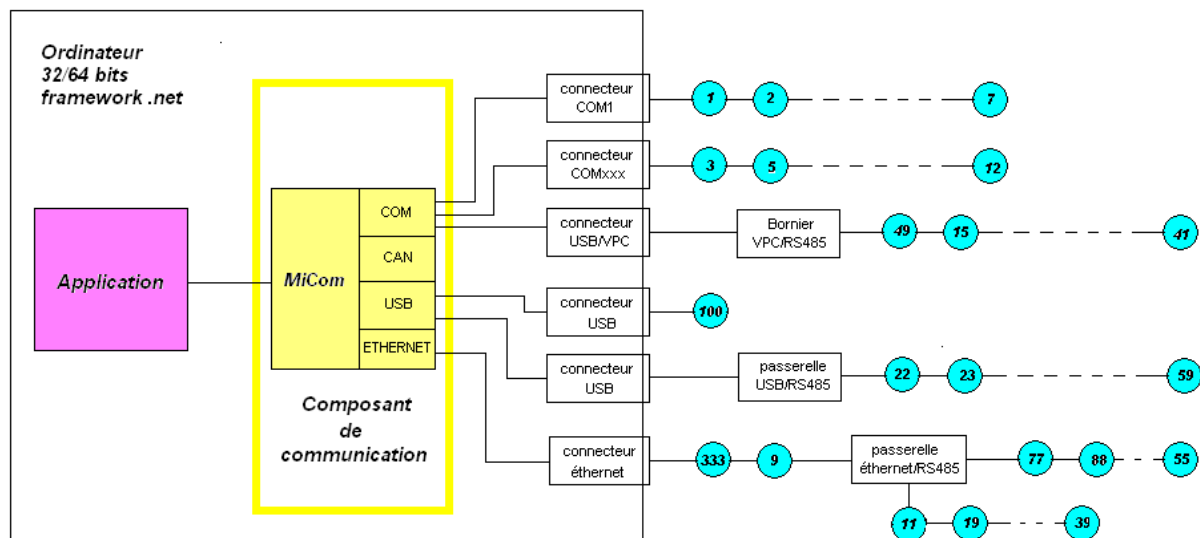
1er principe

Chaque module (contrôleur ou servo-moteur) dispose d'une adresse propre configurable.
(0 en sortie usine)

2e principe

Chaque module présent sur le réseau sera vu, du point de vue applicatif, selon une adresse unique, quelque-soit la topologie du réseau disponible.
(liaison COM, CAN, Usb ou ethernet)

2.2. Le composant de communication MiCom



Il permet d'assurer la communication de base des modules Midi-ingénierie avec les ordinateurs de type PC sous les systèmes d'exploitation Windows XP et ultérieurs, 32 et 64 bits.



Sont pris en charge les divers moyens de communication aujourd'hui communément disponibles:

- liaison port Com
- liaison port Com virtuel
- liaison Usb/Hid
- liaison ethernet/Tcp/IP
- liaison Can

Le programmeur est déchargé du soucis de la gestion du protocole de ligne et peut faire fi de toute connaissance sur la topologie du réseau, l'accès au module se faisant par la seule connaissance de son adresse.

Cependant, un accès direct aux modules (contrôle d'axe), par le seul composant de communication nécessite l'utilisation du langage spécifique aux modules (commandes/réponses) tel que décrit dans le manuel utilisateur du module auquel il convient de se référer. Se connecter au site <http://www.midi-ingenierie.com/>

2.3. Les composants de contrôle d'axe xMacxx

Ils permettent, chacun, d'assurer le contrôle d'un type d'axe, c.a.d que le contrôle se fait par l'utilisation des propriétés et des méthodes du composant sans soucis du langage propre de l'axe (commandes et réponses) ni du protocole de communication employé.

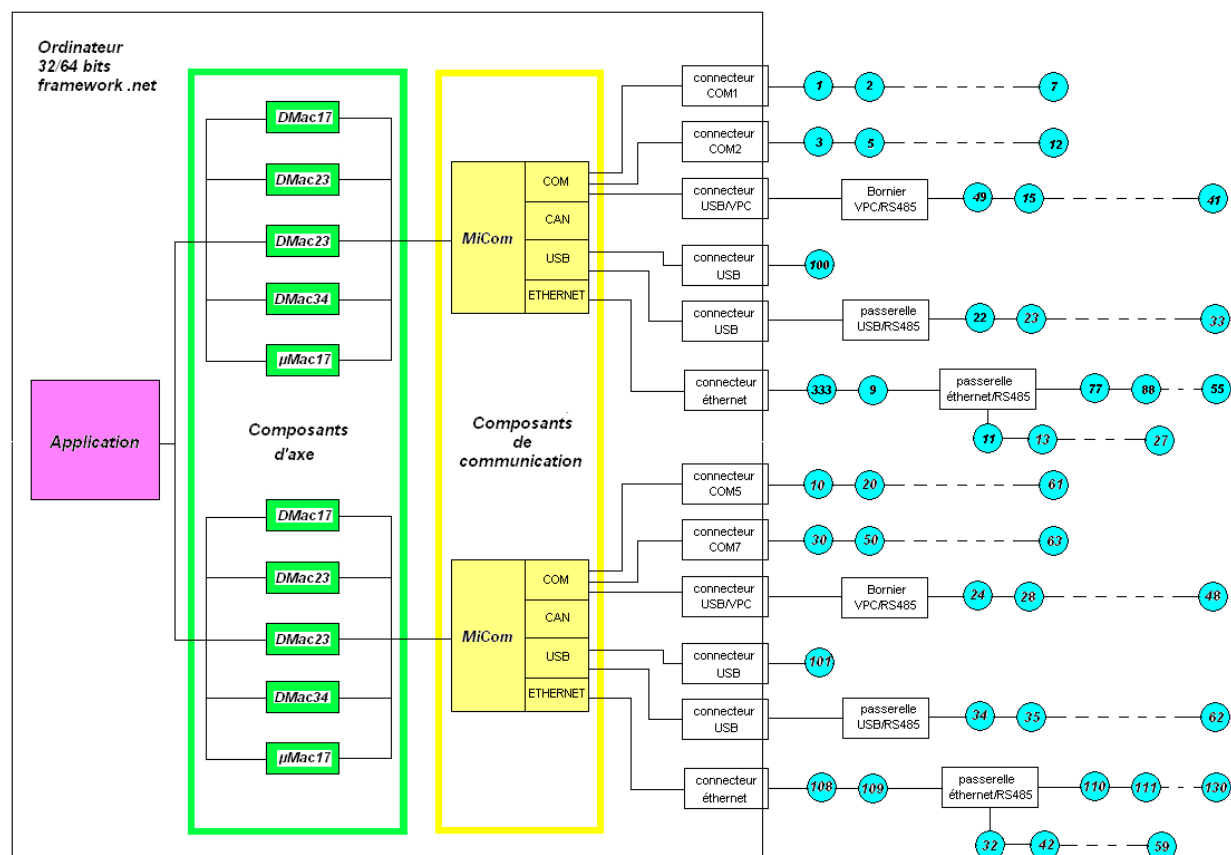
L'utilisation des composants d'axe nécessite l'utilisation d'au moins une instance du composant de communication.

Autant de composants d'axe que de type d'axe disponibles:

- µMAC17
- DMac17
- DMac23
- DMac34
- BMac
- BMac_H



2.4. Exemple d'application avec double session du composant de communication



Exemple de topologie avec l'utilisation de 2 sessions du composant de communication



3. Le composant de communication

3.1. Généralités

Le composant **MiCom** expose un ensemble de propriétés et de méthodes permettant de gérer la communication sur un réseau hétérogène où le repère incontournable est l'**adresse intrinsèque** (ou **adresse physique**) du module.

Cette adresse est une adresse attachée au module. C'est réellement une adresse physique (micro-commutateurs accessibles sur le module) ou une adresse programmable (programmée en mémoire non volatile du module) à défaut à 0 en sortie usine.

Principe d'unicité de l'adresse intrinsèque : un seul module au maximum à une adresse donnée sur le réseau.

Dans un tel système, pour communiquer avec un module, l'application peut ne gérer qu'uniquement l'adresse du module à l'exclusion d'une quelconque référence à un type de communication (port COM, port COM virtuel(Usb), Usb, ligne CAN ou ligne Ethernet), un baudrate ou un protocole.

Dans cette approche des choses, du point de vue de l'application l'**adresse intrinsèque (physique) du module est unique**. (ou en d'autres termes, l'application gère un réseau avec un éventail unique de modules d'adresse allant de **0 à 65535**)

3.2. Canal (ou ligne) de communication et pathName

On définit le **canal de communication** comme étant l'entité de communication attachée à un **pathName**. Le **pathName** est l'identité unique attribuée par le système au canal de communication.

Le **canal** est physiquement une liaison (ou ligne de communication) de type mono-point ou multi-points:

- **mono-point** : 1 seul module sur le canal, c'est le cas des modules avec liaison Usb(hid) ou ethernet "direct"
- **multi-points** : plusieurs modules possibles sur le canal:
 - c'est le cas des modules avec liaison RS485 branchés sur un port COM, un virtual port COM ou une passerelle Usb(hid)/RS485 ou ethernet/RS485 avec 64 modules possibles sur la ligne.
 - c'est également le cas des modules avec liaison Can avec 127 modules possibles sur la ligne.

Pour un port Com, le pathName est de la forme "COMx" avec x de 1 à 255

Pour un port Can, le pathName est de la forme "CANx" avec x de 1 à ...

Pour un canal Usb ou ethernet, le pathName est plus complexe, on utilisera les formes génériques "USB" et "ETH"

Deux applications différentes ne pourront pas communiquer sur le même canal de communication.



3.3. Le système d'adressage des modules

3.3.1. Les principes de base :

Adresse physique :

- les modules Midi-ingénierie disposent d'une adresse "physique" spécifique à Midi-ingénierie permettant de les repérer dans un réseau indépendamment du type de liaison utilisée (RS485, Usb, CAN, ethernet). Cette adresse peut être hard (switchs) ou logiciel (Eeprom).

Types de module :

- On distinguera 2 types de modules :
- les **anciens modules** qui disposent d'une liaison de type:
 - RS485 avec une adresse physique sur 6 bits, c.a.d de 0 à 63
 - CAN avec une adresse physique sur 7 bits, c.a.d de 1 à 127
- les **nouveaux modules** qui disposeront d'une adresse physique sur 16 bits, c.a.d de 0 à 65535

Compatibilité :

- la compatibilité sera assurée pour permettre le fonctionnement combiné des anciens et des nouveaux modules en tout point du réseau.

Types de liason :

- monopoint:** Usb(hid) ou ethernet "direct", dans ce cas le module est de type nouveau avec adresse 16 bits
- multipoint:** port COM, Usb(Virtual port Com), passerelle Usb(hid)/RS485 ou ethernet/RS485, port CAN, dans ce cas le module peut être de type nouveau (adresse 16bits) ou ancien (adresse 6/7bits)

Adresse de dialogue :

- l'adresse de dialogue est l'adresse qui précède systématiquement tout message transmis à un module quelque-soit le type de ligne utilisée.
- tout message de réponse est émis par le module précédé de la même adresse de dialogue, ceci à l'identique des transmissions avec les anciens modules, à savoir :
 - 6bits de 0 à 63 en RS485, Usb et ethernet
 - 7bits de 0 à 127 pour le CAN
- l'adresse de dialogue est formulée ainsi :
 - adresse de dialogue (@_{6bits}) = adresse physique (@_{16bits}) [modulo 64] en RS485, Usb, ethernet
 - adresse de dialogue (@_{7bits}) = adresse physique (@_{16bits}) [modulo 128] en CAN
- le driver de communication assure la correspondance entre adresse physique , port de communication et adresse de dialogue.**

Pour les modules de type nouveau :

- la réponse à la commande d'identification @_{6bits}RV offre un champ supplémentaire précisant l'adresse 16bits du module.
- le module (de type nouveau) répondra à la commande @_{6bits}RV dans les cas où:
- @_{6bits} = @_{16bits} [modulo 64] en RS485
- @_{7bits} = @_{16bits} [modulo 128] en CAN
- quelque-soit @_{6bits} en Usb et ethernet monopoint



3.3.2. Utilisation de l'adresse 16 bits :

. unicité de l'adresse de module du point de vue de l'applicatif .

(c.a.d : 1 seul module correspondant à l'utilisation d'une adresse)

. adresse utilisateur + adresse de groupe = adresse de module + adresse de port .

avec:

- **adresse utilisateur** qui est l'adresse utilisée par l'applicatif
- **adresse de module** qui est l'adresse physique présente sur le module.
- **adresse de groupe** [optionnelle] qui permet de répartir un ensemble de modules entre différentes instances d'un même applicatif. (sur la même machine ou sur des machines différentes lorsqu'il s'agit d'un lien ethernet)
- **adresse de port** [optionnelle] qui permet d'installer des modules d'adresses physiques identiques sur des ports de communication différents. (COMx CANx USB ETH)

adresse de groupe et **adresse de port** se définissent au niveau du composant de communication pour être associées à des ports de communication.

*Exemple pour une **adresse de groupe**:*

Pour une 1ere instance d'application, le port Ethernet est associé à une adresse de groupe **0000h** (valeur à défaut)

Dans ce cas le module d'adresse physique **8203h** installé sur le port Ethernet sera atteint par l'utilisation de l'adresse **8203h**

Pour une 2eme instance de l'application, le port Ethernet est associé à une adresse de groupe **1000h**

Dans ce cas le module d'adresse physique **9203h** installé sur le port Ethernet sera atteint par l'utilisation de l'adresse **8203h**

*Exemple pour une **adresse de port**:*

Le port COM1 est associé à une adresse de port **0000h** (valeur à défaut)

Dans ce cas le module d'adresse physique **00h** installé sur le port COM1 sera atteint par l'utilisation de l'adresse **0000h**

Le port Usb est associé à une adresse de port **3000h**

Dans ce cas le module d'adresse physique **0000h** installé sur le port Usb sera atteint par l'utilisation de l'adresse **3000h**

3.3.3. BroadcastMessage (messages à destination générale) :

Lorsqu'une application transmet un message avec l'adresse 16bits a la valeur -1, le message est véhiculé sans adresse et sans souci de résultat sur l'ensemble des canaux de communication ouverts (au moins une communication a déjà eu lieu sur chacun de ces canaux).

On pourra également utiliser la méthode de dialogue **DialogueAllModules()** spécifiquement destinée à la transmission de message généraux.



3.3.4. Accès multi-thread :

De par sa structure (gestion de listes...), le composant de communication ne permet pas un réel accès simultané aux propriétés et méthodes d'une **même session** du composant pour différents threads d'une **même application**.

Dans ce cas, les accès seront simplement traités de manière **séquentielle**, c.a.d que l'accès d'un thread est automatiquement différé tant que l'accès en cours d'un autre thread n'est pas terminé.

Comment réaliser **un accès multi-thread réel** au composant ?

Il convient pour cela d' **attribuer à chaque thread une session différente du composant de communication**.

Dans ce cas, chaque session possède ses propres listes et chaque thread accède, **en temps partagé** (automatiquement géré par l'OS Windows), aux différentes propriétés et méthodes indépendamment de tout autre thread.

On pourra ainsi réaliser, par exemple, des communications simultanées sur des canaux de communication différents.



3.4. Les propriétés

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

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

Résumé :

Propriété qui expose la version logicielle du composant.

Exprimée en numéro de version x.x.x.x suivi de la date de compilation jj.mm.aaaa

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

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

Résumé :

Propriété qui expose l'état de la connexion au réseau.

'true' lorsque la connexion est établie.

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

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

Résumé :

Propriété qui expose la liste ordonnée des ports Com détectés.

Expression de type "COMx" .

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

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

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

Résumé :

Propriété qui expose la liste ordonnée des ports Can détectés.

Expression de type "CANx" .

Exemple: "CAN1","CAN2" .

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

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

Résumé :

Propriété qui expose la liste ordonnée des canaux Usb spécifiques Midi-ingénierie détectés.

Expression de type path Usb/hid .

Exemple de path usb/hid: hid#vid_10c4.pid_ea80....

Exemple:

"hid#vid_05a4&pid_9862&.....",.....,"hid#vid_062a&pid_0000&....."

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

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

Résumé :

Propriété qui expose la liste ordonnée des canaux ethernet spécifiques Midi-ingénierie détectés.

Expression de type adresse IP.adresse de port .

Exemple: "192.168.0.77.10001", ... , "192.168.0.125.10001" .



public [int](#) **nbrPortsComDetected** { get; }
Membre de [MidiIngenierie.Com.MiCom](#)

Résumé :

Propriété qui expose le nombre de ports Com détectés.

public [int](#) **nbrPortsCanDetected** { get; }
Membre de [MidiIngenierie.Com.MiCom](#)

Résumé :

Propriété qui expose le nombre de ports Can détectés.

Retourne -1 lorsque le système Can est indisponible sur la machine.

public [int](#) **nbrPortsUsbDetected** { get; }
Membre de [MidiIngenierie.Com.MiCom](#)

Résumé :

Propriété qui expose le nombre de canaux Usb spécifiques Midi-ingénierie détectés.

Retourne -1 lorsque le système Usb est indisponible sur la machine.

public [int](#) **nbrPortsEthDetected** { get; }
Membre de [MidiIngenierie.Com.MiCom](#)

Résumé :

Propriété qui expose le nombre de canaux ethernet spécifiques Midi-ingénierie détectés.

Retourne -1 lorsque le système ethernet est indisponible sur la machine.

public [string\[\]](#) **baudratesEnabled** { set; get; }
Membre de [MidiIngenierie.Com.MiCom](#)

Résumé :

Propriété qui donne accès en lecture/écriture à la liste des baudrates RS485 autorisés.

Exprimé en bits/s .

Exemple: "9600", ... , "38400" .

Attention: une liste vierge désactive toute connexion de type RS485.

public [string\[\]](#) **baudratesCanEnabled** { set; get; }
Membre de [MidiIngenierie.Com.MiCom](#)

Résumé :

Propriété qui donne accès en lecture/écriture à la liste des baudrates CAN autorisés.

Exprimé en kbits/s .

Exemple: "1000" .

Attention: une liste vierge désactive toute connexion de type CAN.

public [string\[\]](#) **protocolesEnabled** { set; get; }
Membre de [MidiIngenierie.Com.MiCom](#)

Résumé :

Propriété qui donne accès en lecture/écriture à la liste des protocoles autorisés.

Exemple: "XON" pour utilisation du protocole Xon/Xoff

"CON" pour utilisation du protocole console

Attention: une liste vierge désactive toute connexion.



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

Résumé :

Propriété qui donne accès en lecture/écriture à la liste des types de modules autorisés.

Exemple: "xMac", ... , "Mac" .

Liste des modules sélectionnables :

xMac pour les modules de type µMac, DMac, BMac et RMac

Mac pour les modules de type Mac

F30xx pour les sécateurs

Attention: une liste vierge autorise le type de module à défaut xMac.

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

Résumé :

Propriété qui donne accès en lecture/écriture à la liste des ports autorisés.

Les ports sont repérés par leur path complet ou par leur path générique.

Exemples de path complet: "COM1" "CAN1" .

Exemples de path générique: "COM" "CAN" "USB" "ETH".

Les ports Usb et ethernet ne sont utilisés ici que par leurs path génériques.

Exemple d'utilisation de la propriété: "COM1","ETH" .

Attention: une liste vierge autorise l'accès à tout port à l'exclusion de ceux répertoriés par la propriété 'portsDisabled'.

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

Résumé :

Propriété qui donne accès en lecture/écriture à la liste des ports non autorisés.

Les ports sont repérés par leur path complet ou par leur path générique.

Exemples de path complet: "COM1" "CAN1" .

Exemples de path générique: "COM" "CAN" "USB" "ETH".

Les ports Usb et ethernet ne sont utilisés ici que par leurs path génériques.

Exemple d'utilisation de la propriété: "CAN","USB" .

Attention: une liste vierge n'a aucune incidence sur la sélection des ports.

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

Résumé :

Propriété qui donne accès en lecture/écriture à la liste de sélection des adresses de base des ports.

L'adresse de base d'un port est telle que: adr utilisateur + [adr groupe] = adr port + adr module.

cette adresse permet à une application de se connecter à des modules d'adresses physiques identiques sur des canaux de communication (ports) distincts.

Les ports sont repérés par leur path complet ou par leur path générique.

Exemples de path complet: "COM1" "CAN1" .

Exemples de path générique: "COM" "CAN" "USB" "ETH".

Les ports Usb et ethernet ne sont utilisés ici que par leurs path génériques.

Exemple d'utilisation de la propriété: "COM1 100","ETH 1000" .

Valeur à défaut de l'adresse de base d'un port: 0.



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

Résumé :

Propriété qui donne accès en lecture/écriture à une adresse de groupe pour l'application considérée. L'adresse de groupe pour une application est telle que: adr utilisateur + adr groupe = [adr port] + adr module.

cette adresse permet, par exemple, à 2 instances d'une même application de se connecter à des modules sur un canal de communication (ports) commun.

cette adresse n'a d'intérêt que pour une utilisation des ports Usb ou ethernet, les ports COMx ou CANx ne pouvant être physiquement partagés par 2 applications.

Valeur à défaut: 0.

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

Résumé :

Propriété qui donne accès en lecture à la liste de configuration des modules.

Chaque élément de la liste donne accès aux arguments de configuration d'un module.

Il s'agit des arguments adresse et [path] (optionnel) fournis par la méthode SetModule(adr,[path]) et utilisés pour établir la connexion.

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

Lorsque la méthode 'SetOnLine()' est utilisée, l'ensemble des modules de la liste de configuration sont installés, c.a.d qu'une connexion est réalisée (ou tentée) selon les arguments fournis.

Cette liste est constituée de manière globale lors de l'utilisation de la méthode 'LoadConfig()' en chargeant le fichier de configuration dont le nom est fourni par la propriété 'configFileName'.

Ce chargement global peut également être automatique lors de l'utilisation de la méthode 'SetOnLine()' lorsque la propriété 'loadConfigOnSetOnLine' est activée.

Elle peut également être constituée élément par élément par l'utilisation de la méthode 'SetModule()'.

Cette liste est sauvegardée de manière globale lors de l'utilisation de la méthode 'SaveConfig()' pour constituer le fichier de configuration dont le nom est fourni par la propriété 'configSaveFileName'.

La liste de configuration des modules est limitée à 255 éléments.

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

Résumé :

Propriété qui expose la liste des modules installés.

Chaque élément de la liste expose les arguments d'installation d'un module.

Il s'agit des arguments adresse,path,baudrate,protocole et type réellement utilisés pour établir la connexion.

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

Remarques:

- Un module peut faire partie de la liste de configuration et ne pas apparaître dans la liste des modules installés si celui-ci n'a pas pu être installé.(ex: le module n'est pas présent). Inversement un module installé fait obligatoirement partie de la liste de configuration des modules.

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

Résumé :

Propriété qui donne accès en lecture/écriture au chemin/nom du fichier de configuration.



```
public string configSaveFileName { set; get; }  
Membre de MidilIngenierie.Com.MiCom
```

Résumé :

Propriété qui donne accès en lecture/écriture au chemin/nom du fichier de sauvegarde de la configuration. Ce chemin peut être identique à celui du fichier de configuration (voir la propriété 'configFileName').

Remarques:

- les noms des fichiers de configuration et de sauvegarde pourront être identiques.

```
public bool loadConfigOnSetOnLine { set; get; }  
Membre de MidilIngenierie.Com.MiCom
```

Résumé :

Propriété booléenne qui permet de déclencher automatiquement le chargement du fichier de configuration lors de l'utilisation de la méthode 'SetOnLine()'.
Le chemin du fichier de configuration est programmé dans la propriété 'configFileName'.

```
public bool loadConfigFromConfigFile { set; get; }  
Membre de MidilIngenierie.Com.MiCom
```

Résumé :

Propriété booléenne qui permet de déclencher immédiatement le chargement du fichier de configuration. L'écriture de la valeur 'true' déclenche le chargement.
La valeur de la propriété est automatiquement remise à 'false' (action de type bouton poussoir).
Le chemin du fichier de configuration est programmé dans la propriété 'configFileName'.

Remarques:

- équivalent à l'utilisation de la méthode LoadConfig()

```
public bool saveConfigToConfigFile { set; get; }  
Membre de MidilIngenierie.Com.MiCom
```

Résumé :

Propriété booléenne qui permet de déclencher immédiatement la sauvegarde de la configuration. L'écriture de la valeur 'true' déclenche la sauvegarde.
La valeur de la propriété est automatiquement remise à 'false' (action de type bouton poussoir).
Le chemin du fichier de sauvegarde est programmé dans la propriété 'configSaveFileName'.

Remarques:

- équivalent à l'utilisation de la méthode SaveConfig()

```
public int configFileReport { get; }  
Membre de MidilIngenierie.Com.MiCom
```

Résumé :

Propriété qui expose le numéro de la rubrique éronnée du fichier de configuration.



public int visualComponentsRefreshPeriod { set; get; }

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

Résumé :

Propriété période de rafraichissement globale des composants visuels.

Valeur exprimée en 0.1s .

Chaque composant visuel inscrit est rafraichi au rythme de cette période.

public string loggingEnable{ set; get; }

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

Résumé :

Propriété qui permet de choisir un mode de logging, c.a.d un tracé des accès au composant et des communications réalisées.

Le tracé est réalisé dans le fichier "MidilIngenierie.Com.log" dans le répertoire de l'application.

La gestion du fichier de logging est laissée à la discrétion de l'utilisateur (suppr, copie, renommer...)

Valeurs attendues de l'argument:

- "ALL" ou "all" tous les évènements seront tracés.
- "OnUsingComponent" ou "OnU" les accès aux propriétés et méthodes du composant seront tracés.
- "OnAllCommunication" ou "OnA" toutes les communications seront tracées.
- "OnCommunicationError" ou "OnC" seules les communications en erreur seront tracées.
- "OFF" ou "off" aucun évènement ne sera tracé.



3.5. Les méthodes

3.5.1. Méthodes d'accès 'standard' au driver de communication

Ces méthodes permettent à l'utilisateur de gérer un ensemble de modules selon leurs seules adresses physiques en faisant fi de toute connaissance topologique du réseau.

Les arguments entre [] sont optionnels.

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

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

Résumé :

Ajoute ou remplace un module d'adresse donnée à l'élément de configuration du réseau.

Lorsque le réseau est activé (propriété 'isOnLine' à true) le module est installé (tentative de connexion).

Lorsque le module est déjà présent dans la liste, il en est d'abord supprimé de manière identique à l'utilisation de la méthode ReleaseModule().

Un maximum de 256 modules est configurable/installable.

Paramètres :

address: est l'adresse du module spécifié.

configModule: est la configuration désirée (path).

Retourne :

error number:

- 0 si ok .

- Default.CONFIG_MODULE_UNEXPECTED si l'argument configModule est erroné.

- Default.MODULE_IS_NOT_DETECTED si le module n'est pas détecté.

- Default.MODULE_IS_NOT_INSTALLED si la liste de configuration est saturée (maximum de 256 éléments).

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

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

Résumé :

Retrouve la configuration en cours d'un module d'adresse donnée.

Il s'agit de la configuration programmée par la méthode 'SetModule()'.

Paramètres :

address: est l'adresse du module spécifié.

rep: est la configuration relue (path).

errorEventEnabled: false pour localement inhiber la procédure OnErrorOnUseMethod.

Retourne :

le compte-rendu d'erreur (0 si ok)

Exemple:

```
rep = "COM"
```



public int GetModuleInstalled(int address, ref string rep [, bool errorEventEnabled])
 Membre de [MidilIngenierie.Com.MiCom](#)

Résumé :

Retrouve les arguments d'installation d'un module d'adresse donnée.
 Il s'agit des arguments path, baudrate, protocole et type réellement utilisés pour établir la connexion.

Paramètres :

address: est l'adresse du module spécifié.
 rep: est la configuration relue (path baudrate protocole type).
 errorEventEnabled: false pour localement inhiber la procédure OnErrorOnUseMethod.

Retourne :

le compte-rendu d'erreur (0 si ok)

Exemple:

rep = "COM3 38400 XON DMAC23"

public int ReleaseModule(int address)
 Membre de [MidilIngenierie.Com.MiCom](#)

Résumé :

Suppression d'un module d'adresse donnée de l'élément de configuration du réseau.
 Le module est déconnecté.

Paramètres :

address: est l'adresse du module spécifié.

Retourne :

toujours 0 (ok)

public int ReleaseAllModules()
 Membre de [MidilIngenierie.Com.MiCom](#)

Résumé :

Suppression de l'ensemble des modules de la configuration du réseau.
 Les modules sont déconnectés.

Retourne :

toujours 0 (ok)

public int SetOnLine()
 Membre de [MidilIngenierie.Com.MiCom](#)

Résumé :

Permet d'activer la connexion réseau.
 Cette activation est nécessaire en début d'application avant toute tentative de communication.
 Tous les modules préalablement installés sont désinstallés.
 Tous les modules appartenant à l'élément de configuration du réseau seront installés (avec tentative de connexion)
 La propriété 'isOnLine' est mise à la valeur true.
 Lorsque la propriété 'loadConfigOnSetOnLine' est activée, le fichier de configuration est préalablement chargé.

Retourne :

le compte-rendu d'erreur (0 si ok)



public int SetOffLine()

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

Résumé :

Permet de désactiver l'ensemble des connexions réseau.

L'élément de configuration du réseau est conservé.

Ainsi tous les modules installés préalablement sont désinstallés (déconnectés) et pourront être réinstallés (reconnectés) à la prochaine utilisation de la méthode SetOnLine().

La propriété 'isOnLine' est mise à la valeur false.

Retourne :

toujours 0

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

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

Résumé :

Assure une séquence de dialogue (commande/réponse) avec le module d'adresse donnée.

L'argument 'cmd' contient la commande interprétable par le module.

L'argument 'rep' retourne la réponse du module lorsque celle-ci existe.

L'argument 'status' retourne des informations d'état du module lorsque la commande est correctement interprétée.

L'argument 'errorEventEnabled' permet localement d'autoriser/inhiber la procédure OnErrorOnUseMethod.

(se référer au manuel utilisateur du module pour le détail des commandes/réponses interprétables).

(se référer à la note d'application MI_v0_AN03_fr.pdf pour le détail des informations du status).

Paramètres :

address: est l'adresse du module spécifié.

cmd: est la commande destinée au module.

rep: est la réponse du module lorsque celle-ci existe.

status: est le status du module après interprétation de la commande.

errorEventEnabled: false pour localement inhiber la procédure OnErrorOnUseMethod.

Retourne :

le compte-rendu d'erreur (0 si ok)

Remarques:

- lorsque l'argument *adresse* a la valeur -1, le message de commande est véhiculé sans notion d'adresse (**BroadcastMessage**) et sans souci de résultat sur l'ensemble des canaux de communication ouverts. (il est, dans ce cas, préférable d'utiliser la méthode **DialogueAllModules()**)

- lorsque l'argument *adresse* a une valeur supérieure à 65635 (0FFFF en hexa) la parité des poids forts indique l'utilisation locale d'un protocole spécifique, à savoir :

> adresse de 10000h à 1FFFFh pour l'utilisation automatique du protocole (syntaxe) Mac

> adresse de 20000h à 2FFFFh pour l'utilisation automatique du protocole Console

> adresse de 30000h à 3FFFFh pour l'utilisation automatique du protocole sécateur F30xx

public int DialogueAllModules(string cmd)

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

Résumé :

Permet l'émission d'une commande globale sur l'ensemble des modules déjà détectés.

Cette méthode permet, notamment, de mettre en oeuvre les commandes de type synchronisation des mouvements, mouvements directs avec la commande "SYNC TOP" et mouvements interpolés avec la commande "SYNC INTERPOL".

ATTENTION: la synchronisation en temps réelle ne peut avoir lieu que sur un canal de communication commun !

**Paramètres :**

cmd: est la commande destinée à l'ensemble des modules déjà détectés.

Retourne :

toujours 0

public int LoadConfig()

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

Résumé :

Assure le chargement de la configuration réseau à partir du fichier de configuration.

Le path du fichier est programmé dans la propriété 'configFileName'.

Tous les modules préalablement installés sont désinstallés.

Retourne :

0 si le chargement est totalement réalisé.

Si le fichier est erroné, la propriété 'configFileReport' contient le numéro de la rubrique erronée.

Remarques:

- après le chargement de la configuration les modules relevant de cette configuration ne seront installés qu'à la faveur de l'utilisation de la méthode **SetOnLine()** qui permet d'amorcer la communication.

- le chargement de configuration peut également résulter d'une écriture de la valeur **true** dans la propriété **loadConfigFromConfigFile** ou encore être réalisé de manière automatique à la faveur de l'utilisation de la méthode **SetOnLine()** associé à l'utilisation de la valeur **true** de la propriété **loadConfigOnSetOnLine**

public int SaveConfig()

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

Résumé :

Assure la sauvegarde de la configuration réseau dans le fichier de sauvegarde de la configuration.

Le path du fichier est programmé dans la propriété 'configSaveFileName'.

Retourne :

0 si la sauvegarde est totalement réalisée.

Remarques:

- la sauvegarde de configuration peut également résulter d'une écriture de la valeur **true** dans la propriété **saveConfigToConfigFile**.

public **string** GetErrorString(**int** num_err)

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

Résumé :

Obtient un message explicite correspondant à un numéro d'erreur spécifié.

Paramètres :

num_err: est le numéro d'erreur spécifié.

Retourne :

le message explicite.

Exemple:

"défaut de timeout en réception"

public **string**[] GetLastErrorInfos()

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



Résumé :

Obtient, sous forme d'un tableau de chaînes de caractères, les informations concernant la dernière erreur relevée.

Retourne :

- 1ère information: le numéro de l'erreur sous forme d'entier signé.
- 2ème information: un message explicitant le type d'erreur.
- 3ème information: le nom de la méthode ayant générée l'erreur.
- 4ème information: l'adresse du module concerné.
- 5ème information: l'argument principal de la méthode.

public [int](#) StartVisualComponentsRefresh()

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

Résumé :

Démarre le processus de rafraichissement cyclique de l'ensemble des composants visuels inscrits au service de rafraichissement.
Le processus n'est pas démarré lorsqu'aucun composant n'est inscrit.

Retourne :

Le nombre de composants inscrits.

public [int](#) StopVisualComponentsRefresh()

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

Résumé :

Arrête le processus de rafraichissement cyclique de l'ensemble des composants visuels inscrits au service de rafraichissement.

Retourne :

Le nombre de composants inscrits.

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

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

Résumé :

Inscrit un composant visuel au service de rafraichissement cyclique.

Paramètres :

process: est le processus délégué exécuté au rythme de rafraichissement précisé par la propriété *visualComponentsRefreshPeriod*.

Retourne :

Le nombre de composants inscrits.

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

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

Résumé :

Supprime l'inscription d'un composant visuel au service de rafraichissement cyclique.

Paramètres :

process: est le processus délégué concerné.



Retourne :

Le nombre de composants inscrits.

Remarques:

- lorsque **process** n'est pas précisé, l'ensemble des inscriptions est supprimé et le processus de rafraichissement est arrêté.

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

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

Résumé :

Télécharge vers les modules le fichier de nom spécifié.

Le fichier est exploré pour analyser les éventuelles directives de téléchargement.

Les lignes de commandes sont transmises aux modules selon l'adresse spécifiée sur la ligne de commande.

Paramètres :

fileName: est le chemin/nom du fichier à télécharger.

errorEventEnabled: false pour localement inhiber la procédure OnErrorOnUseMethod.

Retourne :

0 si ok

public int StopUploadFile()

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

Résumé :

Arrêt forcé du téléchargement de fichier.

ATTENTION: l'arrêt forcé doit être utilisé dans un thread différent de celui du téléchargement pour avoir un effet.

Retourne :

toujours 0



Installation d'un module:

- le module est recherché par scutation selon l'argument **path** précisé ou non et selon les éléments des propriétés **baudratesEnabled** et **protocolesEnabled**.
- Un module correctement installé sera ajouté à la propriété **modulesInstalled** donnant la liste des modules installés. (une installation incorrecte provient essentiellement du fait que le module est physiquement absent)
- [**path**] est l'arguments[optionnel] précisant la nom spécifique ou générique du canal à emprunter. (exemples de canaux spécifiques: COM1 CAN1 , exemples de canaux génériques: Com Can Usb Eth)
- lorsque l'argument **path** d'un module est précisé, le module est installé dans la mesure où le path est en cohérence avec les éléments des propriétés **portsEnabled** et **portsDisabled**.
- lorsque l'argument **path** d'un module n'est pas précisé, le module est recherché parmi l'ensemble des paths en cohérence avec les éléments des propriétés **portsEnabled** et **portsDisabled**.
- lorsque l'argument **path** d'un module est le path d'un module déjà installé (par ex en RS485), les arguments de configuration de ligne **baudrate** et **protocole** seront ceux utilisés pour le module précédemment installé, autrement dit, un canal multi-modules ne supporte que des modules répondant au même baudrate et protocole.
- la recherche s'arrête au 1er module trouvé répondant à l'adresse recherchée.

Service de rafraichissement des composants visuels:

- avant de démarrer le processus de rafraichissement l'application doit **inscrire** les composants visuels désirés au service de rafraichissement .
- Pour cela l'applicatif inscrit, pour chaque composant, le délégué correspondant à la méthode à exécuter à chaque période de rafraichissement à l'aide de la méthode **AddVisualComponentsRefresh()**.
- le délégué respecte le prototype **public void DelegateVisualComponentsRefresh()**
- la propriété **visualComponentsRefreshPeriode** est garnie avec la valeur désirée en 1/10s
- le processus est démarré à l'aide de la méthode **StartVisualComponentsRefresh()** .
- **Attention:** chaque délégué est exécuté au rythme de la période de rafraichissement, c.a.d que l'ensemble des rafraichissements des composants inscrits est fait pendant cette période. Il est de la responsabilité de l'utilisateur de configurer une période raisonnable selon le nombre de délégués inscrits pour ne pas pénaliser le temps de réaction de l'applicatif. (par exemple, une période à 100ms avec 10 délégués inscrits conduit à 1 une opération de rafraichissement toutes les 10ms (100ms /10)



3.5.2. Les méthodes d'accès "expert" au driver de communication

Ces méthodes donnent à l'utilisateur un accès direct au driver. Leur utilisation demande une totale maîtrise de la topologie du réseau pour savoir associer les arguments path , baudrate, protocole et adresse du module.

```
public int Open(string path, string baudrate, string protocol, string prefix)
```

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

Résumé :

Méthode de type expert permettant de l'ouverture d'un canal de communication.

Paramètres :

path: est le canal de communication à ouvrir.

baudrate: est le baudrate de communication pour une liaison de type RS485.

protocol: est le protocole de communication pour une liaison de type RS485.

prefix: est un argument destiné à gérer des commandes spécifiques au canal de communication et/ou à préfixer l'argument 'cmd' de la méthode Dialogue().

Retourne :

le handler dédié au canal de communication ouvert (0 ou négatif si nok)

```
public int Dialogue(int hCom, string cmd, ref string rep, ref int status)
```

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

Résumé :

Méthode de type expert qui assure une séquence de dialogue (commande/réponse) sur un canal de communication.

L'argument 'hCom' est le handler dédié au canal de communication ouvert selon la méthode 'Open()'.

L'argument 'cmd' contient une commande interprétable par des modules.

La commande doit être précédée de l'adresse du module de destination.

Une commande sans adresse est destinée à l'ensemble des modules présents sur le canal de communication.

L'argument 'rep' retourne la réponse du module adressé lorsque celle-ci existe.

L'argument 'status' retourne des informations d'état du module lorsque la commande est correctement interprétée.

(se référer au manuel utilisateur du module pour le détail des commandes/réponses interprétables).

(se référer à la note d'application MI_v0_AN03_fr.pdf pour le détail des informations du status).

Paramètres :

hCom: est le handler dédié au canal de communication.

cmd: est la commande à transmettre sur le canal de communication. Lorsque celui-ci est défini, l'argument 'prefix' de la méthode Open() précède automatiquement la commande.

rep: est la réponse du module adressé lorsque celle-ci existe.

status: est le status du module après interprétation de la commande.

Retourne :

le compte-rendu d'erreur (0 si ok)

```
public int Close(int hCom)
```

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

Résumé :

Méthode de type expert qui permet la fermeture d'un canal de communication.

L'argument 'hCom' est le handler dédié au canal de communication ouvert selon la méthode 'Open()'.

Paramètres :

hCom: est le handler dédié au canal de communication.

**Retourne :**

le compte-rendu d'erreur (0 si ok)

3.5.3. Le préfixe de commande dans les méthodes expert

C'est une chaîne de caractères **précédée et suivie** du caractère **/**.

Le préfixe, lorsqu'il est utilisé, est directement placé devant la commande à transmettre au module dans l'argument **cmd** de la méthode **Dialogue()**.

Le préfixe peut également être programmé dans l'argument **prefix** de la méthode **Open()**. Dans ce cas il est automatiquement intégré à l'argument **cmd** à chaque utilisation de la méthode **Dialogue()**.

Exemple :

Dialogue(adresse, **"/Mac23/**Move_to 123456", ref rep, ref status);

Le préfixe permet de configurer certains paramètres du canal de communication ou d'informer le driver de l'utilisation, par exemple, d'un protocole spécifique.

/Mac23/

Les commandes seront automatiquement transférées au module selon la syntaxe utilisée par les modules de type **Mac23**.

/Mac34/

Les commandes seront automatiquement transférées au module selon la syntaxe utilisée par les modules de type **Mac34**.



3.6. Les évènements

Les évènements suivants permettent à l'utilisateur de gérer les erreurs de communication notamment celles générées lors de l'accès en écriture aux propriétés du composant.

3.6.1. Evènement sur écriture d'une propriété

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

Résumé :

Evenement déclenché sur erreur lors de l'accès en écriture de propriété.

public string Ev1 { set; get; }

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

Résumé :

Chaine de caractères donnant le nom de la propriété concernée.

public string Ev2 { set; get; }

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

Résumé :

Chaine de caractères donnant des informations sur le constat d'erreur.

public string Ev3 { set; get; }

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

Résumé :

Chaine de caractères donnant des conseils pour éviter l'erreur.

public int Ev4 { set; get; }

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

Résumé :

Numéro représentatif de la propriété concernée.

public int Ev5 { set; get; }

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

Résumé :

Toujours nul.



3.6.2. Evènement sur utilisation d'une méthode

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

Résumé :

Evenement déclenché sur erreur lors de l'utilisation d'une méthode.

public string Ev1 { set; get; }

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

Résumé :

Chaine de caractères donnant le nom de la méthode concernée.

public string Ev2 { set; get; }

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

Résumé :

Chaine de caractères donnant des informations sur le constat d'erreur.

public string Ev3 { set; get; }

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

Résumé :

Chaine de caractères donnant l'argument principal de la méthode.

public int Ev4 { set; get; }

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

Résumé :

Numéro représentatif du type d'erreur.

public int Ev5 { set; get; }

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

Résumé :

Argument adresse de la méthode (-1 si l'argument adresse n'existe pas ou si adressage en broadcast).



4. Les composants d'axe

4.1. Généralités

Chaque composant d'axe est le reflet d'un type d'axe Midi-ingénierie, c.a.d que l'ensemble des propriétés et des méthodes exposées par le composant reflète l'ensemble des variables et des commandes du module telles que décrites dans le **manuel utilisateur** du type d'axe considéré disponible sur le site <http://www.midi-ingenierie.com/>.

classe composant

types d'axe géré

DMac17
DMac23
DMac34
BMac
BMac_H

DMAC17
DMAC23-1 DMAC23-2
DMAC34-1 DMAC34-2
BMAC
BMAC_H

4.2. Propriétés inhérentes à la gestion de composant

Ce sont des propriétés spécifiques à la gestion propre du composant dotNet et qui n'ont pas, par conséquent, de correspondance en terme de variable d'axe.

```
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

Obtient ou spécifie la référence du composant de communication responsable du routage des commandes/réponses entre le composant d'axe et le module physique.

```
public int Address { set; get; }  
Membre de MidiIngenierie.Axe.MIAxe
```

Résumé :

Obtient ou définit la valeur de l'adresse attribuée à l'axe considéré
Chaque axe doit avoir une adresse UNIQUE

La propriété _Dialog doit avoir été spécifiée au préalable

```
public bool AxeOnLine { get; }  
Membre de MidiIngenierie.Axe.MIAxe
```

Résumé :

Obtient l'information comme quoi l'axe concerné a bien été physiquement localisé par la couche Dialogue

Le composant de communication est référencé par la propriété _Dialog.



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

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

Résumé :

Précise le type du module

Le composant modélise une gamme de module MIDI-INGENIERIE.

La propriété Type permet de préciser le type dans sa gamme

Cette propriété est plutôt orientée Design

```
public string Type_real { get; }
```

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

Résumé :

Récupère l'information Type du module détecté par la couche Dialog

Obtient, pour un axe détecté (AxeOnLine True), le type réel du module.

Attention: une divergence avec la propriété **Type[design]** peut entrainer des problèmes de fonctionnement.

```
public string FileName { get; }
```

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

*Obtient le chemin et nom complet du fichier assembly **MidilIngenierie.Axe.dll** utilisé.*

```
public string Version { get; }
```

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

*Obtient le code version de l'assembly **MidilIngenierie.Axe.dll** utilisé.*

```
public string Identity { get; }
```

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

Résumé :

Obtient l'identification complète de l'axe considéré

L'axe doit être accessible en dialogue , ce qui est impossible en mode design

En design , si l'axe est décrit dans une configuration pré_existante son identification sera retrouvée

Obtient, pour un axe détecté (AxeOnLine True), l'identification complète du module.

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

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

Résumé :

Permet de définir un Label associé au composant concerné

Obtient ou spécifie une information qui identifie l'axe dans le contexte applicatif.

```
public string Path { get; }
```

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

Résumé :

path property

Obtient, pour un axe détecté (AxeOnLine True), le canal de communication tel qu'établit par le composant de communication.



4.3. Les propriétés

Ce sont les propriétés exposées par le composant qui reflètent l'ensemble des variables du module et certaines commandes telles que décrites dans le **manuel utilisateur** du type d'axe considéré disponible sur le site <http://www.midi-ingenierie.com/>.

Exemples:

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

Résumé :

Obtient ou définit la valeur de la variable #HIGH_SPEED

c'est une valeur exprimée en 100ème tour/min comprise entre 0 et 400 000

rappel :

la variable #HIGH_SPEED permet de définir la vitesse de consigne de tous les déplacements en mode position

ainsi que la vitesse maximale des déplacements en mode vitesse

variable correspondante: #HIGH_SPEED ou #HSP paramètre high speed (10^2 tour/mn)

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

Résumé :

Obtient ou définit la valeur de la variable #LOW_SPEED

c'est une valeur exprimée en 100ème tour/min comprise entre 0 et 400 000

rappel :

la variable #LOW_SPEED permet de définir la vitesse d'approche utilisée en fin de déplacement pour la gestion du positionnement en mode position

variable correspondante: #LOW_SPEED ou #LSP paramètre low speed (10^2 tour/mn)

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

Résumé :

Obtient la valeur de la variable #ERROR

c'est une valeur dont les bits représentent l'état du module

rappel:

chaque famille de module expose une collection de bits d'erreur : réf documentation

variable correspondante: #ERROR ou #ERR valeur binaire

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

Résumé :

moveTo

commande correspondante: MOVE_TO ou MTO paramètre position (10^4 tour)

On retrouve, dans le manuel utilisateur, la description de chacune des variables ou commandes avec ses unités et valeurs aux limites.



4.4. Les méthodes

Ce sont les méthodes exposées par le composant qui reflètent certaines commandes du module telles que décrites dans le **manuel utilisateur** du type d'axe considéré disponible sur le site <http://www.midi-ingenierie.com/>.

Exemples:

```
public int Move_Interpol(int moveInterpolArg)
```

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

Résumé :

Provoque l'émission de la commande MOVE_INPERPOL moveinterpolArg pour le module concerné. En mode interpolé, chaque commande est mémorisée dans la FIFO d'interpolation. Les mouvements seront lancés par la commande SYNCHRO INTERPOL.

Paramètres :

moveInterpolArg: Déplacement relatif à effectuer (en incréments moteur) pendant le temps #INTERPOL_TIME.

Retourne :

compte rendu positionné par la couche dialogue

commande correspondante: MOVE_INTERPOL ou MIN paramètre de déplacement (10⁻⁴tour)

```
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

commande correspondante: STOP ou STO

On retrouve, dans le manuel utilisateur, la description de chacune des commandes avec ses arguments, unités et valeurs aux limites.



5. Annexe A – Répertoire des défauts du composant de communication

5.1. Annexe A1 – Défauts de communication

-5	str_ON_PARAM_BAUD	Impossible d'ouvrir le port de communication - L'argument 'baudrate' est erroné.
-4	str_ALREADY_USED	Impossible d'ouvrir le port de communication - Le port est déjà utilisé par une autre application.
-3	str_ALREADY_OPENED	Le port de communication est déjà ouvert dans l'application.
-2	str_ON_OPEN	Impossible d'ouvrir le port de communication - Le port selon l'argument path donné n'existe pas ...
-1	str_ON_PATH	Impossible d'ouvrir le port de communication - L'argument path (chemin) est erroné.
31	str_NOT_OPENED	Défaut de communication - Le port n'a pas été ouvert.
32	str_ON_HANDLE	Défaut de communication - Le handle fourni est hors limite.
33	str_MESSAGE_TO_LONG	Défaut de communication - Le message en émission est trop long.
34	str_MESSAGE_EMPTY	Défaut de communication - Le message en émission est vide.
35	str_IO_EXCEPTION	Défaut de communication - Un 'IO exception' du module de communication est survenu.
90	str_DEVICE_DISCONNECTED	Défaut de communication - Le module est (ou a été) déconnecté.
91	str_VCP_DISCONNECTED	Défaut de communication - Le port de communication virtuel (VCP) est déconnecté.
100	str_EMISS_IMPOSSIBLE	Défaut de communication - L'opération d'émission est impossible.
101	str_TO_EMISS	Défaut de communication - Timeout en émission.
120	str_ON_EMISS	Défaut de communication - Défaut de l'opération d'émission.
200	str_REC_IMPOSSIBLE	Défaut de communication - L'opération de réception est impossible.
201	str_TO_REC_ACK	Défaut de communication - Timeout en réception (ACK).
202	str_TO_REC_ETX	Défaut de communication - Timeout en réception (ETX).
203	str_TO_REC	Défaut de communication - Timeout en réception.
210	str_ON_REC_ACK	Défaut de communication - Défaut sur le protocole d'acquittement, le caractère 'ACK' n'est pas reçu.
211	str_ON_REC_XOFF	Défaut de communication - Défaut sur le protocole d'acquittement, le caractère 'XOFF' n'est pas reçu.
212	str_ON_REC_XERR	Défaut de communication - Le module n'a pas correctement interprété la commande, réception du caractère 'XERR'.
213	str_ON_REC_NAK	Défaut de communication - Défaut sur le protocole d'acquittement, réception du caractère 'NAK'.
214	str_ON_REC_ETX	Défaut de communication - Défaut sur le protocole de réponse, le caractère 'ETX' n'est pas reçu.
220	str_ON_REC	Défaut de communication - Défaut de l'opération de réception.
230	str_ON_REC_LONG	Défaut de communication - La longueur du message reçu est incohérente.
231	str_ON_REC_CHKS	Défaut de communication - Le checksum du message reçu est incohérent.
1000	str_CONFIG_MODULE_UNEXPECTED	Défaut d'utilisation de la méthode - Configuration de module erronée.
1001	str_MODULE_IS_NOT_CONFIGURED	Défaut d'utilisation de la méthode - Le module n'est pas configuré.
1002	str_MODULE_IS_NOT_INSTALLED	Défaut d'utilisation de la méthode - Le module n'est pas installé.
1003	str_MODULE_IS_NOT_DETECTED	Défaut d'utilisation de la méthode - Le module n'est pas détecté.
1004	str_ADRESSE_MODULE_UNEXPECTED	Défaut d'utilisation de la méthode - L'argument 'adresse' est erroné.
1005	str_UNEXPECTED_HANDLE	Défaut d'utilisation de la méthode - Tentative d'utilisation d'un type de driver inconnu.
1006	str_CONFIG_FILE_IS_NOT_DEFINE	Défaut d'utilisation de la méthode - Le nom du fichier de configuration n'est pas défini.
1007	str_CONFIG_FILE_IS_NOT_EXISTING	Défaut d'utilisation de la méthode - Le fichier de configuration n'existe pas.
1008	str_CONFIG_FILE_IS_ON_ERROR	Défaut d'utilisation de la méthode - Le fichier de configuration est erroné.
1009	str_SAVE_CONFIG_FILE_IS_NOT_DEFINE	Défaut d'utilisation de la méthode - Le fichier de sauvegarde de la configuration n'existe pas.
1010	str_SAVE_CONFIG_FILE_IS_NOT_CREATED	Défaut d'utilisation de la méthode - Le fichier de sauvegarde de la configuration ne peut pas être créé.
1011	str_FILE_IS_NOT_DEFINE	Défaut d'utilisation de la méthode - Le nom du fichier n'est pas défini.
1012	str_FILE_IS_NOT_EXISTING	Défaut d'utilisation de la méthode - Le fichier n'existe pas.
1013	str_FILE_IS_ON_ERROR	Défaut d'utilisation de la méthode - Le fichier est erroné.
	str_DEFAULT	Défaut non répertorié

5.2. Annexe A2 - Défauts en écriture de propriété

Propriétés	numéro représentatif	
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 - Utilisation de protocoles spécifiques

6.1. Le protocole de gestion des modules de type Mac

Les modules de type Mac ont une syntaxe de langage totalement différente de celle des modules actuels de type **xMac** (µMac, DMac, BMac et RMac)

Le composant de communication peut se charger de la conversion des langages aussi bien en transmission qu'en réception. Grâce à ce convertisseur, les modules de type Mac peuvent être gérés de manière totalement transparente selon la syntaxe du langage **DMac** avec toutefois les limitations intrinsèques aux capacités propres aux modules Mac. Pour faciliter d'éventuelles migrations de programmes la syntaxe de langage de type **Simpa** disponible sur le driver Midi-ingénierie d'ancienne génération est également conservée.

L'utilisateur aura donc la possibilité d'un triple accès au module Mac :

- en langage **expert** c.a.d utilisation du langage directement interprétable par le module sans intervention du convertisseur.

Notice pour le contrôle MAC23 V24 et MAC34 V24 en mode expert

ref : **mac23_34_ex_v5_um_fr.pdf**

- en langage de syntaxe type **Simpa** avec utilisation du convertisseur.

Manuel d'utilisation du Module Mac23

ref : **mac23_v9_um_fr.pdf**

- en langage de syntaxe type **DMac** avec utilisation du convertisseur.

Notice ci-jointe

6.1.1. Utilisation locale du convertisseur Mac :

Le convertisseur est activé pour la seule communication en cours.

- par utilisation de la méthode expert **Dialogue(hCom,adr/cmd,rep,status)**

Dans ce cas l'argument **cmd** de la méthode doit être préfixé par :

- /Mac23/ pour les modules de type Mac23
- /Mac34/ pour les modules de type Mac34

Exemple d'accès au module Mac d'adresse 5 :

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

- par utilisation de la méthode **DialogueModule(adr,cmd,rep,...)**

Dans ce cas l'argument **adr** sera utilisé avec l'offset **10000h**

Exemple d'accès au module Mac d'adresse 5 :

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

6.1.2. Utilisation globale du convertisseur Mac :

Le convertisseur est automatiquement activé lors de la recherche d'un module.

- par sélection du type Mac dans la propriété **typesModuleEnabled**

Dans ce cas au moins un des items suivants doit faire partie de la liste de la propriété :

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



Exemple d'accès au module Mac d'adresse 5 :

typesModuleEnabled = Array["xMac", "Mac"]

DialogueModule(5, "Move_to 12345", ref rep,)

(sans offset sur l'argument **adr**)

6.1.3. Le convertisseur de commandes pour modules Mac :

Commandes :

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)

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

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



hal(t)	gs	@ @180000000000
mt0 param (move_to)	ga param	@ @1300PPPPPPPP
mt0 0 (move_to)	gh	@ @140000000000
msh param (move_speed)	gf param	@ @150000SSVVVV
mon param1 param2 (move_on)	gt param1 param2	@ @1600DDDDVVVV

Questions:

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

qa

Mac

@ @29xx00000000

@ @2Axx00000000

@ @240000000000

@ @200000000000

@ @210000000000

@ @250000000000

@ @250000000000

@ @260000000000

@ @260000000000

@ @220000000000

@ @230000000000

@ @270000000000

@ @270000000000

@ @270000000000

@ @270000000000

@ @270000000000



6.2. Le protocole Console

C'est le protocole Console utilisable par certain types de modules Midi ingénierie.

Il est décrit dans le document :

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

ref : *mi_v0_an03_fr.pdf*

6.2.1. Utilisation locale du protocole console:

- par utilisation des méthodes expert **Open("path","baudrate","CON")**, **Dialogue()** et **Close()**

Dans ce cas le protocole devient actif pour le path utilisé

L'argument **protocole** de la méthode **Open()** doit être de la forme :

- **"Console"** seuls les 3 premiers caractères sont significatifs

Exemple d'accès au module d'adresse 5 en protocole console:

```
Open("COM3", " ", "console")
Dialogue(hcom, "05Move_to 123456", ref rep, ref status)
Close(hcom)
```

- par utilisation de la méthode **DialogueModule(adr,cmd,rep,...)**

Dans ce cas l'argument **adr** sera utilisé avec l'offset **20000h** et **le protocole est activé pour le path de communication intrinsèquement lié au module**.

Exemple d'accès au module d'adresse 5 en protocole console:

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

6.2.2. Utilisation globale du protocole console :

Le protocole est automatiquement activé lors de la recherche d'un module.

- par sélection du protocole Console dans la propriété **protocolesEnabled**

Dans ce cas l'item suivant doit faire partie de la liste de la propriété :

- **"CON"**

Exemple d'accès au module d'adresse 5 en protocole console:

```
protocolesEnabled = Array["Xon","Con"]
DialogueModule(5, "Move_to 12345", ref rep, ....) (sans offset sur l'argument adr)
```



6.3. Le protocole de gestion des sécateurs de type F30xx

C'est un protocole compatible avec le protocole Console avec cependant une syntaxe spécifique aux sécateurs Infaco de type F30xx.

Dans ce type de liaison, **charge à l'utilisateur de maîtriser la syntaxe sécateur**, aucun convertisseur n'est fourni.

Le protocole et la syntaxe sont décrits dans le document :

Supplément au manuel de l'ensemble Sécateur F3010

ref :

Seule une utilisation locale est possible.

6.3.1. Utilisation locale du protocole F30xx:

- par utilisation des méthodes expert **Open("path","38400","CON")**, **Dialogue()** et **Close()**

Dans ce cas le protocole devient actif pour le path utilisé

Les arguments **baudrate** et **protocole** de la méthode **Open()** doivent être de la forme :

- **"38400"** pour l'argument **baudrate**
- **"Console"** pour l'argument **protocole**, seuls les 3 premiers caractères sont significatifs

Exemple d'accès au module sécateur :

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

- par utilisation de la méthode **DialogueModule(adr,cmd,rep,...)**

Dans ce cas l'argument **adr** sera utilisé avec l'offset **30000h** et **le protocole est activé pour le path de communication intrinsèquement lié au module sécateur.**

Exemple d'accès au module sécateur d'adresse 5 :

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

Remarque : le sécateur est un module qui ne gère pas d'adresse, en conséquence il doit être branché seul sur un canal de communication. L'adresse précisée n'a d'intérêt qu'au sens du composant de communication.