INFRANOR

Gem Drive Studio

Guide de mise en œuvre du serveur de communication

Indice de révision : **1.2**

INFRANOR

Sommaire

1. Introduction

Les logiciels PC développés pour le paramétrage des variateurs Infranor de dernière génération sont conçus sur la base d'une architecture modulaire de type client/serveur.

Un des intérêts de cette architecture est la possibilité de développer des modules clients indépendants utilisant un même serveur.

Le module Serveur de Communication a pour rôle de gérer les accès physiques au bus de terrain en adaptant le protocole de communication aux différents périphériques (CANopen, RS232, ...).

Le présent document a pour objectif de fournir toutes les informations permettant de développer son propre module client en utilisant le module serveur de communication.

2. Configuration minimale

Le module serveur est destiné à être installé sur un PC dont la configuration minimale est la suivante:

- Processeur 1 GHz,
- 512 Mo de mémoire RAM,
- système d'exploitation Windows© XP (Service Pack 2) ou Serveur 2003
- Microsoft .NET Framework V3.0 (ou ultérieur) installé.

3. Installation du module serveur

Le kit d'installation contient:

- L'installateur du serveur.
- Un modèle de client (code source) écrit en VB.NET.
- Une documentation.

Décompresser le contenu du fichier zip et lancer le fichier Setup.exe.
Suivre la procédure pour installer le serveur sur le PC.

Les sources du modèle de client peuvent être édités dans l'environnement Microsoft Visual Studio.
Une version express gratuite est disponible en téléchargement sur le site de Microsoft.

Lors de l'installation du serveur sur le PC, quatre clés seront créées dans la base de registre de Windows:

Une clé contenant le nom de l'exécutable permettant de lancer le serveur

Clé: HKEY_USER
Sous-clé: .Default\Software\DriveServer\Infos
Nom de la clé: DriveServerName

Une clé contenant le chemin d'accès à cet exécutable

Clé: HKEY_USER
Sous-clé: .Default\Software\DriveServer\Infos
Nom de la clé: DriveServerPath

Une clé contenant la version du serveur

Clé: HKEY_USER
Sous-clé: .Default\Software\DriveServer\Infos
Nom de la clé: DriveServerVersion

Une clé contenant le chemin d'accès aux fichiers dictionnaires d'objets

Clé: HKEY_USER
Sous-clé: .Default\Software\DriveServer\Infos
Nom de la clé: DictionariesPath

Les modules clients pourront par la suite utiliser ces clés pour le lancement/arrêt du serveur.

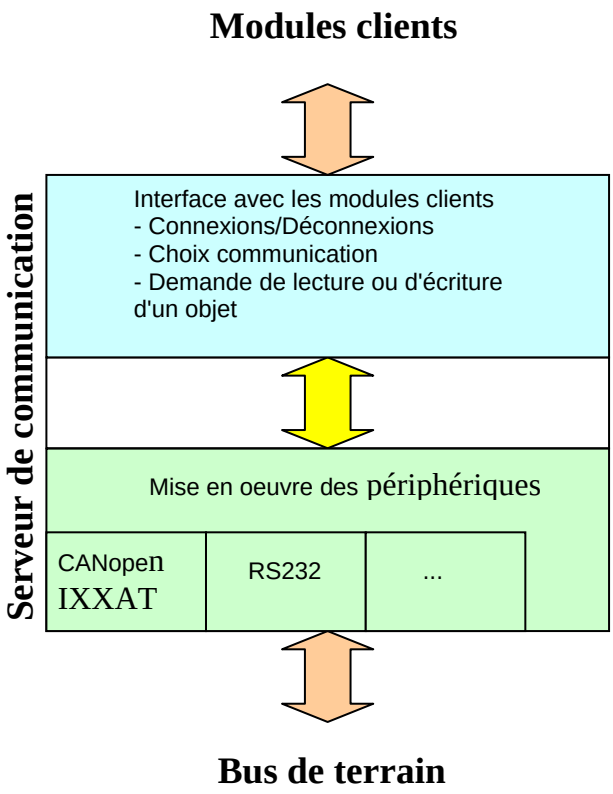
4. Description du serveur

4.1. Architecture

Le serveur est constitué:

- d'une partie dédiée à la communication entre le serveur et les modules clients,

- d'une partie en charge de gérer les périphériques hardware (RS232, CANopen1, CANopen2, ...).



La gestion des périphériques physiques est entièrement assurée par le serveur. De ce fait, tout nouveau périphérique implémenté au niveau du serveur n'entrainera pas d'obligation de recompiler les modules clients.

L'ajout d'un nouveau canal de communication se traduira par le développement d'un nouveau module interne au serveur et fera l'objet d'une nouvelle version de ce dernier, et la compatibilité sera assurée avec les versions précédentes.

Le serveur reste indépendant vis à vis du type des données. Pour chaque paramètre (objet CAN), le serveur lit ou écrit une suite d'octets sur le périphérique sélectionné, et c'est le client appelant qui devra mettre en forme les données en fonction du type de l'objet. L'information de type est donnée par le dictionnaire d'objets correspondant au modèle et à la version de l'appareil.

4.2. Environnement de développement

Le serveur de communication a été développé sous l'environnement .NET de façon à offrir le maximum de compatibilité avec les systèmes actuels.

Les modules clients pourront être écrits en utilisant un des langages .NET (C#, VB.NET, C++, Delphi.NET....).

La communication entre le serveur de communication et les modules clients est basée sur la techonologie WCF ou Windows Communication Foundation. Cette technologie est apparue avec la version 3.0 du Framework .NET.

WCF fournit un modèle de programmation unifiée permettant de construire des applications distribuées.

Cette technologie s'articule autours de 3 éléments importants:

A/ Le service

Un service WCF est une entité logicielle implémentant un contrat. Ces services sont listés dans une interface (au sens objet du terme) qui reprend la liste des opérations exposées par chaque service (contrat de service).

B/ L'adresse

L'adresse est simplement une URL qui définit l'endroit où se trouve le service

C/ Le protocole ou "binding"

C'est la méthode utilisée pour communiquer avec le service WCF. Le Framework .NET 3.0 propose 9 "bindings" différents:

- BasicHttpBinding
- WSHttpBinding
- WSDualHttpBinding
- WSFederationHttpBinding
- NetTcpBinding
- NetNamedPipeBinding
- NetMsmqBinding
- NetPeerTcpBinding
- MsmqIntegrationBinding

4.3. Fichiers requis

4.3.1. Fichiers installés

Concrètement, les services développés dans le serveur de communication sont compilés dans un fichier bibliothèque de type DLL (**DriveService.dll**). et exposés via une interface (**DriveInterface.dll**).

Les services du serveur de communication nécessitent un hôte sur lequel ils pourront être exécutés. C'est le rôle de l'exécutable **DriveHost.exe**.

Pour développer un module client et utiliser les service du serveur, il faut ajouter une référence au fichier DriveInterface.dll.
Le démarrage du serveur se fera en lançant le fichier exécutable DriveHost.exe.

4.3.2. Les fichiers de configuration du serveur et du client

La configuration d'un service Windows Communication Foundation (WCF) avec un fichier de configuration permet de fournir des données de point de terminaison et de comportement du service au moment de l'exécution plutôt qu'à la compilation.

Ces fichiers doivent être placés dans le même répertoire que l'exécutable. Il permettent de définir les points de terminaison des services, ainsi que leur comportement.

Côté client :

Le fichier de configuration (app.config) doit être placé dans le même répertoire que l'exécutable. Voici un exemple de son contenu:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="longTimeoutBinding"
                 receiveTimeout="infinite"
                 sendTimeout="infinite">
          <reliableSession enabled ="true" inactivityTimeout ="infinite"
ordered="false"/>
          <security mode="None"/>
        </binding>
      </netTcpBinding>
    </bindings>
    <client>
      <endpoint
        address="net.tcp://localhost:8018/AllDriveServices"
        binding="netTcpBinding"
        bindingConfiguration="longTimeoutBinding"
        contract="DriveInterface.IDriveService"
        name="NetTcpBinding_IDriveService" />
    </client>
  </system.serviceModel>
</configuration>
```

Côté serveur :

Le fichier de configuration est installé dans le même répertoire que le fichier DriveHost.exe.
Voici son contenu:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="longTimeoutBinding"
                 receiveTimeout="infinite"
                 sendTimeout="infinite">
          <reliableSession enabled ="true" inactivityTimeout ="infinite"
ordered="false"/>
          <security mode="None"/>
        </binding>
      </netTcpBinding>
```

```
    </bindings>
    <services>
      <service name="DriveService.AllDriveServices">
        <endpoint
            address="net.tcp://localhost:8018/AllDriveServices"
            bindingConfiguration="longTimeoutBinding"
            binding="netTcpBinding"
            contract="DriveInterface.IDriveService" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Dans cet exemple, pour utiliser les services du serveur de communication, les clients se connecteront à l'adresse `net.tcp://localhost:8018/AllDriveServices` et dialogueront en utilisant des sockets TCP.

Les fichiers de configuration peuvent être modifiés. Par exemple, il est possible de rajouter des balises <endpoint> avec des adresses différentes et des "binding" différents.

Ainsi, en ajoutant les lignes suivantes dans le fichier App.config, le service sera également disponible en http:

```
<endpoint address="http:// localhost:8018/AllDriveServices "
binding="basicHttpBinding" contract=" DriveInterface.IDriveService "/>
```

5. Liste des fonctions

Fct_AddClientReference

Description

Cette fonction permet de référencer un module client sur le serveur.

Paramètres d'entrée:

- Utilisation des Callbacks (type booléen)

Paramètre de sortie

Aucun

Valeur de retour

- Numéro de client (type Short)

Note: (Ce numéro sera à utiliser par la suite pour tous les échanges avec le serveur

Fct_RemoveClientReference

Description

Cette fonction permet de déréférencer le module client du serveur.

Paramètres d'entrée

- Numéro de client (type Short)

Paramètres de sortie

Aucun

Valeur de retour

- Nombre de clients restant encore référencés sur le serveur (type Byte)

Fct_ReadServerFeatures

Description

Cette fonction permet de connaître la liste des fonctionnalités implantées sur le serveur de communication.

Paramètres d'entrée

Aucun

Paramètres de sortie

Aucun

Valeur de retour

- Mot de 32 bits, chaque bit représente une fonctionnalité supportée (type UInteger)

Le récapitulatif suivant indique la liste des fonctions implantées, pour les différentes versions du serveur. Si le bit correspondant est à 1, alors la fonction est implémentée.

Version 1.0 (première version)
Bit 0: Simulateur basique (uniquement lecture)

Fct_CommConfig

Description

Cette fonction permet de configurer la communication en affichant une fenêtre regroupant les différents types de périphériques disponibles.

Paramètres d'entrée

- Numéro de client (type Short)

Paramètres de sortie

- Type de communication choisi (type SByte) :

    0: Liaison série
    1: CANopen (périphériques IXXAT)

- Nom du périphérique choisi (type String)

- Vitesse de communication (type UInteger)

Valeur de retour

- Etat de la communication (type SByte) :

    0: Périphérique stoppé
    1: Périphérique démarré
    -1: Périphérique en défaut

Sub_ReadCommConfig

Description

Cette fonction permet de lire l'état de la communication. Les informations retournées permettent d'éviter d'appeler la fonction Fct_CommConfig si un périphérique est déjà démarré et que le client veut rester sur le même périphérique.

Paramètres d'entrée

- Aucun

Paramètres de sortie

- Type de communication choisi (type SByte) :
    0: Liaison série
    1: CANopen (périphériques IXXAT)

- Nom du périphérique choisi (type String)

- Vitesse de communication (type UInteger)

- Etat de la communication (type SByte) :

    0: Périphérique stoppé
    1: Périphérique démarré
    -1: Périphérique en défaut

Sub_StopPeripheral

Description

Cette fonction permet d'arrêter le périphérique démarré.

Paramètres d'entrée

- Aucun

Paramètres de sortie

- Aucun

Fct_ReadPort

Description

Cette fonction permet de lire un paramètre (objet CANopen) sur un des appareils connectés au bus de terrain.
Paramètres d'entrée

- Numéro de client (type Short)

- Numéro de nœud (adresse CAN) de l'appareil (type Short)

- Index du paramètre en hexadécimal (type String)

- Sous-index du paramètre en hexadécimal (type string)

- Valeur du "timeout" en secondes (optionnel, valeur = 1 par défaut) (type Short)

Paramètres de sortie

- Tableau d'octets contenant la valeur lue (type Byte)

Valeur de retour

- Etat de la commande (type Integer) :

    - 1: Retour OK
    - 2: Serveur occupé (dans ce cas, réitérer la demande)
    - 3: Retour en erreur
    - Autres valeurs : Code "Abort" (cf CANopen communication Profile DS-301)

Fct_WritePort

Description

Cette fonction permet d'écrire un paramètre (objet CANopen) sur un des appareils connectés au bus de terrain.

Paramètres d'entrée

- Numéro de client (type Short)

- Numéro de nœud (adresse CAN) de l'appareil (type Short)

- Index du paramètre en hexadécimal (type String)

- Sous-index du paramètre en hexadécimal (type string)

- Tableau contenant les octets à écrire (type Byte)

- Valeur du "timeout" en secondes (optionnel, valeur = 1 par défaut) (type Short)

Paramètres de sortie

- Aucun

Valeur de retour (type Integer)

- Etat de la commande :

  - 1: Retour OK
  - 2: Serveur occupé (dans ce cas, réitérer la demande)
  - 3: Retour en erreur
  - Autres valeurs : Code "Abort" (cf CANopen communication Profile DS-301).

6. Développement d'un module client

6.1. Pré-requis

Le module client doit faire référence au fichier interface **DriveInterface.dll** et aux mécanismes de la technologie WCF regroupés dans l'espace de nom **System.ServiceModel**.

Le fichier DriveInterface.dll se trouve sous le répertoire d'installation du serveur.

6.2. Dictionnaires d'objets

Les dictionnaires d'objets permettent de connaître la liste et les caractéristiques des paramètres propres à un appareil. Ces dictionnaires sont gérés en versions et peuvent être modifiés en fonction des évolutions de firmware.
Concrètement, un dictionnaire d'objets est un fichier au format XML qui contient une zone d'en-tête permettant une identification des versions, suivi d'une liste des paramètres et de leur description.

Pour chaque paramètre, le dictionnaire d'objets précise :

- L'index
- Le sous-index
- Le Nom
- Le type (*)
- Le type d'accès (**)
- Les valeurs limites
- La possibilité ou non de "mapper" cet objet dans un message PDO
- Le comportement (modification sous ou hors asservissement, ...)
- La classe (paramètre moteur, régulateur, de communication, d'application, ...)

(*) Valeurs possibles:

- 0x02: Nombre entier signé de 8 bits (Integer8)
- 0x03: Nombre entier signé de 16 bits (Integer16)
- 0x04: Nombre entier signé de 32 bits (Integer32)
- 0x05: Nombre entier non signé de 8 bits (Unsigned8)
- 0x06: Nombre entier non signé de 16 bits (Unsigned16)
- 0x07: Nombre entier non signé de 32 bits (Unsigned32)
- 0x09: Chaîne de caractères (VisibleString)
- 0x0A: Caractère (OctetString)

(**) Valeurs possibles:

- rw: Accessible en lecture ou écriture
- ro: Accessible uniquement en lecture
- wo: Accessible uniquement en écriture

Le dictionnaire contient en fait les informations qui sont normalement contenues dans le fichier EDS (Electronic Data Sheet) disponible pour tout appareil respectant le protocole CANopen. Les

informations contenues dans le dictionnaire d'objets sont plus complètes que celles contenues dans le fichier EDS.

Pour chaque nouvelle version du serveur de communication, le répertoire contenant les dictionnaires d'objets associés aux appareils sera mis à jour.
Ce répertoire se trouve sous le dossier "Dictionaries" du répertoire d'installation du serveur.

Le nom de ce répertoire est donné par une clé inscrite dans la base de registre de Windows lors de l'installation du serveur (cf. Chapitre Installation du module serveur)

6.3. Mise en œuvre du serveur

6.3.1. Création du canal de communication WCF

Pour créer un canal de communication avec le serveur, il est nécessaire d'utiliser l'objet ChannelFactory qui va permettre de générer une classe qui autorisera un client à envoyer et/ou recevoir vers et depuis un service.

Ce canal peut être uniquement défini du client vers le serveur, ou bien dans les deux sens. Dans ce cas, on parlera de "duplex channel". La communication du serveur vers le client est effectuée au moyen de procédures de type "callback".

Le code suivant (exemple en VB .NET) permet de créer ce canal de communication entre le client et le serveur. On peut noter que cet objet est du type de l' interface (le contrat du service).

```
'Normal channel
'Public myChannelFactory As ServiceModel.ChannelFactory(Of
DriveInterface.IDriveService) = Nothing

'Duplex channel (by using the callback procedures)
Public myChannelFactory As ServiceModel.DuplexChannelFactory(Of
DriveInterface.IDriveService) = Nothing

Public myService As DriveInterface.IDriveService

Dim Instance As New InstanceContext(New CbCallbackClass())

myChannelFactory = New DuplexChannelFactory(Of
DriveInterface.IDriveService)(Instance, "NetTcpBinding_IDriveService")

myService = myChannelFactory.CreateChannel()
```

La ligne de code suivante permet de refermer le canal de communication :

```
myChannelFactory.Abort()
```

6.3.2 Lancement du serveur

Le premier client doit lancer le serveur. Pour cela, les actions à effectuer sont les suivantes :

- ⇨ Vérifier si le serveur est installé en contrôlant la présence des clés (DriveServerName et/ou DriveServerPath) dans la base des registres.
- ⇨ Vérifier si le serveur est déjà lancé en vérifiant si son nom (cf. clé DriveServerName) figure dans la liste des processus actifs de Windows.
- ⇨ Lancer le serveur (cf. clé DriverServerPath) s'il ne l'est pas déjà.

Rappel: Le serveur n'est pas une application Windows (application fenêtrée). C'est une DLL qui contient des procédures appelées par les clients. Lancer le serveur consiste en fait à créer un canal de communication (au sens Windows Communication Foundation) entre ce serveur et les modules clients.

Quand le serveur est lancé, une nouvelle icone apparait dans la zone de notification de la barre des tâches de Windows. En cliquant sur cette icone, on fait apparaître un menu qui permet d'arrêter le serveur.

6.3.3. Connexion au serveur

Lorsqu'un client se connecte au serveur, il se référence sur ce dernier et reçoit un numéro de client qui sera par la suite utilisé dans tous les échanges entre le module client et le module serveur.

6.3.4 Choix d'un périphérique de communication

Pour lancer une commande de lecture ou d'écriture sur un des appareils présents sur le bus de terrain, il est nécessaire de choisir un périphérique de communication parmi tous les périphériques disponibles (RS232, CAN,...).

Lorsque le premier client se connecte au serveur (référencement) alors le serveur démarre automatiquement le dernier périphérique utilisé si celui-ci est disponible.

Pour utiliser un périphérique différent, il est possible, depuis le module client d'appeler une procédure du serveur qui va ouvrir une fenêtre de configuration de la communication.
A partir de cette fenêtre, l'utilisateur peut stopper le périphérique en cours, s'il est démarré, puis choisir un autre type de périphérique et lancer son initialisation.

Note: Quand un utilisateur sélectionne un périphérique de communication et appui sur le bouton permettant d'initialiser ce périphérique, celui-ci est immédiatement démarré par le serveur, et s'il y a d'autres modules clients, ces derniers devront utiliser ce même périphérique tant que celui-ci n'aura pas été arrêté. Il est cependant possible de stopper le périphérique en cours depuis n'importe quel client connecté au serveur.

6.3.5. Utilisation

Lorsqu'un module client est connecté au serveur et qu'un périphérique est démarré, il est alors possible d'utiliser les fonctions permettant de lire ou d'écrire des paramètres sur un appareil connecté au bus de terrain (cf. Chapitre Liste des fonctions pour les explications détaillées).

6.3.6. Déconnexion et arrêt du serveur

Quand un client se déconnecte du serveur, il se déréférence de celui-ci et il libère son numéro de client.

Se déconnecter du serveur ne signifie pas l'arrêt du serveur. Cependant, la procédure de déconnexion appelée par le client précise en retour le nombre de clients restant connectés. Si ce nombre est égal à 0, il est recommandé d'arrêter le serveur puisque le processus n'est plus utilisé.

Côté serveur, si un périphérique de communication est démarré, il est automatiquement stoppé après déconnexion du dernier client.

Pour arrêter le serveur, il suffit de stopper le processus dans Windows à partir des clés précédemment décrites.

6.3.7 Les "callbacks"

Les callbacks sont des procédures du client appelées par le serveur.
La mise en œuvre de ces procédures n'est effective que si le paramètre d'entrée de la procédure de référencement sur le serveur (`Fct_AddClientReference`) est positionné à 1.

Attention: Le module client doit obligatoirement contenir la déclaration de ces procédures. Si une de ces procédures n'est pas utilisée, le corps de la procédure pourra être vide, mais elle devra tout de même être déclarée.

La liste des callbacks actuellement implémentée est la suivante:

Sub_ServerNotification

Description : Cette procédure n'est utilisée que dans le cadre de projets Gem Drive Studio. Elle permet d'informer un module client que le projet a été modifié

Paramètres d'entrée

- Type de notification (type Short)

    - 0: La notification indique que le client doit mémoriser sa configuration projet
    - 1: La notification indique que le fichier projet a été modifié. Le nouveau fichier projet est passé en paramètre, ainsi que la nouvelle liste des appareils connectés
    - 2: La notification indique que le module client doit s'arrêter.

- Chemin d'accès au fichier projet (type string).

- Liste des nodeID des appareils connectés séparés par des "#" (type string).

Paramètres de sortie

- Aucun.

Sub_ServerCommandProgress

Description : Dés que la taille d'un transfert (lecture ou écriture) dépasse 50 octets, cette procédure est appelée par le serveur toutes les secondes pour indiquer le nombre d'octets total du transfert et le nombre d'octets restant à transférer.

Paramètres d'entrée

- Nombre d'octets à transférer (type Integer)

- Nombre d'octets restant à transférer (type Integer)

Paramètres de sortie

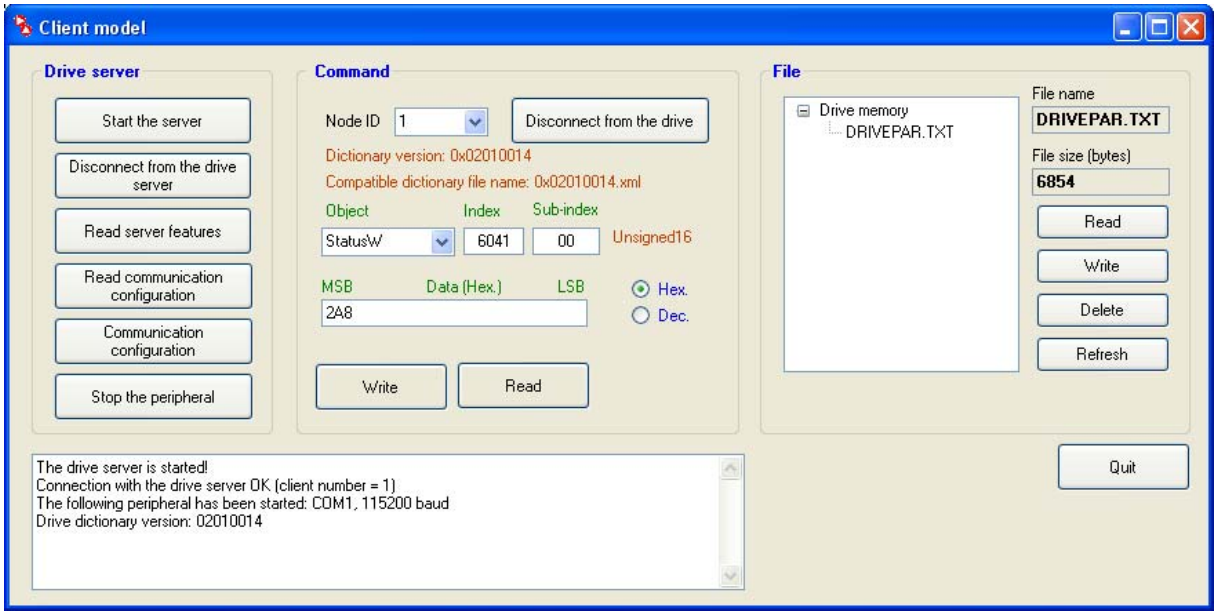
- Aucun.

6.4. Exemple: Un modèle de client

L'installateur du serveur de communication intègre un projet Visual Studio nommé ClientModel. Comme son nom l'indique, ce projet contient les fichiers sources commentés correspondant à un module client type.

Pour compiler ce projet, il est nécessaire de rajouter le fichier **DriveInterface.dll** dans la liste des références. Pour rappel, ce fichier se trouve dans le répertoire d'installation du serveur de communication.

L'interface principale contient plusieurs boutons permettant de mettre en œuvre les différentes possibilités du serveur de communication :



Les différentes actions possibles sont :

A/ Démarrage du serveur

=> Utilisation des clés d'installation du serveur pour lancer/contrôler le processus du serveur

B/ Connexion/déconnexion au serveur

=> Création du canal de communication WCF
=> Référencement du client auprès du serveur (Attribution d'un numéro de client).
=> Déréférencement du client (libération du numéro de client).

C/ Choix du périphérique de communication

=> Affichage de la fenêtre de configuration de la communication.
=> Démarrage/Arrêt du périphérique.

D/ Identification d'un appareil sur le bus

=> Connexion à un appareil en utilisant son adresse.
=> Lecture de la version de son dictionnaire d'objets et association avec un fichier dictionnaire présent dans la librairie.
=> Création d'une liste d'objets CANopen à partir de l'analyse du dictionnaire d'objets.

E/ Lecture/écriture d'un paramètre sur un appareil

=> Lecture d'un objet avec mise en forme en fonction du type
=> Ecriture d'un objet

F/ Mise en œuvre du système de fichiers d'un appareil

=> Lecture de la liste des fichiers présents dans un appareil
=> Lecture/écriture/effacement d'un fichier

Le détail des objets permettant d'accéder au système de fichiers est précisé en annexe de ce document.

7. Annexe : Objets permettant d'accéder aux fichiers

7.1. Nom du fichier

Il existe plusieurs opérations possibles sur les fichiers contenus dans l'appareil. Avant chaque opération, le nom du fichier doit être indiqué à l'appareil.

Objet : Index 0x5F40, Sous-index 0x0
Type : String
Valeur : Nom du fichier au format 8.3 (Exemple: DRIVEPAR.TXT)

7.2. Ouverture d'un fichier en lecture

Objet : Index 0x5F42, Sous-index 0x1
Type : Integer32
Valeur à écrire : Taille du fichier en octets.

L'utilisation de cet objet nécessite qu'un nom de fichier ait été préalablement précisé avec l'objet 0x5F40/0x0.

7.3. Lecture d'un fichier

Objet : Index 0x5F48, Sous-index 0x0
Type : Unsigned8
Valeur lue : Tableau d'octets.

L'utilisation de cet objet nécessite qu'un nom de fichier ait été préalablement ouvert avec l'objet 0x5F42/0x1.

7.4. Création d'un fichier

Objet : Index 0x5F42, Sous-index 0x2
Type : Integer32
Valeur à écrire : Taille du fichier en octets.

L'utilisation de cet objet nécessite qu'un nom de fichier ait été préalablement précisé avec l'objet 0x5F40/0x0.

7.5. Ecriture d'un fichier

Objet : Index 0x5F49, Sous-index 0x0
Type : Unsigned8
Valeur à écrire : Tableau d'octets

L'utilisation de cet objet nécessite qu'un fichier ait été préalablement créé avec l'objet 0x5F42/0x2

7.6. Effacement d'un fichier

Objet : Index 0x5F42, Sous-index 0x4
Type : Integer16
Valeur à écrire : 0

L'utilisation de cet objet nécessite qu'un nom de fichier ait été préalablement précisé avec l'objet 0x5F40/0x0.

7.7. Fermeture d'un fichier

Objet : Index 0x5F42, Sous-index 0x3
Type : Integer32
Valeur à écrire : 0

L'utilisation de cet objet nécessite qu'un fichier ait été préalablement ouvert en lecture ou en écriture.

7.8. Recherche d'un fichier

Objet : Index 0x5F42, Sous-index 0x5
Type : Integer16
Valeur à lire : Etat du fichier

Valeurs de retour :

-2	Nom de fichier incorrect
-1	Le fichier est déjà ouvert
0	Le fichier n'existe pas
1	Le fichier existe
2	Le fichier est corrompu

L'utilisation de cet objet nécessite qu'un nom de fichier ait été préalablement précisé avec l'objet 0x5F40/0x0.

7.9. Liste des fichiers

Objet : Index 0x5F4A, Sous-index 0x0
Type : String
Valeur à lire : Liste et taille des fichiers

La chaîne de caractère retournée contient la liste et la taille des fichiers présents dans la mémoire de l'appareil.

Exemple :

DRIVEPAR.TXT 2621
USER_PAR.TXT 1582
2 file(s).

7.10. Codes de retour

Les codes de retour suivants concernent les opérations d'ouverture, fermeture, lecture, écriture et effacement d'un fichier :

0	FILE_OK	L'opération s'est correctement déroulée
-1	FILE_EXIST	Un fichier portant le même nom existe déjà
-2	FILE_ACCESS	Le fichier est déjà ouvert
-3	DSK_FULL	La mémoire est saturée
-4	FILE_NOTFOUND	Le fichier n'existe pas
-7	FILE_RDERROR	Erreur de lecture
-5	FILE_EOF	Fin du fichier
-6	FILE_NOTOPENED	Le fichier n'est pas ouvert
-8	FILE_WRERROR	Erreur d'écriture
-9	FILE_OPENED	Le fichier est ouvert
-10	FILE_NAME	Nom du fichier incorrect
-11	FILE_TOO_BIG	Taille du fichier trop importante
-12	FILE_CRC32	Erreur de CRC32