UTILISATION DE LA BIBLIOTHEQUE DE FONCTIONS DYNAMIQUE OLEPRN.DLL AVEC C#

I. Introduction

Au cours de ce tutoriel, je vais vous aider (enfin je l'espère) à concevoir une application permettant d'obtenir des informations sur chacune des imprimantes installées sur votre réseau. Pour ce faire nous allons utiliser la bibliothèque de fonctions dynamique oleprn.dll. Celle-ci renferme des fonctions qui permettent de dialoguer avec vos imprimantes avec le protocole SNMP. Nous allons ainsi pouvoir récupérer un certain nombre d'informations comme l'état de l'imprimante ou encore des consommables.

II. Le protocole SNMP

Avant de commencer à coder notre application, je voudrai vous donner quelques informations sur ce protocole.

1. Définition

SNMP (Simple Network Management Protocol) est un protocole permettant de communiquer avec différents périphériques sur un réseau si ceux-ci le prennent en charge. Cela permet donc aux administrateurs réseau de pouvoir diagnostiquer rapidement une faille touchant un de ces éléments sur le réseau. Dans ce tutoriel nous allons nous intéresser seulement aux imprimantes, mais ce protocole permet de mettre en place une surveillance des hubs, des switchs, etc…

On communique avec ces périphériques par l'intermédiaire d'objets gérables qui sont rangés dans une table ordonnée que l'on appelle MIB (« Management Information Base »). La MIB contient donc différents objets renfermant une information spécifique pouvant être lue ou mise à jour apr l'intermédiaire du protocole SNMP via un identifiant que l'on nomme OID.

Pour plus d'informations sur ce protocole je vous renvoie sur la page wiki en anglais qui donne plus d'informations que la versions française :
http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol

2. Activer le protocole sur les imprimantes

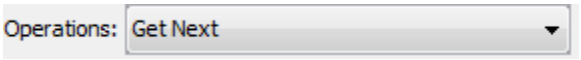
Normalement toutes imprimantes récentes sont compatibles avec ce protocole, mais celui-ci n'est pas forcément activer par défaut, il convient donc de le faire par l'intermédiaire de l'interface de gestion web de l'imprimante.

3. Logiciel de navigation de MIB

Différents logiciels spécialisé existent pour effectuer une navigation de MIBs. Parmi les plus connus on peut citer Nagios, Cacti, etc… Je vous propose un petit logiciel plus simple à utiliser. Il s'agit de MIB Browser édité par la société iReasoning Network et il existe en plusieurs versions dont une gratuite pour un usage personnel qui est la Free Personal Edition. Cette version est téléchargeable à l'adresse suivante : http://ireasoning.com/downloadmibbrowserlicense.shtml

Lorsque l'application est démarrée il suffit de renseigner l'adresse ip dans le champ texte suivant :

Une fois ceci effectué il devient possible de faire afficher les différentes OIDs :



En sélectionnant l'opération « Walk», vous aurez à votre disposition la totalité des OIDs disponibles pour ce périphérique, Voici un aperçu :

Result Table

Name/OID	Value
sysDescr.0	Xerox Phaser 8550DP;PS 3.11.0,Net 24.38.04.28.2005,Eng ...
sysObjectID.0	.1.3.6.1.4.1.253.8.62.1.19.6.3.1
sysUpTime.0	27 minutes 59 seconds (167975)
sysContact.0	
sysName.0	Phaser 8550DP
sysLocation.0	
sysServices.0	72
ifNumber.0	2
ifIndex.1	1
ifIndex.2	2

Les autres opérations possibles sont :

- « Get Next » permet de retourner la valeur OID du prochain objet dans l'arbre…
- « Get » permet de déterminer la valeur d'un objet en donnant en paramètre la valeur de l'OID. Bien entendu l'OID doit exister pour que cette commande aboutisse.
- « Get Bulk » permet de récupérer une liste de valeurs à a partir d'une liste d'OIDs. C'est une version améliorée de la commande « Get » idéale pour de très grandes MIBs.
- « Set » permet de modifier la valeur d'une des variable de la MIB

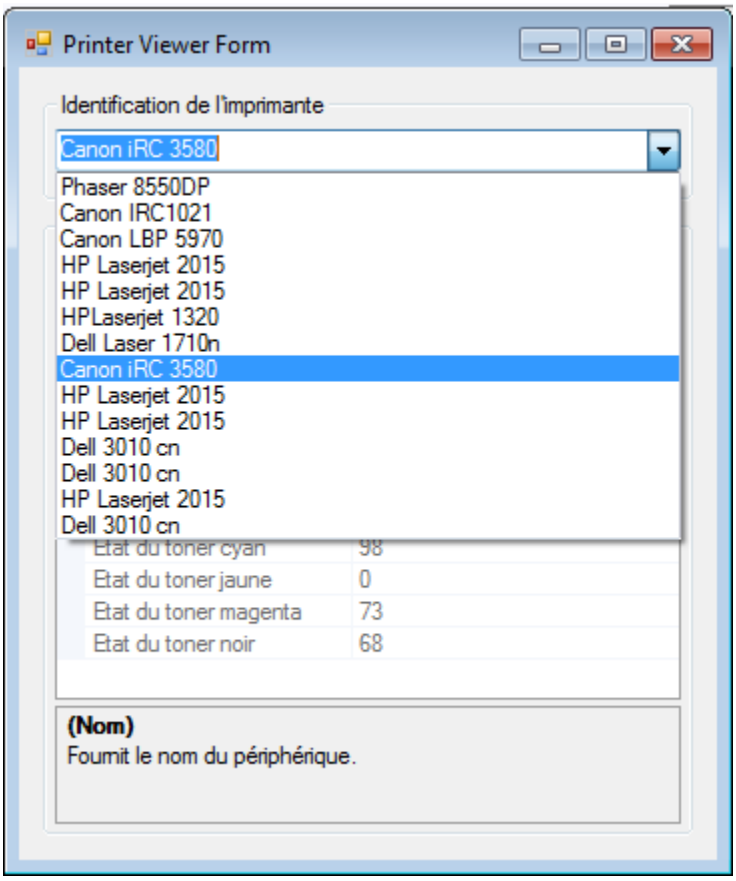
III. Codage de l'application

Maintenant nous allons passer aux choses sérieuses. Nous allons concevoir une petite applications nous permettant de récupérer les informations suivantes :

- Le nom, l'adresse IP, et l'emplacement à partir d'un fichier xml ,
- L'état de l'imprimante et des consommables par l'intermédiaire de la bibliothèque de fonctions dynamique oleprn.dll

1. Interface graphique

Avant de nous lancer dans le vif du sujet, arrêtons nous quelques instants sur l'interface graphique de notre application. Voilà le résultat final :



Commencez dans un premier temps par créer un projet de type Windows Form en C# que vous pouvez nommer à votre convenance. Me concernant, je lui ai donné comme nom : PrinterMgmt.

Pour réaliser ce type d'interface, allez dans le designer de la fiche principal et ajoutez-y les éléments suivants :

- Deux composants GroupBox ;
- Dans le GoupBox du haut ajoutez-y un composant Label et une Combobox ;
- Dans le GroupBox du bas ajoutez-y un composant propertyGrid.

C'est à peu près tout ce qu'il vous faut après vous pourrez personnaliser l'interface comme bon vous semble…

2. Le fichier xml répertoriant la liste des imprimantes

Maintenant intéressons nous à la source de données qui va nous permettre de lister les imprimantes présentes sur votre réseau. J'ai choisi ce type de source car il est universel et peut donc être utilisé par n'importe quel langage de programmation sur n'importe quelle plateforme voici la structure de notre fichier xml.

```
<?xml version="1.0" encoding="utf-8"?>

<ArrayOfPrinter xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <Printer Nom="nom_imprimante1" IpAddress="@ip1">

        </Printer>

        <Printer Nom=="nom_imprimante2"  IpAddress="@ip2"  Emplacement="Bureau par exemple"

        </Printer>

</ArrayOfPrinter>
```

Comme vous pouvez le constater pour chaque imprimante sur le réseau on crée un nouveau nœud avec pour attribut son nom, son adresse IP ainsi que son emplacement mais cet attribut est facultatif…

Bien entendu libre à vous de générer cet xml par l'intermédiaire d'un accès WMI ou tout autre moyen (en utilisant un logiciel permettant de scruter les adresses ip de votre réseau et capable de reconnaître le type de périphérique)…

Cette source de données va nous permettre d'alimenter notre combobox ainsi que notre propertyGrid avec des informations de base…

3. Codage de la classe Printer

Venons en au codage de l'application. L'essentiel de notre code va être fournit par la classe Printer qui va nous permettre de récupérer les informations sur chaque imprimante…

Avant de commencer ajouter un nouveau type de projet à votre solution : Une bibliothèque de classe. Nommez celui-ci « PrinterType », par exemple…

Ajoutez un nouvel élément de type classe que vous allez nommer Printer…

Nous allons avoir besoin d'une référence à la bibliothèque de fonctions oleprn.dll. Il faut donc l'ajouter par l'intermédiaire du menu contextuel. Sélectionner ensuite l'onglet COM et rechercher dans la liste « oleprn 1.0 type Library ». une fois valider vous devriez retrouver la référence dans la partie explorateur de solutions.

Nous avons à présent tous les éléments en notre possessions, attardons nous un instant sur les propriétés que nous voulons créer dans notre classe Printer :

- Nom : permet de récupérer le nom de l'imprimante (information fournie par notre source de données xml);
- IPAdress : permet de récupérer l'adresse IP de l'imprimante (information fournie par notre source de données xml) ;
- Emplacement : permet de récupérer l'emplacement de l'imprimante (information fournie par notre source de données xml);
- Status : permet de récupérer l'état de l'imprimante à un moment donné (information fournie par l'utilisation de notre référence) ;
- BlackTonerStatus : permet d'obtenir le pourcentage restant pour la cartouche noire de l'imprimante (information fournie par l'utilisation de notre référence) ;
- MagentaTonerStatus : permet d'obtenir le pourcentage restant pour la cartouche magenta de l'imprimante (information fournie par l'utilisation de notre référence) ;
- CyanTonerStatus : permet d'obtenir le pourcentage restant pour la cartouche cyan de l'imprimante (information fournie par l'utilisation de notre référence) ;
- YellowTonerStatus : permet d'obtenir le pourcentage restant pour la cartouche jaune de l'imprimante (information fournie par l'utilisation de notre référence) ;

Cette classe va nous permettre de diffuser les informations sur un élément et en particulier pour notre propertyGrid. Chacune des propriétés que l'on vient de voir alimentera ce composant.

Attardons nous maintenant à créer nos getter, ses propriétés seront en lecture seule pour les besoins de ce tutoriel.

Voyons la déclaration de la classe Printer au moins pour nos trois premières propriétés pour commencer :

```
namespace PrinterType
{
    public class Printer : ICloneable
    {

        //membre de la classe permettant d'identifier l'adresse IP d'une
      imprimante sur le réseau
        private string ipAddress;
        //membre de la classe permettant d'identifiant le nom associé à une
imprimante
        private string nom;
        //membre de la classe permettant de savoir où est située l'imprimante
        private string emplacement;
```

Voyons maintenant nos getters/setters car ces propriétés vont être initialisées à l'aide de notre source de données xml…

```
        [DisplayName("(Nom)")]
        [Category("(Identification)")]
        [Description("Fournit le nom du périphérique.")]
        [ReadOnly(true)]
        [XmlAttribute()]


        public string Nom
        {
            get
            {
                return nom;
            }
            set
            {
                nom = value;
                //onPropertyChanged("Nom");
            }
        }


        [DisplayName("Adresse IP")]
        [Category("(Identification)")]
        [Description("Fournit l'adresse IP de l'imprimante.")]
        [ReadOnly(true)]
        [XmlAttribute()]


        public string IpAddress
        {
            get
            {
                return ipAddress;
            }
```

```
            set
            {
                ipAddress = value;
            }
        }

        [DisplayName("Emplacement")]
        [Category("(Identification)")]
        [Description("Fournit le nom du périphérique.")]
        [ReadOnly(true)]
        [XmlAttribute()]

        public string Emplacement
        {
            get
            {
                return emplacement;
            }
            set
            {
                emplacement = value;
            }
        }
```

Nous avons sous-entendu dans la partie précédente que ces propriétés étaient alimentées par un fichier xml et qu'elles vont être affichées dans un composant PropertyGrid. Pour ce faire nous utilisons les attributs suivants :

- [DIsplayName] qui permet d'afficher le nom de la propriétés dans la PropertyGrid ;
- [Category] qui permet de regrouper nos propriétés dans des catégories, ici Identification et Statut ;
- [Description] qui permet d'afficher une petite légende sur la propriété sélectionnée dans la PropertyGrid ;
- [ReadOnly] permet de déterminer si la propriété est modifiable ou non ;
- [XmlAttribute] qui est un attribut permettant de fixer le type de donnée source de la propriété.

Le reste est classique, et nous nous y attarderons pas…

Nous allons maintenant passer aux cinq autres propriétés qui concernent le statut de l'imprimante et l'état des consommables…

Cette partie est plus complexe que la précédente car elle fait appel aux fonctions de la bibliothèque OLEPRN.dll. Pour les utiliser n'oubliez de référencer celle-ci dans la partie import de votre fichier :

```
using OLEPRNLib;
```

Voici la déclaration des propriétés :

```
   //membre de la classe permettant d'identifier l'état d'une imprimante à un
instant donné

   private string status;

   //membre de la classe permettant d'identifier l'état des consommbles à un
instant donné
   private int blackTonerStatus;
   private int yellowTonerStatus;
   private int cyanTonerStatus;
   private int magentaTonerStatus;
```

Maintenant voyons les getters associées

```
[DisplayName("Statut")]
        [Category("(Statut de l'imprimante)")]
        [Description("Renseigne le statut actuel de l'imprimante.")]
        [ReadOnly(true)]
        public string Status
        {

            get
            {



                return getStatus(ipAddress);
            }


        }



        [DisplayName("Etat du toner noir")]
        [Category("(Statut des consommables)")]
        [Description("Renseigne le statut actuel de l'imprimante.")]
        [ReadOnly(true)]

        public string BlackTonerStatus
        {

            get
            {
                return getTonerStatus(ipAddress, Nom , 1);
            }


        }


        [DisplayName("Etat du toner jaune")]
        [Category("(Statut des consommables)")]
        [Description("Renseigne le statut actuel de l'imprimante.")]
```

```
        [ReadOnly(true)]

        public string YellowTonerStatus
        {

            get
            {
                return getTonerStatus(ipAddress, Nom, 2);
            }


        }


        [DisplayName("Etat du toner cyan")]
        [Category("(Statut des consommables)")]
        [Description("Renseigne le statut actuel de l'imprimante.")]
        [ReadOnly(true)]

        public string CyanTonerStatus
        {

            get
            {
                return getTonerStatus(ipAddress, Nom, 3);
            }


        }


        [DisplayName("Etat du toner magenta")]
        [Category("(Statut des consommables)")]
        [Description("Renseigne le statut actuel de l'imprimante.")]
        [ReadOnly(true)]
        //[XmlAttribute()]
        //[DataGridViewColumn(3, DataGridViewColumnType.TextBox, "Etat de
l'imprimante")]
        public string MagentaTonerStatus
        {

            get
            {
                return getTonerStatus(ipAddress, Nom, 4);
            }


        }
```

Comme vous pouvez le constater les valeurs des propriétés sont obtenu par deux fonctions getStatus (pour obtenir l'état de l'imprimante et getTonerStatus (pour obtenir le pourcentage d'encre restant…

Passons sans plus tarder à l'explication de la méthode de classe getStatus, dont voici le code :

```
public string getStatus(string ipAdress)
        {
```

```
            SNMP snmp;
            int DeviceId = 1;
            int retries = 1;
            int TimeoutInMS = 2000;
            string Result1Str;
            string status;
            try
            {


                string[] ErrorMessageText = new string[8];

                ErrorMessageText[0] = "service recquis";
                ErrorMessageText[1] = "Eteinte";
                ErrorMessageText[2] = "Bourrage papier";
                ErrorMessageText[3] = "porte ouverte";
                ErrorMessageText[4] = "pas de toner";
                ErrorMessageText[5] = "niveau toner bas";
                ErrorMessageText[6] = "plus de papier";
                ErrorMessageText[7] = "niveau de papier bas";

                snmp = new SNMP();

                snmp.Open(ipAdress, CommunityString, retries, TimeoutInMS);
                uint WarningErrorBits =
snmp.GetAsByte(String.Format("25.3.5.1.2.{0}", DeviceId));
                uint statusResult =
snmp.GetAsByte(String.Format("25.3.2.1.5.{0}", DeviceId));

                switch (statusResult)
                {
                    case 2: Result1Str = "OK";
                        break;
                    case 3: Result1Str = "Avertissement: ";
                        break;
                    case 4: Result1Str = "Test: ";
                        break;
                    case 5: Result1Str = "Hors de fonctionnement: ";
                        break;
                    default: Result1Str = "Code Inconnu: " + statusResult;
                        break;


                }
                string Str = "";
                if ((statusResult == 3 || statusResult == 5))
                {
                    int Mask = 1;
                    int NumMsg = 0;
                    for (int i = 0; i < 8; i++)
                    {
                        if ((WarningErrorBits & Mask) == Mask)
                        {
                            if (Str.Length > 0)
                                Str += ", ";
                            Str += ErrorMessageText[i];
                            NumMsg = NumMsg + 1;
```

```
                    }
                    Mask = Mask * 2;
                }
            }
            status = Result1Str + Str;
            snmp.Close();
        }
        catch (Exception)
        {
            status = "Informations non disponibles...";

        }

        return status;
    }
```

Tout d'abord définissons les ressources dont nous avons besoin dans la MIB pour récupérer l'état complet de notre imprimante :

- L'état de l'imprimante ;
- Et en fonction de l'état de l'imprimante récupérer l'erreur ou l'avertissement.

L'état de l'imprimante est récupérable via les ressources respectives suivantes :

- hrDeviceStatus dont l'OID est 1.3.6.1.2.1.25.3.2.1.5 ;
- hrPrinterDetectedErrorState dont l'OID est 1.3.6.1.2.1.25.3.5.1.2 .

Ces ressources sont censées être des standard donc c'est pour cette raison que ces ressources ont des noms bien définis.

La première ressource nous renvoie un nombre entre 1 et 5 dont voici les significations :

- 1 : erreur inconnu ;
- 2 : OK en état de fonctionnement ;
- 3 : avertissement : l'imprimante peut fonctionner mais un seuil d'alerte a été transmis ;
- 4 : est en test ;
- 5 : Hors de fonctionnnement l'état le plus critique.

Pour que la ressource hrPrinterdetectedErrorState nous renvoie une valeur il faut donc que que hrDeviceStatus nous renvoie une valeur égale à 3 ou 5. Cette valeur est codée sur un octet. La valeur dépend donc du bit dont la valeur est à 1, les autres étant positionnés à la valeur 0 :

- Si bit 0 (valeur renvoyée 1) : service recquis ;

- Si bit 1 (valeur renvoyée 2) : Eteiente ;
- Si bit 2 (valeur renvoyée 4) : Bourrage Papier ;
- Si bit 3 (valeur renvoyée 8) : porte ouverte ;
- Si bit 4 (valeur renvoyée 16) : plus de toner ;
- Si bit 5 (valeur renvoyée 32) : niveau toner bas ;
- Si bit 6 (valeur renvoyée 64) : plus de papier ;
- Si bit 7 (valeur renvoyée 128) : niveau de papier bas.

C'est pour cette raison que nous avons déclaré le tableau de chaîne de caractères ***ErrorMessageText.***

Donc maintenant que nous savons ce que nous devons récupérer il faut mettre en place un accès SNMP. Rien de plus simple il suffit de créé une instance de l'objet SNMP et d'ouvrir la connexion grâce à la fonction Open qui prend en argument l'adresse IP du périphérique, la CommunityString qui permet d'assurer un niveau de sécurité étendu, puis ensuite nous avons le nombre d'essais de connexion voulu dans notre cas, on essayera de se connecter une seule fois, et le nombre de milisecondes demandées pour la tentative de connexion. Dans notre cas, et c'est le cas par défaut notre CommunityString est fixée à « public ». En effet dans ce petit projet nous nous connectons aue pour obtenir des informations. Si vous voulez aller plus loin il est nécessaire de mettre en place un niveau de sécurité avancé en définissant un accès SNMP v2 ou v3…

Une fois que la communication avec l'imprimante est établie (si elle l'est pas on renvoie la chaîne de caractère « informations non disponible » que l'on gère par l'intermédiaire d'un bloc try…catch… pour éviter tout plantage) nous devons donc récupérer les valeurs définies précédemment.

Pour cela nous avons une fonction bien utile qui est GetAsByte prenant en paramètre l'OID que nous avons défini précédemment…

Donc nous récupérons les valeurs voulus respectivement dans les variables WarningErrorsBits et statusResult…

En fonction de la valeur stockée dans statusResult nous commençons à assigner à notre variable de type chaîne de caractère (Result1Str). L'étape suivante consiste à tester la valeur de statusResult pour savoir si elle est égale à 3 ou 5 auquel cas nous devons extraire le type d'erreur ou le type d'avertissement…

L'étape suivante consiste à créé un masque nous permettant de récupérer le bit qui est à 1 dans l'octet reccueilli. Il nous permet donc de compléter notre chaîne de retour avec le tableau ErrorMessageText en lui attribuant comme index le numéro de bit trouvé par le masque…

Il suffit ensuite de fermer la connexion avec la fonction Close().

Voyons maintenant comment récupérer l'état des consommables (toners).

Nous avons besoin de récupérer les valeurs fournies par les OIDs suivantes :

- pour le toner noir .1.3.6.1.2.1.43.11.1.1.9.1.1 pour récupérer le niveau actuel et .1.3.6.1.2.1.43.11.1.1.8.1.1 pour récupérer le niveau maximal ;
- pour le toner jaune .1.3.6.1.2.1.43.11.1.1.9.1.2 pour récupérer le niveau actuel et .1.3.6.1.2.1.43.11.1.1.8.1.2 pour récupérer le niveau maximal;
- pour le toner cyan .1.3.6.1.2.1.43.11.1.1.9.1.3 pour récupérer le niveau actuel et .1.3.6.1.2.1.43.11.1.1.8.1.3 pour récupérer le niveau maximal ;
- pour le toner magenta .1.3.6.1.2.1.43.11.1.1.9.1.4 pour récupérer le niveau actuel et .1.3.6.1.2.1.43.11.1.1.9.1.4 pour récupérer le niveau maximal.

Attention ces valeurs devraient être standard mais ne le sont pas en pratique, c'est pour cette raison que j'ai effectué une conversion pour les imprimante Dell de la série 3010cn, cela devient donc :

- pour le toner cyan .1.3.6.1.2.1.43.11.1.1.9.1.1 donc même traitement pour récupérer le niveau maximal;
- pour le toner magenta.1.3.6.1.2.1.43.11.1.1.9.1.2 donc même traitement pour récupérer le niveau maximal;
- pour le toner jaune.1.3.6.1.2.1.43.11.1.1.9.1.3 donc même traitement pour récupérer le niveau maximal;
- pour le toner noir .1.3.6.1.2.1.43.11.1.1.9.1.4 donc même traitement pour récupérer le niveau maximal.

Donc reportez-vous à la notice du fabricant pour récupérer les bons OIDs

Voyons maintenant comment ceci est mis en œuvre…

```
public string getTonerStatus(string ipAdress, string printerName, int
tonerNumber)
        {
            int DeviceId = 1;
            int retries = 1;
            int TimeoutInMS = 2000;
            int tonerNumberDell = 0;
            string status;
            try
            {
                SNMP snmp = new SNMP();
                snmp.Open(ipAdress, CommunityString, retries, TimeoutInMS);
                switch (tonerNumber)
                {
                    case 1:
                        tonerNumberDell = 4;
                        break;
```

```
                    case 2:
                        tonerNumberDell = 3;
                        break;
                    case 3:
                        tonerNumberDell = 1;
                        break;
                    case 4:
                        tonerNumberDell = 2;
                        break;
                }

                switch (printerName)
                {
                    case "Dell 3010 cn":
                        currentlevel =
Convert.ToUInt32(snmp.Get(".1.3.6.1.2.1.43.11.1.1.9.1." +
tonerNumberDell.ToString()));
                        maxlevel =
Convert.ToUInt32(snmp.Get(".1.3.6.1.2.1.43.11.1.1.8.1."+
tonerNumberDell.ToString()));
                        break;
                    default:
                        currentlevel =
Convert.ToUInt32(snmp.Get(".1.3.6.1.2.1.43.11.1.1.9.1." +
tonerNumber.ToString()));
                        maxlevel =
Convert.ToUInt32(snmp.Get(".1.3.6.1.2.1.43.11.1.1.8.1." +
tonerNumber.ToString()));
                        break;
                }
                uint remaininglevel = (currentlevel * 100 / maxlevel);
                status = remaininglevel.ToString();
                snmp.Close();
            }
            catch (Exception)
            {

                status = "Informations non disponibles...";
            }

            return status;
        }
```

Comme vous pouvez le constater nous avons besoin que d'une seule méthode générique ***getTonerStatus*** car les OIDs se ressemblent…

Il suffit d'indiquer en paramètres de la fonction le numéro (1, 2, 3 ou 4) pour récupérer les valeurs des OIDs correspondants…

La connexion SNMP se fait de la même façon que précédemment. Pour récupérer les valeurs des OIDs voulus nous allons utiliser une autre fonction qui est Get qui prend en argument la chaîne de caractère correspondant à l'OID demandé. Elle renvoie une valeur de type Object qu'il est nécessaire de convertir en type Uint32.

```
currentlevel = Convert.ToUInt32(snmp.Get(".1.3.6.1.2.1.43.11.1.1.9.1." +
tonerNumber.ToString()));
```

Lorsqu'on a récupéré les deux valeurs il suffit de réaliser l'opération qui va nous donner le pourcentage d'encre restant :

```
uint remaininglevel = (currentlevel * 100 / maxlevel);
```

Nous en avons terminé avec la classe Printer. Rien ne vous empêche de récupérer d'autres valeurs…

4. Codage de la classe Printers

Pour commencer ajoutez une autre classe à votre bibliothèque de classe et nommez la Printers

La classe Printers va hériter de la Printer, car elle va nous permettre de récupérer une liste d'imprimantes.

C'est dans cette classe que nous allons récupérer toutes les données fournies par notre source de données xml…

Dans notre exemple nous récupérons des données mais nous n'allons pas en sauvegarder…

Voici le code de la classe :

```
public class Printers :List<Printer>, ICloneable
    {
        public string Xml
        {

            set
            {
                try
                {

                    using (XmlReader xmlReader = XmlReader.Create(new
StringReader(value)))
                    {
                        this.Clear();
                        XmlSerializer xmlSerializer = new
XmlSerializer(this.GetType());
                        Printers printers =
xmlSerializer.Deserialize(xmlReader) as Printers;
                        this.AddRange(printers);

                    }
                }
                catch
                {
                    //Ne rien faire
                }

            }
        }
```

```
    }
}
```

On commence par initialiser un Lecteur sur le fichier xml… Le « désérialiseur » nous permet d'extraire les données contenues (attributs) dans le fichier xml pour chaque nœud. Ces données sont celles que nous avons vu dans la classe Printer (nom, adresse IP, emplacement)…

Toutes ces données sont ajoutées à notre liste d'imprimantes…

5. Codage de la fiche principale

Maintenant nous allons nous pencher sur le comportement de l'application. Comme nous avons vu le cœur de notre application est notre combobox qui récupère la liste des noms de chaque imprimante au démarrage de l'application :

```
private void Form1_Load(object sender, EventArgs e)
        {

            printerBindingNavigator.Visible = false;
            nomFichier = "C:\\printers.prn";
            printerBindingSource.DataSource = OuvrirFichier(nomFichier);
            printerBindingNavigator.BindingSource = printerBindingSource;

            comboBox1.DataSource = printerBindingNavigator.BindingSource;
            comboBox1.DisplayMember = "Nom";
        }
```

Nous avons besoin d'ajouter un composant de type BindingSource et un composant de type BindingNavigator à notre projet. Nous n'avons pas besoin de rendre visible notre navigateur de données donc on met sa valeur Visible à false…

Notre BindingSource va récupérer les données issues de notre fichier xml… Ceci est effectué par l'intermédiaire de notre fonction OuvrirFichier qui prend en paramètre le nom de fichier :

```
public Printers OuvrirFichier(string NomFichier)
        {
            Printers result = null;
            using (System.IO.StreamReader sr = new
System.IO.StreamReader(NomFichier))
            {
                result = new Printers();
                result.Xml = sr.ReadToEnd();

            }
            return result;
        }
```

Elle renvoie notre liste d'imprimante avec ses attributs comme nous l'avons vu précédemment et l'assigne donc à notre source de données.

Cette source de données va être ensuite assignée à notre navigateur. Il est ensuite possible de récupérer le nom de chaque imprimante grâce à la méthode ***DisplayMember***…

Lorsque l'utilisateur sélectionne une nouvelle imprimante dans la combobox il faut que la propertyGrid modifie son contenu en fonction de la source de données… Ceci est fait grâce à la méthode CurrentChanged de la liaison de données.

```
private void printerBindingSource_CurrentChanged(object sender, EventArgs e)
        {
            this.prnPropertyGrid.SelectedObject = (sender as
BindingSource).Current;
        }
```

IV. CONCLUSION

Nous avons vu comment réaliser une petite application simple permettant de scruter l'état de nos imprimantes installées en réseau en utilisant la bibliothèque de fonctions oleprn.dll. Vous avez vu de même comment utiliser un fichier xml pour alimenter un composant de type propertyGrid. Nous avons vu qu'il est simple de récupérer des données sur des imprimantes grâce au protocole SNMP. Mais attention certains constructeurs ne respectent pas les standards, il faut donc réaliser un petit travail de recherche.