

Remerciements

Je tiens à remercier toutes personnes qui ont, de près ou de loin, contribué à l'accomplissement de ce travail de fin d'études.

Je remercie également toutes les personnes ressources pour l'aide et le soutien qu'elles m'ont témoigné tout au long de mon cheminement.

Merci à toute l'équipe d'Omni-Métrie qui m'a accueilli et aidé durant mon stage.

Merci également à Mr. Simon et Mr. Comblin qui m'ont aidé à prendre des décisions et à faire des choix.

Une de mes pensées va aux membres de ma famille ainsi qu'à mes amis qui m'ont soutenu et encouragé jusqu'au bout de ce travail.

Je ne me permettrai pas de clore ces remerciements sans saluer les professeurs de l'Institut Helmo Saint-Laurent pour le savoir transmis tout au long de ces années.

Introduction

Le choix du stage de fin de 3^{ème} fut l'aboutissement de trois années d'études. Ce qui m'a tout de suite plu dans l'automatisation, c'est sa diversité. Je n'allais pas me confiner dans un domaine précis mais au contraire apprendre une multitude de choses venant de domaines tout aussi variés.

Mon choix fut guidé par la possibilité d'étendre encore mes connaissances à deux sujets qui m'étaient jusqu'alors inconnus : l'OPC et la domotique.

Lors de mon stage, j'ai réalisé un logiciel permettant d'adapter le système de l'entreprise au système OPC. Au travers du TFE, je vous décrirai tout mon cheminement au travers de 15 semaines au sein de l'entreprise Omni-MétriX.

Le TFE est divisé en trois parties, la première présentera l'entreprise Omni-MétriX.

La deuxième partie parlera de l'étude et de l'analyse de mon projet : L'OPC, le système Omni-MétriX ainsi que les solutions disponibles pour la réalisation de mon projet.

Ensuite, la troisième expliquera le développement de ma réalisation.

Table des matières

Remerciements.....	1
Introduction	2
Table des matières.....	3
Présentation de l'entreprise	5
<i>Fondation d'Omni-Métrix.....</i>	<i>5</i>
<i>Secteur d'activités d'Omni-Métrix et ses réalisations</i>	<i>6</i>
<i>Le futur d'Omni-Métrix.....</i>	<i>7</i>
Etude et analyse du projet OPC.....	8
<i>Sujet de mon stage.....</i>	<i>8</i>
<i>Le système OPC</i>	<i>8</i>
Qu'est-ce que l'OPC ?.....	8
La fondation OPC	9
Spécifications	9
OPC DA	10
Fonctionnement : trois options.....	12
Avantages ou inconvénients ?	17
OPC UA.....	18
<i>Etude du système Omni-Métrix.....</i>	<i>19</i>
Vue d'ensemble	19
Les interfaces	21
Le CAN (Control Area Network)	23
<i>Etude du projet</i>	<i>25</i>
Créer un serveur OPC DA	25
Créer un driver pour un serveur DA.....	26
Solution « passerelle »	28
Réalisation.....	29
<i>Vue d'ensemble</i>	<i>29</i>

<i>Les objets utilisés</i>	31
L'objet « OPCServer »	31
L'objet « OPCGroups »	32
L'objet « OPCGroup »	33
L'objet « OPCItems »	34
L'objet « OPCItem »	35
<i>Le programme</i>	35
Présentation de l'interface :	37
Initialisation	38
Connexion de la socket	39
Connexion au serveur	40
Connexion au groupe	41
Connexion à l'Item	43
Data Arrival	46
Data Change	48
Supervision	50
Notify Icon	51
Conclusion	52
Bibliographie	53
Livres	53
Sites internet	53
Autres	53
Liste des figures	54
Annexes	56
<i>Annexe 1 : Le programme</i>	57
Feuille « Main »	57
Module « Devil »	64
Module « Fct »	66
Module « Ini »	74
<i>Annexe 2 : Les spécifications OPC</i>	75

Présentation de l'entreprise

Fondation d'Omni-Métrie



Omni-Métrie est une petite société vouée à un grand avenir. Tout a commencé avec la rencontre en 2004 entre deux hommes : Alain Willems et Jacques Stevens.

Mr. Willems a réalisé des études de commerce qui lui ont permises de travailler avec des marques d'éclairage telles que Idee Design Licht (DE) et Lucent Lighting (UK). On lui doit notamment la réalisation de l'éclairage de bâtiments tels que l'Esplanade de Louvain-la-Neuve (Fig. 1) ou du shopping de Maasmechelen.

Mr. Stevens, quant à lui, a commencé dans le bâtiment comme entrepreneur pour terminer ingénieur électronicien. Il a trouvé le moyen de réaliser l'entièreté de la domotique dans un bâtiment avec un seul système, ce qui reste un grand défi encore aujourd'hui.

Lorsqu'ils se rencontrent, Mr. Willems comprend le potentiel du système et ils décident de s'associer pour fonder Omni-Métrie SPRL en mai 2009.



Figure 1-Esplanade de Louvain-la-Neuve

Qu'est-ce que la domotique ?

Selon le dictionnaire Larousse, la domotique est l'ensemble des techniques visant à intégrer à l'habitation tous les automatismes en matière de sécurité, de gestion d'énergie, de communication, etc.

Secteur d'activités d'Omni-Métrix et ses réalisations



Figure 2-Médiacité

Omni-Métrix a d'abord été une société qui a beaucoup travaillé dans l'exportation et l'importation d'éclairages architecturaux d'origine allemande et anglaise. La société a équipé des bâtiments tels que le complexe Médiacité de Liège, le campus de la Haute Ecole de la Province de Liège situé à Jemeppe ou encore le Palais de justice de la cour Européenne.

De nos jours, la domotique est réalisée de telle façon qu'un automate central commande un ensemble de capteurs et d'interrupteurs. Les capteurs servent par exemple à détecter une présence ou mesurer la température ambiante et les interrupteurs

à allumer un éclairage ou déclencher le chauffage. Le système, dit centralisé, comporte un désagrément : un seul automate contrôle l'entièreté du système, donc lorsque l'automate tombe en panne, toute la gestion de la maison est alors hors-service. Imaginez que cet automate gère aussi le système de sécurité du bâtiment et vous pourrez imaginer l'ampleur de la problématique.

Le système Omni-Métrix lui est « décentralisé », ce qui veut dire qu'il est composé d'un ensemble d'interfaces qui gèrent chacune une petite partie de la gestion du bâtiment. Lors d'une panne, seule l'interface défectueuse est alors en défaut. Ce système est non seulement plus sécurisant mais aussi plus simple à réaliser, à maintenir et corriger. Avec un système où un automate gère tout, il faut programmer « ce tout » dans l'automate et ce programme peut-être lourd à réaliser et corriger. Avec le système Omni-Métrix, chaque interface contient sa partie du programme, il est donc plus simple de cibler un défaut et de le déboguer.

Voici maintenant quelques réalisations pratiques de l'entreprise :

Le **Moving Light** permet de diminuer la facture en électricité.

Dans un entrepôt classique de grande surface, on bascule un interrupteur, toutes les lumières s'allument, ce qui n'est pas très économique. Des zones non utilisées sont éclairées. Avec le Moving Light, toute personne entre dans l'entrepôt avec un capteur qui permet de la localiser. On arrive donc à isoler une zone à éclairer : celle où la personne se trouve. Les lampes s'allument chacune à leur tour en fonction du déplacement de la personne.

Le **Smart Office** aide les grandes entreprises qui souhaitent faire des économies avec une meilleure gestion des bureaux. Chaque employé porte sur lui un badge qui contient des informations sur sa morphologie. Lorsqu'il arrive au travail, il badge à l'entrée des locaux et choisit un bureau. Ce bureau va automatiquement s'adapter à sa morphologie (par exemple en montant ou descendant le niveau de la table). De plus, la lumière ne s'allume qu'au-dessus de son bureau et il est possible d'envisager le démarrage de l'ordinateur de son bureau.



Figure 3-Smart Office

A la fin de la journée, cet employé badge à nouveau, l'éclairage et l'ordinateur de son bureau s'éteignent automatiquement. Le soir la femme de ménage badge et toutes les tables se lèvent au maximum pour lui faciliter la tâche. Donc, plus de pertes d'énergie avec les lumières et les ordinateurs qui restent allumés inutilement. En plus, l'ergonomie est réfléchi ce qui améliore le confort du travail.

Le futur d'Omni-MétriX

Vous l'aurez compris, la domotique est un secteur en pleine expansion car c'est économique, écologique et offre un meilleur confort. Omni-MétriX n'a donc pas de soucis à se faire, la petite entreprise a encore de longs jours devant elle.

L'objectif de la société est d'adapter sa gamme domotique industrielle aux particuliers. Car si les interfaces sont adaptées aux besoins des entreprises, elles restent fort chères pour les particuliers qui devraient rentabiliser leur installation sur un plus grand laps de temps.

Etude et analyse du projet OPC

Sujet de mon stage

Aujourd'hui, les entreprises sont composées d'une multitude de machines, d'automates allant de la petite imprimante jusqu'à la chaîne de montage. Afin d'utiliser ses capacités au maximum, l'entreprise doit interconnecter tous ces appareils. Un automate devrait, par exemple, pouvoir imprimer lui-même un rapport d'erreur. L'automate doit pouvoir communiquer avec toutes ses jauges. Une base de données doit pouvoir stocker les informations de nombreux appareils différents. Le problème c'est que chaque marque a son propre mode de communication. Il est donc difficile de mettre en relation tous les appareils d'une entreprise. Pour réaliser cette mise en commun, une référence émerge en ce moment, c'est l'OPC.

Mon stage consiste à rendre le système Omni-Métrix compatible avec le système OPC. Grâce à la compatibilité avec le système OPC, Omni-Métrix pourra réaliser ses interfaces avec les grandes marques de la domotique, la rendant plus attractive.

Le système OPC

Qu'est-ce que l'OPC ?

OPC signifiait initialement **OLE for Process Control**. OLE est un protocole et un système d'objets mis au point par Microsoft pour permettre à des applications utilisant des formats différents de dialoguer. OPC visait donc à relier les applications Windows avec le matériel et les logiciels permettant le contrôle de processus industriels.

Aujourd'hui, on a gardé le terme OPC, cependant ce n'est plus le OLE qui est utilisé car il n'est plus à jour. OPC s'est basé sur les protocoles OLE, DCOM et COM et utilise dans ses dernières versions le protocole « .NET », plus performant et plus compatible que ses prédécesseurs.

OPC n'est pas qu'un protocole de communication, mais une solution globale pour accéder aux données des différents appareils de terrain dans une usine.

Un appareil de terrain représente tout appareil se trouvant sur le terrain de l'entreprise et ayant une quelconque action sur la production de l'entreprise. Par exemple un appareil de terrain peut être un automate ou un capteur (de température, de distance, etc...).

OPC est un ensemble de spécifications libre de droit, éditées par le regroupement d'utilisateurs au sein de la fondation OPC.

Les spécifications OPC définissent des « interfaces COM » qui traitent les différents domaines du « process control ».

La fondation OPC

La fondation OPC est une organisation mondiale chargée de promouvoir la technologie OPC. Elle compte aujourd'hui 427 membres à travers le monde dont les plus importants fabricants en contrôle de processus industriels.

Les membres de la fondation OPC sont divisés en groupes de travail qui améliorent les spécifications OPC. La collaboration des membres de la fondation OPC dépend d'initiatives personnelles au travers des groupes de travail.

Le conseil d'administration de la fondation OPC comporte sept membres : Thomas Burke, le président de la fondation OPC, Reinhold Achatz, Grant Wilson, Nobuaki Konishi, David Eisner, Ken Hall et Russ Agrusa.

Process Control : Control de processus industriel.

Spécifications

OPC est basé sur un principe simple, celui du client et du serveur. Le serveur va offrir des services au client demandeur, comme au restaurant lorsque qu'en tant que client vous commandez un plat au serveur.

Dans le modèle OPC, il y a plusieurs types de serveurs destinés à fournir différents services :

- DA : ou Data Access permet d'accéder à des données.
- A&E : ou Alarm and Event gère les alarmes et les événements spéciaux.
- HDA : ou Historical Data Access permet d'archiver et d'accéder à des données fournies par un serveur DA.

Les spécifications sont utilisées par la fondation OPC pour définir des normes de communications. Les trois principales sont citées ci-dessus : DA, A&E et HDA.

Entre autre, cela veut dire qu'il faut trois serveurs différents pour réaliser la gestion des données. Ce n'est pas très pratique mais la fondation OPC y a pensé et a réalisé l'OPC UA (Unified Architecture voir page 18).

OPC DA

Attardons-nous maintenant sur la spécification DA (Data Access) dont je me servirai dans mon programme. Le serveur DA permet d'accéder en temps réel aux données d'un réseau de PLC connecté ou de n'importe quelle autre source d'informations. Comme cité plus haut, OPC DA est basé sur le principe client-serveur. C'est le serveur DA qui va collecter les informations de l'appareil de terrain.

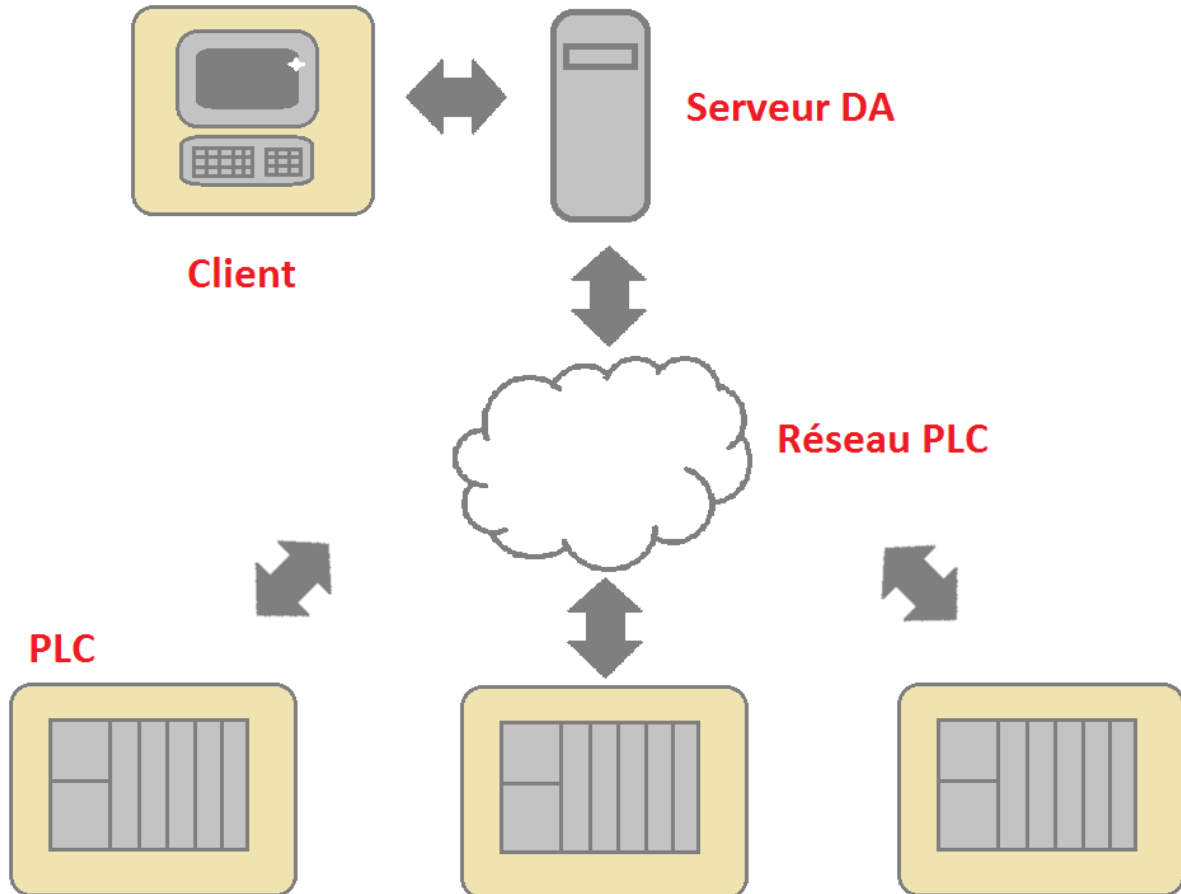
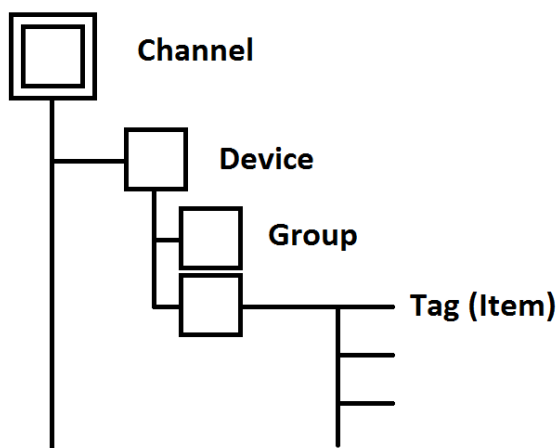


Figure 4-OPC DA

PLC : (Programmable Logic Controller) est un automate programmable industriel (API).

Un serveur DA peut être vu sous la forme d'une base de données. Pour faire simple, le serveur DA représenté ci-dessus est un ordinateur avec un logiciel qui tourne continuellement. Cet ordinateur sert d'intermédiaire entre le réseau de PLC et le réseau OPC. Le serveur DA contient en fait une liste d'Items contenant chacun une donnée venant des PLC. Chaque liste d'Items ou « Tags » est organisée sous la forme d'une structure. Voici la structure schématique des serveurs Kepware :



Channel : Connexion avec le ou les Devices.
 Device : Appareil ou PLC.
 Group : C'est un groupement logique de Tags.
 Tag : C'est une valeur.

Figure 5-Structure du serveur OPC

Il est donc possible de voir un serveur DA comme une base de données organisée. Lorsqu'une des données d'un des PLC est modifiée, que la réception est moins bonne ou qu'elle est devenue inexistante, l'Item correspondant est mis à jour dans le serveur DA. Ainsi une multitude de clients pouvant dialoguer en « OPC » peuvent se connecter au serveur DA. Ils obtiennent rapidement les données des Items modifiés qui ne sont autres que des données venant des PLC.

Fonctionnement : trois options

Pour bien comprendre le fonctionnement, commençons par un exemple. Voici une entreprise qui dispose d'un IHM, d'un historique de processus, d'un API, d'une surveillance de processus et d'un moteur.

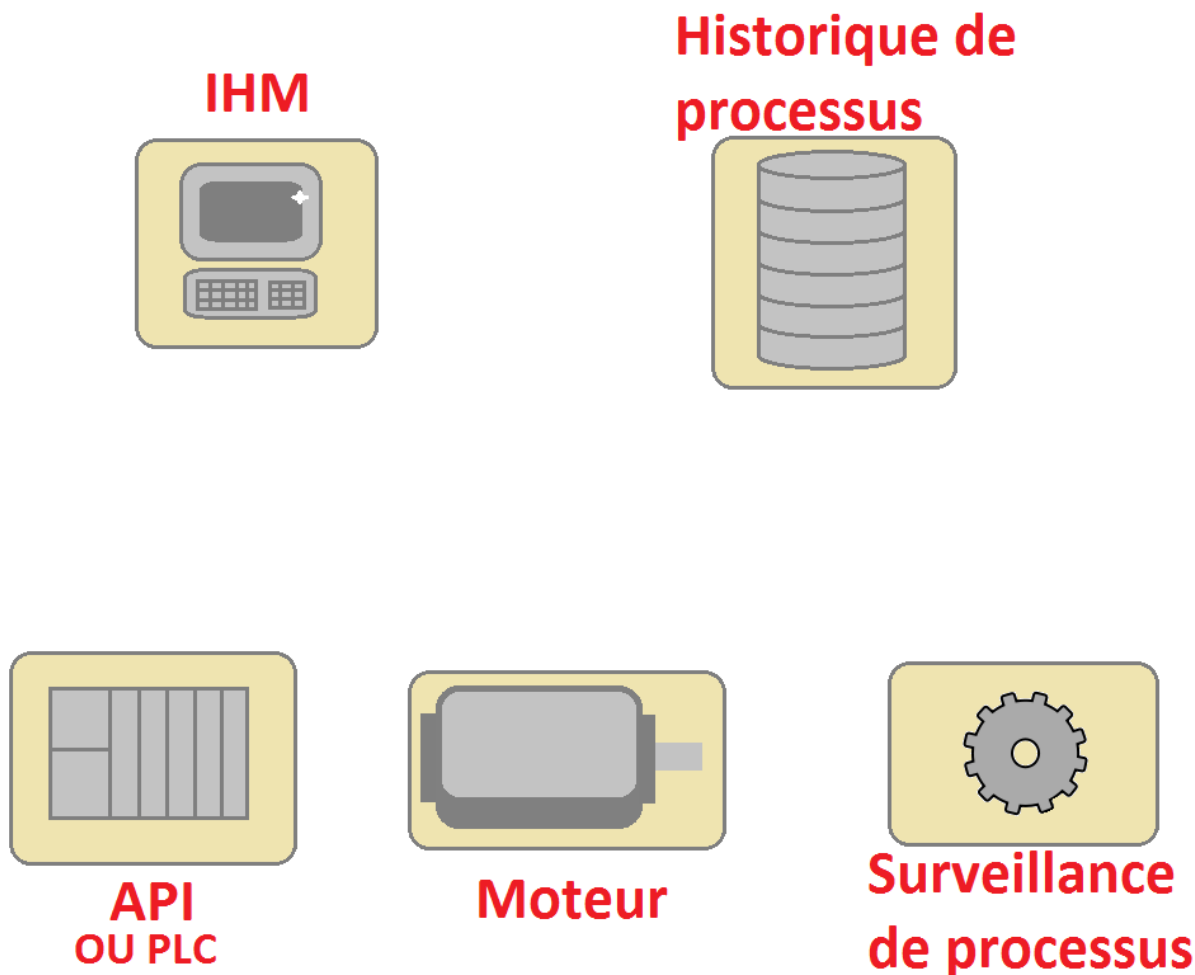


Figure 6-Entreprise sans OPC

IHM : L'IHM (Interface Homme-Machine) est un ensemble de moyens et d'outils mis en œuvre pour qu'un humain puisse communiquer avec une machine. Par exemple, un ordinateur avec un logiciel permettant de contrôler un automate.

Historique de processus : L'historique de processus est une base de données qui permet de les stocker en temps réel.

API : C'est un Automate Programmable Industriel. PLC (Programmable Logic Controller) est l'équivalent anglais de l'API.

Surveillance de processus : La surveillance de processus permet surveiller le déclenchement d'évènements. Par exemple : si une cuve est remplie, pour éviter qu'elle ne déborde, le processus de remplissage est arrêté et les vannes de vidange s'ouvrent.

Une entreprise a un certain nombre de besoins :

- **L'échange d'informations** : L'ensemble des sous-systèmes de l'entreprise doivent pouvoir communiquer entre eux. Un contrôle d'accès doit pouvoir recevoir des informations sur le système de sécurité incendie et inversement. Dans l'entreprise servant d'exemple plus haut, il faut un échange de données avec le PLC et la base de données (historique de processus).
- **L'interopérabilité** : Le système mis en place dans l'entreprise doit pouvoir fonctionner avec d'autres systèmes présents ou futurs et ce sans restriction d'accès ou de mise en œuvre. La pérennité du matériel de l'entreprise doit être garantie ainsi que son évolution.
- **La réduction des coûts** : L'entreprise a besoin d'un système bon marché avec un faible coût d'entretien et qui lui fasse gagner du temps.

Pour pallier à ces besoins, trois options s'offrent à nous :

- La solution propriétaire
- Créer un driver pour chaque appareil
- La solution OPC

La solution propriétaire :

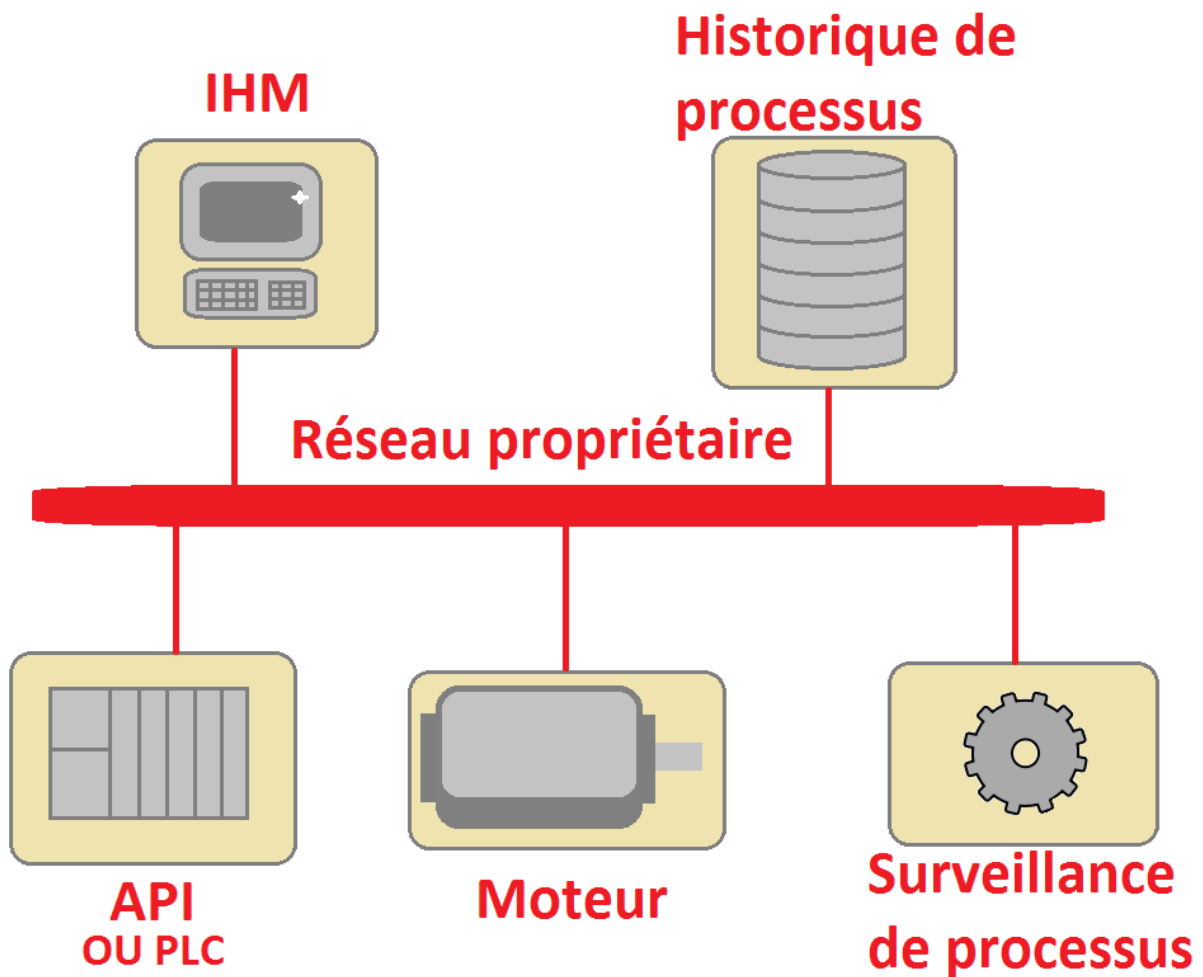


Figure 7-Entreprise avec réseau propriétaire

Elle existe et fonctionne mais n'est pas recommandée. Il suffit de prendre une seule marque d'appareil. Il n'y aura pas de problème de compatibilité mais il faut une multitude d'appareils différents. Peu d'entreprises sont capables de fournir une telle diversité. Dans notre exemple, il faudrait un fabricant d'ordinateurs qui est aussi fabricant d'automates, de moteurs qui de surcroît possède des bases de données compatibles avec son réseau ainsi qu'une gestion d'évènements. De plus, l'entreprise se retrouve « coincée » dans une marque. Elle ne pourra ajouter aucun appareil d'une autre marque sous peine d'incompatibilité qui demanderait la création d'un driver. La création du driver nous amène à la deuxième solution.

Créer un driver pour chaque appareil :

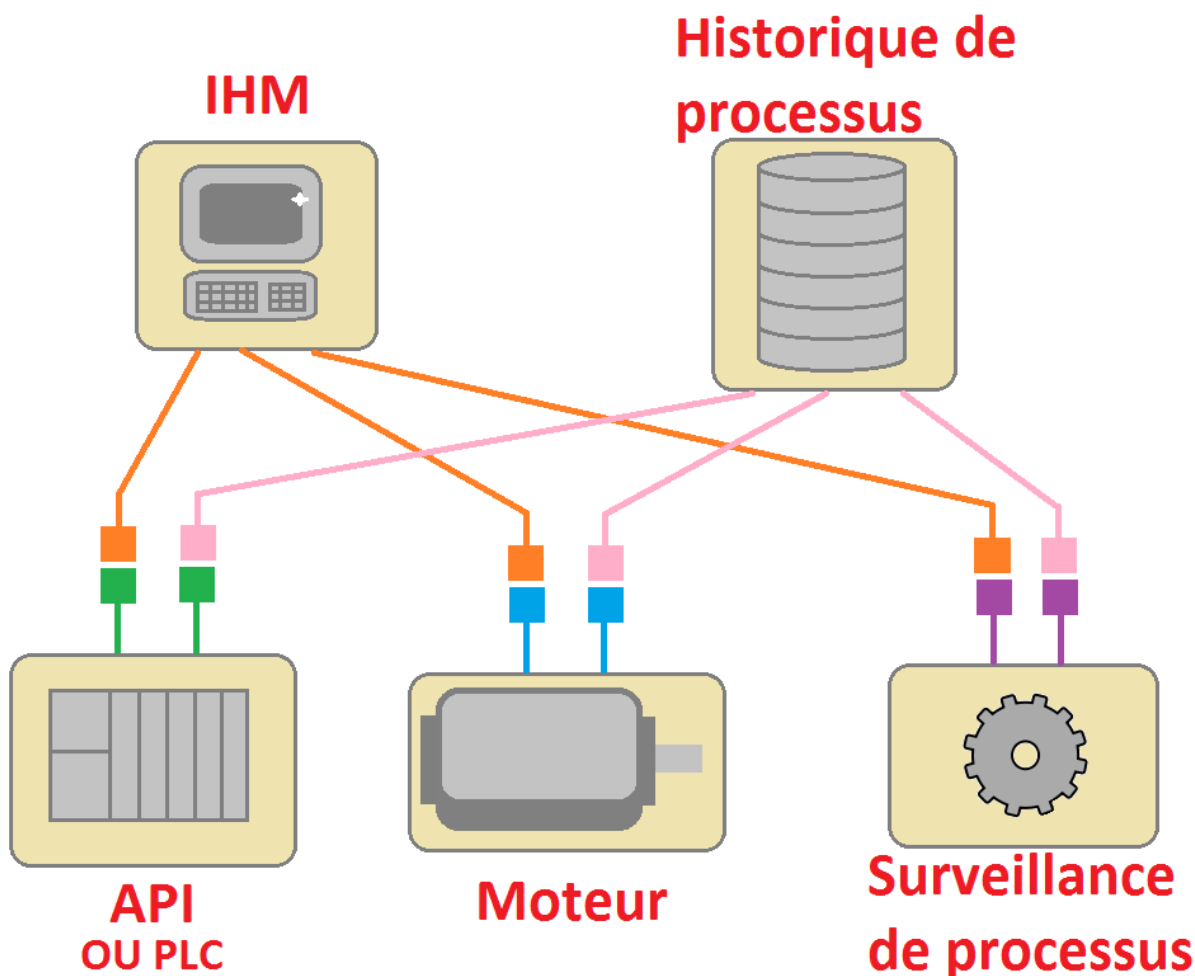


Figure 8-Solution DRIVER

Cette solution consiste à, comme elle l'indique, créer un driver pour chaque appareil. Chaque couleur représente un type de communication. Cependant comme vous pouvez le constater, il y a de nombreux câblages qui demandent beaucoup de drivers (un pour chaque liaison entre deux appareils).

Driver : ou pilote est un programme qui permet à un logiciel d'interagir avec un périphérique (comme un automate).

Cette solution est très onéreuse et demande beaucoup de temps. Si l'on rajoute un appareil, il faut à nouveau créer un driver. Même si cette solution permet d'échanger des informations, elle a un cruel manque d'interopérabilité et ne peut réduire efficacement ses coûts. Cette solution n'est donc pas recommandable non plus.

La solution OPC :

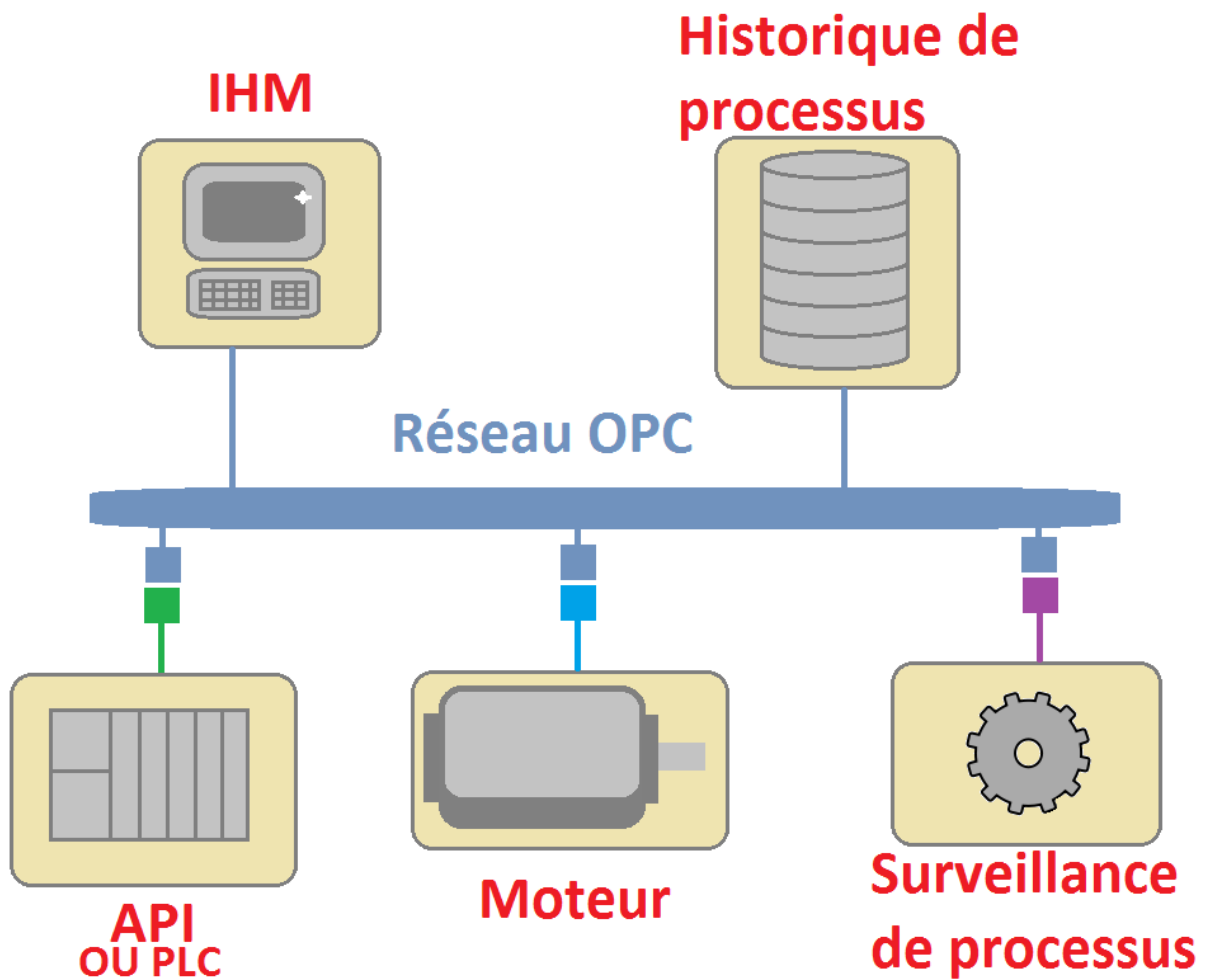


Figure 9-Entreprise avec OPC

Chaque carré mis l'un en face de l'autre représente un serveur OPC Da ou UA si l'on veut utiliser les dernières normes OPC. Tous les appareils peuvent communiquer entre eux, le réseau OPC étant la partie supérieure en bleu. On voit tout de suite que c'est plus clair que la solution précédente. Elle est moins coûteuse et demande moins de temps à mettre en œuvre.

Avantages ou inconvénients ?

Est-ce vraiment mieux d'ajouter des serveurs ? Est-ce vraiment moins cher ? Est-ce fiable ?

OPC est devenu une référence. La plupart des logiciels de supervision ou de contrôle de processus sont compatibles OPC. Toutes les marques se rendent compatibles OPC parce que c'est fiable, que c'est assez simple à configurer et moins chère que la deuxième solution. Le schéma précédent montrait de manière épurée l'OPC. Voici maintenant le réseau OPC physique appliqué au réseau de l'entreprise Omni-Métrix (le DevilNet) :

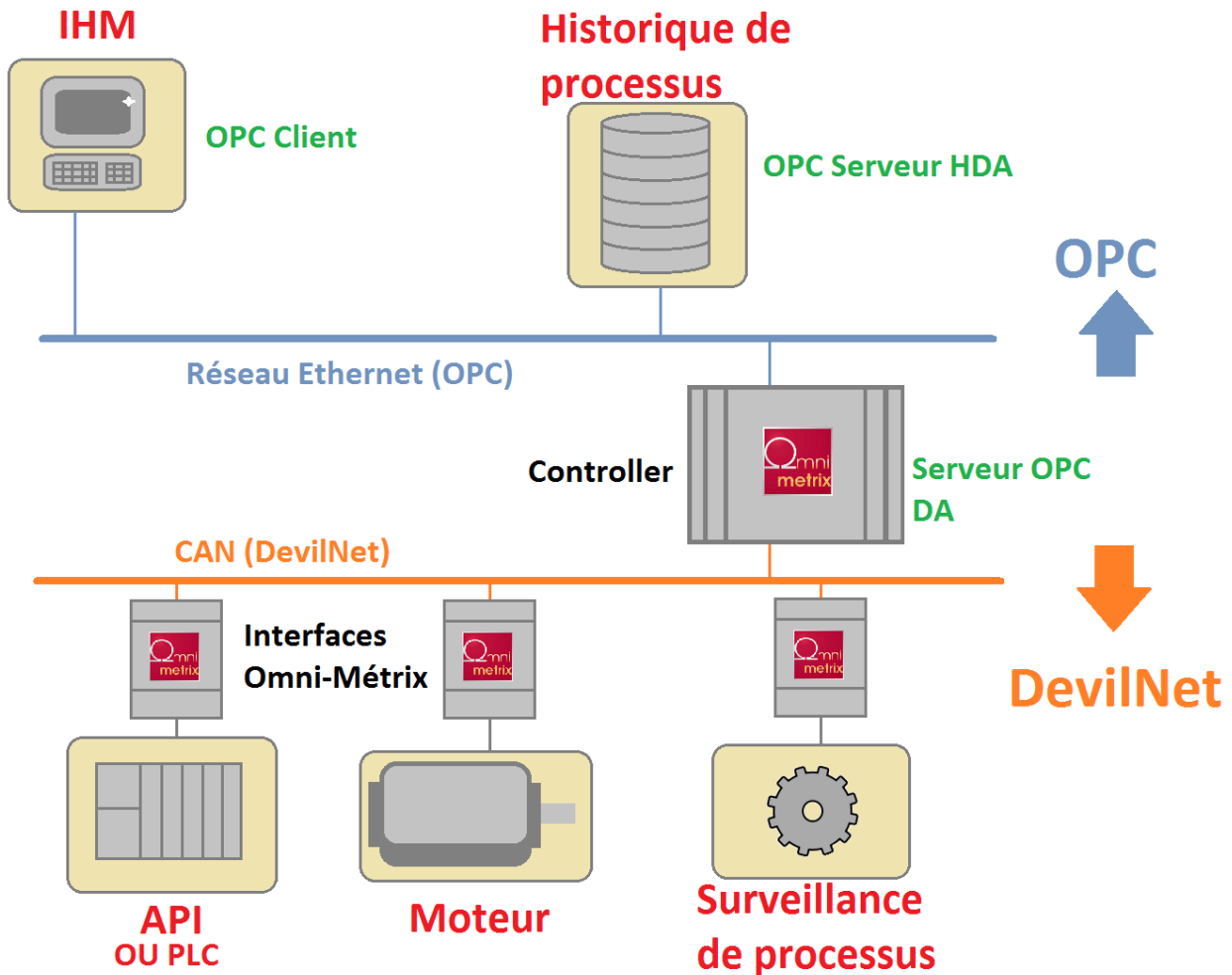


Figure 10-Entreprise Omni-Métrix avec OPC

Grâce à OPC, Omni-Métrix pourra maintenant s'interfacer à des grandes marques comme Siemens.

OPC UA

Les dernières normes OPC sont aujourd'hui déterminées par OPC UA c'est-à-dire OPC Unified Architecture. L'OPC UA regroupe dans un seul espace mémoire une donnée compatible avec les spécifications DA (Data Access), A&E (Alarm and Events) et HDA (Historical Data Access). Autrement dit, il n'y a plus besoin que d'un serveur UA pour réaliser de l'acquisition de données (DA), de la gestion d'évènements (A&E) et de l'archivage de données (HDA).

Etude du système Omni-MétriX

Vue d'ensemble

Comme dit précédemment, Omni-MétriX est un système décentralisé composé d'interfaces fixées sur un rail DIN et relié par un bus, le DevilNet. Ces interfaces sont des intelligences déportées qui permettent de diviser la gestion complète d'un bâtiment. Il permet de mieux cibler une erreur en cas de problème. Le contrôleur n'est là que pour les gestions demandant une supervision.

Interfaces

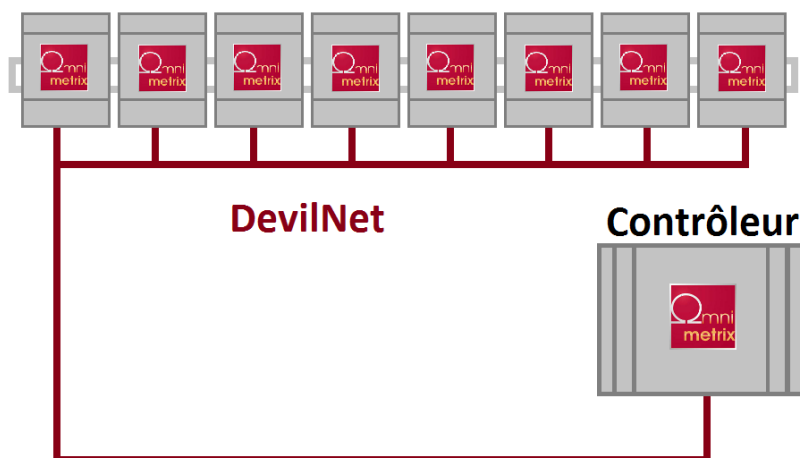
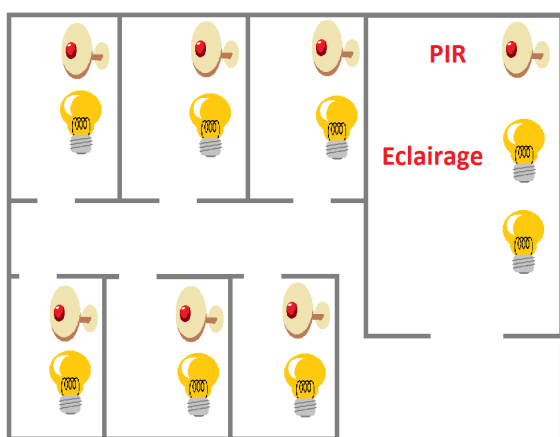


Figure 11-Les interfaces

Les avantages ne s'arrêtent pas là :

Pour réaliser la gestion de tout un bâtiment, surtout s'il est grand, il faut énormément de câbles. Cependant avec le système décentralisé, chaque interface peut être placée près de ses entrées/sorties.

Voici le schéma d'une entreprise désirant automatiser son éclairage. Pour ce faire, on dispose d'un PIR et d'au moins un éclairage (représenté par une ampoule) par pièce.



Un PIR est un détecteur de présence infra-rouge.

Figure 12-Entreprise simple

À présent, voyons ce que donnerait l'installation centralisée :

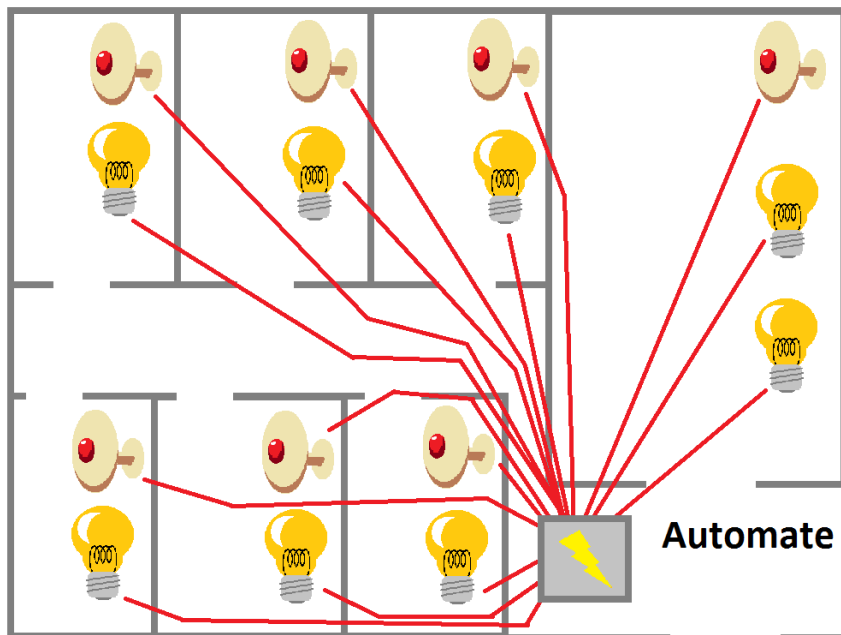


Figure 13-Entreprise avec système centralisé

Sur cette image, j'ai représenté en rouge les câbles qui relient les capteurs et les éclairages à l'automate. Sans prendre en compte les murs, il faut déjà une grande quantité de câbles. Voyons maintenant l'installation avec le système décentralisé :

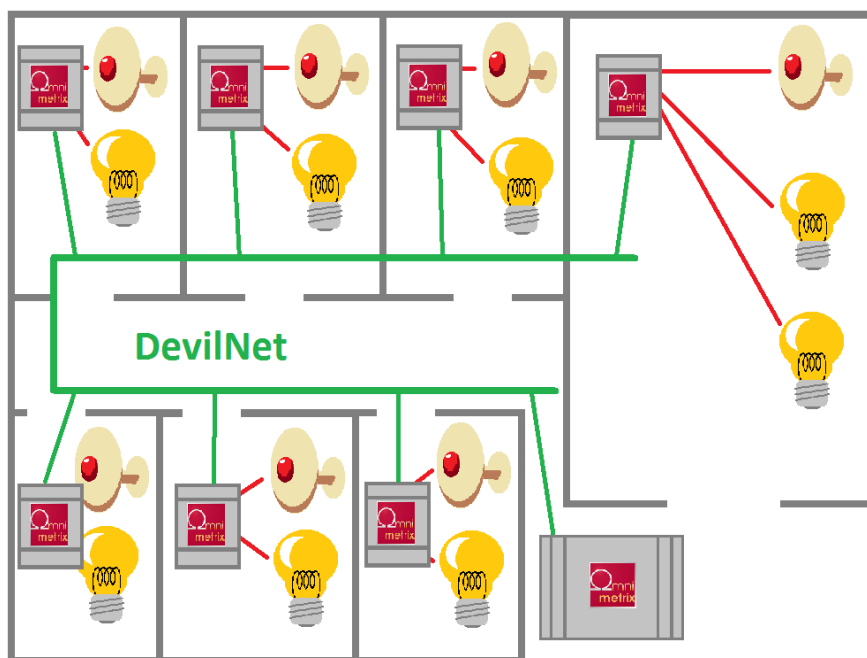


Figure 14-Entreprise avec le système Omni-Métrie

Il y a besoin d'une quantité moindre de câbles car les interfaces sont placées à proximité de leurs entrées/sorties et reliées entre elles par le DevilNet. Ce dernier point est très utile, surtout dans les grandes entreprises où les bâtiments font quelques centaines de mètres.

Les interfaces

Les interfaces sont les points clés du système Omni-Mérix.

Voici maintenant la présentation d'une des interfaces :

La 4 DI ou 4 digital input :



Figure 15-Interface 4 DI

Comme nous pouvons le constater, cette interface est constituée de 2 rangées de connecteurs. Ceux du dessous sont destinés à l'alimentation et la communication avec le DevilNet. Les connecteurs du dessus sont quant à eux, destinés aux entrées digitales. Cette interface est alimentée en 24V et reçoit des entrées dont la tension varie entre 0 et 5V. Le schéma suivant montre les détails du câblage.

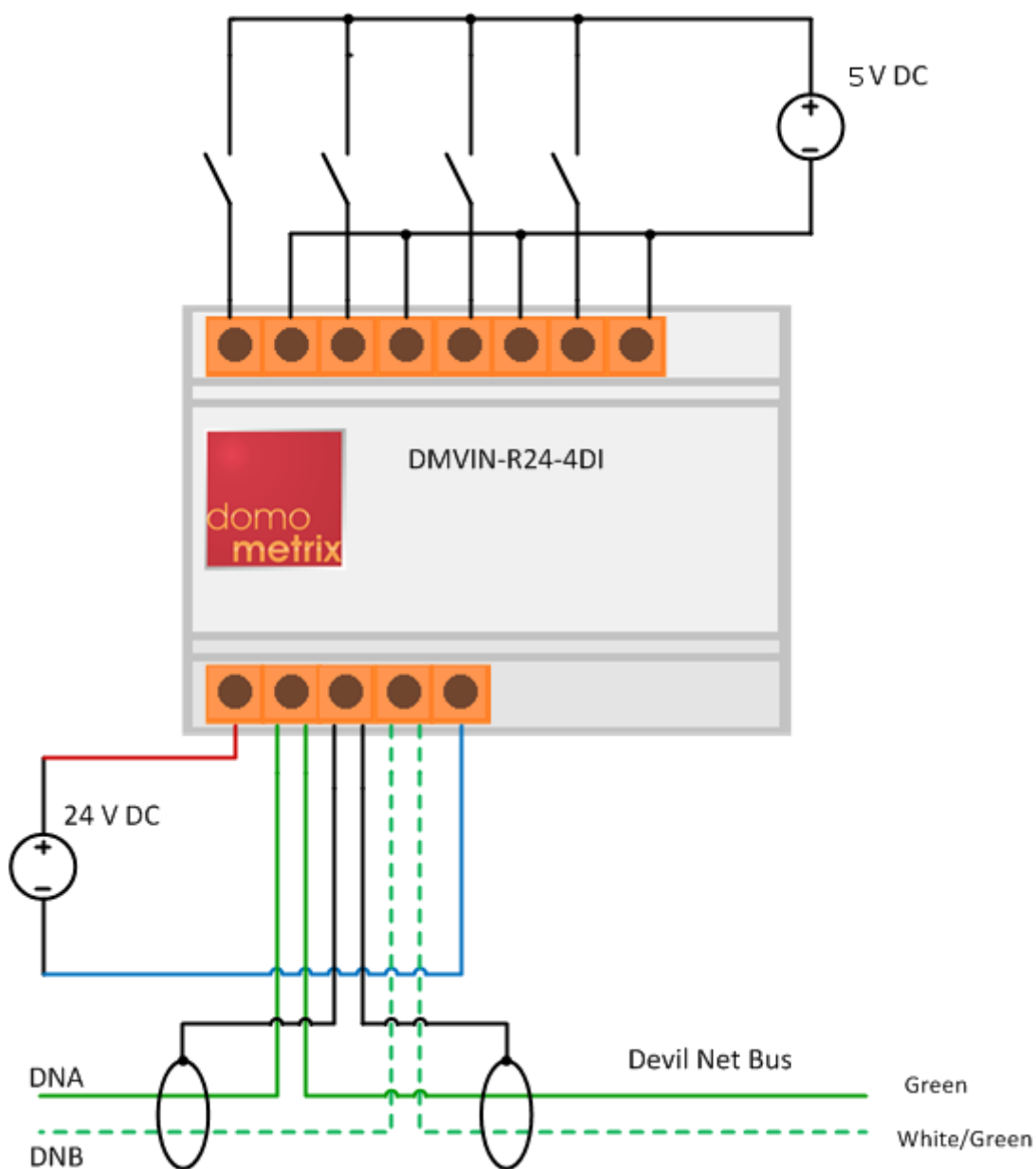


Figure 16-Schéma de câblage d'une interface 4 entrées

Toutes les interfaces ont en commun les mêmes connexions au DevilNet. Que ce soit pour un volet, un lecteur de badge, l'éclairage, la climatisation, et toute autre application domotique dans le bâtiment, il existe une interface Omni-Mérix.

Ces interfaces ont leur propre microprocesseur. Le modèle de microprocesseur est commun à toutes les interfaces, seul le hardware diffère.

Pour les applications de télémaintenance qui nécessitent des bases de données ou si on désire commander le système Omni-Mérix par l'intermédiaire d'un SCADA, il nous faut un contrôleur.

Télémaintenance : Selon le dictionnaire Larousse : Technique de maintenance d'un système informatique par un centre de maintenance auquel il est relié par un réseau de télécommunication.

C'est la visualisation de mesures et la commande à distance d'un système.

SCADA : Le SCADA (Supervisory Control And Data Acquisition) est un système de gestion à distance d'un système industriel.

Le contrôleur est en fait un petit ordinateur mis aux normes industrielles. Il est relié au bus DevilNet par une connexion RS-232 et à un réseau par la connexion Ethernet. Il sert donc d'intermédiaire entre l'utilisateur du SCADA ou la base de données et le DevilNet. Dans le cadre de mon projet, je devrais ajouter au contrôleur un logiciel qui, une fois installé, servira d'intermédiaire entre le DevilNet et le réseau OPC.

Le CAN (Control Area Network)



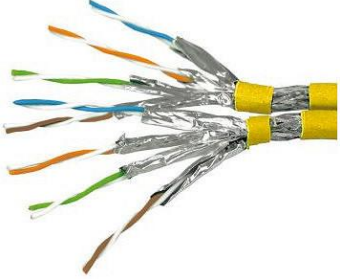
Le bus DevilNet utilise le Bus CAN comme support. Il a d'abord été développé par BOSCH. C'est un bus de terrain très sécurisé et rapide : le risque d'erreur est pratiquement inexistant et la vitesse du bus va jusqu'à 1 Mégabit/ Seconde sur de courtes distances. Le CAN est doté d'un Protocol ouvert qui permet l'utilisation de programmes tels que « DEVICE NET », « CAN OPEN » ou encore « CAN KINGDOM ».

Le protocole DevilNet basé sur la norme CAN (iso 11998), permet les connexions bidirectionnelles entre les interfaces. Lorsque le contrôleur envoie une demande à une interface, l'interface renvoie une confirmation indiquant qu'elle a reçu la demande mais qu'en plus, elle l'a exécutée avec succès.

Les possibilités d'adressage du bus sont pratiquement infinies puisqu'il accepte environ 537 millions d'adresses. Sur chaque adresse, il est possible réaliser plus de 64.000 fonctions différentes.

Les câbles CAT 5e, 6 et 7 sont utilisés pour transmettre les informations du DevilNet. Les CAT 6 et 7 sont justifiés lorsqu'il existe de nombreuses perturbations en milieu industriel, par exemple.

Voici une courte présentation de ces trois câbles :

 <p>Figure 17-CAT 5e</p>	<p>CAT 5 e : La catégorie 5 est un câble UTP c'est-à-dire Unshielded Twisted Pair. C'est du câble non blindé mais largement suffisant pour câbler l'intérieur des bureaux où il n'y a pas beaucoup de perturbations. Il offre une bande passante allant jusque 125MHz. Il est souvent utilisé pour câbler les réseaux domestiques.</p>
 <p>Figure 18-CAT 6</p>	<p>CAT 6 : La catégorie 6 est aussi un câble UTP mais dont le diamètre des brins conducteurs a été augmenté pour permettre une meilleure conduction. Ainsi il dispose d'une bande passante de 250MHz.</p>
 <p>Figure 19-CAT 7</p>	<p>CAT 7 : La catégorie 7 fait partie des câbles SFTP c'est-à-dire Shielded Foiled Twisted Pair. Non seulement ce câble est blindé, ce qui lui permet de se protéger des basses fréquences, mais en plus il est muni de paires torsadées avec écrantage par feuillard. Ce dernier, protège le CAT 7 des hautes fréquences et évite le rayonnement du câble vers son environnement.</p>

Bande passante : La bande passante est une zone de fréquences pour lesquelles la réponse d'un appareil est supérieure à une atténuation maximum.

Etude du projet

Pour réaliser l'interfaçage OPC, trois solutions sont possibles.
Voici les avantages et les inconvénients de chacune :

Créer un serveur OPC DA

Il est possible de créer soi-même un serveur OPC DA (ou UA), cependant il faudra faire face à quelques soucis. En effet, pour pouvoir accéder aux outils de création d'un serveur OPC, il faut s'inscrire sur le site de la fondation (www.opcfoundation.org) et s'enregistrer comme membre de la fondation OPC. Cette enregistrement n'est pas gratuit et coûte cher pour les petites entreprises.

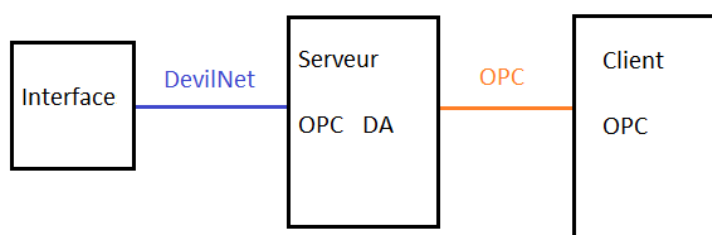


Figure 20-Solution serveur DA

L'image ci-dessus illustre ce que donnerait le réseau de l'entreprise avec cette solution. Le DevilNet, relié avec un contrôleur, communique avec un logiciel serveur OPC DA, qui n'est en fait qu'une sorte de base de données. Le client OPC peut alors lire des données dans le serveur DA qui contient une copie de l'état des données de l'interface. S'il le désire, le client peut demander au serveur d'écrire une donnée dans l'interface.

Créer un driver pour un serveur DA

Il existe déjà des serveurs DA tout faits, qui fonctionnent parfaitement et sont régulièrement mis-à-jour. Encore une fois ce n'est pas gratuit mais c'est simple à réaliser. J'ai mis à titre d'exemple, le serveur Kepware que j'ai utilisé pour tester mon client.

The screenshot shows the KEPServerEX - Runtime application interface. It is divided into three main sections:

- Section 1 (Left):** A tree view showing the hierarchy of data items. It includes 'Channel1', 'Device1', 'Data Type Examples', and 'Simulation Examples'. 'Device1' is circled in black.
- Section 2 (Right):** A table displaying the details of the selected tags. The table has columns for Tag Name, Address, Data Type, Scan Rate, Scaling, and Description. The tags listed are Bool1 through TagJu.
- Section 3 (Bottom):** An event log showing a series of system events with columns for Date, Time, Source, and Event. The log is circled in black.

Tag Name	Address	Data Type	Scan Rate	Scaling	Description
Bool1	K0004.00	Boolean	100	None	
Bool2	K0004.01	Boolean	100	None	
Bool5	R0004.01	Boolean	100	None	
Bool6	R0004.02	Boolean	100	None	
Tag1	R0001	Short	100	Linear	Ramping Read/Write tag used to verify client connection
Tag11	K0010	Short	100	Linear	
Tag2	R0002	Short	100	Linear	Ramping Read/Write tag used to verify client connection
Tag3	R0003	Short	100	Linear	Ramping Read/Write tag used to verify client connection
TagJu	R0010	Short	100	Linear	

Date	Time	Source	Event
26/04/2011	8:33:44	KEPServerEX\...	Alarms & Events Plug-in V5.3.156.0
26/04/2011	8:33:46	KEPServerEX\...	Runtime service started.
27/04/2011	8:33:55	KEPServerEX\...	Runtime performing exit processing.
27/04/2011	8:35:18	KEPServerEX\...	Kepware Communications Server 5.3
27/04/2011	8:35:20	KEPServerEX\...	Simulator device driver loaded successfully.
27/04/2011	8:35:25	KEPServerEX\...	Starting Simulator device driver.
27/04/2011	8:35:25	Simulator	Simulator Device Driver V5.3.156.0
27/04/2011	8:35:25	KEPServerEX\...	Advanced Tags Plug-in V5.3.156.0
27/04/2011	8:35:25	KEPServerEX\...	Data Logger Plug-in V5.3.156.0
27/04/2011	8:35:25	KEPServerEX\...	Oracle Connector Plug-in V5.3.156.0
27/04/2011	8:35:25	KEPServerEX\...	Alarms & Events Plug-in V5.3.156.0
27/04/2011	8:35:26	KEPServerEX\...	Runtime service started.
27/04/2011	8:42:40	KEPServerEX\...	Kepware Communications Server 5.3
27/04/2011	8:42:42	KEPServerEX\...	Simulator device driver loaded successfully.
27/04/2011	8:42:45	KEPServerEX\...	Starting Simulator device driver.
27/04/2011	8:42:45	Simulator	Simulator Device Driver V5.3.156.0
27/04/2011	8:42:45	KEPServerEX\...	Advanced Tags Plug-in V5.3.156.0
27/04/2011	8:42:45	KEPServerEX\...	Data Logger Plug-in V5.3.156.0
27/04/2011	8:42:45	KEPServerEX\...	Oracle Connector Plug-in V5.3.156.0
27/04/2011	8:42:45	KEPServerEX\...	Alarms & Events Plug-in V5.3.156.0
27/04/2011	8:42:45	KEPServerEX\...	Runtime service started.
27/04/2011	9:34:09	KEPServerEX\...	Configuration session started by Julien as Default User (R/W)
27/04/2011	9:34:15	KEPServerEX\...	Configuration session assigned to Julien as Default User ha...
27/04/2011	14:45:42	KEPServerEX\...	Configuration session started by Julien as Default User (R/W)

Figure 21-Interface serveur Kepware

En 1 on peut voir la hiérarchisation des données qui se trouve en 2. Les données sont appelées « Item » ou « Tag ». C'est comme si dans une base de données, il y avait un groupe appelé « Channel1 » dans lequel se trouve le sous-groupe « Device1 » qui contient les différents Tags. Dans la partie numéro 3, on peut voir le journal du serveur avec toutes les actions qui se sont bien ou mal déroulées.

Ces serveurs sont fournis avec de nombreux drivers compatibles avec les principaux standards. Cependant le DevilNet est unique pour le moment et il n'existe aucun driver tout fait. Dans cette solution il faut donc créer un driver pour interfacier le DevilNet et le serveur OPC. Si l'on arrive à faire le driver pour le serveur Kepware, il n'est pas dit qu'il fonctionnera avec un serveur d'une autre marque. Or si l'entreprise possède déjà un serveur OPC d'une autre marque, elle serait obligée de racheter le serveur compatible avec notre driver ou bien c'est à nous de refaire un autre driver compatible. Bref, cette solution reste fort inconfortable.

Solution « passerelle »

Cette solution est celle retenue pour mon projet de stage. Puisque dans l'OPC, il faut toujours un serveur DA, trouvons un moyen universel de communiquer avec lui. Je m'explique :

Pour ne pas devoir programmer notre propre serveur OPC ou commencer à créer des drivers « à la pelle », il faut réaliser un logiciel qui traduirait les données du DevilNet en un langage compréhensible de tous les serveurs OPC, c'est-à-dire le langage OPC tout simplement.

L'idée est de réaliser un traducteur qui lorsqu'il reçoit des données du DevilNet, il les envoie au serveur OPC et à l'inverse, si le serveur reçoit une commande d'un client OPC, il l'envoie au DevilNet.

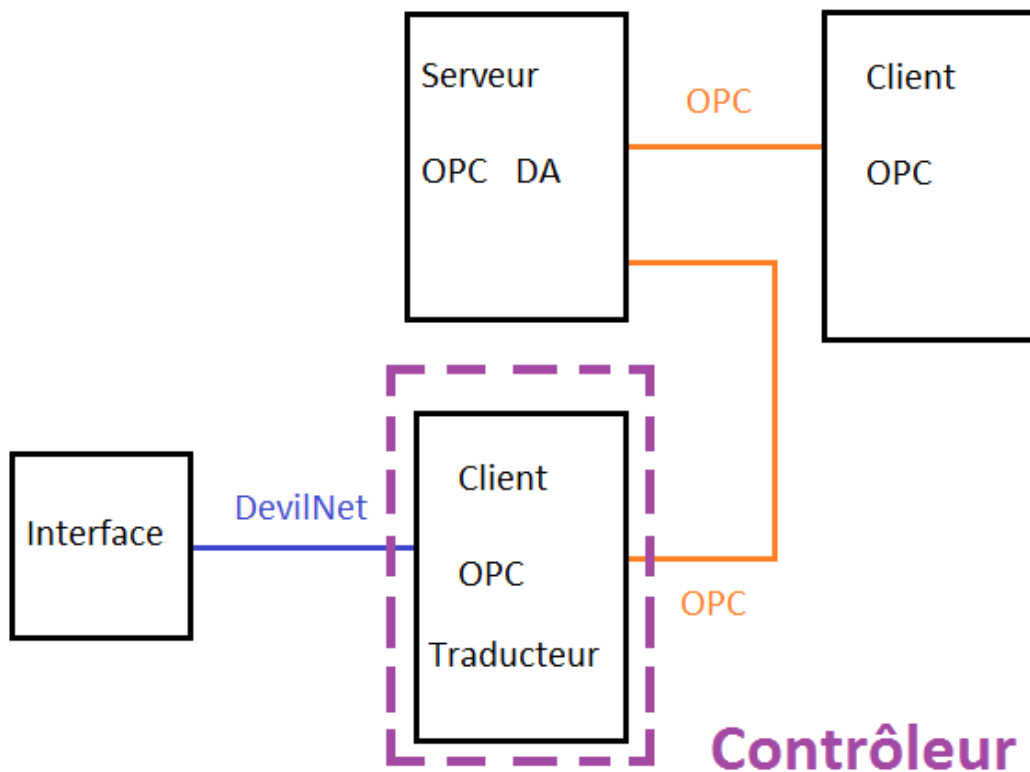


Figure 22-Solution "passerelle"

Réalisation

Vue d'ensemble

Réaliser le client OPC ne fut pas chose facile car ça demande de grandes connaissances en programmation et les exemples de clients OPC ne fonctionnent pas.

J'ai débuté mon étude sur les clients OPC en C++, un langage que je connais. Cependant l'exemple fournit par Kepware ainsi que ceux disponibles sur le net ne fonctionnent pas.

Pour pallier à ce problème, j'ai donc pris le client Kepware en Visual Basic qui lui fonctionne.

Voyons maintenant comment réaliser mon client :

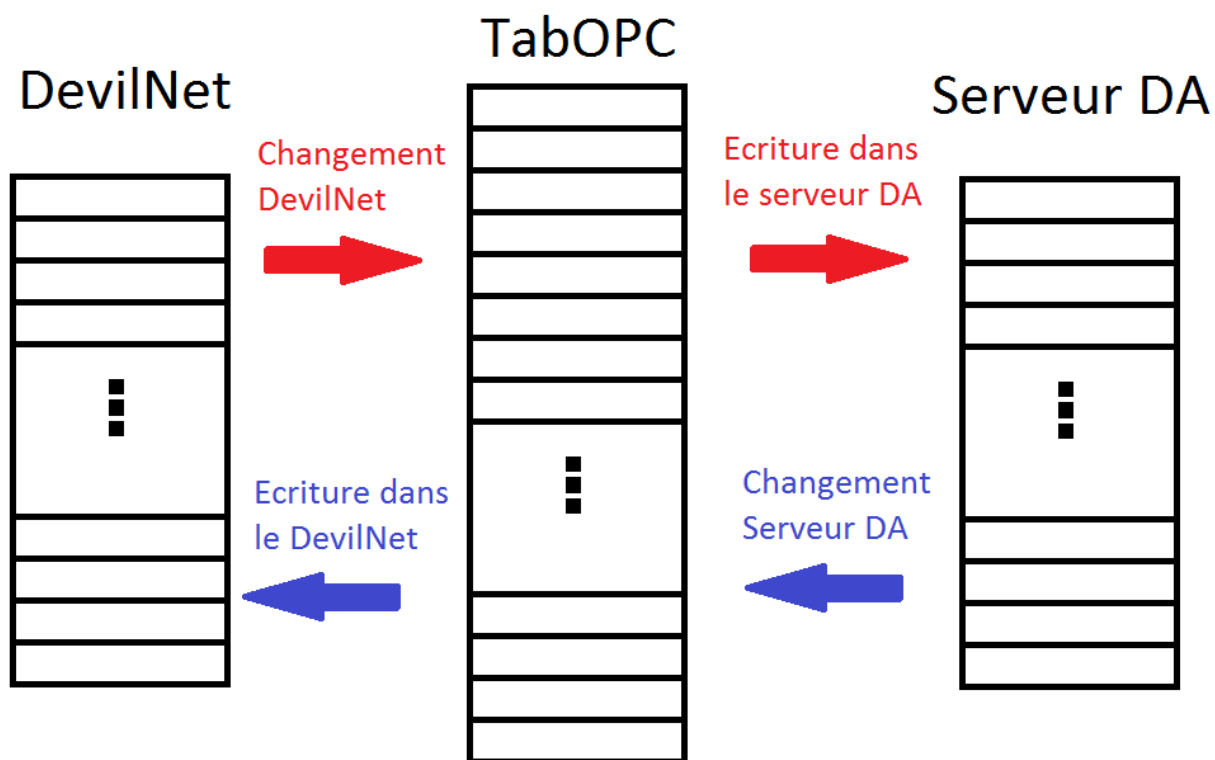


Figure 23-Vue d'ensemble du client OPC

Dans mon client OPC se trouve un tableau qui contient des données mémorisées. Dans le serveur DA et le DevilNet se trouvent les données en temps réel. Lorsqu'une valeur de commande du DevilNet est modifiée, automatiquement, mon logiciel devra renvoyer la valeur modifiée au serveur DA. Inversement, si une valeur de commande se trouvant dans le serveur DA est modifiée, mon client renvoie la nouvelle valeur au DevilNet.

Pour être plus clair, si un PIR du DevilNet passe de l'état OFF à l'état ON parce qu'il a détecté une présence. Le contrôleur détecte le changement par l'intermédiaire de mon client OPC et va mettre à jour, sur le serveur OPC, l'item correspondant. Inversement, le contrôleur détectera un changement sur le serveur OPC grâce à mon client OPC. Il s'aperçoit de la modification et envoie par l'intermédiaire du DevilNet, la commande à l'interface contenant la sortie à modifier.

De plus, un serveur gère déjà le DevilNet. Il suffit de lui envoyer une séquence pour qu'il commande une interface et renvoie alors sont ID.

Les objets OPC ont de nombreux attributs et méthodes. Les attributs OPC qui nous intéressent sont uniquement l'Item ID et la valeur de l'Item.

Pour échanger des valeurs avec le serveur DA, je dois m'y connecter. Mon client utilisera l'architecture comme ci-dessous. Dans la programmation, il faut d'abord connecter mon client au serveur DA qui crée un objet serveur OPC. L'objet OPC serveur contient une collection d'objets « OPCGroup ». Chaque objet « OPCGroup » contient une collection d'objets Items.

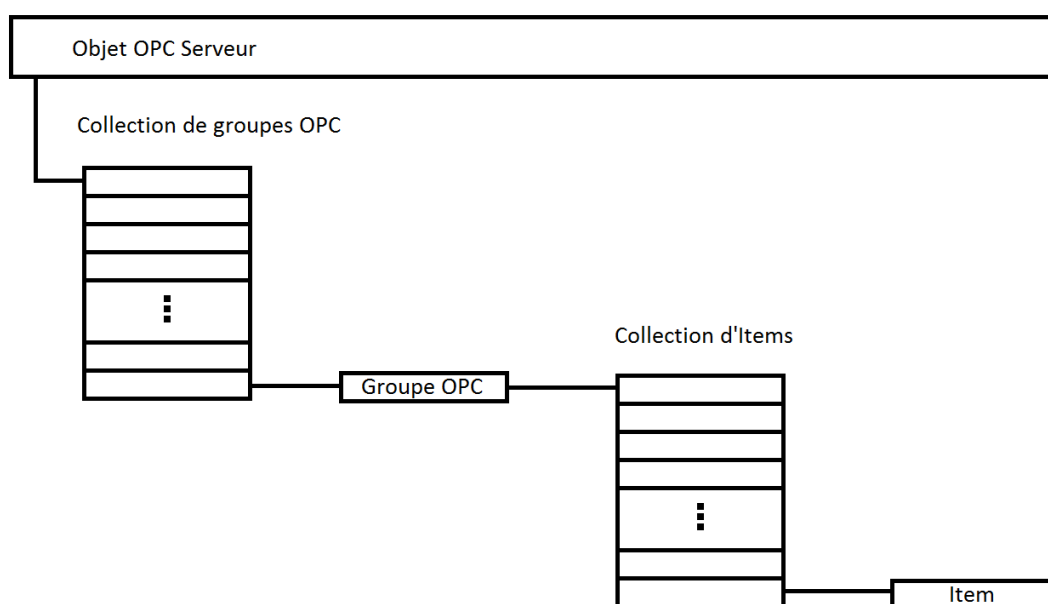


Figure 24-Structure des objets OPC du client

Un objet est un ensemble de méthodes et d'attributs. Une méthode étant un bloc de codes permettant de réaliser une action et l'attribut une valeur. Par exemple : L'objet collection d'Items a une méthode « AddItem » qui permet de lui ajouter un Item et un attribut appelé « Count » qui permet de connaître le nombre d'Items dans l'objet.

Les objets utilisés

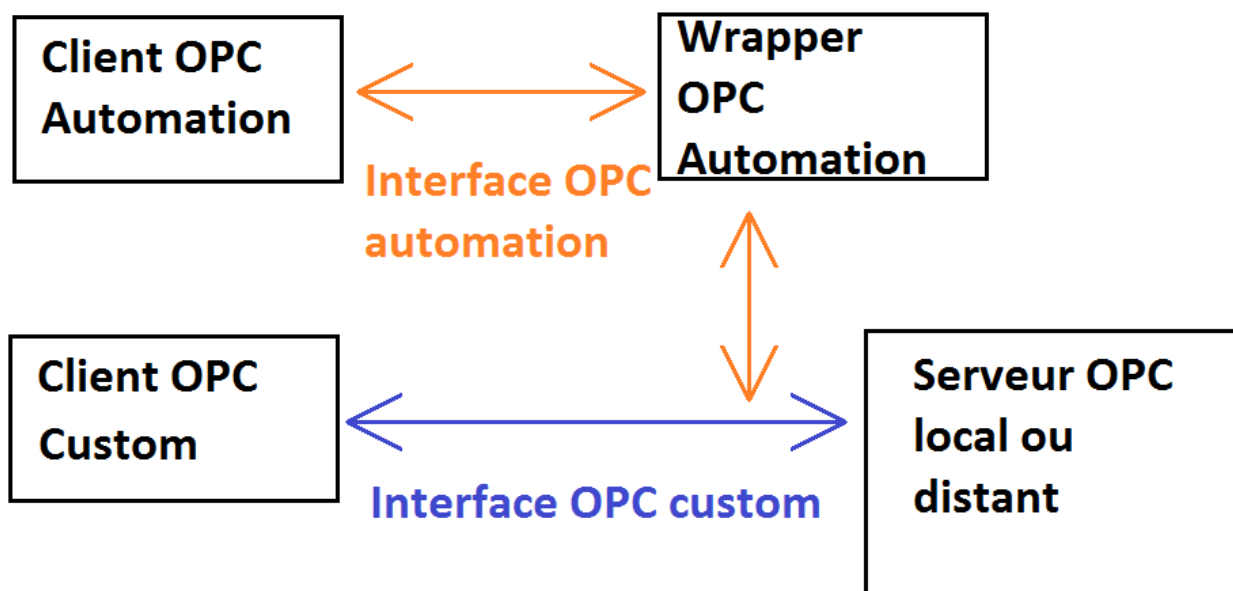
L'objet « OPCServer »

OPC peut utiliser deux interfaces différentes. L'interface custom est plus performante et l'interface automation est plus lente.

L'environnement de développement Visual Basic ne permet pas l'accès direct aux interfaces COM. L'interface COM étant celle utilisée par OPC.

Pour accéder à OPC, le Visual Basic doit passer par un « Wrapper ». L'interface custom demande exclusivement le C++ car il est capable de communiquer directement avec l'interface COM.

Voilà pourquoi j'ai dû utiliser un « OPCServer » Automation.



Une fois connecté à un serveur OPC DA, l'objet « OPCServer » peut être utilisé pour obtenir des informations sur ce serveur et peut manipuler la collection d'objets « OPCGroup ».

Chaque objet utilisé comporte de nombreux attributs, méthodes et parfois des événements. Ceux-ci sont soit utilisés dans le programme, soit ils sont importants. Toutes les informations sur les objets OPC se trouvent dans les Spécifications OPC DA 2.01 (voir annexe 2).

Attributs :

- **ServerState** : Une variable de type « Long » qui indique l'état du serveur. Chaque valeur de la variable correspond à un état du serveur.
- **ServerName** : C'est le nom du serveur OPC auquel est connecté l'objet « OPCServer ». Ce nom est celui utilisé par la méthode « Connect ».

- OPCGroups : C'est la collection d'objets « OPCGroup » appartenant à l'objet serveur.

Méthodes :

- GetOPCServers : Cette méthode permet d'obtenir la liste des serveurs OPC disponibles sur le réseau. Les noms sont transmis dans un tableau de « string ».
- Connect : Grâce au nom du serveur (=ProgID), cette méthode connecte l'objet « OPCServer » à un serveur OPC DA.
- Disconnect : Permet à un objet « OPCServer » de se déconnecter d'un serveur OPC DA.
- GetErrorString : Converti un numéro d'erreur et une « string » lisible. Le serveur renvoie la « string » à l'endroit spécifié dans la propriété au niveau du serveur « LocalID ».

L'objet « OPCGroups »

L'objet « OPCGroups » est une collection d'objets « OPCGroup » avec des méthodes qui les créent, les suppriment et les manipulent. Cet objet contient des attributs pour l'objet « OPCGroup » par défaut. Quand un objet « OPCGroups » est ajouté, le groupe par défaut est défini avec ces attributs. Ces attributs par défaut peuvent être changés, ça n'affecte pas les groupes déjà présents mais permet d'ajouter des groupes avec un état initial différent. Ceci réduit le nombre de paramètres requis pour appeler la méthode « Add ».

Attributs :

- Parent : Retourne la référence de l'objet « OPCServer » parent.
- DefaultGroupsActive : Définit si le groupe ajouté sera actif ou pas.
- DefaultGroupUpdateRate : Définit le taux de mise à jour du groupe qui sera ajouté.
- DefaultGroupDeadband : Définit la bande morte du futur groupe ajouté. Elle est exprimée en un pourcentage (valeur de 0 à 100).
- Count : Donne le nombre de groupes contenus dans l'objet « OPCGroups ».

Méthodes :

- Add : Crée un nouvel objet « OPCGroup » et l'ajoute à la collection « OPCGroups ». Les attributs du nouveau groupe sont déterminés par les attributs par défaut de l'objet « OPCGroups ». Une fois le groupe ajouté, les attributs de l'objet peuvent être modifiés.
- Remove : Supprime un groupe grâce à sa clé.
- RemoveAll : Supprime tous les groupes et ses objets « OPCItems » pour préparer la fermeture du serveur.

L'objet « OPCGroup »

« OPCGroup » fournit au client le moyen d'organiser ses données.

Attributs :

- Parent : Retourne la référence de l'objet « OPCServer » parent.
- Name : C'est le nom du groupe.
- IsActive : Détermine si le groupe est actif ou pas.
- IsSubscribed : Détermine le contrôle asynchrone du groupe. Si le « IsSubscribed » est actif (à l'état ON), le groupe pourra recevoir l'évènement « DataChange ».
- ClientHandle : C'est un chiffre « Long » définissant de manière unique le groupe. Il permet d'y accéder rapidement.
- Deadband : Bande morte du groupe en pourcentage.
- UpdateRate : Taux de mise à jour du groupe.
- OPCItems : C'est la collection d'objets « OPCItem » du groupe.

Méthodes :

- SyncRead : Effectue une lecture synchrone de la valeur, de la qualité et du « timestamp » pour un ou plusieurs Items d'un groupe.
- SyncWrite : Effectue une écriture synchrone de la valeur d'un ou plusieurs Items d'un groupe.
- AsyncRead : Effectue une lecture asynchrone de la valeur, de la qualité et du « timestamp » pour un ou plusieurs Items d'un groupe. Le résultat de la lecture est retourné via l'évènement « AsyncReadComplete » associé à l'objet « OPCGroup ».
- AsyncWrite : Effectue une écriture asynchrone de la valeur d'un ou plusieurs « Items » d'un groupe. Le résultat de l'écriture est retourné via l'évènement « AsyncWriteComplete » associé à l'objet « OPCGroup ».

Evènements :

- DataChange : L'évènement « DataChange » se déclenche lorsque la valeur ou la qualité d'un « Item » du groupe change. L'évènement est réglé sur le taux de mise à jour du groupe, il ne saurait donc pas survenir plus vite. « DataChange » est affecté par l'état actif du groupe et de l'« Item » auquel il est rattaché.
- AsyncReadComplete : Indique que la lecture asynchrone est terminée et retourne la manière dont elle s'est déroulée.

- AsyncWriteComplete : Indique que l'écriture est terminée et retourne la manière dont elle s'est déroulée.

L'objet « OPCItems »

L'objet « OPCItems » est une collection d'objets « OPCItem ». Les méthodes qu'il contient permettent de créer, de supprimer et de manipuler les objets « OPCItem ». « OPCItems » contient des attributs pour l'objet « OPCItem » par défaut. Quand un objet « OPCItem » est ajouté, il prend les attributs par défaut de l'objet « OPCItems ». Ces attributs peuvent être changés par la suite, ça n'affecte pas les « OPCItems » déjà présents mais permet d'en ajouter avec un état initial différent. Ceci réduit le nombre de paramètres requis pour appeler la méthode « AddItem ».

Attributs :

- Parent : Permet d'obtenir les références de l'objet « OPCGroup » parent.
- DefaultsActive : Détermine si l'« Item » ajouté sera actif par défaut.
- Count : Contient le nombre d'« Items » présents dans la collection.

Méthodes :

- Item : Permet de sélectionner une « Item » particulier de la collection.
- AddItem : Permet d'ajouter un « Item » à la collection avec les attributs par défaut contenus dans l'objet « OPCItems ». Ces attributs peuvent être modifiés par la suite.
- AddItems : Permet d'ajouter plusieurs « Items » à la collection avec les attributs par défaut contenus dans l'objet « OPCItems ». Ces attributs peuvent être modifiés par la suite.
- Remove : Supprime un « OPCItem » de la collection.
- SetClientHandles : Change le « Client Handle » de un ou plusieurs « Items ». Le « Client Handle » est une indexation des « Items » créée par le programmeur pour y accéder rapidement.
- SetDataType : Permet de fixer le type de données d'un ou plusieurs « Items ».

L'objet « OPCItem »

L'objet « OPCItem » représente une connexion avec une source de données du serveur OPC. A chaque donnée est associée une valeur, une qualité et un « Time Stamp ».

Attributs :

- Parent : Contient les références de l'objet « OPCGroup » parent.
- ClientHandle : Valeur de type « Long » associée à un « Item » pour pouvoir rapidement y accéder.
- ItemID : Identificateur unique de l' « Item ».
- IsActive : Active l'état d'acquisition de données de l' « Item » sélectionné.
- Value : C'est la valeur de l' « Item » sélectionné.
- Quality : Dernière qualité connue de l' « Item » sélectionné.
- TimeStamp : Dernier « Time Stamp » connus de l' « Item » sélectionné.
- CanonicalDataType : Retourne la forme canonique de l' « Item » sélectionné, c'est-à-dire sous la forme d'une variable de type « Integer ».

Méthodes :

- Read : Cette méthode permet d'effectuer un appel bloquant pour lire les trois données d'un « Item » contenu dans un serveur OPC DA. L'appel bloquant veut dire que le processus s'arrêtera jusqu'à ce que l'appel et l'action aient été réalisés.
- Write : Réalise un appel bloquant pour écrire une valeur dans l' « Item » sélectionné d'un serveur OPC DA.

Le programme

Voici la description des différentes étapes de mon programme.

Avant de commencer le programme, il a fallu aller chercher des outils VB supplémentaires.

Le client OPC a besoin des références suivantes :

- Microsoft DAO 3.51 Object Library
- OPC DA Automation Wrapper 2.02 (C'est une DLL rajoutée.)
- PaintCtl Type Library

Ainsi que les composants suivants :

-
- Microsoft Common Dialog Control 6.0
 - Microsoft Common Controls 5.0 (sp2)
 - Microsoft Winsock Control 6.0
 - NotifyIcon (C'est un module OCX, OLE Control Extension, qui me permet de gérer une icône en barre système.)

Présentation de l'interface :

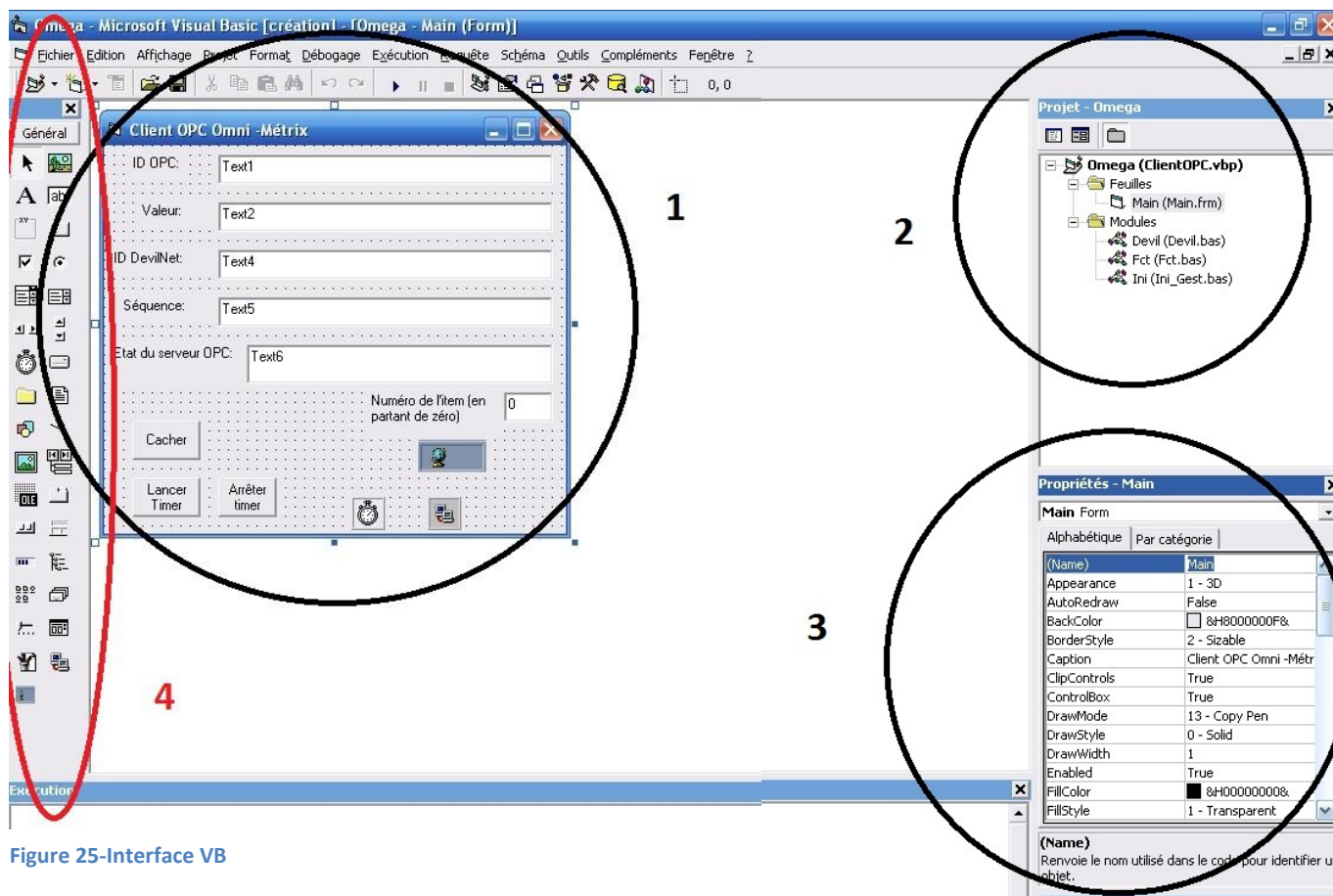


Figure 25-Interface VB

Voici donc l'interface Visual Basic 6.0.

- En 1, on peut voir ce qu'affichera mon programme une fois lancé. Mon programme VB (Visual Basic) est composé d'une feuille et de plusieurs fonctions.
- En 2, la feuille « Main » est en même temps une interface visuelle (en 1) et des lignes de code. Les modules sont uniquement des lignes de code composées de fonctions.
- En 3, ce sont les propriétés de la feuille « Main ». Par exemple son nom : « Main », et son « Caption », autrement dit le nom qui s'affiche dans la barre de titre : « Client OPC Omni-Métrix » (en 1).
- En 4, ce sont les composants ajoutés précédemment.

Initialisation

Une des propriétés de mon programme est configurée de telle façon qu'il démarre sur la feuille « MAIN ».
Dans les lignes, ci-dessous, plusieurs données sont déclarées comme public pour que je puisse y accéder de n'importe quelle feuille ou module (les phrases précédées de « ' » sont des commentaires) :

Public WithEvents OPCServeur As OPCServer	'Serveur auquel on se connecte
Public WithEvents OPCGroupes As OPCGroups	'Collection de groupe
Public WithEvents OPCGroup As OPCGroup	'Groupe sélectionné
Public OPCItems As OPCItems	'Collection d'Items OPC
Public OPCItem As OPCItem	'OPC Item sélectionné
Public CollItems As New Collection	'Collection d'Items

```
*****
'ItemIndex est le nombre d'Items contenus dans le tableau "TabOPC" en partant de
' zéro
*****
```

Public ItemIndex As Long

Les objets sont déclarés « WithEvents » pour permettre des évènements comme « Data Arrival » (page 46).
Tous les objets OPC, cités ci-dessus, seront les objets utilisés pour la connexion OPC.
Maintenant que les objets sont créés, passons à la sélection du mode d'affichage, et initialisation de
ItemIndex et Colltems.

```
*****
'Si dans le fichier "config", le mode caché est à 1, le programme est automatiquement
'caché dans la barre système.
*****
```

```
If Ini.LireINI("ModeCahé", "Mode") = "0" Then
    Main.Visible = False
Else
    Main.Visible = True
    Fct.Cacher
End If
```

```
*****
'Initialisation
*****
```

```
ItemIndex = 0
Set Colltems = Nothing
```

Connexion de la socket

Une socket est une interface logiciel avec les services du système d'exploitation permettant de communiquer avec un réseau utilisant le protocole TCP/IP. Une socket est composée d'une adresse IP et d'un Port.

En VB, l'objet qui permet de gérer les sockets s'appelle : « Winsock ». Je lui ai donné le nom « TCPDevil ». Voici la marche à suivre pour se connecter au serveur du DevilNet.

```
Public Function ConnectionAuDevilNet()
```

```
Dim Adr As String
Dim Port As String
```

```
*****
'Pour se connecter au DevilNet, j'ai utilisé une socket. Son adresse IP et son
```

'port sont précisés dans le fichier config. La socket peut être utilisée en TCP
'ou en UDP, je l'ai utilisée en TCP.

```
Main.TcpDevil.RemoteHost = Ini.LireINI("DevilNet", "Adr")
Main.TcpDevil.RemotePort = Ini.LireINI("DevilNet", "Port")
Main.TcpDevil.Protocol = sckTCPProtocol
Main.TcpDevil.Connect
```

End Function

Connexion au serveur

Connectons-nous au serveur OPC grâce à l'objet « OPCServer » public précédemment créé. À la création de l'objet serveur OPC, un objet « OPCGroups » lui est attribué.

En rouge la situation dans la hiérarchie OPC :

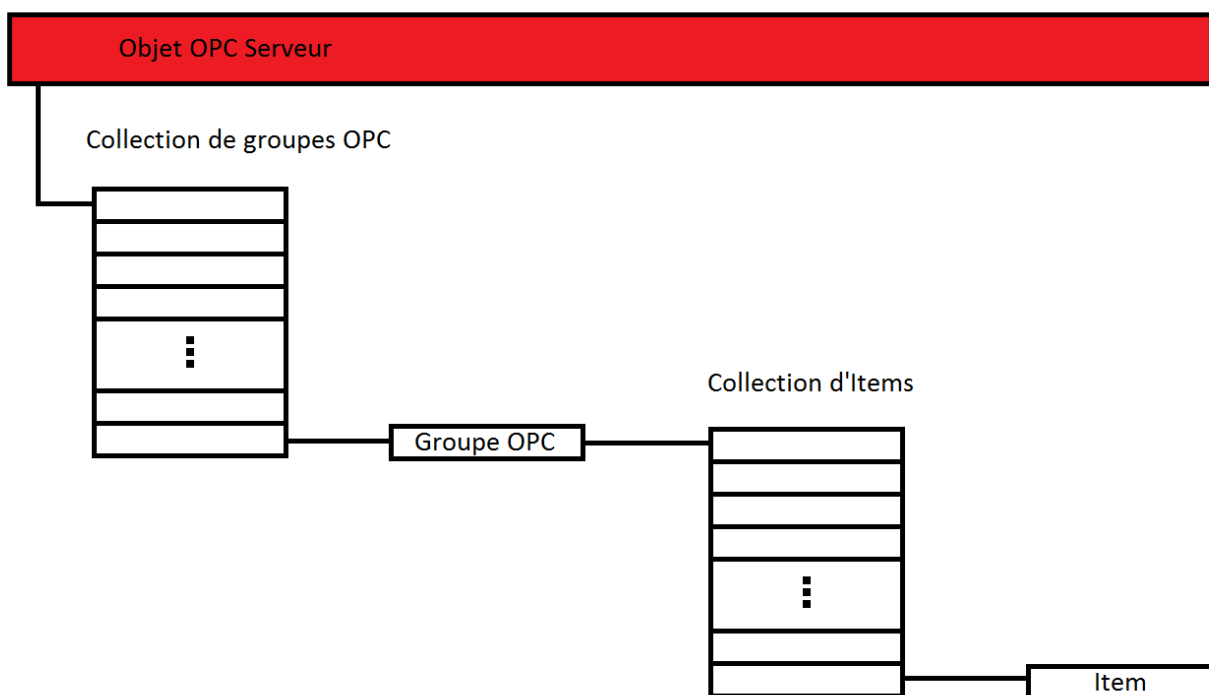


Figure 26-Connexion au serveur OPC

Pour se connecter au serveur OPC, il faut son Prog ID. Dans le cas du serveur Kepware, il s'appelle « Kepware.KEPServerEX.V5 ». Il me suffit alors de demander à mon objet serveur OPC de se connecter au serveur OPC ayant pour Prog ID « Kepware.KEPServerEX.V5 ». Le Prog ID du serveur se trouve dans un fichier « .ini » nommé : « config.ini ». J'utilise le module INI pour lire ce fichier.


```
Public Function FctServeurConnection()
```

```
Dim ProgID As String
Dim Variable As Variant
```

```
'*****
'Recherche le progID du serveur dans le fichier config.
'*****
```

```
ProgID = Ini.LireINI("ServeurOPC", "Nom")
```

```
Set Main.OPCServeur = New OPCServer
```

```
Main.OPCServeur.Connect ProgID
```

```
End Function
```

Connexion au groupe

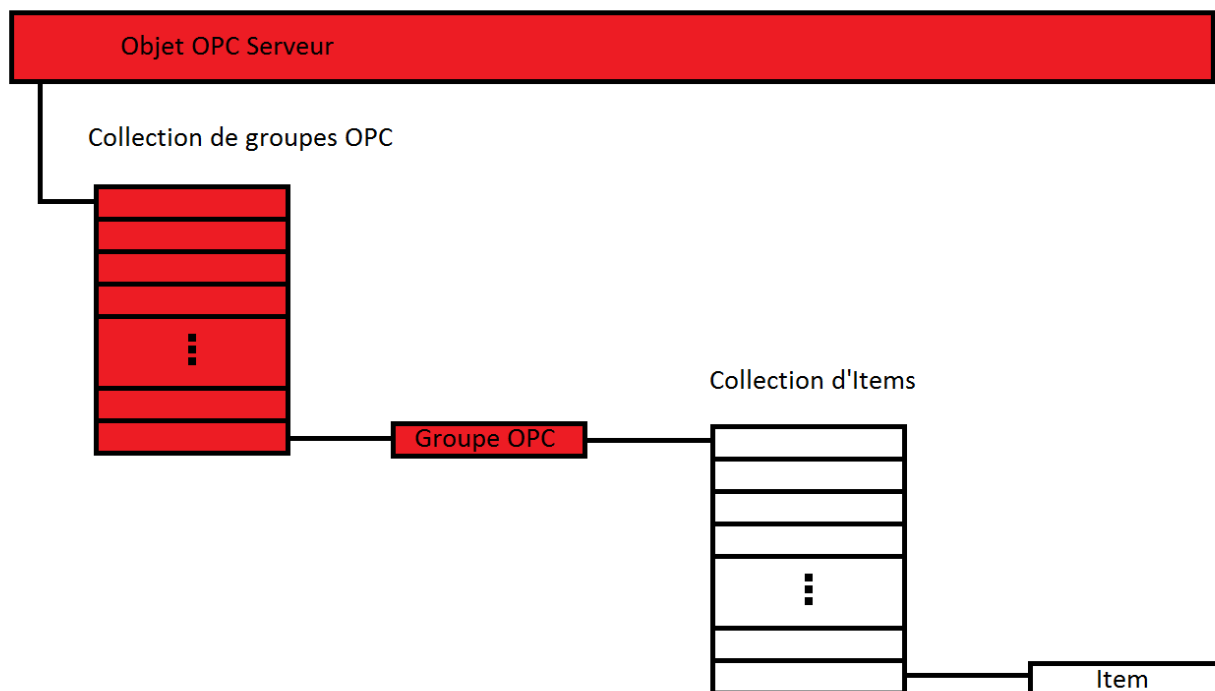


Figure 27-Connexion au groupe OPC

Pour ajouter un groupe, il faut qu'il soit ajouté au serveur auquel on se connecte. J'ai repris la collection de groupes, je l'ai déclarée publique et j'ai réalisé un « set » dessus. La collection de groupes publique devient alors la collection de groupes appartenant à l'objet serveur OPC

Public Function FctNouveauGroup()

Dim Nom As String

```
*****  
'Ajout de la collection de groupes au serveur  
*****
```

Set Main.OPCGroups = Main.OPCSeurveur.OPCGroups

```
*****  
'Initialisation des valeurs par défauts des propriétés des groupes  
*****
```

Main.OPCGroups.DefaultGroupsActive = True
Main.OPCGroups.DefaultGroupUpdateRate = 100
Main.OPCGroups.DefaultGroupDeadband = 0

```
*****  
'Ajout du groupe se trouvant dans le fichier config  
*****
```

Nom = Ini.LireINI("Group", "Nom")

Set Main.OPCGroup = Main.OPCGroups.Add(Nom)

Dans la dernière ligne de code, j'ajoute à la collection de groupes, un groupe dont j'aurai préalablement choisi le nom. Comme pour le serveur, le nom se trouve dans le fichier « config.ini ».

Connexion à l'Item

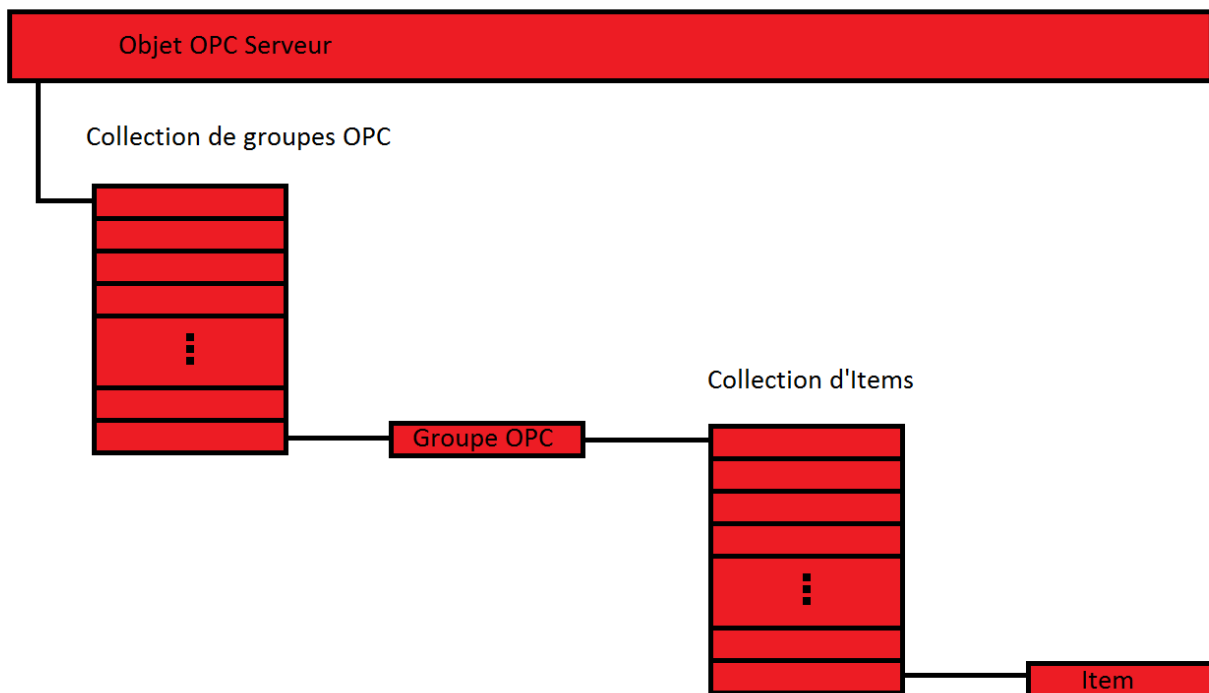


Figure 28-Connexion à l'item OPC

L'ajout d'un Item se fait de façon similaire à celui d'un groupe. D'abord faire correspondre une collection d'Items au groupe sélectionné avec l'instruction « set ». Ensuite, il faut donner un Item ID à l'Item.

L'Item ID est un nom qui permet de l'identifier de manière unique.

Dans le cas du serveur Kepware : « Channel ».« Device ».« Group ».« Nom du Tag »

Par exemple :

Channel= ShowRoomAns

Device= Salon

Groupe= Devil vers OPC

Nom du Tag= TL1

Item ID = ShowRoomAns.Salon.Devil vers OPC.TL1

Pour s'adresser rapidement à un Item OPC on utilise un « Client Handle ». Le but étant de trouver un nom qui définit de façon unique un Item. J'ai pris une solution simple, j'ai donné un chiffre à chaque Item. Ce chiffre commence à zéro, le « Client Handle » correspond au numéro de la case de mon tableau « Tab OPC ».

```
Public Function FctNouvelItem(NomTabOPC As String)
```

```
Dim NbrOPC As Integer      'Correspond au nombre d'Items dans le fichier
Dim ItemID As String
Dim ClientHandle As Long   'Le client handle doit désigner de façon unique un
                           'Item OPC. J'ai choisi d'y faire correspondre un chiffre.
```

```
Dim i As Integer
Dim j As Integer
Dim Item As OPCItem
```

```
*****
'Recherche du nombre d'Items OPC à ajouter au tableau.
*****
```

```
NbrOPC = Ini.LireINI(NomTabOPC, "Nbr")
```

```
ClientHandle = Main.ItemIndex 'Si cette fonction n'est pas utilisée pour la
                              'première fois, l'index repart du dernier Item
                              'OPC enregistré dans le tableau.
```

```
*****
'Dimensionnement du tableau "OPC".
*****
```

```
ReDim TabOPC(ClientHandle + NbrOPC)
```

```
For i = 1 To NbrOPC
```

```
ItemID = Ini.LireINI(NomTabOPC, CStr(i)) 'Rechercher l'ItemID de l'Item
```

```
Set Main.OPCItem = Main.OPCItems.AddItem(ItemID, ClientHandle) 'Ajout d'un nouvel Item au groupe principal
```

```
Main.OPCItem.IsActive = True           'Donne au nouvel Item la valeur active et un
Main.OPCItem.RequestedDataType = 0     'type "natif".
```

```

Main.CollItems.Add Main.OPCItem, CStr(ClientHandle) 'Ajout de l'Item à la collection d'Items
                                                    '(plus simple à manipuler).
TabOPC(ClientHandle).ItemID = Main.OPCItem.ItemID 'Ajout de l'ItemID et la valeur de l'Item au tableau.

TabOPC(ClientHandle).Valeur = CByte(Main.OPCItem.Value)

*****
'Insertion de l'ID dans le tableau.
*****

TabOPC(ClientHandle).DeviID = Ini.LireINI("ID", CStr(i))

ClientHandle = ClientHandle + 1
Next i

Main.ItemIndex = ClientHandle

*****
'Insertion des séquences.
*****

For i = 1 To NbrOPC

    Fct.TabOPC(CStr(i - 1)).Seq = Ini.LireINI("DeviNet", CStr(i))

Next i

End Function
    
```

Les séquences sont des commandes que j’envoie au serveur du DeviNet pour changer l’état d’une entrée/sortie. Les séquences des Items sont inscrites dans mon fichier « config.ini ».

Pour créer le tableau central de mon programme « TabOPC », j’ai dû réaliser une structure qui me permettrait d’avoir les 4 informations qui m’intéressent : Item ID, Valeur, Devi ID et la Séquence. Chaque ligne de mon tableau a la structure que j’ai choisie.

```

Public TabOPC() As StrucOPC

*****
'Structure du tableau OPC
*****

Type StrucOPC
    ItemID As String
    Valeur As Byte
    DeviID As String
    Seq As String
End Type
    
```

Data Arrival

Le « Data Arrival » est un évènement qui est déclenché lorsque l'objet Winsock (TcpDevil) reçoit une donnée. Concrètement, à l'envoi d'une séquence au serveur du DevilNet, il renvoie le Devil ID et la valeur de l'entrée/sortie modifiée. Lors de l'évènement « Data Arrival », celui-ci appelle la fonction « Compare ». Elle va comparer la valeur modifiée de l'Item et celle en mémoire dans le tableau pour voir s'il faut modifier la valeur du serveur OPC DA.

Si les deux valeurs ne correspondent pas, la fonction « Compare » se charge de modifier la valeur du tableau « TabOPC » puis de modifier la valeur du serveur DA. C'est nécessaire pour éviter un phénomène de boucle que j'expliquerai au « Data Change » (page48).

```
Public Function Compare(Str As String) ' "Str" est la chaine de caractères reçue à l'évènement "Data Change".

    Dim Pos As Long
    Dim i As Long
    Dim j As Long

    Pos = 1

    For i = 0 To Main.ItemIndex ' Cette première boucle est là pour chercher
                                ' tous les IDs que l'on pourrait recevoir en
                                ' une seule string.

        Pos = InStr(Pos, Str, "#ID") ' Recherche dans une chaine de caractère (string).

        If Pos = 0 Then

            Exit For
        End If

        Pos = Pos + 3

    For j = 0 To Main.ItemIndex ' Cette deuxième boucle recherche la correspondance
                                ' entre l'ID du DevilNet et l'ID Devil d'un Item du
                                ' tableau "TabOPC".

        If Fct.TabOPC(CStr(j)).DevilID = Mid$(Str, Pos, 4) Then

            If Fct.TabOPC(CStr(j)).Valeur <> CByte(Mid$(Str, Pos + 5, 2)) Then
                Fct.Ecrire CLng(j), CByte(Mid$(Str, Pos + 5, 2))

                *****
                ' Les débugs qui suivent me permettent de comprendre ce que je reçois
                ' et comment mon programme a trié l'information.
                *****

                Debug.Print ("DATAARRIVAL:")
                Debug.Print (Fct.TabOPC(CStr(j)).ItemID)
                Debug.Print ("TabOPC " + CStr(Fct.TabOPC(CStr(j)).Valeur) + "DevilNet " + (CStr(Mid$(Str, Pos + 5,
                2))))

                Debug.Print (CByte(Mid$(Str, Pos + 5, 2)))
            End If
        End If
    End For
End Function
```

```
Fct.TabOPC(CStr(j)).Valeur = CByte(Mid$(Str, Pos + 5, 2)) 'Modifie la valeur du bon Item dans le
tableau

Fct.Ecrire j, CByte(Mid$(Str, Pos + 5, 2)) 'Ecrit la nouvelle valeur dans le serveur OPC DA

End If

End If

Next j

Next i

End Function
```

Data Change

L'évènement "Data Change" est appelé lors du changement d'un Item. Cet évènement fait partie de l'objet OPCGroup, c'est pour cette raison qu'il est déclaré « WithEvents ». Précédemment, j'ai parlé du « Data Arrival ». Lorsque la fonction « compare » a modifié la valeur du « TabOPC » et puis modifié la valeur du serveur DA. Le « Client Handle » de l'Item modifié est envoyé à l'évènement. Une fois le « Client Handle » acquis, le programme vérifie la valeur de l'Item avec celle du « TabOPC ». Si elles sont identiques, il n'y aura pas d'envoi de séquences pour modifier les entrées/sorties du DevilNet.

C'est pour éviter tout phénomène de boucle que les comparaisons se font avant toute modification du serveur OPC DA vers le DevilNet et inversement.

```
Private Sub OPCGroup_DataChange(ByVal TransactionID As Long, ByVal NumItems As Long, ClientHandles() As Long,
ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)
```

```
Dim i As Integer
Dim Item As OPCItem
Dim Bool As Boolean
```

```
For i = 1 To NumItems
```

```
*****
'Sélectionne le bon Item de la collection grâce au clienthandles.
Set Item = Main.ColItems.Item(CStr(ClientHandles(i)))
*****
```

```
*****
'Puisque dans mon tableau je n'utilise que des bytes, je dois faire une conversion
'spéciale lorsque je converti un bool vers un byte.
*****
```

```
If Item.CanonicalDataType = vbBoolean Then
```

```
    If Fct.TabOPC(CStr(ClientHandles(i))).Valeur > 0 Then
        Bool = True
    Else
        Bool = False
    End If
```

```
    If Bool <> Item.Value Then
        'N'envoie la séquence que si la valeur du tableau est
        'différente de celle du serveur OPC DA.
        'Ça évite les boucles dans le programme.
```

```
        Debug.Print ("DATACHANGE:")
        Debug.Print (Fct.TabOPC(CStr(ClientHandles(i))).ItemID)
        Debug.Print (CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq))
```

```
        If Item.Value = True Then
            Fct.TabOPC(CStr(ClientHandles(i))).Valeur = 1 'Ecrit la nouvelle valeur dans le tableau
```



```

Main.TcpDevil.SendData (CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq)) 'Envoie la
nouvelle valeur au DevilNet
Else
Fct.TabOPC(CStr(ClientHandles(i))).Valeur = 0 'Ecrit la nouvelle valeur dans le tableau
Main.TcpDevil.SendData (CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq)) 'Envoie la
nouvelle valeur au DevilNet
End If
End If

Else 'Si c'est un Byte:
If Fct.TabOPC(CStr(ClientHandles(i))).Valeur <> Item.Value Then 'N'envoie la séquence que si la
'valeur du tableau est différente de celle du
'serveur OPC DA. Ca évite les boucles dans le programme.

Debug.Print ("DATACHANGE:")
Debug.Print (Fct.TabOPC(CStr(ClientHandles(i))).ItemID)
Debug.Print (CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq))

Fct.TabOPC(CStr(ClientHandles(i))).Valeur = Item.Value
Main.TcpDevil.SendData (CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq))
End If

End If

Next i

End Sub

```

Supervision

Pour pouvoir superviser les valeurs de mon tableau, j'ai réalisé une lecture des informations grâce à un « Timer ». A chaque fois que le temps du « Timer » est écoulé, les lignes de codes qui suivent vont effectuer une lecture des informations contenues dans « TabOPC » et le statut du serveur OPC.

```
Private Sub Timer1_Timer()
```

```
Dim It As OPCItem
```

```
*****
```

```
'Visionner les valeurs du tableau
```

```
*****
```

```
Main.Text1.Text = TabOPC(Main.Text3.Text).ItemID
```

```
Main.Text2.Text = TabOPC(Main.Text3.Text).Valeur
```

```
Main.Text4.Text = TabOPC(Main.Text3.Text).DeviID
```

```
Main.Text5.Text = TabOPC(Main.Text3.Text).Seq
```

```
*****
```

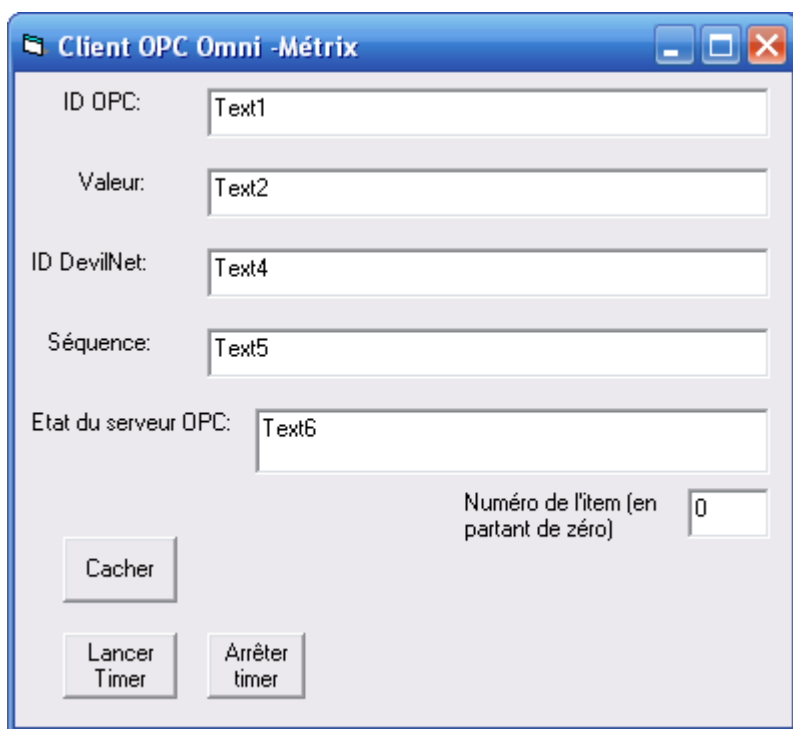
```
'Donne le status du serveur OPC
```

```
*****
```

```
Fct.StatusduServeur
```

```
End Sub
```

Voici ce qui s'affiche à l'écran :



Bien évidemment, les « Text+numéro » sont remplacés par la vraie valeur. Il y a un bouton pour lancer le « Timer » et un pour l'arrêter afin de ne pas utiliser les ressources de l'ordinateur inutilement. Cette interface n'est utile pratiquement que pour le débogage et sera la plupart du temps cachée dans la barre système comme on peut le faire avec le bouton « Cacher ».

Figure 29-Interface du client OPC

Notify Icon

Puisque mon programme doit pouvoir tourner en arrière-plan discrètement, je l'ai adapté pour qu'il s'exécute dans la barre système. Ainsi, lors de son démarrage, il s'y cache automatiquement. Pour atteindre cet objectif, j'ai eu besoin du composant « NotifyIcon.OCX », et d'ajouter un objet « NotifyIconCtrl ». Je lui ai attribué une image pour qu'une petite icône Omni-Métrix apparaisse dans la barre système.

Un clic sur le bouton « Cacher » me permet d'appeler la fonction « Cacher » qui réduit mon programme dans la barre système et cliquer deux fois sur l'icône d'Omni-Métrix la fait réapparaître.

```
Public Function Cacher()
```

```
    Main.Visible = False  
    Main.NotifyIconCtrl1.Enabled = True  
    Main.NotifyIconCtrl1.Visible = True
```

```
End Function
```

```
Private Sub NotifyIconCtrl1_DblClick(Button As Integer)
```

```
    '*****
```

```
    'Affiche le programme à l'écran lorsque l'on click 2x sur l'icône dans la barre système.
```

```
    '*****
```

```
    Main.NotifyIconCtrl1.Visible = False  
    Main.NotifyIconCtrl1.Enabled = False
```

```
    Main.Visible = True
```

```
End Sub
```

Conclusion

Les différentes étapes méthodologiques ont été plus fructueuses que je n'osais l'imaginer.

Apprendre à programmer, taper, enchaîner, organiser des lignes de codes, telle est l'étape essentielle que j'ai franchie.

Mes connaissances en programmation OPC, Visual Basic sont maintenant réelles.

La programmation d'une interface visuelle a aussi été réalisée.

J'ai mis en pratique la notion de socket vue au cours de télécommunication en utilisant Winsock.

L'étape la plus cruciale était d'affronter un sujet complètement inconnu et de gérer mon stress de départ.

Ces quinze semaines m'ont fait vivre la transition entre le monde étudiant et le monde du travail.

Je connais mieux maintenant la vie d'une petite entreprise foisonnant de projets.

Mon client est terminé, Omni-Métrieux peut s'interfacer à OPC et ainsi interagir avec de grandes marques.

Le DevilNet n'est plus confiné dans son protocole et peut maintenant s'adapter à tous les réseaux.

Bibliographie

Livres

Michel Condemine, **OPC Le Livre**

Wolfgang Mahnke, Stefan-Helmut Leitner, Matthias Damm, **OPC Unified Architecture**, Springer

Sites internet

www.OPCFondation.org

www.4CE-Industry.com

www.Microsoft.com

www.Matrikon.com

www.Kepware.fr

www.Omni-metrix.com

Autres

Dictionnaire Larousse

Spécification OPC 2.01

Présentation OPC de Technifutur

Liste des figures

Figure 1-Esplanade de Louvain-la-Neuve	5
Figure 2-Médiacité.....	6
Figure 3-Smart Office.....	6
Figure 4-OPC DA	10
Figure 5-Structure du serveur OPC.....	11
Figure 6-Entreprise sans OPC	12
Figure 7-Entreprise avec réseau propriétaire.....	14
Figure 8-Solution DRIVER	15
Figure 9-Entreprise avec OPC	16
Figure 10-Entreprise Omni-Métrix avec OPC	17
Figure 11-Les interfaces.....	19
Figure 12-Entreprise simple.....	19
Figure 13-Entreprise avec système centralisé.....	20
Figure 14-Entreprise avec le système Omni-Métrix	20
Figure 15-Interface 4 DI.....	21
Figure 16-Schéma de câblage d'une interface 4 entrées	22
Figure 17-CAT 5e	24
Figure 18-CAT 6	24
Figure 19-CAT 7	24
Figure 20-Solution serveur DA.....	25
Figure 21-Interface serveur Kepware	26

Figure 22-Solution "passerelle"	28
Figure 23-Vue d'ensemble du client OPC	29
Figure 24-Structure des objets OPC du client.....	30
Figure 25-Interface VB.....	37
Figure 26-Connexion au serveur OPC	40
Figure 27-Connexion au groupe OPC	41
Figure 28-Connexion à l'item OPC.....	43
Figure 29-Interface du client OPC.....	50

Annexes

Annexe 1 : Le programme

Feuille « Main »

```

'*****
'*
'* Nom du programme: CLientOPC
'* Version: 1.00
'* Date: 01/06/2011
'* Programmeur: J.Simon
'*
'*****
'*
'* Description
'*
'* Ce programme permet d'échanger des données entre le DevilNet et OPC.
'* Lorsque une valeur d'un Item du serveur OPC DA auquel il est branché change,*
'* le ClientOPC envoie une sequence correspondante au DevilNet. Si une valeur *
'* du DevilNet est modifiée, le ClientOPC envoie la nouvelle valeur au serveur *
'* OPC DA. Tous les paramètres de configurations se trouve dans le fichier : *
'* "config.ini"
'*****
'*
'* Modifications:                               Date:
'*****
'*
'*
'*
'*****

```

Option Explicit

```

Public WithEvents OPCServeur As OPCServer      'Serveur auxquels on se connecte
Public WithEvents OPCGroups As OPCGroups      'Collection de groupe
Public WithEvents OPCGroup As OPCGroup        'Groupe sélectionné
Public OPCItems As OPCItems                  'Collection d'items OPC
Public OPCItem As OPCItem                    'OPC item sélectionné
Public ColItems As New Collection             'Collection d'items

```

```

'*****
'ItemIndex est le nombre d'items contenus dans le tableau "TabOPC" en partant de
'zéro
'*****

```

```

Public ItemIndex As Long
Private Sub Command1_Click()

```

```

'*****
'Affiche les inforamtions d'un item contenues dans "TabOPC": ID OPC, ID
DevilNet,
'valeur et séquence
'*****

```

```
Main.Timer1.Enabled = True

End Sub

Private Sub Command2_Click()

'*****
'Appel la fonction qui cache le programme dans la barre système.
'*****
Fct.Cacher
End Sub

Private Sub Command3_Click()

'*****
'Désactive le timer1 qui permet de mettre à jour l'affichage des informations
'sur les items du "TabOPC"
'*****
Main.Timer1.Enabled = False

End Sub

Private Sub Form_Load()

'*****
'Dans cet évènement se trouve les actions principales exécutées par le programme
'*****

'*****
'Si dans le fichier "config", le mode caché est à 1, le programme est
automatiquement
'caché dans la barre système.
'*****

If Ini.LireINI("ModeCahé", "Mode") = "0" Then
Main.Visible = False
Else
Main.Visible = True
Fct.Cacher
End If

'*****
'Initialisation
'*****
```

```
ItemIndex = 0  
Set ColItems = Nothing
```

```
Devil.ConnectionAuDevilNet
```

```
*****  
'Crée la connection avec le serveur
```

```
*****  
Fct.FctServeurConnection
```

```
*****  
'Assigne la collection de groupes au serveur et en sélectionne un pour y  
ajouter  
'une collection d'items.
```

```
*****  
Fct.FctNouveauGroup
```

```
*****  
'Ajout de la collection d'items au serveur.
```

```
*****  
Set Main.OPCItems = Main.OPCGroup.OPCItems
```

```
*****  
'Ajout des items à la collection d'items.
```

```
*****  
Fct.FctNouvelItem "TabOPC"
```

```
End Sub
```

```
Private Sub Form_Unload(Cancel As Integer)
```

```
*****  
'Déconnecte le client du serveur une fois le programme terminé.
```

```
*****  
OPCServeur.Disconnect
```

```
End Sub
```

```
Private Sub NotifyIconCtrl11_DblClick(Button As Integer)
```

```
*****
```

'Affiche le programme à l'écran lorsque l'on click 2x sur l'icone dans la barre
'system.

```
*****
Main.NotifyIconCtrl1.Visible = False
Main.NotifyIconCtrl1.Enabled = False

Main.Visible = True
```

End Sub

```
*****
'"Data change" est un évènement déclenché par la modification d'une valeur au
'niveau du serveur OPC DA.
*****
```

```
Private Sub OPCGroup_DataChange(ByVal TransactionID As Long, ByVal NumItems As
Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long,
TimeStamps() As Date)
```

```
    Dim i As Integer
    Dim Item As OPCItem
    Dim Bool As Boolean
```

```
    For i = 1 To NumItems
```

```
*****
        'Sélectionne le bon item de la collection grâce au clienthandles.
*****
        Set Item = Main.ColItems.Item(CStr(ClientHandles(i)))
```

```
*****
        'Etablit la conversion de la valeur de l'Item en un
byte.
```

```
*****
        If Item.CanonicalDataType = vbBoolean Then

            If Fct.TabOPC(CStr(ClientHandles(i))).Valeur
> 0 Then

                Bool = True
            Else
                Bool = False
            End If
```

```
                If Bool <> Item.Value Then
'N'envoie la séquence que si la valeur du tableau est différente de celle du
'serveur OPC DA. Ca évite les boucles dans le programme.
```

```

Debug.Print ("DATACHANGE:")
Debug.Print
(Fct.TabOPC(CStr(ClientHandles(i))).ItemID)
Debug.Print
(CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq))

If Item.Value = True Then
Fct.TabOPC(CStr(ClientHandles(i))).Valeur = 1 'Ecrit la nouvelle valeur dans le
tableau
Main.TcpDevil.SendData
(CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq)) 'Envoie la nouvelle valeur au
DevilNet
Else
Fct.TabOPC(CStr(ClientHandles(i))).Valeur = 0 'Ecrit la nouvelle valeur dans le
tableau
Main.TcpDevil.SendData
(CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq)) 'Envoie la nouvelle valeur au
DevilNet
End If
End If

Else
'Si c'est un Byte:
If Fct.TabOPC(CStr(ClientHandles(i))).Valeur
<> Item.Value Then 'N'envoie la séquence que si la valeur du tableau est
différente de celle du
'serveur OPC DA. Ca évite les boucles dans le programme.
Debug.Print ("DATACHANGE:")
Debug.Print
(Fct.TabOPC(CStr(ClientHandles(i))).ItemID)
Debug.Print
(CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq))

Fct.TabOPC(CStr(ClientHandles(i))).Valeur
= Item.Value
Main.TcpDevil.SendData
(CStr(Fct.TabOPC(CStr(ClientHandles(i))).Seq))
End If

End If

Next i

End Sub

```

```
Private Sub TcpDevil_Connect()
```

```
    MsgBox ("Connecté")
```

```
    '*****
    'Activer IsSubscribed revient à activer l'évènement "data change"
    'de l'OPCGroup.
    '*****
    Main.OPCGroup.IsSubscribed = True
```

```
End Sub
```

```
'*****
'"Data arrival" correspond à l'évènement qui se produit lorsque une donnée
'arrive à la socket du client OPC.
'*****
```

```
Private Sub TcpDevil_DataArrival(ByVal bytesTotal As Long)
```

```
    Dim strData As String
```

```
    Main.TcpDevil.GetData strData 'Récupère la string envoyée à la socket.
    Fct.Compare strData
```

```
End Sub
```

```
'*****
'Le timer1 est là pour permettre un petit affichage de données dans la forme
main.
'*****
```

```
Private Sub Timer1_Timer()
```

```
    Dim It As OPCItem
```

```
    '*****
    'Visionner les valeurs du tableau.
    '*****
```

```
    Main.Text1.Text = TabOPC(Main.Text3.Text).ItemID
    Main.Text2.Text = TabOPC(Main.Text3.Text).Valeur
    Main.Text4.Text = TabOPC(Main.Text3.Text).DevilID
    Main.Text5.Text = TabOPC(Main.Text3.Text).Seq
```

```
'*****
'Donne le statut du serveur OPC.
'*****
```

Fct.StatusduServeur

End Sub

Module « Devil »

```

*****
!*
!* Nom du programme: CLientOPC
!* Version: 1.00
!* Date: 01/06/2011
!* Programmeur: J.Simon
!*
*****
!*
!* Description
!*
!* Ce programme permet d'échanger des données entre le DevilNet et OPC.
!* Lorsque une valeur d'un Item du serveur OPC DA auquel il est branché change,*
!* le ClientOPC envoie une sequence correspondante au DevilNet. Si une valeur *
!* du DevilNet est modifiée, le ClientOPC envoie la nouvelle valeur au serveur *
!* OPC DA. Tous les paramètres de configurations se trouve dans le fichier : *
!* "config.ini"
*****
!*
!* Modifications:                               Date:
*****
!*
!*
!*
*****

```

Option Explicit

```

*****
' Cette fonction établit la connection au DevilNet
*****

```

Public Function ConnectionAuDevilNet()

```

    Dim ADR As String
    Dim Port As String

```

```

*****
    ' Pour se connecter au DevilNet, j'ai utilisé une socket. Son adresse IP et
son
    ' port sont précisés dans le fichier config. La socket peut être utilisée en
TCP
    ' ou en UDP, je l'ai utilisée en TCP.

```

```

*****
    Main.TcpDevil.RemoteHost = Ini.LireINI("DevilNet", "Adr")
    Main.TcpDevil.RemotePort = Ini.LireINI("DevilNet", "Port")
    Main.TcpDevil.Protocol = sckTCPProtocol
    Main.TcpDevil.Connect

```


End Function

Module « Fct »

```

!*****
!*
!* Nom du programme: CLientOPC
!* Version: 1.00
!* Date: 01/06/2011
!* Programmeur: J.Simon
!*
!*****
!*
!* Description
!*
!* Ce programme permet d'échanger des données entre le DevilNet et OPC.
!* Lorsque une valeur d'un Item du serveur OPC DA auquel il est branché change,*
!* le ClientOPC envoie une sequence correspondante au DevilNet. Si une valeur *
!* du DevilNet est modifiée, le ClientOPC envoie la nouvelle valeur au serveur *
!* OPC DA. Tous les paramètres de configurations se trouve dans le fichier : *
!* "config.ini"
!*****
!*
!* Modifications:                               Date:
!*****
!*
!*
!*
!*
!*****

```

Option Explicit

```

!*****
!Structure du tableau OPC
!*****

```

```

Type StrucOPC
    ItemID As String
    Valeur As Byte
    DevilID As String
    Seq As String
End Type

```

```

Public TabOPC() As StrucOPC

```

```

!*****
!Se connect avec le serveur dont le ProgID est contenu dans le fichier config.ini
!*****

```

```

Public Function FctServeurConnection()

```

```
Dim ProgID As String
Dim Variable As Variant
```

```
'*****
'Recherche le progID du serveur dans le fichier config.ini
'*****
```

```
ProgID = Ini.LireINI("ServeurOPC", "Nom")
```

```
Set Main.OPCServeur = New OPCServer
```

```
Main.OPCServeur.Connect ProgID
```

```
End Function
```

```
'*****
'Renvoie le status du serveur OPC
'*****
```

```
Public Function StatusduServeur()
```

```
Dim chiffre As Long
```

```
' Get the Server State property of the connected server
```

```
' The value returned will be one of the following
```

```
' OPC_STATUS_RUNNING - 1 - Server is running normally
```

```
' OPC_STATUS_FAILED - 2 - Vendor specific fatal error has occurred
```

```
' OPC_STATUS_NOCONFIG - 3 - Server Running but no Configuration Data
```

```
available
```

```
' OPC_STATUS_SUSPENDED - 4 - Server is suspended and not receiving data
```

```
' OPC_STATUS_TEST - 5 - Server in test mode
```

```
' OPC_STATUS_DISCONNECTED - 6 - Server has disconnected
```

```
chiffre = Main.OPCServeur.ServerState
```

```
Select Case chiffre
```

```
Case 1
```

```
Main.Text6.Text = "Fonctionne Normalement"
```

```
Case 2
```

```
Main.Text6.Text = "Erreur fatal"
```

```
Case 3
```

```
Main.Text6.Text = "Fonctionne sans retour configuration de donnée
```

```
valable"
```

```
Case 4
```

```
Main.Text6.Text = "Serveur à l'arrêt et ne reçoit pas de données"
```

```

Case 5
    Main.Text6.Text = "Serveur en mode test"
Case 6
    Main.Text6.Text = "Serveur déconnecté"

```

```
End Select
```

```
End Function
```

```

'*****
'Ajoute le groupe dont le nom est contenu dans le fichier config.ini
'*****

```

```
Public Function FctNouveauGroup()
```

```
    Dim Nom As String
```

```

'*****
'Ajout de la collection de groupes au serveur
'*****

```

```
    Set Main.OPCGroupes = Main.OPCServeur.OPCGroupes
```

```

'*****
'Initialisation des valeurs par défauts des propriétés des groupes
'*****

```

```

    Main.OPCGroupes.DefaultGroupIsActive = True
    Main.OPCGroupes.DefaultGroupUpdateRate = 100
    Main.OPCGroupes.DefaultGroupDeadband = 0

```

```

'*****
'Ajout du groupe se trouvant dans le fichier config
'*****

```

```
    Nom = Ini.LireINI("Group", "Nom")
```

```
    Set Main.OPCGroup = Main.OPCGroupes.Add(Nom)
```

```
End Function
```

```

'*****
'Fonction qui permet d'écrire une valeur dans le serveur OPC
'*****

```

```
Public Function Ecrire(ClientHandle As Long, Valeur As Variant)
```

```
    Dim NumItems As Long          'le nombre d'item
```

```

Dim ServerHandles(1) As Long      'tableau contient le numéro de l'item à
modifier

Dim TransactionID As Long        'ID de transaction

Dim Errors() As Long            'tableau de long qui décrit l'état d'écriture
de                               'chaque item

Dim Cancel As Long              'génééré par le serveur TransactionID, cancel
transaction                       'permet au client d'interrompre la

Dim Item As OPCItem             'Item OPC

Dim Value(1) As Variant         'Tableau de valeurs à transmettre

NumItems = 1
TransactionID = 1

Set Item = Main.ColItems.Item(CStr(ClientHandle))
ServerHandles(1) = Item.ServerHandle
Value(1) = Valeur

Main.OPCGroup.AsyncWrite NumItems, ServerHandles, Value, Errors,
TransactionID, Cancel

End Function

'*****
'Création des tableaux + ajout des items
'*****

Public Function FctNouvelItem(NomTabOPC As String)

Dim NbrOPC As Integer
Dim ItemID As String
Dim ClientHandle As Long      'Le client handle doit désigner de façon unique
un                               'item OPC. J'ai choisi d'y faire correspondre un
chiffre.

Dim i As Integer
Dim j As Integer
Dim Item As OPCItem

'*****
'Recherche du nombre d'item OPC à ajouter au tableau.
'*****

```

```

NbrOPC = Ini.LireINI(NomTabOPC, "Nbr")

ClientHandle = Main.ItemIndex 'si cette fonction n'est pas utilisée pour la
item                          'première fois, l'index repart du dernier
                              'OPC enregistré dans le tableau.

'*****
'Dimensionnement du tableau "OPC".
'*****

ReDim TabOPC(ClientHandle + NbrOPC)

For i = 1 To NbrOPC
    ItemID = Ini.LireINI(NomTabOPC, CStr(i))
'Recherche de l'ItemID de l'item
    Set Main.OPCItem = Main.OPCItems.AddItem(ItemID, ClientHandle) 'Ajout
d'un nouvel item au group principale

    Main.OPCItem.IsActive = True 'Donne au nouvel item la
valeur active
    Main.OPCItem.RequestedDataType = 0 'type comme "natif"

    Main.ColItems.Add Main.OPCItem, CStr(ClientHandle) 'Ajout de l'item à la
collection d'items
manipuler 'plus simple à

    TabOPC(ClientHandle).ItemID = Main.OPCItem.ItemID 'Ajout de l'itemID et
de la valeur de l'item au tableau

    TabOPC(ClientHandle).Valeur = CByte(Main.OPCItem.Value)

'*****
'Insertion de l'ID dans le tableau.
'*****

TabOPC(ClientHandle).DevidID = Ini.LireINI("ID", CStr(i))

ClientHandle = ClientHandle + 1

```

Next i

Main.ItemIndex = ClientHandle

'*****
'Insertion des séquences

'*****

For i = 1 To NbrOPC

Fct.TabOPC(CStr(i - 1)).Seq = Ini.LireINI("DevilNet", CStr(i))

Next i

End Function

'*****
'Cette fonction est appelée lorsque il y a un "data arrival" à la socket.
'Concrètement, elle retrouve l'item du tableau auquel correspond l'ID du DevilNet
'qu'il à reçus. Ensuite, la fonction compare change la valeur de cette item.
'*****

Public Function Compare(Str As String)

Dim Pos As Long
Dim i As Long
Dim j As Long

Pos = 1

For i = 0 To Main.ItemIndex
chercher

en

'Cette première boucle est là pour

'tout les IDs que l'on pourrait recevoir

'une seule string.

Pos = InStr(Pos, Str, "#ID")

If Pos = 0 Then

Exit For
End If

Pos = Pos + 3

For j = 0 To Main.ItemIndex
correspondance

'Cette deuxième boucle recherche la

```

'entre l'ID du DevilNet et l'ID Devil
d'un item du
' tableau "TabOPC".
If Fct.TabOPC(CStr(j)).DevilID = Mid$(Str, Pos, 4) Then

    If Fct.TabOPC(CStr(j)).Valeur <> CByte(Mid$(Str, Pos + 5,
2)) Then
        Fct.Ecrire CLng(j), CByte(Mid$(Str, Pos +
5, 2))

'*****
'Les débogs qui suivent me permettent de
comprendre ce que je reçois
'et comment mon programme à trié
l'information
'*****
Debug.Print ("DATAARRIVAL:")
Debug.Print (Fct.TabOPC(CStr(j)).ItemID)
Debug.Print ("TabOPC " +
CStr(Fct.TabOPC(CStr(j)).Valeur) + "DevilNet " + (CStr(Mid$(Str, Pos + 5, 2))))
Debug.Print (CByte(Mid$(Str, Pos + 5,
2)))

Fct.TabOPC(CStr(j)).Valeur =
CByte(Mid$(Str, Pos + 5, 2)) 'Modifie la valeur du bon item dans le tableau

Fct.Ecrire j, CByte(Mid$(Str, Pos + 5,
2)) 'Ecrit la nouvelle valeur dans le serveur OPC DA

End If

End If

Next j

Next i

End Function

'*****
'Fonction pour cacher le programme dans la barre système.
'*****

Public Function Cacher()

```



```
Main.Visible = False  
Main.NotifyIconCtrl1.Enabled = True  
Main.NotifyIconCtrl1.Visible = True
```

```
End Function
```

Module « Ini »

```
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias
"GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As
Any, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As
Long, ByVal lpFileName As String) As Long
```

```
Private Declare Function WritePrivateProfileString Lib "kernel32" Alias
"WritePrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName
As Any, ByVal lpString As Any, ByVal lpFileName As String) As Long
```

```
Function LireINI(Entete As String, Variable As String) As String
    Dim Retour As String
    Fichier = App.Path & "\" & "config.ini"
    Retour = String(255, Chr(0))
    LireINI = Left$(Retour, GetPrivateProfileString(Entete, ByVal Variable, "",
Retour, Len(Retour), Fichier))
End Function
```

```
Function EcrireINI(Entete As String, Variable As String, Valeur As String) As
String
    Fichier = App.Path & "\" & "config.ini"
    WriteINI = WritePrivateProfileString(Entete, Variable, Valeur, Fichier)
End Function
```

```
' Pour l'executer ex :
'EcrireINI("MonEntete", "MaVariable", "MaValeur")
'LireINI("MonEntete", "MaVariable")
```

Annexe 2 : Les spécifications OPC



**Data Access Automation Interface Standard
Version 2.01
January 6, 1999**

Synopsis: This specification is an interface for developers of OPC clients and OPC Data Access Servers. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of servers and clients by multiple vendors that shall inter-operate seamlessly together. This document defines the OPC Data Access OLE Automation interface for developers of OPC clients and OPC Data Access Servers. The purpose of this specification is to provide an OLE Automation interface for the OPC Data Access Server Custom Interface Functionality

<u>Documentation Type</u>	<u>Industry Standard Specification</u>		
<u>Title:</u>	<u>OPC Data Access Automation Specification</u>	<u>Date:</u>	<u>January 6, 1999</u>
<u>Version:</u>	<u>2.01</u>	<u>Soft</u>	<u>MS-Word</u>
		<u>Source:</u>	<u>opcda20_auto.doc</u>
<u>Author:</u>	<u>OPC Foundation</u>	<u>Status:</u>	<u>Release</u>

Trademarks: Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here. Required Runtime Environment: This specification requires Windows 95/98 (with DCOM installed), Windows NT 4.0 or later. It is recommended that Windows NT 4.0 machines be run with SP3, or later.

NON-EXCLUSIVE LICENSE AGREEMENT

The OPC Foundation, a non-profit corporation (the “OPC Foundation”), has established a set of standard OLE/COM interface protocols intended to foster greater interoperability between automation/control applications, field systems/devices, and business/office applications in the process control industry.

The current OPC specifications, prototype software examples and related documentation (collectively, the “OPC Materials”), form a set of standard OLE/COM interface protocols based upon the functional requirements of Microsoft’s OLE/COM technology. Such technology defines standard objects, methods, and properties for servers of real-time information like distributed process systems, programmable logic controllers, smart field devices and analyzers in order to communicate the information that such servers contain to standard OLE/COM compliant technologies enabled devices (e.g., servers, applications, etc.).

The OPC Foundation will grant to you (the “User”), whether an individual or legal entity, a license to use, and provide User with a copy of, the current version of the OPC Materials so long as User abides by the terms contained in this Non-Exclusive License Agreement (“Agreement”). If User does not agree to the terms and conditions contained in this Agreement, the OPC Materials may not be used, and all copies (in all formats) of such materials in User’s possession must either be destroyed or returned to the OPC Foundation. By using the OPC Materials, User (including any employees and agents of User) agrees to be bound by the terms of this Agreement.

LICENSE GRANT:

Subject to the terms and conditions of this Agreement, the OPC Foundation hereby grants to User a non-exclusive, royalty-free, limited license to use, copy, display and distribute the OPC Materials in order to make, use, sell or

otherwise distribute any products and/or product literature that are compliant with the standards included in the OPC Materials.

All copies of the OPC Materials made and/or distributed by User must include all copyright and other proprietary rights notices include on or in the copy of such materials provided to User by the OPC Foundation.

The OPC Foundation shall retain all right, title and interest (including, without limitation, the copyrights) in the OPC Materials, subject to the limited license granted to User under this Agreement.

WARRANTY AND LIABILITY DISCLAIMERS:

User acknowledges that the OPC Foundation has provided the OPC Materials for informational purposes only in order to help User understand Microsoft's OLE/COM technology. THE OPC MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. USER BEARS ALL RISK RELATING TO QUALITY, DESIGN, USE AND PERFORMANCE OF THE OPC MATERIALS. The OPC Foundation and its members do not warrant that the OPC Materials, their design or their use will meet User's requirements, operate without interruption or be error free.

IN NO EVENT SHALL THE OPC FOUNDATION, ITS MEMBERS, OR ANY THIRD PARTY BE LIABLE FOR ANY COSTS, EXPENSES, LOSSES, DAMAGES (INCLUDING, BUT NOT LIMITED TO, DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL, SPECIAL OR PUNITIVE DAMAGES) OR INJURIES INCURRED BY USER OR ANY THIRD PARTY AS A RESULT OF THIS AGREEMENT OR ANY USE OF THE OPC MATERIALS.

GENERAL PROVISIONS:

This Agreement and User's license to the OPC Materials shall be terminated (a) by User ceasing all use of the OPC Materials, (b) by User obtaining a superseding version of the OPC Materials, or (c) by the OPC Foundation, at its option, if User commits a material breach hereof. Upon any termination of this Agreement, User shall immediately cease all use of the OPC Materials, destroy all copies thereof then in its possession and take such other actions as the OPC Foundation may reasonably request to ensure that no copies of the OPC Materials licensed under this Agreement remain in its possession.

User shall not export or re-export the OPC Materials or any product produced directly by the use thereof to any person or destination that is not authorized to receive them under the export control laws and regulations of the United States.

The Software and Documentation are provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARS 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor/ manufacturer is the OPC Foundation, P.O. Box 140524, Austin, Texas 78714-0524.

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, the OPC Materials.

Revision 2.0 Highlights

This revision replaces the Data Access Automation Interface previously documented in the OPC Data Access 1.0A Specification. Basically the automation interface architecture was redesigned to address ease of use by Visual Basic Programmers, and to take advantage of the technology improvements, inclusive of automation events and object support for the WithEvents keyword.

Revision 2.01 January 6, 1999 Highlights

As noted elsewhere a draft version of the Specification was inadvertently circulated as Version 2.0. The 'correct' version 2.0 has been relabeled as 2.01 (this document), redated and republished. The basic changes between the draft dated October 14, 1998 and this document include:

- Removal of AsyncRefreshComplete (event for refresh follows custom interface architecture, with data returned in DataChange Event);

- Change to AsyncCancelComplete to return the Transaction ID associated with the method being canceled;

- Changing reference to NumItems to Count;

- Correction to OPC error numbering;

- Adding NON-EXCLUSIVE LICENSE AGREEMENT Section;

- Minor formatting changes.

1 Introduction

1.1 Background

A standard mechanism for communicating to numerous data sources, either devices on the factory floor, or a database in a control room is the motivation for this specification. The standard mechanism would consist of a standard automation interface targeted to allow Visual Basic applications, as well as other automation enabled applications to communicate to the above named data sources.

Manufacturers need to access data from the plant floor and integrate it into their existing business systems. Manufacturers must be able to utilize off the shelf tools (SCADA Packages, Databases, spreadsheets, etc.) to assemble a system to meet their needs. The key is open and effective communication architecture concentrating on data access, and not the types of data. We have addressed this need by architecting and specifying a standard automation interface to the OPC Data Access Custom interface to facilitate the needs of applications that utilize an automation interface to access plant floor data.

1.2 Purpose

What is needed is a common way for automation applications to access data from any data source like a device or a

database. The OPC Data Access Automation defines a standard by which automation applications can access process data. This interface provides the same functionality as the custom interface, but in an "automation friendly" manner.

Given the common use of Automation to access other software environments (e.g.: RDBMS, MS Office applications, WWW objects), this interface has been tailored to ease application development, without sacrificing functionality defined by the Custom interface.

The figure below shows an Automation client calling into an OPC Data Access Server using a 'wrapper' DLL. This wrapper translates between the custom interface provided by the server and the automation interface desired by the client. Note that in general the connection between the Automation Client and the Automation Server will be 'In Process' while the connection between the Automation Server and the Custom Server may be either In Process, Local or Remote.

Automation Client

OPC Automation Wrapper



COM / DCOM



OPC Custom Interface Server

Figure 1-1. Custom and Automation Client Applications Interfacing to OPC Servers

10

1.3 Scope

This document specifies a revised version of the OLE Automation interface that was specified in Release 1.0 of the OPC specification. There were several reasons for these revisions. The most important are as follows:

- Make the interface easier to use by the Visual Basic Programmer
- Take advantage of newer features of Visual Basic (such as events)
- Allow the creation of a common wrapper DLL which could be shared by all vendors

This document assumes that the reader is familiar with the information provided on the OPC Data Access Custom Interface Specification. That document provides an Overview of the OPC functionality as well as detailed descriptions of the behavior of the various functions.

We have deliberately not duplicated that information in an attempt to maintain consistency.

1.4 References

Kraig Brockschmidt, Inside OLE, Second Edition, Microsoft Press, Redmond, WA, 1995. Microsoft Systems Journal, Q&A, April 1996, pp. 89-101. OLE Automation Programming Reference, Microsoft Press, Redmond, WA, 1996.

OLE 2 Programming Reference, Vol. 1, Microsoft Press, Redmond, WA, 1994.

OPC Data Access Custom Interface Standard, Version 2.0, OPC Foundation 1998.

1.5 Audience

This specification is intended as reference material for developers of OPC Automation Clients that require the functionality of the OPC Data Access Custom Interface. The developer needs some knowledge of basic Automation concepts and terminology.

2 Architecture

The fundamental design goal is that this interface is intended to work as a 'wrapper' for existing OPC Data Access Custom Interface Servers providing an automation friendly mechanism to the functionality provided by the custom interface.

2.1 Functional Requirements

The automation interface provides nearly all of the functionality of the required and optional Interfaces in the OPC Data Access Custom Interface. If the OPC Data Access Custom server supports the interface, the functions and properties at the automation level will work. Automation interfaces generally do not support optional capabilities in the same way that the custom interface does. If the underlying custom interface omits some optional functionality then the corresponding automation functions and properties will exhibit some reasonable default behavior as described in more detail later in this document.

The interfaces are fully supported by VC++ and Visual Basic 5.0. They allow any application which has an OLE Automation Interface (e.g. VB, VC++, and VBA enabled applications) to access the OPC Interface, according to the limitations of the respective application.

The interface described in this specification specifically does NOT support VBScript or Java Script. A separate wrapper could be developed to accommodate the needs of VBScript and Java Script. However such an effort is outside the scope of this specification.

2.2 OPC Automation Server Object Model

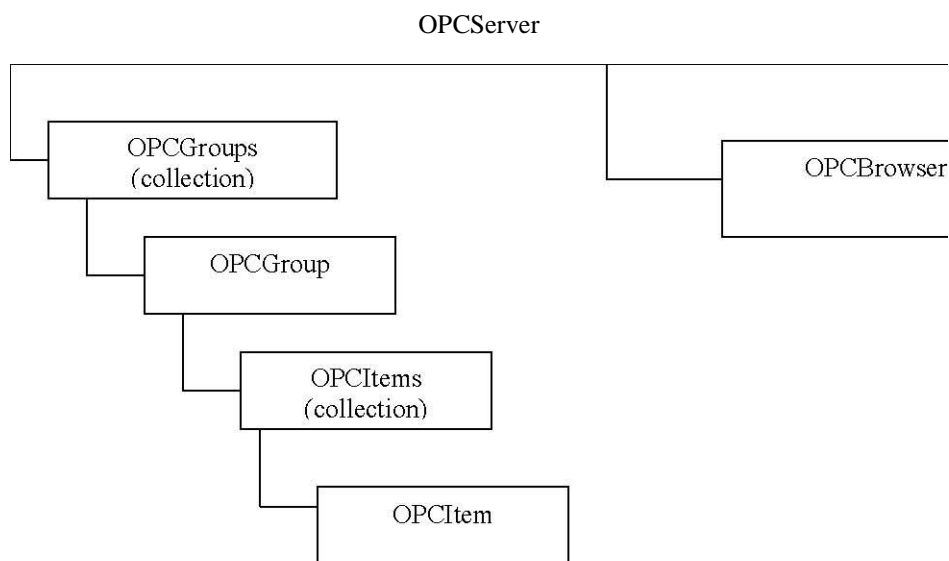


Figure 2-1. Automation Object Hierarchy

Object	Description
OPCServer	An instance of an OPC Server. You must create an OPCServer object before you can get references to other objects. It contains the OPCGroups Collection and creates OPCBrowser objects.

OPCGroups	An Automation collection containing all of the OPCGroup objects this client has created within the scope of the OPCServer that the Automation Application has connected to via the OPCServer.Connect()
OPCGroup	An instance of an OPCGroup object. The purpose of this object is to maintain state information and provide the mechanism to provide data acquisition services for the OPCItem Collection object that the OPCGroup object references.
OPCItems	An Automation collection containing all of the OPCItem objects this client has created within the scope of the OPCServer, and corresponding OPCGroup object that the Automation Application has created.
OPCItem	An automation object that maintains the item's definition, current value, status information, last update time. Note the Custom Interface does not provide a separate Item Object.
OPCBrowser	An object that browses item names in the server's configuration. There exists only one instance of an OPCBrowser object per instance of an OPC Server object.

2.3 OPC Data Access Automation Object Model

The OPCServer object provides a way to access (read/write) or communicate to a set of data sources. The types of sources available are a function of the server implementation.

13 An OPC Automation client connects to an OPC Automation Server that communicates to the underlying data source

(e.g. OPC Data Access Custom Servers) through the functionality provided by the automation objects described here. The OPCServer provides an (OPCGroups) automation collection object to maintain a collection of OPCGroup Objects. The OPCGroup object allows clients to organize the data they want to access. An OPCGroup can be activated and deactivated as a unit. An OPCGroup also provides a way for the client to 'subscribe' to the list of items so that it can be notified when they change. The OPCGroup Object provides an OPCItems collection of OPCItems. The OPCItem object provides a connection to a single data item in the underlying data source.

2.4 Data Synchronization

There is a requirement that the VB client be able to read or receive data such that the value, quality, and timestamp information are kept in sync. Basically the client needs to be assured that the quality of the data and the timestamp matches the value.

If a client obtains values using any of the Read methods it can be assured that Value, Timestamp, and Quality properties

will be in synch with each other. If a client obtains data by registering for DataChange events, then the Value, Timestamp, and Quality will be in sync within the scope of the EventHandler routine.

If a client mixes these two approaches it will be impossible for the client to ensure that the item properties are exactly in sync since an event which changed the properties could occur between the time the client accesses the various properties.

2.5 Introduction to Exceptions and Events

2.5.1 Exceptions

Most properties and methods described here communicate with an OPC Custom Server. In OLE Automation, there is no easy way to return an error when accessing a property. The best way to resolve this is for the automation server to generate an exception if such an error occurs in the underlying data source. This means that the client needs to have exception logic in place to handle errors.

Errors that occur when setting a property are reported using the standard Visual Basic Err object. Refer to Appendix A -OPC Automation Error Handling for more details on handling errors.

2.5.2 Events

The automation interface supports the event notification mechanism that is provided with Visual Basic 5.0. The Automation server triggers events in response to Async Refresh, Async Read and Async Write Method calls. In addition, Automation server triggers events when data changes according to the client specification. The implementation assumes that the Automation Client is equipped to deal with these events.

2.6 Arrays

By convention, the OPC Automation interface assumes that arrays are 1 based. If an array is passed to a function that is

larger than the Count or NumItems parameter, only Count or NumItems elements will be used, starting at index 1. This only applies to parameters for functions and events within the automation interface. This does not apply to item values, where the data type for the item value is itself an array.

To avoid errors it is suggested that VB code use “Option Base 1”.

2.7 Collections

OLE Automation collections are objects that support Count, Item, and a hidden property called _NewEnum. Any object that has these properties as part of the interface can be called a collection. In VB, a collection can be iterated using either of two idioms.

The first method explicitly uses Count and Item to index the elements of the collection.

```
For I=1 To
    object.Countel
    ement =
    object.Item(I)
    'or...element =
    object( I
)Next I
```

The second method iterates through the available items using the hidden _NewEnum function:

```
For Each element In
    object `do something
    with elementNext
    element
```

The For Each method of iterating a collection is faster than the explicit Item method. Item can also be used to access a particular index, such as Item(3). It doesn't need to be used within a loop.

2.8 Optional Parameters

Optional parameters are denoted by the keyword “Optional”. Optional parameters may be omitted from a method call if the default behavior is acceptable. OLE Automation requires that optional parameters be Dim'd as Variant, though they may hold a string, array, etc.

2.9 Method Parameters

Method parameters are assumed to be passed ByVal unless specified to be ByRef. ByRef parameters get filled in by the method and passed back.

2.10 Type Library

VB uses the OPC Automation Type Library to define the following interfaces. Make sure that (in Visual Basic 5.0) Properties | References has “OPC Automation 2.0” checked.

About the OPC Data Access Automation Wrapper DLL

The OPC foundation has provided a reference sample of the Data Access Automation interface for the OPC foundation members use in providing an automation interface to OPC data access custom interface servers. The reference sample is provided as a DLL complete with the Visual C++ source code. Vendors may provide the DLL directly with their product.

Vendors that choose to modify the source code, or even just build the DLL from the source code(unchanged) must do the following prior to including or shipping the DLL.

- 1 The name of the OPC automation DLL must be changed from OPCDAAuto.dll to a vendor specific unique name.
- 2 The name of the OPC automation IDL(opcauto.idl) file should be changed to a vendor specific unique name.
- 3 The helpstring ("OPC Automation 2.0") in the IDL file must be changed to reflect your vendor specific OPC automation interface. This is the name that shows up in the Automation Type Library. Visual Basic applications that use your vendor build OPC automation interface DLL will include the DLL by using the type library.
- 4 All guid's in the IDL file must be changed to new values that are generated by using the Guidgen tool. This is required to prevent the vendor built automation interface library from being confused with another vendors built automation library or the OPC foundation provided automation library.

The vendor is encouraged to not change the existing automation interfaces. If additional functionality is desired, a new object and interface should be added and should replicate all the functionality of the existing object that is being added to.

The OPC foundation has also provide a visual basic sample that demonstrates usage of the Data Access Automation interface. This sample is intended only to demonstrate the functionality of the OPC data access automation interface.

4 OPC Data Access Automation Objects & Interfaces

4.1 OPCServer Object

Description A client creates the OPCServer Automation object. The client then 'connects' it to an OPC Data Access Custom Interface (see the 'Connect' method). The OPCServer object can now be used to obtain general information about an OPC server and to create and manipulate the collection of OPCGroup objects.'

Syntax OPCServer

Remarks The WithEvents syntax enables the object to support the declared events for the particular object. For the OPCServer, the only event defined is the ServerShutDown. The OPCGroup (described later) has all the events associated with DataChange and the events as required to support the Asynchronous methods of the OPCGroup object.

Example Dim WithEvents AnOPCServer As OPCServer Set AnOPCServer = New OPCServer

4.1.1 Summary of Properties

StartTime	CurrentTime	LastUpdateTime
MajorVersion	MinorVersion	BuildNumber
VendorInfo	ServerState	LocaleID
Bandwidth	OPCGroups	PublicGroupNames
ServerName	ServerNode	ClientName

4.1.2 Summary of Methods

GetOPCServers	Connect	Disconnect
CreateBrowser	GetErrorString	QueryAvailableLocaleIDs
QueryAvailableProperties	GetItemProperties	LookupItemIDs

4.1.3 Summary of Events

ServerShutDown		
----------------	--	--

4.1.4 OPCServer Properties

4.1.4.1 StartTime 4.1.4.2 CurrentTime

Description (Read-only) Returns the time the server started running. This is the start time of the server that the client has specified to connect to. Multiple Clients connecting to the same server can be assured that each client will read the same value from the server for this property.

Syntax `StartTime As Date`

Remarks The automation server is expected to use the custom interface `GetStatus ()` to obtain the values for this property as well as many of the other properties defined as properties of the `OPCServer`. An error occurs if the client has not connected to a Data Access Server via the `Connect` method.

Example

```
Dim AnOPCServerTime As Date AnOPCServerTime = AnOPCServer.StartTime
```

Description (Read-only) Returns the current time from the server. When you access this property, you will get the value that the automation server has obtained from the custom server via the `GetStatus ()` interface.

Syntax `CurrentTime As Date`

Remarks An error occurs if the client has not connected to a Data Access Server via the `Connect` method.

Example

```
Dim AnOPCServerTime As Date AnOPCServerTime = AnOPCServer.CurrentTime
```

4.1.4.3 LastUpdateTime

Description (Read-only) Returns the last update time from the server. When you access this property, you will get the value that the automation server has obtained from the custom server via the `GetStatus()` interface.

Syntax `LastUpdateTime As Date`

Remarks Returns the last time data was sent from the server to a client application. An error occurs if the client has not connected to a Data Access Server via the `Connect` method.

Example

```
Dim AnOPCServerTime As Date AnOPCServerTime = AnOPCServer.LastUpdateTime
```

4.1.4.4 MajorVersion

Description(Read-only) Returns the major part of the server version number (e.g. the “1” in version 1.32). When you access this property, you will get the value that the automation server has obtained

from the custom server via the GetStatus() interface.

Syntax MajorVersion As Integer

RemarksAn error occurs if the client has not connected to a Data Access Server via the Connect method.

```
ExampleDim AnOPCServerMajorVersion As  
StringAnOPCServerMajorVersion =  
Str(AnOPCServer.MajorVersion)
```

4.1.4.5 MinorVersion

Description(Read-only) Returns the minor part of the server version number (e.g. the “32” in version 1.32). When you access this property, you will get the value that the automation server has obtained from the custom server via the GetStatus () interface.

Syntax MinorVersion As Integer

RemarksAn error occurs if the client has not connected to a Data Access Server via the Connect method.

```
ExampleDim AnOPCServerMinorVersion As  
StringAnOPCServerMinorVersion =  
Str(AnOPCServer.MinorVersion)
```

4.1.4.6 BuildNumber

Description(Read-only) Returns the build number of the server. When you access this property, you will get the value that the automation server has obtained from the custom server via the GetStatus () interface.

Syntax BuildNumber As Integer

RemarksAn error occurs if the client has not connected to a Data Access Server via the Connect method.

```
ExampleDim BuildNumber as IntegerBuildNumber =  
AnOPCServer.BuildNumber
```

4.1.4.7 VendorInfo

Description(Read-only) Returns the vendor information string for the server. When you access this property, you will get the value that the automation server has obtained from the custom server via the GetStatus () interface.

Syntax VendorInfo As String

RemarksAn error occurs if the client has not connected to a Data Access Server via the Connect method.

```
ExampleDim info As Stringinfo =  
AnOPCServer.VendorInfo
```

4.1.4.8 ServerState

Description (Read-only) Returns the server's state, which will be one of the OPCServerState values:

Syntax ServerState As Long

Setting	Description
OPC_STATUS_RUNNING	The server is running normally. This is the usual state for a server
OPC_STATUS_FAILED	A vendor specific fatal error has occurred within the server. The server is no longer functioning. The recovery procedure from this situation is vendor specific. An error code of E_FAIL should generally be returned from any other server method.
OPC_STATUS_NOCONFIG	The server is running but has no configuration information loaded and thus cannot function normally. Note this state implies that the server needs configuration information in order to function. Servers which do not require configuration information should not return this state.
OPC_STATUS_SUSPENDED	The server has been temporarily suspended via some vendor specific method and is not getting or sending data. Note that Quality will be returned as OPC_QUALITY_OUT_OF_SERVICE.

OPC_STATUS_TEST The server is in Test Mode. The outputs are disconnected from the real hardware but the server will otherwise behave normally.

Inputs may be real or may be simulated depending on the vendor implementation. Quality will generally be returned normally.

Remarks These are the server states that are described in the OPC Data Access Custom Interface Specification, and returned by an OPC server via the custom interface. Refer to the OPC Data Access Custom Interface Specification IOPCServer::GetStatus() for more details. When you access this property, you will get the value that the automation server has obtained from the custom server via the GetStatus () interface. An error occurs if the client has not connected to a Data Access Server via the Connect method..

```
Example Dim ServerState As LongServerState =
AnOPCServer.ServerState
```

4.1.4.9 LocaleID

Description (Read/Write) This property identifies the locale, which may be used to localize strings returned from the server. . This LocaleID will be used by the GetErrorString method on this interface

Syntax LocaleID As Long

Remarks It should also be used as the 'default' LocaleID by any other server functions that are affected by LocaleID. An error occurs if the client has not connected to a Data Access Server via the Connect method.

```
Example `(getting the property)::Dim
    LocaleID As LongLocaleID =
    AnOPCServer.LocaleID
    `(setting the property):AnOPCServer.LocaleID = LocaleID
```

4.1.4.10 Bandwidth

Description (Read-only) This is server specific. The suggested use is the server's bandwidth as a percentage of available bandwidth. This value will be hFFFFFFFF when the server cannot calculate a bandwidth. When you access this property, you will get the value that the automation server has obtained from the custom server via the GetStatus () interface.

Syntax Bandwidth As Long

Remarks An error occurs if the client has not connected to a Data Access Server via the Connect method.

```
Example Dim Bandwidth As Long Bandwidth = AnOPCServer.Bandwidth
```

4.1.4.11 OPCGroups

Description (Read only) A collection of OPCGroup objects. This is the default property of the OPCServer object.

Syntax OPCGroups As OPCGroups

```
Example `(explicit property specification):Dim groups As OPCGroupsSet groups =
    AnOPCServer.OPCGroups
    `(using the default specification):Dim groups As OPCGroupsSet groups =
    AnOPCServer
```

4.1.4.12 PublicGroupNames

Description (Read-only) Returns the names of this server's Public Groups. These names can be used in ConnectPublicGroup. The names are returned as an array of strings.

Syntax PublicGroupNames As Variant

Remarks An error occurs if the client has not connected to a Data Access Server via the Connect method. An empty list is returned if the underlying server does not support the Public Groups interface, or if there are no public groups defined.

```
Example Dim AllPublicGroupNames As Variant
    AllPublicGroupNames = AnOPCServer.PublicGroupNames
```

4.1.4.13 ServerName

Description (Read-only) Returns the server name of the server that the client connected to via Connect().

Syntax ServerName As String

Remarks When you access this property, you will get the value that the automation server has cached locally. The ServerName is empty if the client is not connected to a Data Access Server.

Example

```
Dim info As String info = AnOPCServer.ServerName
```

4.1.4.14 ServerNode

Description (Read-only) Returns the node name of the server that the client connected to via Connect(). When you access this property, you will get the value that the automation server has cached locally.

Syntax ServerNode As String

Remarks The ServerNode is empty if the client is not connected to a Data Access Server. The ServerNode will be empty if no host name was specified in the Connect method.

Example

```
Dim info As String info = AnOPCServer.ServerNode
```

4.1.4.15 ClientName

Description(Read/Write) This property allows the client to optionally register a client name with the server. This is included primarily for debugging purposes. The recommended behavior is that the client set his Node name and EXE name here.

Syntax ClientName As String

RemarksRecommended to put NodeName and ClientName in the string, separated by a semi-colon (;). Refer to the example below for suggested syntax

Example

```

\'(getting the property):
Dim info As String
info = AnOPCServer.ClientName

\'(setting the property):
AnOPCServer.ClientName =
"NodeName;c:\programfiles\vendor\someapplication.exe"
```

4.1.5 OPCServer Methods

4.1.5.1 GetOPCServers

Description Returns the names (ProgID's) of the registered OPC Servers. Use one of these ProgIDs in the Connect method. The names are returned as an array of strings.

Syntax `GetOPCServers(Optional Node As Variant) As Variant`

Part Description

Node The Node name provides the mechanism to specify the remote node where you want the automation server to give you the list of all the registered OPC servers.

Remarks Refer to the OPC Data Access Custom Interface Standard for specific registry requirements for the custom servers.

Node is optional. The use of a node name makes use of DCOM to access another computer. Acceptable node names are UNC names ("Server"), or DNS names ("server.com", "www.vendor.com", or "180.151.19.75").

Example getting the registered OPC Servers (the real OPC servers) and adding them to a standard VB listbox.

```
Dim AllOPCServers As Variant
AllOPCServers = AnOPCServer.GetOPCServers
For i = LBound(AllOPCServers) To UBound(AllOPCServers)
    listbox.AddItem AllOPCServers(i)
Next i
```

4.1.5.2 Connect

Description Must be called to establish connection to an OPC Data Access Server (that implements the custom interface).

Syntax `Connect (ProgID As String, Optional Node As Variant)`

Part	Description
ProgID	The ProgID is a string that uniquely identifies the registered real OPC Data Access Server (that implements the custom interface).
Node	The Node name can specify another computer to connect using DCOM.

Remarks Each instance of an OPC Automation Server is "connected" to an OPC Data Access Server (which implements the custom interface).

Node is optional. The use of a node name makes use of DCOM to access another computer. Acceptable node names are UNC names ("Server"), or DNS names ("server.com", "www.vendor.com", or "180.151.19.75").

Calling this function will result in the automation wrapper calling CoCreateInstanceEx to create a Data Access Custom(specified by the ProgID)server on the specified machine(StrNodeName).

If this function is called a second time without calling explicitly calling disconnect the automation wrapper will automatically disconnect the existing connection.

See Also Use the GetOPCServers method to find the legal ProgIDs.

```
Example ` Connect to the first
registered OPCServer
returned from
theGetOPCServersDim
AlloPCServers As
VariantAlloPCServers =
AnOPCServer.GetOPCServ
ersAnOPCServer.Connect
(AlloPCServers(1)) `Con
nect to a specific
server on some remote
nodeDim ARealOPCServer
As StringDim
ARealOPCNodeName As
StringARealOPCServer =
"VendorX.DataAccessCus
tomServer"ARealOPCNode
Name =
"SomeComputerNodeName"
AnOPCServer.Connect
(ARealOPCServer,
ARealOPCNodeName)
```

4.1.5.3 Disconnect

Description Disconnects from the OPC server.

Syntax Disconnect ()

RemarksThis allows you to disconnect from a server and then either connect to another server, or remove the object. It is a good programming practice for the client application to explicitly remove the

objects that it created (including all OPCGroup(s), and OPCItem(s) using the appropriate automation method. Calling this function will remove all of the groups and release all references to the underlying OPC Custom Server.

Example `AnOPCServer.Disconnect`

4.1.5.4 CreateBrowser

Description Creates an OPCBrowser object

Syntax `CreateBrowser() As OPCBrowser`

Remarks The OPC Browse interface is an optional interface that is not required to be supported by an OPC Custom interface server. Therefore, an OPCBrowser object will not be returned for OPC Custom interface servers that do not implement the browse interface.

```
Example Dim ARealOPCServer As String Dim ARealOPCNodeName As
String ARealOPCServer =
"VendorX.DataAccessCustomServer" ARealOPCNodeName =
"SomeComputerNodeName" AnOPCServer.Connect (ARealOPCServer,
ARealOPCNodeName) Dim AnOPCServerBrowserObject As
OPCBrowser Set AnOPCServerBrowserObject =
AnOPCServer.CreateBrowser
```

4.1.5.5 GetErrorString

Description Converts an error number to a readable string. The server will return the string in the Locale that is specified in the server level LocaleID property. Refer to the properties of the OPC Server for setting and getting the LocaleID property.

Syntax `GetErrorString(ErrorCode As Long) As String`

Part Description

ErrorCode Server specific error code that the client application had returned from an interface function from the server, and for which the client application is requesting the server's textual representation.

Example `Dim AnOPCServerErrorString As String`

` for this sample, assume while adding some items, we detected that one of the items was `invalid. Not all code included for clarity reasons.

```
AnOPCItemCollection.Add AddItemCount,
AnOPCItemIDs, AnOPCItemServerHandles, AnOPCItemErrors'Get the
error string and display it to tell the user why the item could
not be added
```

```
AnOPCServerErrorString =
```

```
AnOPCServer.GetErrorString(AnOPCItemErrors (index))
```

```
`and more codeErrorBox.Text = AnOPCServerErrorString`and more code
```

4.1.5.6 QueryAvailableLocaleIDs

Description Return the available LocaleIDs for this server/client session. The LocaleIDs are returned as an array of longs.

Syntax QueryAvailableLocaleIDs () As Variant

Example

```
Dim LocaleID As Variant
Dim AnOPCTextSting as String
AnOPCServerLocaleID = AnOPCServer.QueryAvailableLocaleIDs()
For i = LBound(LocaleID) To UBound(LocaleID)
AnOPCTextSting = LocaleIDToString(LocaleID(i))
listbox.AddItem AnOPCTextSting
Next i
```

4.1.5.7 QueryAvailableProperties

Description Return a list of ID codes and Descriptions for the available properties for this ItemID. This list may differ for different ItemIDs. This list is expected to be relatively stable for a particular ItemID. That is, it could be affected from time to time by changes to the underlying system's configuration.

Syntax QueryAvailableProperties (ItemID As String, ByRef Count As Long, ByRef PropertyIDs() as Long, ByRef Descriptions() As String, ByRef DataTypes() As Integer)

Part	Description
ItemID	The ItemID for which the caller wants to know the available properties
Count	The number of properties returned
PropertyIDs	DWORD ids for the returned properties. These IDs can be passed to GetItemProperties or LookupItemIDs
Descriptions	A brief vendor supplied text Description of each Property. NOTE LocalID does not apply to Descriptions.
DataTypes	The datatype which will be returned for this Property by GetItemProperties.

```

Example` Get the available propertiesDim OPCItemID As
StringDim ItemCount As LongDim PropertyIDs() As
LongDim Descriptions() As StringDim DataTypes() As
IntegerDim AnOPCTextSting As StringOPCItemID =
"SomeOPCDataAccessItem"AnOPCServer.QueryAvailablePrope
rties (OPCItemID, ItemCount,
PropertyIDs, Descriptions,
DataTypes)Fori=1To
ItemCountAnOPCTextSting =
Str(PropertyIDs(i)+" "+
Descriptions(i))listbox.AddItem
AnOPCTextStingNext I
    
```

4.1.5.8 GetItemProperties

Description Return a list of the current data values for the passed ID codes.

Syntax `GetItemProperties (ItemID As String, Count As Long, ByRef PropertyIDs() as Long, ByRef PropertyValue() As Variant, ByRef Errors() As Long)`

Part	Description
ItemID	The ItemID for which the caller wants to read the list of properties
Count	The number of properties passed
PropertyIDs	DWORD ids for the requested properties. These IDs were returned by QueryAvailableProperties or obtained from the fixed list described earlier.
PropertyValues	An array of size Count VARIANTS returned by the server, which contain the current values of the requested properties.
Errors	Error array indicating whether each property was returned.

```

ExampleDim OPCItemID as StringDim ItemCount As LongDim
PropertyIDs(3) as LongDim Data() as VariantDim Errors() as
LongDim AnOPCTextSting As String` Set values for ItemCount
    
```

```
and PropertyIDs...AnOPCServer.GetItemProperties (OPCItemID,
ItemCnt, PropertyIDs,
Data, Errors)Fori=1To
ItemCntAnOPCTextSting =
Str (PropertyIDs (i) + "" +
Data (i)listbox.AddItem
AnOPCTextStingNext i
```

4.1.5.9 LookupItemIDs

Description Return a list of ItemIDs (if available) for each of the passed ID codes. These indicate the ItemID, which could be added to an OPCGroup and used for more efficient access to the data corresponding to the Item Properties. An error within the error array may indicate that the passed Property ID is not defined for this item.

Syntax LookupItemIDs (ItemID As String, Count As Long, PropertyIDs () as Long, ByRef NewItemIDs () As String, ByRef Errors () As Long)

ItemID	The ItemID for which the caller wants to lookup the list of properties
Count	The number of properties passed
PropertyIDs	DWORD ids for the requested properties. These IDs were returned by QueryAvailableProperties
NewItemIDs	The returned list of ItemIDs.
Errors	Error array indicating whether each New ItemID was returned.

```
ExampleDim OPCItemID as StringDim Count As LongDim PropertyIDs(1)
as Long
Dim NewItemIDs () as StringDim Errors() as LongDim
AnOPCTextSting As StringOPCItemID =
"SomeOPCDataAccessItem"Count = 1PropertyIDs(1) =
5;AnOPCServer.LookupItemIDs (OPCItemID, Count, PropertyIDs,
NewItemIDs, Errors)Fori=1To
CountAnOPCTextSting =
Str (PropertyIDs (i) + "" +
```

```
NewItemIDs(i)listbox.AddItem
AnOPCTextStingNext i
```

4.1.6 OPCServer Events

4.1.6.1 ServerShutdown

DescriptionThe ServerShutdown event is fired when the server is planning on shutting down and wants to tell all the active clients to release any resources. The client provides this method so that the server can request that the client disconnect from the server. The client should remove all groups and items.

Syntax ServerShutdown (Reason As String)

Part	Description
ServerReason	An optional text string provided by the server indicating the reason for the shutdown.

```
ExampleDim WithEvents AnOPCServer As OPCServerDim ARealOPCServer As
StringDim ARealOPCNodeName As StringSet AnOPCServer = New
OPCServer ` note we need to specify an
example to facilitate creating an object that is'dimensioned
with eventsARealOPCServer =
"VendorX.DataAccessCustomServer"ARealOPCNodeName =
"SomeComputerNodeName"AnOPCServer.Connect(ARealOPCServer,
ARealOPCNodeName)Private Sub AnOPCServer_ServerShutdown(ByRef
aServerReason As
String)` write your client code here to let go of the serverEnd Sub
```

4.2 OPCBrowser Object

Description The OPCBrowser object is a collection of branch or item names that exist in the server. Browsing is optional. If the server does not support browsing, CreateBrowser will not create this object.

Syntax OPCBrowser

RemarksThe properties Filter, DataType, and AccessRights affect the collection at the time a method such as ShowLeafs is called. These properties let the client request a subset of the address space. If the user is browsing names of items to write data to, then the AccessRights property should be set to OPCWritable before calling ShowLeafs. Servers can have either a flat or hierarchical name space. When the namespace is flat, calling the method ShowLeafs fills the collection with the

entire set of names in the server. Hierarchical browsing is a two step process. First, the browse position is set using a Move method, then the names are put into the collection using the Show methods. Calling ShowBranches fills the collection with the branches below the current position. Calling MoveDown with one of these branch names moves the position to that branch. Calling MoveUp moves up one level. Calling MoveToRoot moves all the way to the top level. From any position, branches and leafs can be browsed.

ShowBranches and ShowLeafs should not be called from inside a For Each loop.

The reason for this restriction is when in a For Each loop or a loop to Count the items, basically you would be changing the contents of the collection, and the next item has no meaning. Basically, you should not call ShowBranches and ShowLeafs while looping through the Browse

object's collection. It is legal to call ShowLeafs while in a loop on some other collection.

```
Example Dim WithEvents AnOPCServer As OPCServer

Dim ARealOPCServer As String

Dim ARealOPCNodeName As String

Dim AnOPCServerBrowser As OPCBrowser

Dim SomeName As VariantSet AnOPCServer = New OPCServer
ARealOPCServer = "VendorX.DataAccessCustomServer"

ARealOPCNodeName = "SomeComputerNodeName"

AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)

Set browser = AnOPCServer.CreateBrowser
AnOPCServerBrowser.ShowBranches

AnOPCServerBrowser.MoveDown(AnOPCServerBrowser.Item(1)
) )
AnOPCServerBrowser.DataType = vbInteger
AnOPCServerBrowser.ShowLeafs

'1**method for getting all
namesForI=1To
AnOPCServerBrowser.Countname =
AnOPCServerBrowser.Item( I )

' Or...
```



```

    name = AnOPCServerBrowser ( I )listBox.Add name
Next
'2 method for getting all names
For Each name In AnOPCServerBrowserlistBox.Add name
Next name

```

4.2.1 Summary of Properties

Organization	Filter	DataType
AccessRights	CurrentPosition	Count

4.2.2 Summary of Methods

Item	ShowBranches	ShowLeafs
MoveUp	MoveToRoot	MoveDown
MoveTo	GetItemID	GetAccessPaths

4.2.3 OPCBrowser Properties

4.2.3.1 Organization

Description (Read-only) Returns either OPCHierarchical or OPCFlat.

Syntax Organization As Long

RemarksIf the organization is OPCFlat, then calling ShowBranches or any Move method has no effect. All names will be available after a single call to ShowLeafs.

Example Dim TheOrganization As Long

```

Set AnOPCServerBrowser =
AnOPCServer.CreateBrowserTheOrganization =
AnOPCServerBrowser.Organization

```

4.2.3.2 Filter

Description(Read/Write) The filter that applies to ShowBranches and ShowLeafs methods. This property defaults to "" (no filtering). Servers may use this filter to narrow the list of names. Servers are recommended to support wildcards such as "*".

Syntax Filter As String

See Also Appendix B – Sample String Filter Syntax Function

```

ExampleVB Syntax Example (getting the property):Dim TheFilter As
StringTheFilter = AnOPCServerBrowser.FilterVB Syntax Example

```

```
(setting the property):Dim TheFilter As
StringAnOPCServerBrowser.Filter = "FIC*"
```

4.2.3.3 DataType

Description (Read/Write) The requested data type that applies to ShowLeafs methods. This property defaults to VT_EMPTY, which means that any data type is acceptable.

Syntax DataType As Integer

Remarks Any legal Variant type can be passed as a requested data type. The server responds with names that are compatible with this data type (may be none). This property provides the mechanism such that the client only gets the leafs that are of a certain DataType.

See Also Appendix A - OPC Automation Error Handling Appendix D- Notes On Automation Data Types

Example

```
VB Syntax Example (getting the property): Dim TheDataType As
Long
TheDataType = AnOPCServerBrowser.DataType
VB Syntax Example (setting the property): Dim TheDataType As
Long
AnOPCServerBrowser.DataType = vbInteger
```

4.2.3.4 AccessRights

Description(Read/Write) The requested access rights that apply to the ShowLeafs methods. This property defaults to OPCReadable OR'd with OPCWritable (that is, everything). This property applies to the filtering, i.e. you only want the leafs with these AccessRights.

Syntax AccessRights As Long

Example

```
VB Syntax Example (getting the property):Dim TheAccessRights As
LongTheAccessRights = AnOPCServerBrowser.AccessRightsVB Syntax
Example (setting the property):
Dim TheAccessRights As LongAnOPCServerBrowser.AccessRights =
OPCWritable
```

4.2.3.5 CurrentPosition

Description (Read-only) Current position in the tree. This string will be "" initially at the "root" or if Organization is OPCFlat.

Syntax CurrentPosition As String

RemarksCurrent Position returns the absolute position, therefore the results of this could be used directly by the MoveTo method.

```
ExampleVB Syntax Example (getting the
property):Dim ACurrentPosition As
String
```

```
AnOPCServerBrowser.MoveDown("level_1")
```

```
Set ACurrentPosition = AnOPCServerBrowser.CurrentPosition
```

4.2.3.6 Count

Description (Read-only) Required property for collections. Returns the number of items in the collection.

```
Syntax Count As Long
```

```
ExampleVB Syntax Example (getting the property):Dim AnOPCCount As
LongAnOPCCount = AnOPCServerBrowser.Count
```

4.2.4 OPCBrowser Methods

4.2.4.1 Item

Description Required property for collections. Returns a name indexed by ItemSpecifier. The name will be a branch or leaf name, depending on previous calls to ShowBranches or ShowLeaves. Item is the default for the OPCBrowser.

```
Syntax Item(ItemSpecifier As Variant) As String
```

Part

Description

ItemSpecifier

1-based index into the collection

```
Example AnOPCServerBrowser.ShowBranches
```

```
'1" method for getting all names
```

```
ForI=1To AnOPCServerBrowser.CountSomeName =
AnOPCServerBrowser.Item( I )
```

```
listBox.Add SomeName
Next I
```

```
'2nd method for getting all namesForI=1To
AnOPCServerBrowser.Count

    SomeName = AnOPCServerBrowser( I )listBox.Add SomeName
Next I

'3rd method for getting all names

For Each SomeName In browserlistBox.Add SomeName
Next SomeName
```

4.2.4.2 ShowBranches

Description Fills the collection with names of the branches at the current browse position.

Syntax ShowBranches() Example AnOPCServerBrowser.ShowBranches

4.2.4.3 ShowLeafs

Description Fills the collection with the names of the leafs at the current browse position. Default for Flat is FALSE.

Syntax ShowLeafs(Optional Flat As Variant)

Part	Description
Flat	Defines what the collection should contain.

Settings The Settings for Flat are:

Settings	Description
True	the collection is filled with all leafs at the current browse position, as well as all the leafs that are below the current browse position. Basically we are treated from the current position on down as a flat name space.
False	the collection is filled with all leafs at the current browse position

Remarks The names of leafs in the collection should match the filter criteria defined by DataType, AccessRights, and Filter. Default for Flat is FALSE.

Example AnOPCServerBrowser.MoveDown ("Floor1_Mixing")
AnOPCServerBrowser.ShowLeafs

4.2.4.4 MoveUp

Description Move up one level in the tree.

Syntax `MoveUp()`

Example `AnOPCServerBrowser.MoveUp`

4.2.4.5 MoveToRoot

Description Move up to the first level in the tree.

Syntax `MoveToRoot()` Example `AnOPCServerBrowser.MoveToRoot`

4.2.4.6 MoveDown

Description Move down into this branch.

Syntax `MoveDown(Branch As String)` Example `AnOPCServerBrowser.MoveDown
("Floor1_Mixing")`

4.2.4.7 MoveTo

Description Move to an absolute position.

Syntax `MoveTo(Branches() As String)`

Part Description

Branches Branches are an array of branch names from the root to a particular position in the tree.

Remarks This method is equivalent to calling `MoveToRoot`, followed by `MoveDown` for each branch name in the array.

Example	Dim	<code>branches(3)</code>	As	String
		<code>AnOPCServerBrowser.MoveToRo</code>		
		<code>ot</code>		
		<code>branches(1) =</code>		
		<code>"node"</code>		
		<code>branches(2) =</code>		
		<code>"device"</code>		
		<code>branches(3) =</code>		
		<code>"group"</code>		

```
browser.MoveTo
branches
```

```
Set ACurrentPosition =
AnOPCServerBrowser.CurrentPosition
```

```
\ACurrentPosition is now "node.device.group"
```

4.2.4.8 GetItemID

Description Given a name, returns a valid ItemID that can be passed to OPCItems Add method.

```
Syntax GetItemID(Leaf As String) As String
```

Part

Description

Leaf

The name of a BRANCH or LEAF at the current level.

Remarks The server converts the name to an ItemID based on the current “position” of the browser. It will not correctly translate a name if MoveUp, MoveDown, etc. has been called since the name was obtained.

Example `AnOPCServerBrowser.ShowLeafs`

```
ForI=1To AnOPCServerBrowser.Count
```

```
Set AnOPCItemID = AnOPCServerBrowser.GetItemID(I)
```

```
Next I
```

```
\or
```

```
AnOPCServerBrowser.MoveDown "Mixing"
```

```
Set AnOPCItemID = AnOPCServerBrowser.GetItemID("FIC101.PV")
```

4.2.4.9 GetAccessPaths

Description Returns the strings that are legal AccessPaths for this ItemID. May be Null if there are no AccessPaths for this ItemID or the server does not support them.

```
Syntax GetAccessPaths(ItemID As String) As Variant
```

Part Description

ItemID Fully Qualified ItemID

Remarks AccessPath is the “how” for the server to get the data specified by the ItemID (the what). The client uses this function to identify the possible access paths for the specified ItemID.

```
Example AnOPCServerBrowser.ShowLeafs
        For I=1 To AnOPCServerBrowser.Count
            Set AnAccessPath = AnOPCServerBrowser.GetAccessPaths("FIC101.PV")
        Next I
        `or
        AnOPCServerBrowser.MoveDown "Mixing"
        Set AnAccessPath = AnOPCServerBrowser.GetAccessPaths("FIC101.PV")
```

4.3 OPCGroups Object

Description OPCGroups is a collection of OPCGroup objects, and the methods that create, remove, and manage them.

This object also has properties for OPCGroup defaults. When OPCGroups are added, the DefaultGroupXXXX properties set its initial state. The defaults can be changed to add OPCGroups with different initial states. Changing the defaults does not affect groups that have already been created. Once an OPCGroup is added, its properties can be modified. This reduces the number of parameters required to call the Add method.

Syntax OPCGroups

4.3.1 Summary of Properties

Parent	DefaultGroupIsActive	DefaultGroupUpdateRate
DefaultGroupDeadband	DefaultGroupLocaleID	DefaultGroupTimeBias
Count		

4.3.2 Summary of Methods

Item	Add	GetOPCGroup
Remove	RemoveAll	ConnectPublicGroup
RemovePublicGroup		

4.3.3 Summary of Events

GlobalDataChange		
------------------	--	--

Example The following sample code is necessary for the subsequent
 Syntax Visual Basic Examples to be operational. This code is referred
 Base to as OPCGroupsObjectBase.

```
Dim WithEvents AnOPCServer As OPCServer
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
Dim AnOPCServerBrowser As OPCBrowser
Dim MyGroups As OPCGroups
Dim DefaultGroupUpdateRate As Long
Dim OneGroup As OPCGroup
Dim AnOPCItemCollection As OPCItems
Dim AnOPCItem As OPCItem
Dim ClientHandles(100) As Long
Dim AnOPCItemIDs(100) As String
```

```
Dim AnOPCItemServerHandles() As Long
Dim AnOPCItemServerErrors() As Long
Set AnOPCServer = New OPCServer
Set ARealOPCServer = "VendorX.DataAccessCustomServer"
Set ARealOPCNodeName = "SomeComputerNodeName"
AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)
Set MyGroups = AnOPCServer.OPCGroups
MyGroups.DefaultGroupIsActive = True
Set OneGroup = MyGroups.Add("AnOPCGroupName")
Set AnOPCItemCollection = OneGroup.OPCItems
```

4.3.4 OPCGroups Properties

4.3.4.1 Parent

Description (Read-only) Returns reference to the parent OPCServer object.

```
Syntax Parent As OPCServer
```

4.3.4.2 DefaultGroupIsActive

Description (Read/Write) This property provides the default active state for OPCGroups created using Groups.Add.

Syntax DefaultGroupIsActive As Boolean

Remarks This property defaults to True.

Example

```
VB Syntax Example (getting the property): Dim
DefaultGroupIsActive As Boolean

DefaultGroupIsActive = MyGroups.DefaultGroupIsActive

VB Syntax Example (setting the property):
MyGroups.DefaultGroupIsActive = FALSE
```

4.3.4.3 DefaultGroupUpdateRate

Description (Read/Write) This property provides the default update rate (in milliseconds) for OPCGroups created using Groups.Add. This property defaults to 1000 milliseconds (1 second).

Syntax DefaultGroupUpdateRate As Long

Example

```
VB Syntax Example (getting the property): Dim
DefaultGroupUpdateRate As Long

DefaultGroupUpdateRate = MyGroups.DefaultGroupUpdateRate
```

VB Syntax Example (setting the property):MyGroups.DefaultGroupUpdateRate = 250

4.3.4.4 DefaultGroupDeadband

Description (Read/Write) This property provides the default deadband for OPCGroups created using Groups.Add. A deadband is expressed as percent of full scale (legal values 0 to 100).

Syntax DefaultGroupDeadband As Single

Remarks This property defaults to 0. Error would be generated if value > 100 or less than 0.

Example

```
VB Syntax Example (getting the property): Dim
DefaultGroupDeadband As Single

DefaultGroupDeadband = MyGroups.DefaultGroupDeadband

VB Syntax Example (setting the property):
MyGroups.DefaultGroupDeadband = 10
```

4.3.4.5 DefaultGroupLocaleID

Description (Read/Write) This property provides the default locale for OPCGroups created using Groups.Add.

Syntax DefaultGroupLocaleID As Long

Remarks This property defaults to the Servers LocaleID..

Example

```
VB Syntax Example (getting the property): Dim
DefaultGroupLocaleID As Long

DefaultGroupLocaleID = MyGroups.DefaultGroupLocaleID

VB Syntax Example (setting the property):
MyGroups.DefaultGroupLocaleID =
ConvertLocaleIdStringToLocaleIdLong ("English")
```

4.3.4.6 DefaultGroupTimeBias

Description (Read/Write) This property provides the default time bias for OPCGroups created using Groups.Add.

Syntax DefaultGroupTimeBias As Long

Remarks This property defaults to 0 minutes.

Example

```
VB Syntax Example (getting the property): Dim
DefaultGroupTimeBias As Long

DefaultGroupTimeBias = MyGroups.DefaultGroupTimeBias

VB Syntax Example (setting the property):
MyGroups.DefaultGroupTimeBias = 60
```

4.3.4.7 Count

Description (Read-only) Required property for collections.

Syntax Count As Long

Example

```
VB Syntax Example :For index = 1 to MyGroups.Count ` some code
hereNext index
```

4.3.5 OPCGroups Methods

4.3.5.1 Item

Description Returns an OPCGroup by ItemSpecifier. ItemSpecifier is the name or 1-based index into the collection. Use GetOPCGroup to reference by ServerHandle. Item is the default method for OPCGroups.

Syntax Item(ItemSpecifier As Variant) As OPCGroup

Example

```
VB Syntax Example:Dim AnOPCGroup As OPCGroupSet AnOPCGroup
= MyGroups.Item(3) `OrSet AnOPCGroup = MyGroups("Group3")
```

4.3.5.2 Add

Description Creates a new OPCGroup object and adds it to the collection. The properties of this new group are determined by the current defaults in the OPCServer object. After a group is added, its properties can also be modified.

Syntax `Add(Optional Name As Variant) As OPCGroup`

Part Description

Name Name of the group. The name must be unique among the other groups created by this client. If no name is provided, The server-generated name will also be unique relative to any existing groups.

Remarks If the optional name is not specified, the server generates a unique name. This method will fail if a name is specified but it is not unique. A failure in this case results in the OPCGroup object not being created, and Visual Basic will generate an error when attempting to use the object that has not been set. Refer to Appendix A - OPC Automation Error Handling for information on OPC Automation errors and Exceptions.

Example

```
MyGroups.DefaultGroupIsActive = True

Set OneGroup = MyGroups.Add( "AnOPCGroupName" )
```

4.3.5.3 GetOPCGroup

Description Returns an OPCGroup by ItemSpecifier.

Syntax `GetOPCGroup (ItemSpecifier As Variant) As OPCGroup`

Part Description

ItemSpecifier ItemSpecifier is either the OPCGroup's ServerHandle, or the name of an OPCGroup. Use Item to reference by index.

Example \ If "AnOPCGroupName" has already been Added

```
Set OneGroup = MyGroups.GetOPCGroup( "AnOPCGroupName" )
```

4.3.5.4 Remove

Description Removes an OPCGroup by Key.

Syntax `Remove(ItemSpecifier As Variant)`

Part Description

ItemSpecifier ItemSpecifier is either the OPCGroup's ServerHandle, or the name of an OPCGroup. Use Item to reference by index.

Remarks This method will fail if the group is a public group. Refer to Appendix A - OPC Automation Error Handling for information on OPC Automation Errors and Exceptions.

```
ExampleSet OneGroup = MyGroups.Add( "AnOPCGroupName" ) ` some more code
      hereMyGroups.Remove( "AnOPCGroupName" ) `orSet OneGroup =
MyGroups.Add( "AnOPCGroupName" ) ` some more code
      hereMyGroups.Remove(OneGroup.ServerHandle )
```

4.3.5.5 RemoveAll

Description Removes all current OPCGroup and OPCItem objects to prepare for server shutdown.

Syntax `RemoveAll()`

Remarks This is designed to make thorough sub-object cleanup much easier for clients to ensure all objects are released when the Server object is released. It is equivalent to calling Remove on all remaining OPCItem and OPCGroup objects. OPCBrowser objects are not sub-objects of the server, and they are not removed by this method.

```
ExampleSet OneGroup = MyGroups.Add( "AnOPCGroupName"
)Set OneGroup = MyGroups.Add(
"AnOPCGroupName1" )Set OneGroup =
MyGroups.Add( "AnOPCGroupName2" ) ` some more
code hereMyGroups.RemoveAll
```

4.3.5.6 ConnectPublicGroup

Description Public Groups are pre-existing groups in a server. These groups can be connected rather than added..

Refer to Appendix A - OPC Automation Error Handling for information on OPC Automation errors and Exceptions.

Syntax `ConnectPublicGroup (Name As String) As OPCGroup`

Part	Description
Name	Name of group to be connected.

Remarks This method will fail if the server does not support public groups or the name is not valid

Refer to Appendix A - OPC Automation Error Handling for information on OPC Automation errors and Exceptions

```
Example Set OneGroup = MyGroups.ConnectPublicGroup (
"AnOPCServerDefinedPublicGroup"" )
```

4.3.5.7 RemovePublicGroup

Description Removes the OPCGroup specified by ItemSpecifier.

Syntax `RemovePublicGroup (ItemSpecifier As Variant)`

Part Description

ItemSpecifier The ServerHandle returned by ConnectPublicGroup, or the name of a Public OPCGroup.

RemarksThis method will fail if the server does not support public groups, or if the group has not been connected to via ConnectPublicGroup.

Refer to Appendix A - OPC Automation Error Handling for information on OPC Automation errors and Exceptions

```
ExampleSet OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) `
    some more code hereMyGroups.RemovePublicGroup (
    "AnOPCGroupName" )
```

```
`orSet OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) ` some
more code hereMyGroups.RemovePublicGroup (OneGroup.ServerHandle )
```

4.3.6 OPCGroups Events

4.3.6.1 GlobalDataChange

Description The GlobalDataChangeevent is an event to facilitate one event handler being implemented to receive and process data changes across multiple groups.

Syntax `GlobalDataChange (TransactionID As Long, GroupHandle As Long, NumItems As Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)`

Part	Description
TransactionID	The client specified transaction ID. A non-0 value for this indicates that this call has been generated as a result of an AsyncRefresh. A value of 0 indicates that this call has been generated as a result of normal subscription processing.
GroupHandle	ClientHandle of the OPCGroup Object the changed data corresponds to.
NumItems	The number of items returned
ClientHandles	Array of client item handles for the items
ItemValues	Array of values.
Qualities	Array of Qualities for each item's value.
TimeStamps	Array of UTC TimeStamps for each item's value

Remarks NOTE – it is recommended that the event OnDataChange on the OPCGroup object be used normally. This event has been provided to facilitate one event handler being set up to process data changes for multiple OPCGroup objects. Normally, your application has an individual event handler for each group to receive and process the data changes. This allows you to have one event handler, and then using the GroupHandle, know which Group the event has been fired on behalf of.

This event will be invoked for each OPCGroup object that contains an item, whose value or state of the value has changed since the last time this event, was fired. The individual event on the OPCGroup object is also fired as well. Your application, when using both event handlers will receive the data value twice, once for the individual group event, and once for the AllGroupsDataChange Event.

Example Dim WithEvents AnOPCGroupCollection As OPCGroups

Private Sub AnOPCGroupCollection_GlobalDataChange (TransactionID As Long, GroupHandle As Long, MasterQuality As Long, MasterError As Long, NumItems As Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)

‘ write your client code here to process the data change values

End Sub

4.4 OPCGroup Object

Description The OPC Groups provide a way for clients to organize data. For example, the group might represent items in a particular operator display or report. Data can be read and written. Exception based connections can also be created between the client and the items in the group and can be enabled and disabled as needed. An OPC client can configure the rate that an OPC server should provide the data changes to the OPC client.

Syntax OPCGroup

4.4.1 Summary of Properties

Parent	Name	IsPublic
IsActive	IsSubscribed	ClientHandle
ServerHandle	LocaleID	TimeBias
DeadBand	UpdateRate	OPCItems

4.4.2 Summary of Methods

SyncRead	SyncWrite	AsyncRead
AsyncWrite	AsyncRefresh	AsyncCancel

4.4.3 Summary of Events

DataChange	AsyncReadComplete	AsyncWriteComplete
AsyncCancelComplete		

Example The following sample code is necessary for the subsequent
 Syntax Visual Basic Examples to be operational. . This code is
 Base referred to as OPCGroupObjectBase.

```
Dim WithEvents AnOPCServer As OPCServer
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
Dim AnOPCServerBrowser As OPCBrowser
Dim MyGroups As OPCGroups
Dim DefaultGroupUpdateRate As Long
Dim WithEvents OneGroup As OPCGroup
Dim AnOPCItemCollection As OPCItems
Dim AnOPCItem As OPCItem
Dim ClientHandles(100) As Long
Dim AnOPCItemIDs(100) As String
Dim AnOPCItemServerHandles() As Long
```

```
Dim AnOPCItemServerErrors() As LongSet AnOPCServer = New OPCServerARealOPCServer
= "VendorX.DataAccessCustomServer" ARealOPCNodeName =
"SomeComputerNodeName" AnOPCServer.Connect(ARealOPCServer,
ARealOPCNodeName)Set MyGroups =
AnOPCServer.OPCGroupsMyGroups.DefaultGroupIsActive = TrueSet OneGroup =
MyGroups.Add("AnOPCGroupName")Set AnOPCItemCollection = OneGroup.OPCItemsFor
x=1To AddItemCount
```

```
ClientHandles(x) = x + 1
```

```
AnOPCItemID(x) = "Register_" & xNext xAnOPCItemCollection.AddItem AddItemCount,
AnOPCItemIDs,
AnOPCItemServerHandles, AnOPCItemServerErrors
```

4.4.4 OPCGroup Properties

4.4.4.1 Parent

Description (Read-only) Returns reference to the parent OPCServer object.

Syntax Parent As OPCServer

4.4.4.2 Name

Description (Read/Write) The name given to this group.

Syntax Name As String

Part Description

Name Name of the group. The name must be a unique group name, with respect to the naming of other groups created by this client.

Remarks Naming a group is optional. This property allows the user to specify a name, therefore a unique name must be provided when setting the value for this property. The server will generate a unique name for the group, if no name is specified, on the Add method of the OPCGroups object.

ExampleVB Syntax Example (getting the property):
Dim CurrentValue As String
Set OneGroup = MyGroups.Add("AnOPCGroupName")
CurrentValue = OneGroup.Name
VB Syntax Example (setting the property):
Set OneGroup = MyGroups.Add("AnOPCGroupName")
OneGroup.Name = "aName"

4.4.4.3 IsPublic

Description (Read-only) Returns True if this group is a public group, otherwise False.

Syntax IsPublic As Boolean

ExampleDim CurrentValue As Boolean
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup ("AnOPCGroupName")
some more code here
Set CurrentValue = OneGroup.IsPublic ` to get the value

4.4.4.4 IsActive

Description(Read/Write) This property controls the active state of the group. A group that is active acquires data. An inactive group typically does not continue data acquisition except as required for read/writes.

Syntax IsActive As Boolean

RemarksDefault value for this property is the value from the OPCGroups corresponding default value at time of the Add();


```

ExampleVB Syntax Example (getting the property):Dim
CurrentValue As BooleanSet MyGroups =
AnOPCServer.OPCGroups

Set OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName"
) ` some more code hereSet CurrentValue = OneGroup.IsActive ` to
get the valueVB Syntax Example (setting the property):

Dim CurrentValue As BooleanSet MyGroups =
AnOPCServer.OPCGroupsSet OneGroup =
MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) ` some more
code hereOneGroup.IsActive = True

```

4.4.4.5 IsSubscribed

Description (Read/Write) This property controls asynchronous notifications to the group. A group that is subscribed receives data changes from the server.

Syntax IsSubscribed As Boolean

RemarksDefault value for this property is the value from the OPCGroups corresponding default value at time of the Add();

```

ExampleVB Syntax Example (getting the property):Dim
CurrentValue As BooleanSet MyGroups =
AnOPCServer.OPCGroups

Set OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName"
) ` some more code hereSet CurrentValue = OneGroup.IsSubscribed
` to get the valueVB Syntax Example (setting the property):

Set MyGroups = AnOPCServer.OPCGroupsSet OneGroup =
MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) ` some more
code hereOneGroup.IsSubscribed = True ` to set the value

```

4.4.4.6 ClientHandle

Description(Read/Write) A Long value associated with the group. Its purpose is for the client to quickly locate the destination of data. The handle is typically an index, etc. This handle will be returned to the client along with data or status.

Syntax ClientHandle As LongExample VB Syntax

Example (getting the property):

```

Dim CurrentValue As LongSet MyGroups = AnOPCServer.OPCGroupsSet
OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) `
some more code hereSet CurrentValue = OneGroup.ClientHandle `

```

to get the value
VB Syntax Example (setting the property):

```
Set MyGroups = AnOPCServer.OPCGroupsSet
OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) ` some more
code here
OneGroup.ClientHandle = 1975 ` to set the value
```

4.4.4.7 ServerHandle

Description (Read-only) The server assigned handle for the group. The ServerHandle is a Long that uniquely identifies this group. The client must supply this handle to some of the methods that operate on OPCGroup objects (such as OPCGroups.Remove).

Syntax ServerHandle As Long
Example VB Syntax

Example (getting the property):

```
Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroupsSet
OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) `
some more code here
Set CurrentValue = OneGroup.ServerHandle `
to get the value
```

4.4.4.8 LocaleID

Description (Read/Write) This property identifies the locale, which may be used to localize strings returned from the server. This property's default depends on the value set in the OPCGroups Collection..

Syntax LocaleID As Long
Example VB Syntax

Example (getting the property):

```
Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroupsSet
OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) `
some more code here
Set CurrentValue = OneGroup.LocaleID
VB Syntax Example (setting the property):
```

```
Set MyGroups = AnOPCServer.OPCGroupsSet
OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) ` some more
code here

OneGroup.LocaleID = StringToLocaleID("English")
```

4.4.4.9 TimeBias

Description (Read/Write). This property provides the information needed to convert the time stamp on the data back to the local time of the device.

Syntax TimeBias As Long

```

ExampleVB Syntax Example (getting the property):Dim
    CurrentValue As LongSet MyGroups =
        AnOPCServer.OPCGroups

    Set OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" )

    ` some more code hereSet CurrentValue =
OneGroup.TimeBiasVB Syntax Example (setting the
property):

Set MyGroups = AnOPCServer.OPCGroupsSet OneGroup =
MyGroups.ConnectPublicGroup ( "AnOPCGroupName" ) ` some more code
hereOneGroup.TimeBias = 100
    
```

4.4.4.10 DeadBand

Description (Read/Write) A deadband is expressed as percent of full scale (legal values 0 to 100). This property's default depends on the value set in the OPCGroups Collection.

Syntax DeadBand As Single

```

Example
VB Syntax Example (getting the property): Dim CurrentValue As
Single Set MyGroups = AnOPCServer.OPCGroups

Set OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName"
)
    ` some more code here

Set CurrentValue = OneGroup.DeadBand

VB Syntax Example (setting the property): Set MyGroups =
AnOPCServer.OPCGroups

Set OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName"
)
    ` some more code here

OneGroup.DeadBand = 5
    
```

4.4.4.11 UpdateRate

Description (Read/Write) The fastest rate at which data change events may be fired. A slow process might cause data changes to fire at less than this rate, but they will never exceed this rate. Rate is in milliseconds. This property's default depends on the value set in the OPCGroups Collection. Assigning a value to this property is a "request" for a new update rate. The server may not support that rate, so reading the property may result in a different rate (the server will use the closest rate it does support).

Syntax UpdateRate As Long

Example

```
VB Syntax Example (getting the property): Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroups

Set OneGroup = MyGroups.ConnectPublicGroup ( "AnOPCGroupName" )
    ` some more code here
```

DefaultGroupUpdateRate = OneGroup.UpdateRate

VB Syntax Example (setting the property):Set MyGroups = AnOPCServer.OPCGroupsSet OneGroup = MyGroups.ConnectPublicGroup ("AnOPCGroupName")` some more code hereOneGroup.UpdateRate = 50

4.4.4.12 OPCItems

Description A collection of OPCItem objects. This is the default property of the OPCGroup object.

Syntax OPCItems As OPCItemsExample VB Syntax

Example (getting the property):

```
Dim AnOPCItemCollection As OPCItemsSet MyGroups =
AnOPCServer.OPCGroupsSet OneGroup = MyGroups.ConnectPublicGroup
( "AnOPCGroupName" ) ` some more code hereSet
AnOPCItemCollection = OneGroup.OPCItems
```

4.4.5 OPCGroup Methods

4.4.5.1 SyncRead

Description This function reads the value, quality and timestamp information for one or more items in a group.

```
Syntax SyncRead(Source As Integer, NumItems As Long, ServerHandles()
As Long, ByRef Values() As Variant, ByRef Errors() As Long,
Optional ByRef Qualities As Variant, Optional ByRef TimeStamps
As Variant)
```

Part	Description
Source	The 'data source'; OPC_DS_CACHE or OPC_DS_DEVICE
NumItems	The number of items to be read.
ServerHandles	Array of server item handles for the items to be read

Values	Array of values.
Errors	Array of Long's indicating the success of the individual item reads. This indicates whether the read succeeded in obtaining a defined value, quality and timestamp. NOTE any FAILED error code indicates that the corresponding Value, Quality and Time stamp are UNDEFINED.
Qualities	Variant containing an Integer Array of Qualities.

TimeStamps	Variant containing a Date Array of UTC TimeStamps. If the device cannot provide a timestamp then the server will provide one.
------------	---

Remarks The function runs to completion before returning. The data can be read from CACHE in which case it should be accurate to within the 'UpdateRate' and percent deadband of the group. The data can be read from the DEVICE in which case an actual read of the physical device is to be performed. The exact implementation of CACHE and DEVICE reads is not defined by this specification. When reading from CACHE, the data is only valid if both the group and the item are active. If either the group or the item is inactive, then the Quality will indicate out of service (OPC_QUALITY_OUT_OF_SERVICE). Refer to the discussion of the quality bits later in this document for further information. DEVICE reads are not affected by the ACTIVE state of the group or item.

```
Example Private Sub ReadButton_Click()

    Dim Source As Integer

    Dim NumItems As Long

    Dim ServerIndex As Long

    Dim ServerHandles(10) As Long

    Dim Values() As Variant

    Dim Errors() As Long

    Dim Qualities() As Variant

    Dim TimeStamps() As Variant

    Source = OPC_DS_DEVICE
```

```

NumItems = 10

For ServerIndex=1to NumItems

` set up which items to be read

ServerHandles(ServerIndex) =
AnOPCItemServerHandles(ServerIndex)

Next ServerIndex

OneGroup.SyncRead Source, NumItems, ServerHandles, Values,
Errors,

Qualities, TimeStampsFor ServerIndex=1to NumItems ` process
the values
TextBox(ServerIndex).Text =
Values(ServerIndex)
Next ServerIndex
End Sub

```

4.4.5.2 SyncWrite

Description Writes values to one or more items in a group. The function runs to completion. The values are written to the DEVICE. That is, the function should not return until it verifies that the device has actually accepted (or rejected) the data.

Syntax SyncWrite(NumItems As Long, ServerHandles() As Long, Values()
As Variant, ByRef Errors() As Long)

Part	Description
NumItems	Number of items to be written
ServerHandles	Array of server item handles for the items to be written
Values	Array of values.
Errors	Array of Long's indicating the success of the individual item writes..

Remarks Writes are not affected by the ACTIVE state of the group or item.

```

Example Private Sub WriteButton_Click()

Dim Source As Integer

Dim NumItems As Long

```

```

Dim ServerIndex As Long

Dim ServerHandles() As Long

Dim Values() As Variant

Dim Errors() As Long

NumItems = 10

For ServerIndex=1to NumItems

    ' set up which items to be written

    ServerHandles(ServerIndex) =
    AnOPCItemServerHandles(ServerIndex)

    Values(ServerIndex) = ServerIndex*2 'any random value for
    this

    example would sufficeNext
    ServerIndexOneGroup.SyncWrite NumItems,
    ServerHandles, Values, ErrorsFor ServerIndex=1to
    NumItems ' process the ErrorsTextBox(ServerIndex).Text
    = Errors(ServerIndex)

Next ServerIndexEnd Sub

```

4.4.5.3 AsyncRead

Description Read one or more items in a group. The results are returned via the AsyncReadComplete event associated with the OPCGroup object.

Reads are from 'DEVICE' and are not affected by the ACTIVE state of the group or item.

Syntax AsyncRead(NumItems As Long, ServerHandles() As Long, ByRef Errors() As Long, TransactionID As Long, ByRef CancelID As Long)

Part	Description
------	-------------

NumItems	The number of items to be read.
ServerHandles	Array of server item handles for the items to be read
Errors	Array of Long's indicating the status of the individual items to be read.
TransactionID	The client specified transaction ID. This is included in the 'completion' information provided in the Corresponding Event.
CancelID	A Server generated transaction ID. This is provided to enable the client to cancel the "transaction".

Remarks The AsyncRead requires the OPCGroup object to have been dimensioned with events (Dim WithEvents xxx As OPCGroup) in order for the results of the AsyncRead operation to be returned to the automation client application. The AsyncReadComplete event associated with the OPCGroup object will be fired (called) by the automation server with the results of the AsyncRead operation.

See Also IOPCAsyncIO2::Read from the OPC Data Access Custom Interface Specification

Example Private Sub AsyncReadButton_Click()

Dim NumItems As Long

Dim ServerIndex As Long

Dim ServerHandles(10) As Long

Dim Values() As Variant

Dim Errors() As Long

Dim ClientTransactionID As Long Dim ServerTransactionID As Long

Dim Qualities() As VariantDim TimeStamps() As

VariantNumItems = 10

ClientTransactionID = 1975For ServerIndex=1to NumItems' set up which items to be

readServerHandles(ServerIndex) = AnOPCItemServerHandles(ServerIndex)Next

ServerIndexOneGroup.AsyncRead NumItems, ServerHandles, Errors,

ClientTransactionID , ServerTransactionIDEnd Sub

4.4.5.4 AsyncWrite

Description Write one or more items in a group. The results are returned via the AsyncWriteComplete event associated with the OPCGroup object.

Syntax AsyncWrite(NumItems As Long, ServerHandles() As Long, Values()
As Variant, ByRef Errors() As Long, TransactionID As Long, ByRef
CancelID As Long)

Part	Description
NumItems	The number of items to be written.
ServerHandles	Array of server item handles for the items to be written.
Values	Array of values.
Errors	Array of Long's indicating the status of the individual items to be written.
TransactionID	The client specified transaction ID. This is included in the 'completion' information provided in the Corresponding Event.
CancelID	A Server generated transaction ID. This is provided to enable the client to cancel the "transaction".

RemarksThe AsyncWrite requires the OPCGroup object to have been dimensioned with events (Dim WithEvents xxx As OPCGroup) in order for the results of the AsyncWrite operation to be returned to the automation client application. The AsyncWriteComplete event associated with the OPCGroup object will be fired (called) by the automation server with the results of the AsyncWrite operation.

See Also IOPCAsyncIO2::Write from the OPC Data Access Custom Interface Specification

ExamplePrivate Sub

```

AsyncWriteButton_Click() Dim
NumItems As Long Dim ServerIndex
As Long Dim ServerHandles(10) As
Long Dim Values() As Variant Dim
Errors() As Long Dim
ClientTransactionID As Long

Dim ServerTransactionID As Long NumItems = 10 For
ServerIndex=1 to NumItems
ClientTransactionID = 1957

' set up which items to be write ServerHandles(ServerIndex) =
AnOPCItem ServerHandles(ServerIndex) Values(ServerIndex) =
ServerIndex * 2 'any random value for this

example would suffice Next ServerIndex OneGroup.AsyncWrite

```

NumItems, ServerHandles, Values, Errors,

ClientTransactionID , ServerTransactionIDEnd Sub

4.4.5.5 AsyncRefresh

Description Generate an event for all active items in the group (whether they have changed or not). Inactive items are not included in the callback. The results are returned via the DataChange event associated with the OPCGroup object, as well as the GlobalDataChange event associated with the OPCGroups object.

Syntax AsyncRefresh(Source As Integer, TransactionID As Long,ByRef CancelID As Long)

Part	Description
Source	The 'data source'; OPC_DS_CACHE or OPC_DS_DEVICE
TransactionID	The client specified transaction ID. This is included in the 'completion' information provided in the Corresponding Event.
CancelID	A Server generated transaction ID. This is provided to enable the client to cancel the "transaction".

RemarksThe AsyncRefresh requires the OPCGroup object to have been dimensioned with events (Dim WithEvents xxx As OPCGroup) in order for the results of the refresh operation to be returned to the automation client application. The DataChange event associated with the OPCGroup object will be fired (called) by the automation server with the results of the refresh operation. If the automation client application has dimensioned with events the OPCGroups Object (Dim WithEvents xyz as OPCGroups), then the GlobalDataChange event associated with the OPCGroups object will be fired (called) by the automation server with the results of the refresh operation.

See Also IOPCAsyncIO::Refresh from the OPC Data Access Custom Interface Specification.

```
ExampleDim MyGroups As OPCGroupsDim
    DefaultGroupUpdateRate As LongDim WithEvents
    OneGroup As OPCGroup

    Private Sub AsyncRefreshButton_Click()Dim
    ServerIndex As LongDim Source As LongDim
    ClientTransactionID As Long

    Dim ServerTransactionID As LongClientTransactionID =
    2125Source = OPC_DS_DEVICEOneGroup.AsyncRefresh Source,
    ClientTransactionID
```

ServerTransactionID

End Sub

4.4.5.6 AsyncCancel

Description Request that the server cancel an outstanding transaction. An AsyncCancelComplete event will occur indicating whether or not the cancel succeeded.

Syntax AsyncCancel (CancelID As Long)

Part Description

CancelID The Server generated CancelID that was previously returned by the AsyncRead, AsyncWrite or AsyncRefresh method that the client now wants to cancel.

See Also IOPCAsyncIO2::Cancel from the OPC Data Access Custom Interface Specification

RemarksThe AsyncCancel requires the OPCGroup object to have been dimensioned with events (Dim WithEvents xxx As OPCGroup) in order for the results of the AsyncCancel operation to be returned to the automation client application. The AsyncCancelComplete event associated with the OPCGroup object will be fired (called) by the automation server with the results of the AsyncCancel operation. The client specified transaction ID (TransactionID) will be returned to the automation client application in the AsyncCancelComplete event.

Example

```
Private Sub AsyncCancelButton_Click() Dim ServerIndex As Long Dim
CancelID As Long
CancelID = 1 ' some transaction id returned from one of the async
calls like read, write, or refresh. OneGroup.AsyncCancel CancelID
End Sub
```

4.4.6 OPCGroup Events

4.4.6.1 DataChange

Description The DataChange event is fired when a value or the quality of a value for an item within the group has changed. Note the event will not fire faster than the update rate of the group. Therefore, item values will be held by the server and buffered until the current time + update rate is greater than the time of the previous update (event fired). This is also affected by active states for both Group and Items. Only items that are active, and whose group is active will be sent to the client in an event.

Syntax DataChange (TransactionID As Long, NumItems As Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)

Part	Description
TransactionID	The client specified transaction ID. A non-0 value for this indicates that this call has been generated as a result of an AsyncRefresh. A value of 0 indicates that this call has been generated as a result of normal subscription processing.
NumItems	The number of items returned
ClientHandles	Array of client item handles for the items
ItemValues	Array of values.
Qualities	Array of Qualities for each item's value.
TimeStamps	Array of UTC TimeStamps for each item's value. If the device cannot provide a timestamp then the server will provide one.

Remarks If the item values are changing faster than the update rate, only the most recent value for each item will be buffered and returned to the client in the event.

Example

```
Dim WithEvents AnOPCGroup As OPCGroup Private Sub
AnOPCGroup_DataChange (TransactionID As Long, NumItems As Long,
ClientHandles() As Long, ItemValues() As Variant,
```

58

```
Qualities() As Long, TimeStamps() As Date) ` write your client code here
to process the data change valuesEnd Sub
```

4.4.6.2 AsyncReadComplete

Description This event fires when an AsyncRead is completed.

```
Syntax AsyncReadComplete (TransactionID As Long, NumItems As Long,
ClientHandles() As Long, ItemValues() As Variant, Qualities()
As Long, TimeStamps() As Date, Errors() As Long)
```

Part	Description
TransactionID	The client specified transaction ID.
NumItems	The number of items returned
ClientHandles	Array of client item handles for the items
ItemValues	Array of values.
Qualities	Array of Qualities for each item's value.

TimeStamps	Array of UTC TimeStamps for each item's value. If the device cannot provide a timestamp then the server will provide one.
Errors	Array of Long's indicating the success of the individual item reads. This indicates whether the read succeeded in obtaining a defined value, quality and timestamp. NOTE any FAILED error code indicates that the corresponding Value, Quality and Time stamp are UNDEFINED.

```
ExampleDim WithEvents AnOPCGroup As OPCGroupPrivate Sub
    AnOPCGroup_AsyncReadComplete (TransactionID As Long,
    NumItems As Long, ClientHandles() As Long, ItemValues() As
    Variant,Qualities() As Long, TimeStamps() As Date)' write
    your client code here to process the data change valuesEnd
    Sub
```

4.4.6.3 AsyncWriteComplete

Description This event fires when an AsyncWrite is completed.

Syntax AsyncWriteComplete (TransactionID As Long, NumItems As Long, ClientHandles() As Long, Errors() As Long)

Part	Description
TransactionID	The client specified transaction ID.
NumItems	The number of items returned
ClientHandles	Array of client item handles for the items
Errors	Array of Long's indicating the success of the individual item writes.

```
ExampleDim WithEvents AnOPCGroup As OPCGroupPrivate Sub
    AnOPCGroup_AsyncWriteComplete (TransactionID As Long,
    NumItems As Long, ClientHandles() As Long, ItemValues() As
    Variant,Qualities() As Long, TimeStamps() As Date)' write
    your client code here to process the errorsEnd Sub
```

4.4.6.4 AsyncCancelComplete

Description This event fires when an AsyncCancel is completed.

Syntax AsyncCancelComplete (TransactionID As Long)

Part Description

TransactionID The client specified transaction ID. This is included in the ‘completion’ information provided in the Corresponding Event.

```
ExampleDim WithEvents AnOPCGroup As OPCGroupPrivate Sub
    AnOPCGroup_AsyncCancelComplete (TransactionID As Long) ` write
    your client code here to process the cancelEnd Sub
```

4.5 OPCItems Object

Description This object also has properties for OPCItem defaults. When an OPCItem is added, the DefaultXXXX properties set its initial state. The defaults can be changed to add OPCItems with different initial states. Of course, once an OPCItem is added, its properties can be modified. This reduces the number of parameters required to call the Add method.

Syntax OPCItems

4.5.1 Summary of Properties

Parent	DefaultRequestedDataType	DefaultAccessPath
DefaultIsActive	Count	

4.5.2 Summary of Methods

Item	GetOPCItem	AddItem
AddItems	Remove	Validate
SetActive	SetClientHandles	SetDataTypes

Example The following sample code is necessary for the subsequent Syntax Visual Basic Examples to be operational. This code is referred to as OPCItemsObjectBase.

```
Dim AnOPCServer As OPCServer
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
Dim AnOPCServerBrowser As OPCBrowser
Dim MyGroups As OPCGroups
Dim DefaultGroupUpdateRate As Long
Dim OneGroup As OPCGroup
Dim AnOPCItemCollection As OPCItems
Dim AnOPCItem As OPCItem
Dim ClientHandles(100) As Long
```

```
Dim AnOPCItemIDs(100) As String
Dim AnOPCItemServerHandles(10) As Long
Dim AnOPCItemServerErrors() As Long
Set AnOPCServer = New OPCServer
ARealOPCServer = "VendorX.DataAccessCustomServer"
ARealOPCNodeName = "SomeComputerNodeName"
```

```
AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)Set MyGroups =
AnOPCServer.OPCGroupsMyGroups.DefaultGroupIsActive = TrueSet OneGroup =
MyGroups.Add("AnOPCGroupName")Set AnOPCItemCollection = OneGroup.OPCItems
```

4.5.3 OPCItems Properties

4.5.3.1 Parent

Description (Read-only) Returns reference to the parent OPCGroup object.

```
Syntax Parent As OPCGroup
```

4.5.3.2 DefaultRequestedDataType

Description (Read/Write) The requested data type that will be used in calls to Add. This property defaults to VT_EMPTY (which means the server sends data in the server canonical data type).

```
Syntax DefaultRequestedDataType As Integer
```

Remarks Any legal Variant type can be passed as a requested data type.

See Also [Appendix A - OPC Automation Error Handling](#)
[Appendix D- Notes On Automation Data Types](#)

Example VB Syntax Example (getting the property):

```
Dim CurrentValue As Integer

Dim SomeValue As Integer

CurrentValue = AnOPCItemCollection.DefaultRequestedDataType
```

```
VB Syntax Example (setting the
property):AnOPCItemCollection.DefaultRequestedDataType =
SomeValue
```

4.5.3.3 DefaultAccessPath

Description (Read/Write) The default AccessPath that will be used in calls to Add. This property defaults to

“”

Syntax `DefaultAccessPath As String`

Example VB Syntax Example (getting the property):

```
Dim CurrentValue As String
```

```
Dim SomeValue As String
```

```
CurrentValue = AnOPCItemCollection.DefaultAccessPath
```

```
VB Syntax Example (setting the  
property):AnOPCItemCollection.DefaultAccessPath = SomeValue
```

4.5.3.4 DefaultIsActive

Description (Read/Write) The default active state that will be used in calls to Add. This property defaults to True.

Syntax `DefaultIsActive As Boolean`

Example VB Syntax Example (getting the property):

```
Dim CurrentValue As Boolean
```

```
Dim SomeValue As Boolean
```

```
CurrentValue = AnOPCItemCollection.DefaultIsActive
```

```
VB Syntax Example (setting the  
property):AnOPCItemCollection.DefaultIsActive = SomeValue
```

4.5.3.5 Count

Description (Read-only) Required property for collections.

Syntax `Count As Long`

ExampleVB Syntax Example (getting the property):
Dim CurrentValue As Long
Dim SomeValue As Long
CurrentValue = AnOPCItemCollection.Count

4.5.4 OPCItems Methods

4.5.4.1 Item

Description Required property for collections.

Syntax `Item (ItemSpecifier As Variant) As OPCItem`

Part Description

ItemSpecifier Returns an OPCItem by ItemSpecifier. ItemSpecifier is the 1-based index into the collection

RemarksReturns an OPCItem by ItemSpecifier. ItemSpecifier is the 1-based index into the collection. Use GetOPCItem to reference by ServerHandle.

NOTE: do not confuse the automation 'Item' property with the OPCItem object. The automation 'Item' is a special reserved property used in a generic way by automation collections to refer to the items they contain. The OPCItem is an OPC Automation specific object type that can reside in an 'OPCItems' collection.

4.5.4.2 GetOPCItem

Description Returns an OPCItem by ServerHandle returned by Add. Use the Item property to reference by index.

Syntax `GetOPCItem (ServerHandle As Long) As OPCItem`

Part Description

ServerHandle ServerHandle is the OPCItem's ServerHandle Use Item to reference by index.

Example `Dim AnOPCItem as OPCItem`

```
Set OPCItem =
GetOPCItem(SomeItemServerHandle)
```

4.5.4.3 AddItem

DescriptionCreates a new OPCItem object and adds it to the collection. The properties of this new OPCItem are determined by the current defaults in the OPCItems collection object. After an OPCItem is added, its properties can also be modified.

Syntax `AddItem (ItemID As String, ClientHandle As Long)`

Part	Description
ItemID	Fully Qualified ItemID
ClientHandle	Client handle that will be returned with the

RemarksThis method is intended to provide the mechanism to add one item to the collection at a

time. For adding multiple items use the AddItems method, rather than repetitively calling AddItem for each object to be added.

See Also Appendix A - OPC

Automation Error
 Handling Appendix
 D- Notes On
 Automation Data
 Types Example
 Dim AnOPCItemID
 as String Dim
 AnClientHandle as
 Long
 AnOPCItemID =
 "N7:0"
 AnClientHandle =
 1975
 AnOPCItemCollect
 ion.AddItem
 AnOPCItemID
 AnClientHandle

4.5.4.4 AddItems

are determined by the current defaults in the OPCItems collection object. After an OPCItem is added, its properties can also be modified.

```
SyntaxAddItems (Count As Long, ItemIDs() As String, ClientHandles()  

    AsLong, ByRef ServerHandles() As Long, ByRef Errors() As  

    Long, Optional RequestedDataTypes As Variant, Optional  

    AccessPaths AsVariant)
```

Part	Description
Count	The number of items to be affected
ItemIDs	Array of Fully Qualified ItemID's
ClientHandles	Array of client item handles for the items processed
ServerHandles	Array of server item handles for the items processed

Errors	Array of Long's indicating the success of the individual items operation.
RequestedDataTypes	Optional Variant containing an integer array of Requested DataTypes.
AccessPaths	Optional Variant containing a string array of Access Path's.

See Also [Appendix A - OPC Automation Error Handling](#)
[Appendix D- Notes On Automation Data Types](#)

Example `Dim addItemCount as longDim`

```
AnOPCItemIDs() as StringDim
AnOPCItemServerHandles as longDim
AnOPCItemServerErrors as longDim
AnOPCRequestedDataTypes as variantDim
AnOPCAccessPathss as variantFor x=1To
AddItemCount
```

```
ClientHandles(x) = x +
1AnOPCItemID(x) = "Register_" &
xNext x
```

`AnOPCItemCollection.AddItemCount, AnOPCItemIDs, ClientHandles, AnOPCItemServerHandles, AnOPCItemServerErrors, AnOPCRequestedDataTypes, AnOPCAccessPathss`

` add code to process any errors that are returned from `thethod, individual errors are reported in the Errors array

4.5.4.5 Remove

Description Removes an OPCItem

Syntax `Remove (Count As Long, ServerHandles() As Long, ByRef Errors() As Long)`

Part	Description
Count	The number of items to be removed
ServerHandles	Array of server item handles for the items processed
Errors	Array of Long's indicating the success of the individual items operation.

Example `AnOPCItemCollection.Remove AnOPCItemServerHandles,`

AnOPCItemServerErrors

```
` add code to process any errors that are returned from
`thethod, individual errors are reported in the Errors
array
```

4.5.4.6 Validate

Description Determines if one or more OPCItems could be successfully created via the Add method (but does not add them).

```
SyntaxValidate (Count As Long, ItemIDs() As String, ByRef Errors()
AsLong, Optional RequestedDataTypes As Variant,
OptionalAccessPaths As Variant)
```

Part	Description
Count	The number of items to be affected
ItemIDs	Array of Fully Qualified ItemID's
Errors	Array of Long's indicating the success of the individual items operation.
RequestedDataTypes	Variant containing an integer array of Requested DataTypes.
AccessPaths	Variant containing a string array of Access Path's.

See Also Appendix A - OPC Automation Error Handling
Appendix D- Notes On Automation Data Types

```
Example Dim addItemCount as long
Dim AnOPCItemIDs() as String
```

Dim AnOPCItemServerHandles as longDim

AnOPCItemServerErrors as longDim

AnOPCRequestedDataTypes as variantDim

AnOPCAccessPathss as variant

For x=1To AddItemCountClientHandles(x) = x +

1AnOPCItemID(x) = "Register_" & x

Next xAnOPCItemCollection.Validate AddItemCount, AnOPCItemIDs,AnOPCItemServerErrors, AnOPCRequestedDataTypes, AnOPCAccessPathss

' add code to process any errors that are returned from 'themethod, individual errors are reported in the Errors array

4.5.4.7 SetActive

Description Allows Activation and deactivation of individual OPCItem's in the OPCItems Collection

Syntax SetActive (Count As Long, ServerHandles() As Long, ActiveState As Boolean, ByRef Errors() As Long)

Part	Description
Count	The number of items to be affected
ServerHandles	Array of server item handles for the items processed
ActiveState	TRUE if items are to be activated. FALSE if items are to be deactivated.
Errors	Array of Long's indicating the success of the individual items operation.

Example 'set items to active (TRUE)

```
AnOPCItemCollection.SetActive ItemCount,
AnOPCItemServerHandles,TRUE, AnOPCItemServerErrors
```

' add code to process any errors that are returned from 'themethod, individual errors are reported in the Errors array

4.5.4.8 SetClientHandles

Description Changes the client handles or one or more Items in a Group.

Syntax SetClientHandles (Count As Long, ServerHandles() As Long, ClientHandles() As Long, ByRef Errors() As Long)

Part	Description
Count	The number of items to be affected
ServerHandles	Array of server item handles for the items processed
ClientHandles	Array of new Client item handles to be stored. The Client handles do not need to be unique.
Errors	Array of Long's indicating the success of the individual items operation.

```

Example      For      x=1To      ItemCount

              ClientHandles(x) = x +
              1975

Next                                                x

AnOPCItemCollection.      SetClientHandles      ItemCount,

AnOPCItemServerHandles, ClientHandles,
AnOPCItemServerErrors
    
```

4.5.4.9 SetDataTypes

Description Changes the requested data type for one or more Items

Syntax `SetDataTypes (Count As Long, ServerHandles() As Long, RequestedDataTypes() As Long, ByRef Errors() As Long)`

Part	Description
Count	The number of items to be affected
ServerHandles	Array of server item handles for the items processed
RequestedDataTypes	Array of new Requested DataTypes to be stored.
Errors	Array of Long's indicating the success of the individual items operation.

See Also Appendix A - OPC Automation Error Handling
 Appendix D- Notes On Automation Data Types

```

Example      Dim RequestedDataTypes(100) As
              Long
              For x=1To ItemCount
    
```

RequestedDataTypes(x) = "some vbinteger"

```

Next                                                x

AnOPCItemCollection.SetDataTypes ItemCount, AnOPCItemServerHandles,
    
```

RequestedDataTypes,

AnOPCItemServerErrors

4.6 OPCItem Object

Description An OPC Item represents a connection to data sources within the server. Associated with each item is a Value, Quality and Time Stamp. The value is in the form of a VARIANT, and the Quality is similar to that specified by Fieldbus.

Syntax OPCItem

4.6.1 Summary of Properties

Parent	ClientHandle	ServerHandle
AccessPath	AccessRights	ItemID
IsActive	RequestedDataType	Value
Quality	TimeStamp	CanonicalDataType
EUType	EUInfo	

4.6.2 Summary of Methods

Read	Write	
------	-------	--

4.6.3 OPCItem Properties

4.6.3.1 Parent

Description (Read-only) Returns reference to the parent OPCGroup object.

Syntax Parent As OPCGroup

4.6.3.2 ClientHandle

Description (Read/Write) A Long value associated with the OPCItem. Its purpose is for the client to quickly locate the destination of data. The handle is typically an index, etc. This handle will be returned to the client along with data or status changes by OPCGroup events.

Syntax ClientHandle As Long

Example Dim AnOPCItem as OPCItem

```
Set OPCItem = GetOPCItem(SomeItemServerHandle)
```

```
VB Syntax Example (getting the property):Dim CurrentValue
As LongDim SomeValue As LongCurrentValue =
AnOPCItem.ClientHandle
```

```
VB Syntax Example (setting the
```

```
property):AnOPCItem.ClientHandle = SomeValue
```

4.6.3.3 ServerHandle

Description(Read-only) The server assigned handle for the AnOPCItem. The ServerHandle is a Long that uniquely identifies this AnOPCItem. The client must supply this handle to some of the methods that operate on OPCItem objects (such as OPCItems.Remove).

Syntax ServerHandle As Long

Example Dim AnOPCItem as OPCItem

```
Set OPCItem = GetOPCItem(SomeItemServerHandle)
```

```
VB Syntax Example (getting the property):Dim CurrentValue  
As LongDim SomeValue As LongCurrentValue =  
AnOPCItem.ServerHandle
```

4.6.3.4 AccessPath

Description (Read-only) The access path specified by the client on the Add function..

Syntax AccessPath As String

Example Dim AnOPCItem as OPCItem

```
Set OPCItem = GetOPCItem(SomeItemServerHandle)
```

```
VB Syntax Example (getting the property):Dim CurrentValue  
As StringDim SomeValue As StringCurrentValue =  
AnOPCItem.AccessPath
```

4.6.3.5 AccessRights

Description (Read-only) Returns the access rights of this item.

Syntax AccessRights As Long

Remarks Indicates if this item is read only, write only or read/write.

Example Dim AnOPCItem as OPCItem

```
Set OPCItem = GetOPCItem(SomeItemServerHandle)
```



```
VB Syntax Example (getting the property):Dim CurrentValue
As LongDim SomeValue As LongCurrentValue =
AnOPCItem.AccessRights
```

4.6.3.6 ItemID

Description (Read-only) The unique identifier for this item.

```
Syntax ItemID As StringExample Dim
```

```
AnOPCItem as OPCItemSet OPCItem =
```

```
GetOPCItem(SomeItemServerHandle)
```

```
VB Syntax Example (getting the property):Dim CurrentValue As StringDim
SomeValue As StringCurrentValue = AnOPCItem.ItemID
```

4.6.3.7 IsActive

(Read/Write) State of the Data Acquisition for this item.

Description

Syntax IsActive As Boolean

Remarks FALSE if the item is not currently active, TRUE if the item is currently active

Example Dim AnOPCItem as OPCItem

```
Set OPCItem = GetOPCItem(SomeItemServerHandle)
```

```
VB Syntax Example (getting the property): Dim
CurrentValue As Boolean Dim SomeValue As Boolean
CurrentValue = AnOPCItem.IsActive
```

```
VB Syntax Example (setting the property):
AnOPCItem.IsActive = SomeValue
```

4.6.3.8 RequestedDataType

Description(Read/Write) The data type in which the item's value will be returned. Note that if the requested data type was rejected the OPCItem will be invalid(failed), until the RequestedDataType is set to a valid value.

```
Syntax RequestedDataType As Integer
```

See Also Appendix A - OPC Automation Error Handling Appendix D- Notes On Automation Data Types

```
ExampleDim AnOPCItem as OPCItemSet OPCItem =
GetOPCItem(SomeItemServerHandle)VB Syntax Example (getting the
property):Dim CurrentValue As IntegerDim SomeValue As
IntegerCurrentValue = AnOPCItem.RequestedDataTypeVB Syntax
Example (setting the property):AnOPCItem.RequestedDataType =
SomeValue
```

4.6.3.9 Value

Description (Read-only) Returns the latest value read from the server. This is the default property of AnOPCItem.

Syntax Value As Variant

```
Example Dim AnOPCItem as OPCItemSet OPCItem =
    GetOPCItem(SomeItemServerHandle) VB Syntax Example (getting the
    property):
```

72

```
Dim CurrentValue As Variant Dim SomeValue As Variant CurrentValue =
AnOPCItem.Value
```

4.6.3.10 Quality

Description (Read-only) Returns the latest quality read from the server.

Syntax Quality As Long

Example

```
Dim AnOPCItem as OPCItem Set OPCItem =
GetOPCItem(SomeItemServerHandle) VB Syntax
Example (getting the property): Dim
CurrentValue As Long Dim SomeValue As Long
CurrentValue = AnOPCItem.Quality
```

4.6.3.11 TimeStamp

Description (Read-only) Returns the latest timestamp read from the server.

Syntax TimeStamp As Date

Example

```
Dim AnOPCItem as OPCItem Set OPCItem =
GetOPCItem(SomeItemServerHandle) VB Syntax
Example (getting the property): Dim
CurrentValue As Date Dim SomeValue As Date
CurrentValue = AnOPCItem.TimeStamp
```

4.6.3.12 CanonicalDataType

Description (Read-only) Returns the native data type in the server.

Syntax CanonicalDataType As Integer

See Also Appendix A - OPC Automation Error Handling

```

Example Dim AnOPCItem as OPCItem Set OPCItem =
GetOPCItem(SomeItemServerHandle) VB Syntax
Example (getting the property): Dim
CurrentValue As Integer Dim SomeValue As
Integer CurrentValue =
AnOPCItem.CanonicalDataType
    
```

4.6.3.13 EUType

Description (Read-only) Indicate the type of Engineering Units (EU) information (if any) contained in EUInfo.

Syntax EUType As Integer

See Also OPCITEMATTRIBUTES in the OPC Data Access Custom Interface Specification

Remarks 0 - No EU information available (EUInfo will be VT_EMPTY) 1 - Analog - EUInfo will contain a SAFEARRAY of exactly two doubles (VT_ARRAY | VT_R8) corresponding to the LOW and HI EU range. 2 - Enumerated - EUInfo will contain a SAFEARRAY of strings (VT_ARRAY | VT_BSTR) which contains a list of strings (Example: "OPEN", "CLOSE", "IN TRANSIT", etc.) corresponding to sequential numeric values (0, 1, 2, etc.)

```

Example Dim AnOPCItem as OPCItem Set OPCItem =
GetOPCItem(SomeItemServerHandle) VB
Syntax Example (getting the
property): Dim CurrentValue As
Integer Dim SomeValue As
Integer CurrentValue = AnOPCItem.EUType
    
```

4.6.3.14 EUInfo

Description (Read-only) Variant that contains the Engineering Units information

Syntax EUInfo As Variant

See Also OPCITEMATTRIBUTES in the OPC Data Access Custom Interface Specification

```

Example Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(SomeItemServerHandle)
VB Syntax Example (getting the property):
Dim CurrentValue As Variant
Dim SomeValue As Variant
CurrentValue = AnOPCItem.EUInfo
    
```

4.6.4 OPCItem Methods

4.6.4.1 Read

DescriptionRead makes a blocking call to read this item from the server. Read can be called with only a source (either OPCache or OPCDevice) to refresh the item's value, quality and timestamp properties. If the value, quality and timestamp must be in sync, this method's optional parameters return values that were acquired together.

Syntax Read (Source As Integer, Optional ByRef Value As Variant, Optional ByRef Quality As Variant, Optional ByRef TimeStamp AsVariant)

Part	Description
Source	The 'data source'; OPC_DS_CACHE or OPC_DS_DEVICE
Value	Returns the latest value read from the server
Quality	Returns the latest value read from the server
TimeStamp	Returns the latest timestamp read from the server.

Example Private Sub ReadButton_Click()

```

Dim AnOPCItem as
OPCItem

Set OPCItem =
GetOPCItem(SomeItemServerHandle)

Dim Source As
Integer

Dim Value As Variant

Dim Quality As
Variant

Dim TimeStamp As
Variant

Source =
OPC_DS_DEVICE

AnOPCItem.Read Source, ServerHandles, Value, Quality,
                TimeStamp

                \ process
the values

```

```

        TextBox.Text
t = Value

End
Sub

```

4.6.4.2 Write

Description Write makes a blocking call to write this value to the server.

Syntax Write (Value As Variant)

Part

Description

Value

Value to be written to the data source.

```

Example          Private          Sub          WriteButton_Click()

Dim AnOPCItem as OPCItem

Set OPCItem = GetOPCItem(SomeItemServerHandle)

Dim Value As Variant

Value = 1975

AnOPCItem.Write Value

End Sub

```

5 OPC Data Access Automation Definitions and Symbols

5.1 OPCNamespaceTypes

Symbol	Description
OPCHierarchical	
OPCFlat	

5.2 OPCDataSource

Symbol	Description
OPCCache	
OPCDevice	

5.3 OPCAccessRights

Symbol	Description
OPCReadable	
OPCWritable	

5.4 OPCServerState

Symbol	Description
OPCRunning	The server is running normally. This is the usual state for a server
OPCFailed	A vendor specific fatal error has occurred within the server. The server is no longer functioning. The recovery procedure from this situation is vendor specific. An error code of E_FAIL should generally be returned from any other server method.
OPCNoconfig	The server is running but has no configuration information loaded and thus cannot function normally. Note this state implies that the server needs configuration information in order to function. Servers which do not require configuration information should not return this state.
OPCSuspended	The server has been temporarily suspended via some vendor specific method and is not getting or sending data. Note that Quality will be returned as OPC_QUALITY_OUT_OF_SERVICE.
OPCTest	The server is in Test Mode. The outputs are disconnected from the real hardware but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. Quality will generally be returned normally.
OPCDisconnected	The Automation server object is not connected to an OPC custom interface server

5.5 OPCErrors

OPC Error	Value	Description
OPCInvalidHandle	0xC0040001L	The value of the handle is invalid. Note: a client should never pass an invalid handle to a server. If this error occurs, it is due to a programming error in the client or possibly in the server.
OPCBadType	0xC0040004L	The server cannot convert the data between the specified format/ requested data type and the canonical data type.
OPCPublic	0xC0040005L	The requested operation cannot be done on a public group.
OPCBadRights	0xC0040006L	The Items AccessRights do not allow the operation.

OPCUnknownItemID	0xC0040007L	The item ID is not defined in the server address space (on add or validate) or no longer exists in the server address space (for read or write).
OPCInvalidItemID	0xC0040008L	The item ID doesn't conform to the server's syntax.
OPCInvalidFilter	0xC0040009L	The filter string was not valid
OPCUnknownPath	0xC004000AL	The item's access path is not known to the server.
OPCRange	0xC004000BL	The value was out of range.
OPCDuplicateName	0xC004000CL	Duplicate name not allowed.
OPCUnsupportedRate	0x0004000DL	The server does not support the requested data rate but will use the closest available rate.
OPCClamp	0x0004000EL	A value passed to WRITE was accepted but the output was clamped.
OPCInuse	0x0004000FL	The operation cannot be performed because the object is being referenced.
OPCInvalidConfig	0xC0040010L	The server's configuration file is an invalid format.
OPCNotFound	0xC0040011L	Requested Object (e.g. a public group) was not found.
OPCInvalidPID	0xC0040203L	The passed property ID is not valid for the item.

Appendix A - OPC Automation Error Handling

When a run-time error occurs, the properties of the Visual Basic **Err** object are filled with information that uniquely identifies the error.

If your Visual Basic code is not set up to handle the error using the On Error mechanism, an exception will be generated, and depending on the context (Visual Basic in Debug Mode, or an executable), a message box will be invoked with the following information:

Runtime Error: decimal error number (hex error number)

Method "X" of Object "Y" Failed. (Note when your application is an executable, no value for X and Y are displayed)

Therefore, it is highly recommended by the OPC Foundation, that your application take appropriate steps to catch any OPC Automation errors that may occur as a result of setting properties or invoking methods on the OPC Data Access Automation Objects.

An *error handler* is a routine for trapping and responding to errors in your application. An OPC Automation client should add error handlers for any application functionality that involves setting a property or calling a method of OPC Data Access Automation Objects. The process of designing an error handler involves three steps:

1. Set, or *enable*, an error trap by telling the application where to branch to (which error-handling routine to execute) when an error occurs. The On Error statement enables the trap and directs the application to the label marking the beginning of the error-handling routine.
- 2 Write an error-handling routine that will handle errors from setting properties or from method

invocation on OPC Data Access Automation objects.

3 Exit the error-handling routine. Decide what action your application should take as a result of the error. For example, if you attempted to add a group with a duplicate name (provided from the end user), you could advise the end user that the

group was not added, and to enter a different name. Your application could also take the approach of adding the group again (with a “”), letting the server generate the name.

Watching for Errors

An error trap is enabled when Visual Basic executes the On Error statement, which specifies an error handler. The error trap remains enabled while the procedure containing it is active — that is, until an Exit Sub, Exit Function, Exit Property, End Sub, End Function, or End Property statement is executed for that procedure.

To set an error trap that jumps to an error-handling routine, use a On Error GoTo *line* statement, where *line* indicates the label identifying the error-handling code.

Handling the Errors

The first step in writing an error-handling routine is adding a line label to mark the beginning of the error-handling routine. The line label should have a descriptive name and must be followed by a colon.

The body of the error handling routine contains the code that actually handles the error, usually in the form of a Case or If...Then...Else statement. You need to determine which errors are likely to occur and provide a course of action for each.

The Number property of the Err object contains a numeric code representing the most recent run-time error.

The error number from the Number property on the Err object contains the value that you would call GetErrorString with to convert the error number into a readable string.

A Sample OPC Automation Error Code Fragment

```
Dim AnOpcServer As OPCServer
```

```
Private Sub Command1_Click()
    On Error GoTo testerror
    Set AnOpcServer = New OPCServer ' assuming fuzz does not exist so the connect fails and your VB code goes to the label testerror
    AnOpcServer.Connect ("fuzz")

```

```
Time = AnOpcServer.CurrentTime
Debug.Print Time
```

```
testerror:
Debug.Print Err.Number
End Sub
```

Appendix B – Sample String Filter Syntax Function

Syntax `BOOL MatchPattern(LPCTSTR string, LPCTSTR pattern, BOOL bCaseSensitive)` **Return Value** If *string* matches *pattern*, return is **TRUE**; if there is no match, return is **FALSE**. If either *string* or *pattern* is Null, return is **FALSE**; **Parameters** *string* String to be compared with *pattern*. *pattern* Any string conforming to the pattern-matching conventions described in Remarks. *bCaseSensitive* **TRUE** if comparison should be case sensitive.

Remarks

A versatile tool used to compare two strings. The pattern-matching features allow you to use wildcard characters, character lists, or character ranges, in any combination, to match strings. The following table shows the characters allowed in *pattern* and what they match:

Characters in *pattern* Matches in *string* ? Any

single character.

* Zero or more characters.
Any single digit (0-9).
[*charlist*] Any single character in *charlist*.
[!*charlist*] Any single character not in *charlist*.

A group of one or more characters (*charlist*) enclosed in brackets ([]) can be used to match any single character in *string* and can include almost any character code, including digits.

Note To match the special characters left bracket ([), question mark (?), number sign (#), and asterisk (*), enclose them in brackets. The right bracket (]) can't be used within a group to match itself, but it can be used outside a group as an individual character.

By using a hyphen (-) to separate the upper and lower bounds of the range, *charlist* can specify a range of characters. For example, [A-Z] results in a match if the corresponding character position in *string* contains any uppercase letters in the range A-Z. Multiple ranges are included within the brackets without delimiters.

Other important rules for pattern matching include the following:

An exclamation point (!) at the beginning of *charlist* means that a match is made if any character except the characters in *charlist* is found in *string*. When used outside brackets, the exclamation point matches itself.

A hyphen (-) can appear either at the beginning (after an exclamation point if one is used) or at the end of *charlist* to match itself. In any other location, the hyphen is used to identify a range of characters.

When a range of characters is specified, they must appear in ascending sort order (from lowest to highest). [A-Z] is a valid pattern, but [Z-A] is not.

The character sequence [] is considered a zero-length string ("").

Appendix C - Data Access Automation IDL Specification

```
// OPCAuto.idl : OPC Automation 2.0 interface// Version 2.19.00//// The following
naming is used to make Visual Basic see the correct names:
```

```
// OPCxxx is the name used in the spec, IOPCxxx is an interface
// OPCBrowser, OPCGroups, OPCItems are unchanged from the spec
// OPCServer is the name of the coclass containing IOPCAutoServer
// IOPCAutoServer is the actual interface (IOPCServer is already used!)
// OPCGroup is the name of the coclass
// IOPCGroup is the actual interface
// DIOPCGroupEvent is the group's event disp-interface
```

```
// This file will be processed by the MIDL tool to// produce the type library (OPCAuto.tlb) and
marshalling code.#define DISPID_NEWENUM -4
```

```
import "oaidl.idl";import "ocidl.idl";
```

```
interface OPCBrowser; // Forward referencesinterface OPCGroups;interface OPCGroup;interface
OPCItems;interface OPCItem;
```

```
/**/
```

```
[
    object,
    dual,
    uuid(28E68F90-8D75-11d1-8DC3-3C302A000000),
```

```

helpstring("OPC Server Event"),
pointer_default(unique),
oleautomation

]
interface IOPCServerEvent : IDispatch
{

HRESULT ServerShutDown([in,string] BSTR Reason );

};

//*****

[
    object,
    dual,
    uuid(28E68F9C-8D75-11d1-8DC3-3C302A000000),
    helpstring("OPCGroups Event"),
    pointer_default(unique),
    oleautomation

]
interface IOPCGroupsEvent : IDispatch
{

                [helpstring("Event to update item data from any group")]HRESULT
GlobalDataChange(
[in] LONG TransactionID,[in] LONG GroupHandle,[in] LONG NumItems,[in]
SAFEARRAY(LONG) * ClientHandles,[in] SAFEARRAY(VARIANT)* ItemValues,

                [in] SAFEARRAY(LONG) * Qualities,
                [in] SAFEARRAY( DATE) * TimeStamps);

};

//*****

[
    object,
    dual,
    uuid(28E68F90-8D75-11d1-8DC3-3C302A000001),
    helpstring("OPCGroup Events"),
    pointer_default(unique),
    oleautomation

]
interface IOPCGroupEvent : IDispatch
{

```

```

        [helpstring("Event to notify when active data has
changed")]
HRESULT DataChange([in] LONG TransactionID,[in] LONG NumItems,[in]
SAFEARRAY(LONG) * ClientHandles,[in] SAFEARRAY(VARIANT)*
ItemValues,[in] SAFEARRAY(LONG) * Qualities,[in]
SAFEARRAY(DATE) * TimeStamps);

        [helpstring("Event to update item data when a read request was
completed")]
HRESULT AsyncReadComplete([in] LONG TransactionID,[in] LONG NumItems,[in]
SAFEARRAY(LONG) * ClientHandles,[in] SAFEARRAY(VARIANT)*
ItemValues,[in] SAFEARRAY(LONG) * Qualities,[in]
SAFEARRAY(DATE) * TimeStamps,[in] SAFEARRAY(LONG) * Errors);

        [helpstring("Event to notify when a write request was
completed")]
HRESULT AsyncWriteComplete([in] LONG TransactionID,[in] LONG NumItems,[in]
SAFEARRAY(LONG) * ClientHandles,[in] SAFEARRAY(LONG) * Errors);

        [helpstring("Event to
notify when a cancel transaction request was
completed")] HRESULT
AsyncCancelComplete([in] LONG
TransactionID);};

//*****
[
    uuid(28E68F91-8D75-11d1-8DC3-3C302A000000),
    version(1.0),
    helpstring("OPC Automation 2.0")
]
library OPCAutomation
{

    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    enum OPCNamespaceTypes { OPCHierarchical = 1, OPCFlat};enum OPCDataSource {
    OPCCache = 1, OPCDevice};enum OPCAccessRights { OPCReadable = 1, OPCWritable};enum
    OPCServerState{ OPCRunning = 1, OPCFailed,
        OPCNoconfig, OPCsuspended,OPCTest, OPCDisconnected };
    enum OPCErrors{ OPCInvalidHandle = 0xC0040001L,OPCBadType =
        0xC0040004L,OPCPublic = 0xC0040005L,OPCBadRights =

```

```
0xC0040006L,OPCUnknownItemID = 0xC0040007L,OPCInvalidItemID =
0xC0040008L,OPCInvalidFilter = 0xC0040009L,OPCUnknownPath =
0xC004000AL,OPCRange = 0xC004000BL,OPCDuplicateName =
0xC004000CL,OPCUnsupportedRate = 0x0004000DL,OPCClamp =
0x0004000EL,OPCInuse = 0x0004000FL,OPCInvalidConfig =
0xC0040010L,OPCNotFound = 0xC0040011L,OPCInvalidPID =
0xC0040203L };
```

```
/**/
```

```
// OPCServer Interface
```

```
[
```

```
    object,
    dual,oleautomation,
    uuid(28E68F92-8D75-11d1-8DC3-3C302A000000),
    helpstring("OPCServer Object"),
    pointer_default(unique)
```

```
]
```

```
interface IOPCAutoServer : IDispatch
```

```
{
```

```
    // Properties[propget,helpstring("Time
    the Server
    Started")]HRESULT
    LT
    StartTime([out,
    retval] DATE *
    StartTime );
```

```
        [propget]
```

```
HRESULT CurrentTime([out, retval] DATE * CurrentTime );
```

```
        [propget,helpstring("Last time the server sent data")]HRESULT
```

```
LastUpdateTime([out, retval] DATE * LastUpdateTime );
```

```
        [propget]
```

```
HRESULT MajorVersion([out, retval] short * MajorVersion );
```

```
        [propget]
```

```
HRESULT MinorVersion([out, retval] short * MinorVersion );
```

```
        [propget]
```

```
HRESULT BuildNumber([out, retval] short * BuildNumber );
```

```

        [propget,helpstring("Server Vendor's name")]HRESULT
VendorInfo([out, retval] BSTR * VendorInfo );

        [propget,helpstring("Returns an
OPCServerState")]HRESULT ServerState([out, retval] LONG * ServerState );

        [propget,helpstring("Returns this server's name")]HRESULT
ServerName([out, retval] BSTR * ServerName );

        [propget,helpstring("Returns this server's node")]HRESULT
ServerNode([out, retval] BSTR * ServerNode );

        [propget,helpstring("Identify
the client")]HRESULT ClientName([out, retval]
BSTR * ClientName );[propput]HRESULT
ClientName([in] BSTR ClientName );

        [propget]HR
ESULT LocaleID([out, retval]
LONG * LocaleID
);[propput]HRESULT
LocaleID([in] LONG LocaleID );

        [propget,helpstrin
g("Might possibly be
Percentutilization")]HRESULT Bandwidth([out,
retval] LONG * Bandwidth );

        [id(0),propget,helpst
ring("The collection of
OPCGroupObjects")]HRESULT
OPCGroups([out, retval] OPCGroups **
ppGroups );

        [propget,helpstring("Returns an array of names")]HRESULT
PublicGroupNames([out, retval] VARIANT * PublicGroups );

// Methods

        [helpstring("Returns an array of Server names, optionally on another
node")]
HRESULT GetOPCServers([in, optional] VARIANT Node,[out, retval] VARIANT *
OPCServers );

        [helpstring("Connect to a named OPC Server")]
HRESULT Connect([in, string] BSTR ProgID,[in, optional] VARIANT Node);

        [helpstring("End Connection with OPC
Server")]HRESULT Disconnect();

```

```

        [helpstring("Create
a new OPCBrowser Object")]HRESULT
CreateBrowser([out, retval] OPCBrowser
** ppBrowser );
        [helpstring("Convert an error code to a descriptive
string")]
HRESULT GetErrorString([in] LONG ErrorCode,[out, retval] BSTR * ErrorString );

        [helpstring("The LocaleIDs supported by
this server")]HRESULT QueryAvailableLocaleIDs([out, retval]
VARIANT * LocaleIDs );

HRESULT QueryAvailableProperties([in, string] BSTR ItemID,[out] LONG * Count,[out]
SAFEARRAY(LONG) * PropertyIDs,[out] SAFEARRAY(BSTR) *
Descriptions,[out] SAFEARRAY(SHORT) * DataTypes );

HRESULT GetItemProperties([in, string] BSTR ItemID,[in] LONG Count,[in]
SAFEARRAY(LONG) * PropertyIDs,[out] SAFEARRAY(VARIANT)*
PropertyValues,[out] SAFEARRAY(LONG) * Errors );

HRESULT LookupItemIDs(

        [in, string]          BSTR          ItemID,
        [in]                  LONG          Count,
        [in]                  SAFEARRAY(LONG) * PropertyIDs,
        [out]                 SAFEARRAY(BSTR) * NewItemIDs,
        [out]                 SAFEARRAY(LONG) * Errors );

};

//*****
// OPCServer's Event fired back to the client
[

uuid(28E68F93-8D75-11d1-8DC3-3C302A000000),
nonextensible,
helpstring("OPC Server Event")

]
dispinterface DIOPCServerEvent
{

properties:
methods:
[id(1)] void ServerShutDown(

[in,string] BSTR Reason );

```

```
};
```

```

/*****
// OPCBrowser Interface
[
object,dual, oleautomation,uuid(28E68F94-8D75-11d1-8DC3-3C302A000000),helpstring("OPC
Browser"),
    pointer_default(unique)
]
interface OPCBrowser : IDispatch
{
    // Properties[propget,helpstring("Returns one of
        OPCNamespaceTypes")]HR
        ESULT Organization([out,
        retval] LONG * Organization
        );

        [propget,helpstring("Filter narrows the
search results")]HRESULT Filter([out, retval] BSTR * Filter
);[propput]HRESULT Filter([in] BSTR Filter );

        [propget,helpstring("
Data type used in ShowLeafs (anyVariant
type")]HRESULT DataType([out, retval]
SHORT * DataType );[propput]HRESULT
DataType([in] SHORT DataType );

        [propget,helpstring("Access Rights used in
ShowLeafs ()")]HRESULT AccessRights([out, retval] LONG *
AccessRights );[propput]HRESULT AccessRights([in] LONG
AccessRights );

        [propget,helpstring("Position in the Tree")]HRESULT CurrentPosition([out,
retval] BSTR * CurrentPosition );

        [propget,helpstring("Number of items in the Collection")]HRESULT
Count([out, retval] LONG * Count );

        [propget, restricted, id( DISPID_NEWENUM )]HRESULT _NewEnum([out,
retval] IUnknown ** ppUnk );

// Methods
HRESULT Item(
        [in]                VARIANT ItemSpecifier,
        [out, retval]       BSTR * Item );

```

```

        [helpstring("Get all of the branch names that match the
filter")]
HRESULT ShowBranches();

        [helpstri
ng("Get all of the leaf names that
match thefilter")]HRESULT
ShowLeafs([in, optional] VARIANT
Flat);

        [helpstring("Move up a level in the tree")]HRESULT MoveUp();

        [helpstring("Move up to the top (root) of the tree")]HRESULT
MoveToRoot();

        [helpstring("Move down into this branch")]HRESULT MoveDown(
[in, string] BSTR Branch );

        [helpstring("Move to this absolute position")]HRESULT MoveTo(
[in] SAFEARRAY(BSTR) * Branches );

        [helpstring("Converts a leaf name to an ItemID")]HRESULT GetItemID(
[in, string] BSTR Leaf,
[out, retval] BSTR * ItemID );

        [helpstring("Returns an
array of Access Paths for anItemID")]HRESULT
GetAccessPaths(
[in, string] BSTR ItemID,
[out, retval] VARIANT * AccessPaths );
};

//*****
// OPCGroups Interface
[
object,
dual,oleautomation,
uuid(28E68F95-8D75-11d1-8DC3-3C302A000000),
helpstring("Collection of OPC Group objects"),
pointer_default(unique)

]
interface IOPCGroups : IDispatch

```



```

{
    // Properties[propget,helpstring("Returns the parent
        OPCServer")]HRESULT
        Parent([out, retval]
        IOPCAutoServer ** ppParent );

        [propget]HRESUL
T DefaultGroupIsActive([out, retval]
VARIANT_BOOL
*DefaultGroupIsActive
);[propput]HRESULT
DefaultGroupIsActive([in]
VARIANT_BOOL DefaultGroupIsActive
);

        [propget]HRESULT
DefaultGroupUpdateRate([out, retval] LONG *
DefaultGroupUpdateRate );[propput]HRESULT
DefaultGroupUpdateRate([in] LONG
DefaultGroupUpdateRate );

        [propget]HRESULT
DefaultGroupDeadband([out, retval] float *
DefaultGroupDeadband );[propput]HRESULT
DefaultGroupDeadband([in] float
DefaultGroupDeadband );

        [propget]HRESULT
DefaultGroupLocaleID([out, retval] LONG *
DefaultGroupLocaleID );[propput]HRESULT
DefaultGroupLocaleID([in] LONG
DefaultGroupLocaleID );

        [propget]
HRESULT DefaultGroupTimeBias([out, retval] LONG *
DefaultGroupTimeBias );
        [propput]

HRESULT DefaultGroupTimeBias([in] LONG DefaultGroupTimeBias );

        [propget,helpstring("Number of items in the
Collection")]HRESULT Count([out, retval] LONG * Count );

        [propget, restricted, id( DISPID_NEWENUM )]HRESULT
_NewEnum([out, retval] IUnknown ** ppUnk );

// Methods

```

```

[id(0),helpstring("Returns an OPCGroup by index (starts at1) or
name")]
HRESULT Item([in] VARIANT ItemSpecifier,[out, retval] OPCGroup ** ppGroup );
[helpstring("Adds an OPCGroup to the collection")]
HRESULT Add([in,optional] VARIANT Name,[out, retval] OPCGroup ** ppGroup );
[helpstring("Returns an OPCGroup specified by server handleor
name")]
HRESULT GetOPCGroup([in] VARIANT ItemSpecifier,[out, retval] OPCGroup **
ppGroup );
[helpstring("Remove all groups and their
items")]HRESULT RemoveAll();

[h
elpstring("Removes an
OPCGroup specified by server
handleor name")]HRESULT
Remove([in] VARIANT
ItemSpecifier );
[helpstring("Adds an existing public OPCGroup to the
collection")]
HRESULT ConnectPublicGroup([in] BSTR Name,[out, retval] OPCGroup ** ppGroup );

[helpstrin
g("Removes a public OPCGroup
specified by serverhandle or
name")]HRESULT
RemovePublicGroup([in] VARIANT
ItemSpecifier );
};

//*****
*
// OPCGroup's Events fired back to the client
[
    uuid(28E68F9D-8D75-11d1-8DC3-3C302A000000),
    nonextensible,
    helpstring("OPC Groups Event")
]
dispinterface DIOPCGroupsEvent
{

```

properties:

methods:

[id(1)] void GlobalDataChange(

[in] LONG TransactionID,[in] LONG GroupHandle,[in] LONG NumItems,[in]
SAFEARRAY(LONG) * ClientHandles,[in] SAFEARRAY(VARIANT)* ItemValues,

[in] SAFEARRAY(LONG) * Qualities,
[in] SAFEARRAY(DATE) * TimeStamps);

};

/******

// IOPCGroup Interface

[

object,

dual,oleautomation,

uuid(28E68F96-8D75-11d1-8DC3-3C302A000000),

helpstring("OPC Group Object"),

pointer_default(unique)

]

interface IOPCGroup : IDispatch

{

// Properties[propget,helpstring("Returns the parent
OPCServer")]HRESULT
Parent([out, retval]
IOPCAutoServer ** ppParent);

[propget]HRESULT Name([out, retval] BSTR * Name
);[propput]HRESULT Name([in] BSTR Name);

[propget,helpstring("True if this group is public")]HRESULT IsPublic([out,
retval] VARIANT_BOOL * IsPublic);

[propget,helpstring("True if this group is
active")]HRESULT IsActive([out, retval] VARIANT_BOOL *
IsActive);[propput]HRESULT IsActive([in] VARIANT_BOOL
IsActive);

[propget,helpstring("True if this
group will getasynchronous data updates")]HRESULT
IsSubscribed([out, retval] VARIANT_BOOL * IsSubscribed
);[propput]HRESULT IsSubscribed([in] VARIANT_BOOL
IsSubscribed);

[propget]HRESULT ClientHandle([out,
retval] LONG * ClientHandle);[propput]HRESULT

ClientHandle([in] LONG ClientHandle);

[propget]

HRESULT ServerHandle([out, retval] LONG * ServerHandle);

[propget]

HRESULT LocaleID([out, retval] LONG * LocaleID);

[propput]

HRESULT LocaleID([in] LONG LocaleID);

[propget]HRESUL

T TimeBias([out, retval] LONG *

TimeBias);[propput]HRESULT

TimeBias([in] LONG TimeBias);

[propget]HRESUL

T DeadBand([out, retval] FLOAT *

DeadBand);[propput]HRESULT

DeadBand([in] FLOAT DeadBand);

[propget,helpstring("Rate

data can be returned to an application (in

mSec"))]HRESULT UpdateRate([out, retval] LONG *

UpdateRate);[propput]HRESULT UpdateRate([in]

LONG UpdateRate);

[id(0),propget,helpstring

("Returns the OPCItemscollection")]HRESULT

OPCItems([out, retval] OPCItems ** ppItems);

// Methods

HRESULT SyncRead([in] SHORT Source,[in] LONG NumItems,[in]

SAFEARRAY(LONG) * ServerHandles,[out]

SAFEARRAY(VARIANT) * Values,[out] SAFEARRAY(LONG) *

Errors,[out,optional] VARIANT * Qualities,[out,optional]

VARIANT * TimeStamps);

HRESULT SyncWrite([in] LONG NumItems,[in] SAFEARRAY(LONG) *

ServerHandles,[in] SAFEARRAY(VARIANT) * Values,[out]

SAFEARRAY(LONG) * Errors);

HRESULT AsyncRead([in] LONG NumItems,[in] SAFEARRAY(LONG) *

ServerHandles,[out] SAFEARRAY(LONG) * Errors,[in] LONG

TransactionID,[out] LONG * CancelID);

HRESULT AsyncWrite([in] LONG NumItems,[in] SAFEARRAY(LONG) *

```

ServerHandles,[in] SAFEARRAY(VARIANT) * Values,[out]
SAFEARRAY(LONG) * Errors,[in] LONG TransactionID,[out]
LONG * CancelID);

HRESULT AsyncRefresh([in] SHORT Source,[in] LONG TransactionID,
[out] LONG * CancelID);

HRESULT AsyncCancel( [in] LONG CancelID);

};

//*****
// OPCGroup's Events fired back to the client
[

uuid(28E68F97-8D75-11d1-8DC3-3C302A000000),
nonextensible,
helpstring("OPC Group Events")

]
dispinterface DIOPCGroupEvent
{

properties:
methods:
[id(1)] void DataChange(

[in] LONG TransactionID,[in] LONG NumItems,[in] SAFEARRAY(LONG) * ClientHandles,[in]
SAFEARRAY(VARIANT)* ItemValues,[in] SAFEARRAY(LONG) * Qualities,[in]
SAFEARRAY(DATE) * TimeStamps);

[id(2)] void AsyncReadComplete([in] LONG TransactionID,[in] LONG NumItems,[in]
SAFEARRAY(LONG) * ClientHandles,[in] SAFEARRAY(VARIANT)*
ItemValues,[in] SAFEARRAY(LONG) * Qualities,[in]
SAFEARRAY(DATE) * TimeStamps,[in] SAFEARRAY(LONG) * Errors);

[id(3)] void AsyncWriteComplete([in] LONG TransactionID,[in] LONG NumItems,[in]
SAFEARRAY(LONG) * ClientHandles,[in] SAFEARRAY(LONG) * Errors);

[id(4)] void AsyncCancelComplete([in] LONG CancelID);};

//*****
// OPCItems Collection Interface
[

object,
dual,oleautomation,
uuid(28E68F98-8D75-11d1-8DC3-3C302A000000),
helpstring("Collection of OPC Item objects"),
pointer_default(unique)

```

```

]
interface OPCItems : IDispatch
{

// Properties

                [propget,helpstring("Returns the parent
OPCGroup")]HRESULT Parent([out, retval] OPCGroup ** ppParent );

                [propget]H
RESULT
DefaultRequestedDataType([out,
retval] SHORT
*DefaultRequestedDataType
);[propput]HRESULT
DefaultRequestedDataType([in]
SHORT
DefaultRequestedDataType );

                [propget]HRESU
LT DefaultAccessPath([out, retval]
BSTR * DefaultAccessPath
);[propput]HRESULT
DefaultAccessPath([in, string] BSTR
DefaultAccessPath );

                [propget]HRESUL
T DefaultIsActive([out, retval]
VARIANT_BOOL * DefaultIsActive
);[propput]HRESULT
DefaultIsActive([in] VARIANT_BOOL
DefaultIsActive );

                [propget,helpstring("Number of items in the
Collection")]HRESULT Count([out, retval] LONG * Count );

                [propget, restricted, id( DISPID_NEWENUM )]HRESULT
_NewEnum([out, retval] IUnknown ** ppUnk );

// Methods

                [id(0),helpstring("Returns an OPCItem by index (starts at
1)")]
HRESULT Item([in] VARIANT ItemSpecifier,[out, retval] OPCItem ** pItem );

                [helpstring("Returns an OPCItem specified by server
handle")]

```

```

HRESULT GetOPCItem([in] LONG ServerHandle,[out, retval] OPCItem ** ppItem );
    [helpstring("Adds an OPCItem object to the collection")]

HRESULT AddItem([in, string] BSTR ItemID,[in] LONG ClientHandle,[out, retval]
    OPCItem ** ppItem );
    [helpstring("Adds OPCItem objects to the collection")]

HRESULT AddItems([in] LONG NumItems,[in] SAFEARRAY(BSTR) * ItemIDs,[in]
    SAFEARRAY(LONG) * ClientHandles,[out]
    SAFEARRAY(LONG) * ServerHandles,[out]
    SAFEARRAY(LONG) * Errors,[in, optional] VARIANT
    RequestedDataTypes,[in, optional] VARIANT AccessPaths);
    [helpstring("Removes OPCItem objects from the
collection")]
HRESULT Remove(
[in] LONG NumItems,[in] SAFEARRAY(LONG) * ServerHandles,[out] SAFEARRAY(LONG) *
Errors);
[helpstring("?")]

HRESULT Validate([in] LONG NumItems,[in] SAFEARRAY(BSTR) * ItemIDs,[out]
    SAFEARRAY(LONG) * Errors,[in, optional] VARIANT
    RequestedDataTypes,[in, optional] VARIANT AccessPaths);
[helpstring("Set the active state of OPCItem objects")]

HRESULT SetActive([in] LONG NumItems,[in] SAFEARRAY(LONG) * ServerHandles,[in]
    VARIANT_BOOL ActiveState,[out] SAFEARRAY(LONG) * Errors);
[helpstring("Set the Client handles of OPCItem objects")]

HRESULT SetClientHandles([in] LONG NumItems,[in] SAFEARRAY(LONG) *
    ServerHandles,[in] SAFEARRAY(LONG) * ClientHandles,[out]
    SAFEARRAY(LONG) * Errors);
    [helpstring("Set the Data Types of OPCItem objects")]
HRESULT
SetDataTypes(
    [in] [in] [in]          LONG NumItems, SAFEARRAY(LONG) *
    [out]                  ServerHandles, SAFEARRAY(LONG) *
                           RequestedDataTypes, SAFEARRAY(LONG) *
                           Errors);

};

//*****
// OPCItem Interface
[

```

```

object,
dual,oleautomation,
uuid(28E68F99-8D75-11d1-8DC3-3C302A000000),
helpstring("OPC Item object"),
pointer_default(unique)

]
interface OPCItem : IDispatch
{

    // Properties[propget,helpstring("Returns the parent
    OPCGroup")]HRESULT
    Parent([out, retval] OPCGroup
    ** Parent );

    [propget]HRESULT ClientHandle([out,
    retval] LONG * ClientHandle );[propput]HRESULT
    ClientHandle([in] LONG ClientHandle );

    [propget]
    HRESULT ServerHandle([out, retval] LONG * ServerHandle );

    [propget]
    HRESULT AccessPath([out, retval] BSTR * AccessPath );

    [propget]
    HRESULT AccessRights([out, retval] LONG * AccessRights );

    [propget]
    HRESULT ItemID([out, retval] BSTR * ItemID );

    [propget]
    HRESULT IsActive([out, retval] VARIANT_BOOL * IsActive );
    [propput]
    HRESULT IsActive([in] VARIANT_BOOL IsActive );

    [propget]
    HRESULT RequestedDataType([out, retval] SHORT * RequestedDataType );
    [propput]
    HRESULT RequestedDataType([in] SHORT RequestedDataType );

```



```

        [id(0),propget]
HRESULT Value([out, retval] VARIANT * CurrentValue );

        [propget]
HRESULT Quality([out, retval] LONG * Quality );

        [propget]
HRESULT TimeStamp([out, retval] DATE * TimeStamp );

        [propget]
HRESULT CanonicalDataType([out, retval] SHORT * CanonicalDataType );

        [propget]
HRESULT EUType([out, retval] SHORT * EUType );

        [propget]
HRESULT EUInfo([out, retval] VARIANT * EUInfo );

// Methods
HRESULT Read(
        [in]                SHORT    Source,
        [out,optional]      VARIANT  * Value,
        [out,optional]      VARIANT  * Quality,
        [out,optional]      VARIANT  * TimeStamp);

HRESULT Write(
        [in]                VARIANT  Value);
};

/*****

[
    uuid(28E68F9A-8D75-11d1-8DC3-3C302A000000),
    helpstring("OPC Automation Server")
]
coclass OPCServer

    {[default] interface IOPCAutoServer;[source, default] dispinterface DIOPCServerEvent;
    };

/*****

```

```
[
    uuid(28E68F9E-8D75-11d1-8DC3-3C302A000000),
    helpstring("OPC Automation Groups Collection")
]
coclass OPCGroups
{

    [default] interface IOPCGroups;[source, default] dispinterface
    DIOPCGroupsEvent;};

//*****

[
    uuid(28E68F9B-8D75-11d1-8DC3-3C302A000000),
    helpstring("OPC Automation Group")
]
coclass OPCGroup
{

    [default] interface IOPCGroup;[source, default] dispinterface
    DIOPCGroupEvent;};};
```

Appendix D- Notes On Automation Data Types

The OPC Custom Interface allows servers to support data types including VT_I1, VT_UI2, VT_UI4, as well as arrays of these same data types. For a client that is developed in C++, using these data types is very straight forward, but for an automation client application, these data types are not natively supported. Therefore, we have chosen to provide a logical mapping and conversion to those data types which are more native to automation client applications.

The automation interface shall provide the standard automation data types therefore the requested data types that the automation client requests will be those that are natively supported by the automation applications. The problem comes in, when the automation client either does not specify a requested data type, or the server application rejects the requested data type, and the data is then returned in the servers native canonical data type.

The following is the conversion approach that the automation interface (and corresponding implementation) should provide to facilitate providing data values in the data type representation most suitable for automation applications. A value in the canonical data types representation will be converted to the automation data types according to the table below.

CANONICAL DATA TYPE	AUTOMATION DATA TYPE
VT_I1	VT_I2
VT_UI2	VT_I4
VT_UI4	VT_R8 (or VT_CY)
VT_ARRAY VT_I1	VT_ARRAY VT_I2
VT_ARRAY VT_UI2	VT_ARRAY VT_I4
VT_ARRAY VT_UI4	VT_ARRAY VT_R8 (or VT_CY)

These conversions rules are only applicable when the client either has not specified a requested data type, or the requested data type conversion are rejected by the server application. Note: The

canonical data type that is returned by the automation methods, will indicate the data type natively supported by the server, and not the automation data type (from the table above) that the value will be converted to. (This is for consistency with Custom interface client applications)