C

Langage C# et Plate-forme .NET

Objectifs "Passerelle"

- Acquérir les compétences pour :
  - créer une application Windows
  - créer un service Windows
  - contrôler un service
  - gérer des flux XML
  - se connecter à une base
  - utiliser des composants COM (serveurs)
  - être notifié de l'arrivée d'un fichier
  - écrire dans le journal des évènements

Plan

- I. Présentation de l'architecture .NET
- II. Éléments syntaxiques
  - types de données, tableaux et instructions
- III. Applications Windows
  - notions de base
- IV. Manipulation des objets avec C#
- V. Exemple 1 : passerelle Windows
  - connexion ADO, génération XML
- VI. Exemple 2 : passerelle en service NT
  - service NT, déploiement, DCOM
- VII. Compléments

Architecture .NET

Principes généraux

L’architecture .NET

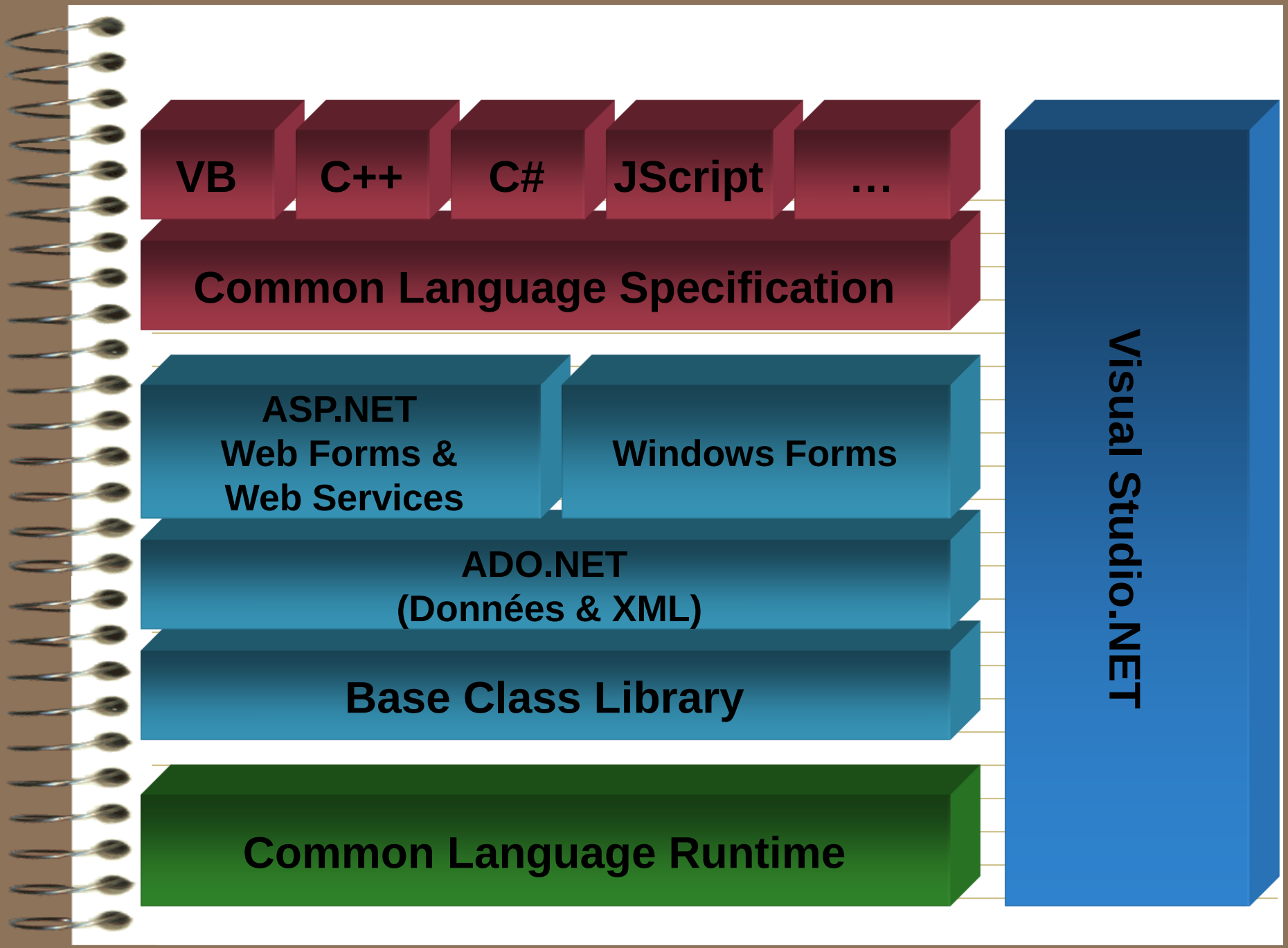
- .NET est une couche Windows (collection de DLL) librement distribuable.
- Cette couche contient des classes et un environnement d’exécution (*run time*).
- Pour un programme qui s’exécute sous contrôle de cet environnement, on parle de mode géré ou managé (*managed code*).

La CLR (1)

- Les services fournis par le *run time* font partie de la CLR (*Common Language Run time*) et assurent :
  - chargement et exécution contrôlé des programme
  - isolation des programmes entre eux
  - vérification de type lors des appels de fonctions
  - conversion de code intermédiaire en code natif à l'exécution (JIT : *Just In Time Compiler*)

La CLR (2)

- accès aux infos sur le code (méta données)
- gestion de la mémoire
- sécurité
- compatibilité avec les DLL et modules COM


VB
C++
C#
JScript
…
Common Language Specification
ASP.NET
Web Forms &
Web Services
Windows Forms
ADO.NET
(Données & XML)
Base Class Library
Common Language Runtime
Visual Studio.NET

Espaces de noms *namesapaces*(1)

- Les classes utilisées par tous les langages .NET sont regroupées en espaces de nom.

System	Accès aux types de base Accès à la console	Int32, Int64, Int16, Byte, Char, String, Float, Double, Console, Type …
System.Collections	Collection d'objets	ArrayList, Hashtable,Queue, Stack, Sortedlist…
System.IO	Accès aux fichiers	File, Directory, Stream, FileStream, BinaryReader, BinaryWriter, TextReader, TextWriter
System.Data.OleDb	Accès ADO aux bases de données	OleDbConnection, OleDbCommand, DataSet
System.Data.SQL	Accès optimisé à SQL Server	SQLConnection, SQLDataSetCommand

Espaces de noms *namesapaces*(2)

System.Net	Accès au réseau	Sockets, TcpClient, TcpListener, UdpClient…
System.Reflection	Accès aux méta données	FileIdInfo, MemberInfo, ParameterInfo…
System.Security	Contrôle de la sécurité	Permissions, Policy, Cryptography ..
System.WinForms	Composant orientés Windows	Button, ListBox, DataGrid, Form, MainMenu, StatusBar…
System.Web.UI.WebControls	Composants orientés Web	Button, ListBox, DataGrid, HyperLink…

Compatibilité des langages .NET

- Une classe .NET peut être utilisé de la même manière dans tous langages .NET
- Une classe peut être écrite dans un langage X
- Une classe peut être dérivé de celle-ci dans un langage Y
- Cette classe dérivée peut être utilisé sans un langage Z
- la manière de créer et utiliser les objets est identique

C# par rapport à C++

- orientation objet plus importante
- code Unicode
- libération automatique d'objet (*garbage collection*)
- sur les tableaux ou les objets les pointeurs sont remplacés par des références
- passage par référence (et non adresse) des arguments
- ...

Éléments syntaxiques

types de données

tableaux

opérateurs

instructions

Premiers pas

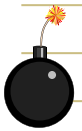
```
Class Prog
{
static void Main()
{
/* code du programme principal */
}
}
```

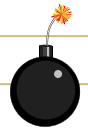
Éléments syntaxiques

- Majuscules <> minuscules (case sensitif)
- Les instructions se terminent par ;
- les espaces de noms dont déclarés par *using*
- // ou /// indique que le reste de la ligne est un commentaire
- /* indique un début de commentaire
- */ indique une fin de commentaire
- un identificateur doit commencer par une lettre ou _

Types entiers (1)

- *byte* : entier non signé codé sur 8 bits, 0 à 255. Classe *System.Byte*
- *sbyte* : entier signé codé sur 8 bits, -128 à 127. Classe *System.SByte*
- *short* : entier signé codé sur 16 bits, ≈ -32 000 à 32 000. Classe *System.Int16*
- *ushort* : entier non signé codé sur 16 bits, 0 à ≈ 65 000. Classe *System.UInt16*

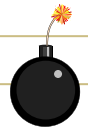




Non conforme à la CLR = peut ne pas être compatible en cas d ’appel d ’un autre langage .NET

Types entiers (2)

- *int* : entier signé codé sur 32 bits, ≈ -2 milliards à 2 milliards. Classe *System.Int32*
- *uint* : entier non signé codé sur 32 bits, de 0 à ≈ 4 milliards. Classe *System.UInt32*
- *long* : entier signé codé sur 64 bits, ≈ -9.2 x 10^{18} à 9.2 x 10^{18}. Classe *System.Int64*
- *ulong* : entier non signé codé sur 64 bits, de 0 à ≈ 18 x 10^{18}. Classe *System.UInt64*



Non conforme à la CLR = peut ne pas être compatible en cas d ’appel d ’un autre langage .NET

Types réels

- *float* : réel codé sur 32 bits, si > 0 de ≈1.4 x 10^{-45} à 3.4 x 10^{38}. Classe *System.Single*
- *double* : réel codé sur 64 bits, si > 0 de ≈4.9 x 10^{-324} à 1.8 x 10^{308}. Classe *System.Double*
- *decimal* : réel codé sur 128 bits, exactitude si < 8 x 10^{28}. Classe *System.Decimal*
- Attention à la précision : si dans une boucle on additionne dix mille fois *0.001* codé en *float*, on obtient *10 000.0004*

Autres types

- *bool* : booléen, *True* ou *False*. Classe *System.Boolean.*
- *char* : caractère codé sur 16 bits Unicode (=ASCI + caractères accentués + autre langues). Classe *System.Char*.
- *string* : chaîne de caractères codé sur 16 bits et d ’agrandit automatiquement. Classe *System.String*

Déclarations et affectations

```
int i;
i = 5;
string s = "Bonjour";
double d = 1.23;
float f = 1.2f;
char c1 = 'A';
char c2 = '\n';          // Nouvelle ligne
char c3 = '\0' ;         // caractère NULL
```

Type enum

```
enum Sexe {masculin, féminin}
....
Sexe s1  ;// s1 est une variable de type Sexe
s1 = Sexe.masculin;
if (s1 == Sexe.masculin) Console.Write("M");
// affiche M
s1++;
if ((int)s1 == 1) Console.Write("F"); // affiche F
```

- La déclaration *enum* doit être hors d'une fonction dans la classe ou hors de la classe

Tableau à une dimension

```
int[] t;          // référencé pas encore alloué
int[] t1, t2;     // référencés pas encore alloués
t1 = new int [3]; // allocation
int[] t3 =new int[3]; //référencé et alloué
t2 = new int [] {5,3,1}; //allocation et initialisation
Console.Write(t2[0]); // affiche 5
Console.Write(t2.Length); // affiche 3
Array.Sort(t2); // tri le tableau
t2.CopyTo(t1,0); //copie t2 dans t1 depuis t2[0]
t3 = t2; // t3 et t2 référence le même tableau
t2[2] = 9; Console.Write(t3[2]); // affiche 9
```

Tableau à n dimensions

```
int[,] tn, tn1;          // référencés pas encore alloués
tn1= new int [2,3]; // allocation
int [,] tn2 =new int[2,3]; //référencé et alloué
tn = new int [,] {{5,3,1}, {2,4,6}}; // initialisation
Console.Write(tn[1,0]); // affiche 2: col n°0, ligne n°1
Console.Write(tn.GetLength(0)); // affiche 2
tn1 = (int[,])tn.Clone(); //copie tn dans tn1
```

Priorité des opérateurs

1		x++ x-- new …
2	un seul opérande	+ - ! ++x --x
3	multiplication – division	* / %
4	addition – soustraction	+ -
5	décalage de bits	<< >>
6	relations	< > >= <= is
7	égalité – inégalité	= = ! =
8	ET binaire	&
9	XOR binaire	^
10	OU binaire	|
11	condition ET	&&
12	condition OU	^^
13	condition	? :
14	assignations	= *= /= += -= …..

Opération entrée-sortie console

```
int i, j;
Console. WriteLine("Somme"); // affiche Somme
Console. Write ("Som"); // affiche Som
Console. WriteLine("me"); // complète Somme
i=1; j=2; Console.WriteLine("Somme = " + (i+j));
// affiche Somme = 3
i=1; j=2; Console.WriteLine("Somme = " + i+j);
// affiche Somme = 12
string s = Console.ReadLine();
i = Int32.Parse(s); Console.WriteLine(i+j);
```

Opérateurs (1)

- Opérateurs arithmétiques : +, -, *, /
- si une des opérandes est réelle, / est une division réelle, sinon c'est une division entière (quotient arrondi inf, % donne le reste. Exemple : *9/-4* donne *-2* et *9%-4* donne *1*
- Post-incrémentation

  (i=j++) équivaut à i=j; j=i+1;
- Pré-incrémentation

  (i=++j) équivaut à j=j+1; i=j;

Opérateurs (2)

- Not booléen: *c=true; b = !c;*
- Opérations binaires (bit à bit) sur entiers non signés : & (and), | (or) , ^ (xor) et ~ (inversion de tous les bits)
- Décalage d'un bit à gauche << (i.e *2)
- Décalage d'un bit à droite >> (i.e /2)

Instruction if

- *if (i==0) une instruction;*
- *if (i==0) une instruction else une instruction;*
- *if (i==0)*

{

une instruction ou plusieurs instructions;

}

else

{

une instruction ou plusieurs instructions;

}

Éléments de conditions

- Conditions avec un booléen :

  if (b) équivaut à *if (b == true)*

  if (!b) équivaut à *if (b == false)*

- Instructions ?

  a = b!=0 ? 1 : 2;

  équivaut à

  if (b!=0) a=1 else a =2;

- Opérateurs logiques

  if (i==1 && j!=2 || k ==0)

  signifie "si i vaut 1 et j différent de 2 ou k vaut 0"

Boucles

- Tant que : *while (i<5)*

  {instructions;}

- Jusqu'à *do*

  {instructions;}

  while (i<5);

- Pour

for (int i=1;i<5;i++)

{instructions;}

for (int i=1, j = 10 ;i<5||j>10;i++,j--)

{instructions;}

Autres instructions

- Passage à l'itération suivante : *continue;*
- Sortie de boucle : *break;*
- Énumération

```
switch(variable)
{case valeur1: instructions; break;
case valeur2 : instructions; break;
default : instructions; break;}
```

Gestion d'erreur

```
int i , j =0 ;
try
   { i = 5/j;}
catch (Exception Err)
   {MessageBox.Show(Err.Message);}
finally
   {MessageBox.Show("Dans tous les cas");}
```

- **Remarque** : l'instruction *finally* est facultative.

Les fonctions

- Pas de passage d'arguments par adresse
- Passage d'arguments par référence seulement pour les tableaux et objets
- Passage d'arguments par valeur ou référence pour les autres types
- *void* signale une fonction qui ne retourne pas de valeurs

Passage d'arguments

- Par valeurs
  - Appel : *int a = 3; f(a);*
  - Description : *static void f(int i) {instructions;}*
- Par référence
  - Appel : *int a = 3; f(ref a);*
  - Description : *static void f(ref int i) {instructions;}*
- De tableau (toujours par réf, *ref* implicite)
  - Appel : *int[] ta = {1,2,3}; f(ta);*
  - Description : *static void f(int [] ti) {instructions;}*

Passage d'arguments variables

- En nombre *f(1,2); f(1,5,1,6); f();*

```
static void f(params int[] p)
{foreach (int a in p) Console.Write (a + " ");}
```

- En nombre et en type: *f(1,2); f(1,5,"z"); f(3.45);*

```
static void f(params object[] p)
{foreach (object a in p) {
if (a is int) Console.WriteLine ( "Entier : "+a);
if (a is double) Console.WriteLine ( "Réel : "+a);
if (a is string) Console.WriteLine ( "Chaine : "+a);}}
```

Applications Windows

Notions de base

Code généré par C#

- Références aux namespaces utilisés pour une application Windows
- Un namespace portant le nom du projet
- A l'intérieur de celui-ci, une classe portant le nom de la fenêtre principale
- A l'intérieur de celle-ci, un fonction Main qui exécute la méthode Run de l'objet Application.

Régions

- Les parties du code peuvent être délimitées dans des "régions" en les bornant par les délimiteurs *#region* et *#endregion*

```
#region Test
int i  = 1;
#endregion
```

- Un bouton de réduction en forme de "-" apparaît alors pour masquer cette partie du code

Quelques méthodes de base

- Affichage de Form2 (non modale)

Form2 f = new Form2() ; Form2.Show();

- Affichage de Form2 (modale)

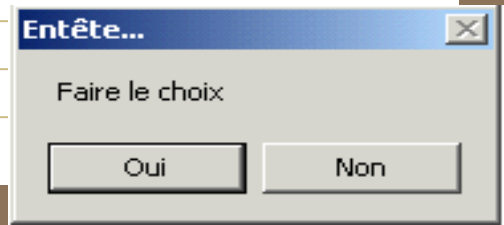
Form2 f = new Form2() ; Form2.ShowDialog();

- Modification propriété d'un contrôle de Form2

Form2 f = new Form2() ; Form2.ShowDialog();

- Affichage d'un boîte de dialogue

MessageBox.Show("Faire le choix", "Entête...",
MessageBoxButtons.YesNo);



Timer

- Il existe deux types de Timer :
  - *Timer Windows* (dans les outils "Windows Form") à utiliser pour les application Windows.
  - *Timer Serveur* (dans les outils "Composants") à utiliser dans les autres cas.
- Initialisation d'un timer

timer1.Interval = 5000;

timer1.Start();

- Programmation d'un timer

private void timer1_Tick(object sender, System.EventArgs e)

{MessageBox.Show("Top" + DateTime.Now);}

Manipulation des objets avec C#

utilisation des classes

Quelques principes

- Il est possible avec C# de mettre en œuvre les principes de l'objet suivant :
  - surcharge de méthodes
  - tableaux d'objets
  - champs et méthodes statiques
  - polymorphisme
  - composition
  - héritage (pas d'héritage multiple)
  - classes abstraites
  - interfaces

Description d'une classe

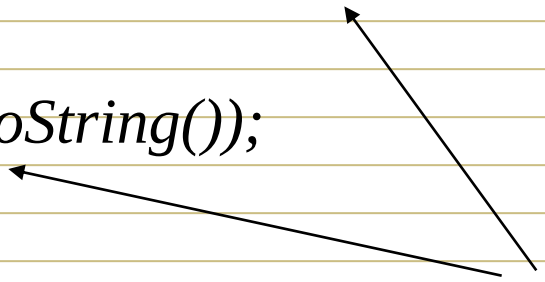
- Classe Personne avec deux attributs (*Nom* et *Prénom*), un constructeur (*Personne()*) et une méthode (*Affiche()*).

```
class Personne
{public string Nom;
public DateTime Naiss;
public Personne(string N, DateTime D) {Nom = N;Naiss=D;}
public void Affiche()
{MessageBox.Show(Nom);}}
```

- Le destructeur est inutile en .NET
- S'il n'est pas décrit, un constructeur est implicite.

Utilisation d'un objet

```
// déclaration
Personne P1;
//instanciation
P1 = new Personne("Dupond",DateTime.Parse("01/01/60"));
P1.Affiche();
MessageBox.Show(P1.Naiss.ToString());
```



Remarque : DateTime est une classe. Des méthodes de conversions en chaîne sont donc nécessaires.

Surcharges de méthodes

- Des méthodes d'une même classe peuvent porter le même nom mais doivent se distinguer par leur type ou nombre d'arguments :

```
class Personne
{...
public void Modifier(String N) {Nom = N;}
public void Modifier(DateTime D)  {Naiss = D;}
public void Modifier(String N, DateTime D)
{Nom = N;Naiss = D;}
}
```

Tableaux d'objets

- Il est nécessaire d'appeler le constructeur pour chaque cellule du tableau :

```
Personne[] TabPers; // déclaration
TabPers = new Personne[10];//allocation
for(int i=0;i<TabPers.Length;i++) // pour chaque élément
TabPers[i]=new Personne("", DateTime.Parse("01/01/1900"));
 // appel du constructeur
```

Attributs et méthodes statiques

- Les attributs ou les méthodes statiques peuvent être appelé indépendamment de toute instance.

```
class Personne
    {public static string Prénom = "A Définir";
    ...
    public static void Test() {MessageBox.Show("Test");}
    ...
    Personne.Test();
    MessageBox.Show(Personne.Prénom);
```

- Remarque : Show de MessageBox et Parse de DateTime sont des méthodes statiques

Polymorphisme

- Des attributs ou des méthodes de classes différentes peuvent porter le même nom. La forme adoptée par l'attribut ou la méthode dépend alors de l'objet sur lequel il ou elle porte.

```
class Acte
    {public string Code;
    public void Affiche() {MessageBox.Show(Code);}}
....
P1 = new Personne("Dupond",DateTime.Parse("01/01/60"));
P1.Affiche();
A1 = new Acte();
A1.Affiche();
```

Composition

- Une classe peut contenir un autre classe

```
class Personne
  {public Personne(string N, DateTime D) {Nom = N;Naiss=D;
  A = new Adresse();}
public class Adresse   { string rue;
                         public string ville ;
                         public string GetRue() {return rue;}}
                         public void Vi(string v){A.ville = v;}
                         public Adresse A;
                         ...}
...
P1.Vi("Paris");
MessageBox.Show (P1.A.ville);
```

Héritage

- A partir d'une classe mère, on peut créer une classe fille enrichie de méthodes et d 'attributs.

```
 class Patient: Personne
{public string Nip;
public Patient (string N, DateTime D, string I) : base(N,D)
    {Nip = I;}
public new void Affiche() {MessageBox.Show(Nip);}}
...
Patient Pat;
Pat = new
    Patient("Dupond",DateTime.Parse("01/01/60"),"0303");
Pat.Affiche();
```

Fonctions virtuelles

- Tout objet de la classe Patient est un objet de la classe Personne. Ainsi, la syntaxe suivante est acceptée :

  Personne Pat1;

  Pat1 = *new Patient("Dupond",DateTime.Parse("01/01/60"),"0303");*

- Pour savoir quelle méthode appelée, C # se base sur la définition stricte de *Pat1*. Ainsi *Pat1* est un objet *Patient*, mais *Pat1.Affiche()* , fait appèle à *Affiche()* de Personne !
- Pour remédier à ce problème, il faut qualifier :
- dans *Personne : public* ***virtual*** *void Affiche();*
- dans *Patient : public* ***override*** *void Affiche();*

Classes abstraites

- Une classe abstraite contient la définition d'une fonction (prototype) sans son implémentation qui sera réalisée dans une classe dérivée. Elle ne peut pas être instanciée.

```
abstract class ProtoSéjour
{string No_Séjour ;
public ProtoSéjour( string Num) {No_Séjour=Num;}
public abstract void Test();}
class Séjour : ProtoSéjour
{public Séjour( string N) : base(N) {}
public override void Test(){MessageBox.Show("OK");}}
```

Interfaces

- On peut définir uniquement les méthodes à implémenter dans la classe

```
interface ObjetMétier
{void Test ();
void Affiche(string S); }
interface Serveur
{void GetVersion();}
class IDT : ObjetMétier, Serveur
{public void Test (){MessageBox.Show("OK");}
public void Affiche(string S) {MessageBox.Show(S);}
public void GetVersion() {MessageBox.Show("Version :");} }
```

Exemple 1 :
Passerelle en mode Windows 32

connexion ADO

timer

génération XML

Présentation Exemple 1

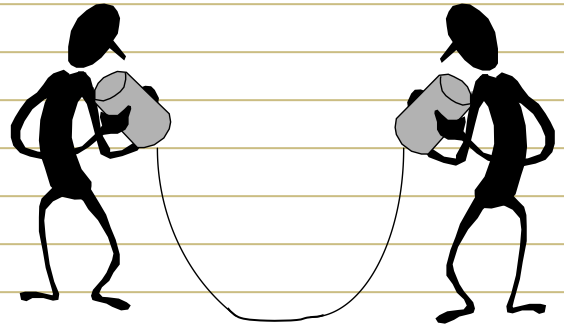
- Toutes les 5 minutes, la passerelle va chercher toutes les informations dans la table idt_patient de PRIMASERVICES. Ces informations sont enregistrées sur le disque dans c:\temp sous un format XML dont le nom est incrémenté à chaque itération.

ADO.NET

Le modèle Objet

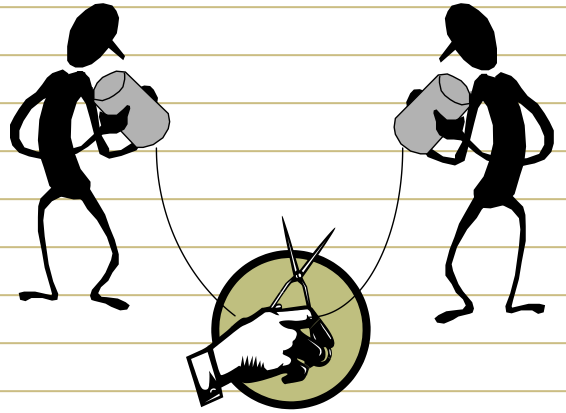
- Les objets « connectés »

  Connection
  Command
  DataReader
  DataAdapter



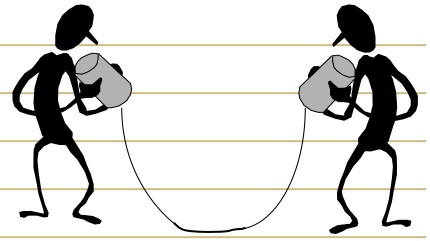
- Les objets non « connectés »

  DataSet
  DataTable
  DataRow
  DataColumn

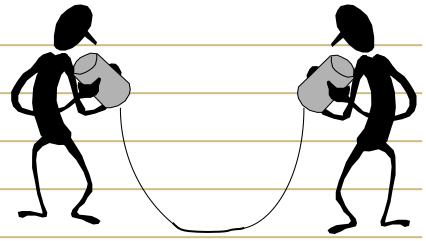


La notion de Data Provider

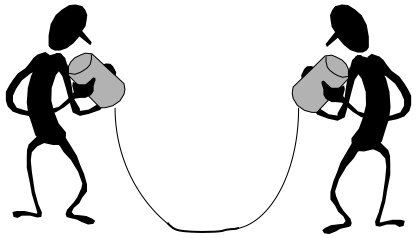
- Implémentation physique du modèle ADO
- Data Provider OLEDB générique
  OleDbConnection
  OleDbCommand
  OleDbDataReader
  OleDbDataAdapter



- Data Provider optimisé pour SQL Server
  SQLConnection
  SQLCommand
  SQLDataReader
  SQLDataAdapter



Objet Connection



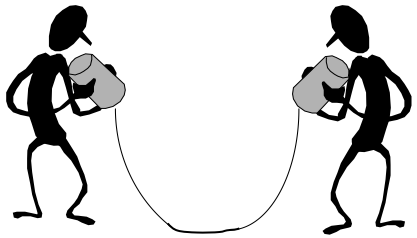
- SQL Server

```
// code de connexion SQL
SqlConnection cn;
string strChaine="Database=;Server = ;uid=;pwd=;"
cn =  new SqlConnection(strChaine);
cn.Open();
```

- Autre SGBD

```
OleDbConnection cn;
string strChaine = "Provider= Microsoft.Jet.OLEDB.4.0; Data
    Source = c:\test.mdb";
cn =  new OleDbConnection (strChaine);
cn.Open()
```

Objet Command



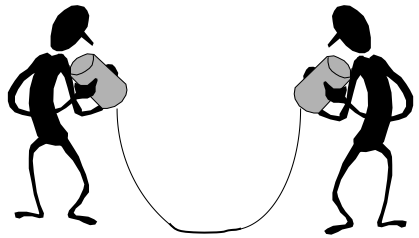
- Permet d'exécuter des requêtes de lecture ou de mise à jour

```
// code de connexion SQL à placer ici
SqlCommand cd;
string strSQL =  "delete authors";
cd = new SqlCommand(strSQL,cn);
int n = cd.ExecuteNonQuery();
```

ADO.NET : Objets de commandes

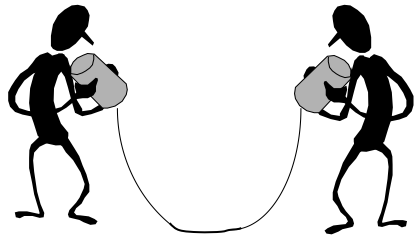
- Objet *SqlDbCommand* pour SQL Serveur *OleDbCommand* pour les autres SGBD.
- Le constructeur *SqlCommand(string Requête SQL)* définit la requête.
- Pour une PS, la propriété *CommandType* doit être à *"StoredProcedure"* et *CommandText* contient le nom de la PS.
- Méthode *ExecuteNonQuery* si pas de retour
- Méthode *ExecuteScalar* si retour 1 valeur
- Méthode *ExecuteReader* si retour n lignes

Exemple



```
using System.Data;
 using System.Data.SqlClient;
...
SqlConnection objConnexion;
SqlCommand objCommande;
string strChaine ="Database=B;Server=S;uid=sa;pwd=";
string strSQL = "SELECT COUNT(*) FROM idt_patient";
int n;
objConnexion = new SqlConnection(strChaine);
objConnexion.Open();
objCommande = new SqlCommand(strSQL,objConnexion);
n = (int)objCommande.ExecuteScalar();
MessageBox.Show("Résultat : " + n);
```

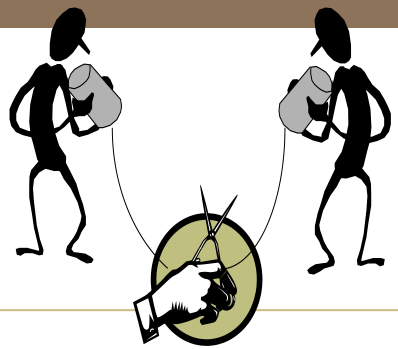
Objet DataReader

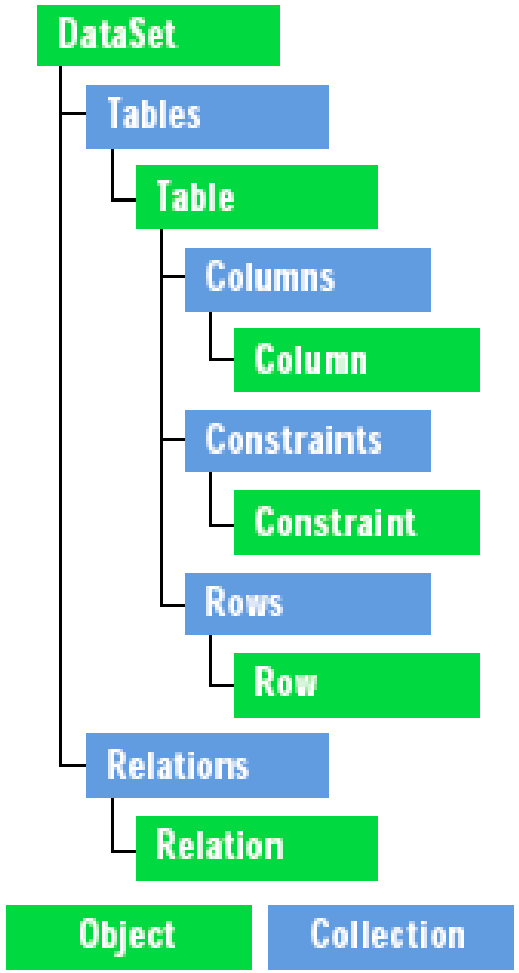


- Comparable à un recordset en mode ReadOnly et ForwardOnly (Curseurs ReadOnly en ADO)

```
// code de connexion SQL à placer ici
SqlCommand cd;
string strSQL =  "select * from t_medecin_t";
cd = new SqlCommand(strSQL,cn);
SqlDataReader dr;
dr = cd.ExecuteReader();
    while (dr.Read())
        {listBox1.Items.Add(dr["nom_medecin"]);}
```

Objet Dataset





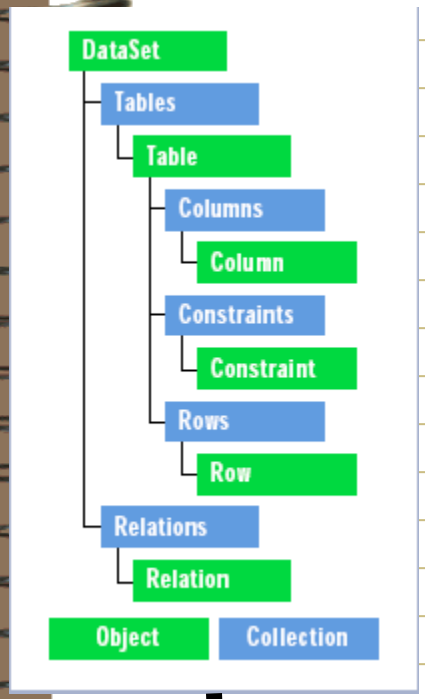
- **Le Dataset est stocké en mémoire côté client (curseur static en ADO)**

Il peut contenir plusieurs tables d'une même source et des données de sources différentes (sorte de "BDD en mémoire")

Il est possible de créer par code des objets (Table, Contrainte, Colonne) dans le Dataset

- **Permet la mise à jour des données**

Objet DataAdapter



DataAdapter

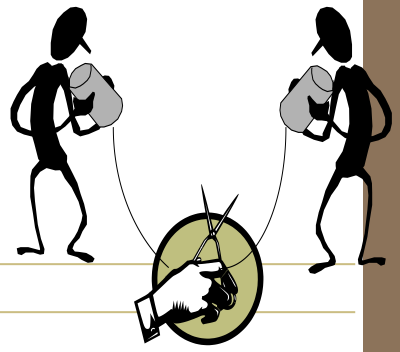
DataConnection

Source
de Données

Fill

Update

DataAdapter - DataSet

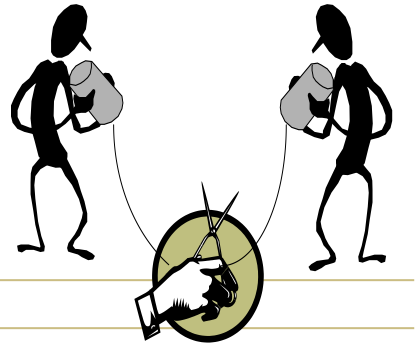


```
// code de connexion SQL à placer ici
 // creation DataAdaptater
SqlDataAdapter da;
da = new SqlDataAdapter("select * from t_medecin_t", cn);
//chargement Dataset
DataSet ds = new DataSet();
da.Fill(ds, "t");
cn.Close();
// affiche le nom du premier médecin :
MessageBox.Show((string)ds.Tables["t"].Rows[0]["nom_mede
    cin"]);
```

objet DataTable

objet DataRow

Objet DataView



- Permet de filtrer, trier une DataTable appartenant au DataSet

```
// code de connexion SQL à placer ici
 // code de creation DataAdaptater  à placer ici
// code de chargement Dataset  à placer ici

DataView dv = new DataView();
dv.Table = ds.Tables["t"];
dv.RowFilter = "code_medecin = 'A'";
dv.Sort = "nom_medecin ASC";
// affiche le nom du premier médecin dans l'ordre alphabétique:
MessageBox.Show(dv[0]["nom_medecin"].ToString());
```

XML

Lecture : Classe XmlTextReader

- Constructeurs : *XmlTextReader ();* ou *XmlTextReader (fichier);* ou *XmlTextReader (stream);*
- Quelques attributs
  - Profondeur du nœud : *int Depth*
  - Nombre d'attributs du nœud : *int AttributCount*
  - Valeur du nœud *: string Value*
  - Type du nœud *: NodeType*
  - Fin de fichier atteinte *: bool EOF*
- Quelques méthodes
  - Contenu de l'attribut : *string GetAttribute ("Nom_attribut");*
  - Lecture et positionnement sur l'elt suivant : *Read();*

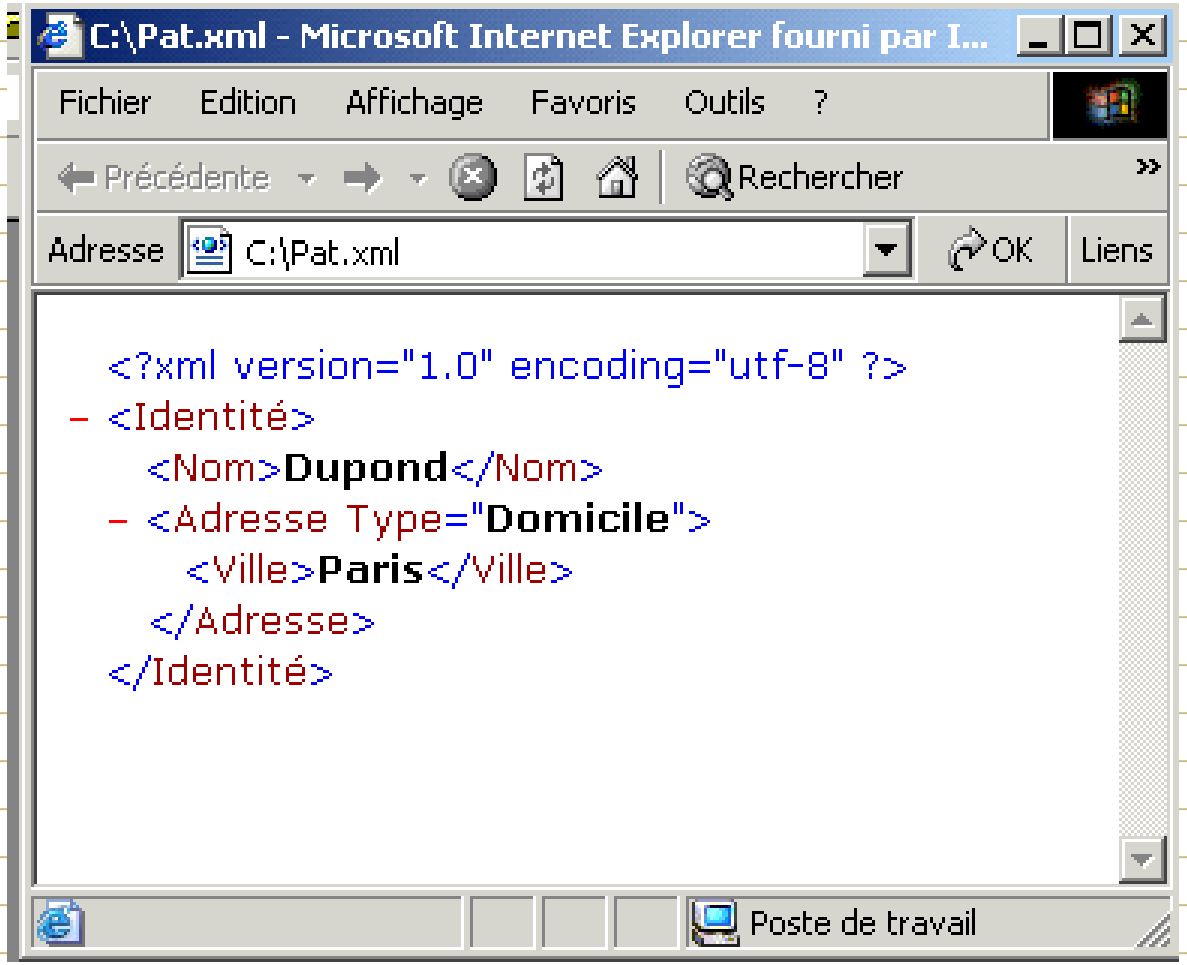
Écriture : Classe XmlTextWriter

- Constructeurs : *XmlTextWriter (fichier, encodage);*
- Quelques attributs
  - Indentation : *enum Formatting*
- Quelques méthodes
  - Commence l'écriture du document: *void WriteStartDocument();*
  - Commence l'écriture de l'élément: *void WriteStartElement(nom);*
  - Écriture d'un élément : *void WriteElementString(nom,valeur);*
  - Écriture d'un attribut : *void WriteAttributeString(nom,valeur);*
  - Termine l'écriture de l'élément : *void WriteEndElement();*
  - Termine l'écriture du document : *void WriteEndDocument();*
  - Force l'écriture disque du document : *void Flush();*
  - Ferme le fichier XML : *void Close();*

Exemple d'écriture

```
using System.Xml;
using System.Text;
...
XmlTextWriter wr = new XmlTextWriter(@"c:\Pat.xml", Encoding.UTF8);
wr.Formatting = Formatting.Indented;
wr.WriteStartDocument();
    wr.WriteStartElement("Identité");
            wr.WriteElementString("Nom","Dupond");
            wr.WriteStartElement("Adresse");
                    wr.WriteAttributeString ("Type","Domicile");
                    wr.WriteElementString("Ville","Paris");
             wr.WriteEndElement();
     wr.WriteEndElement();
 wr.WriteEndDocument();
wr.Flush();
wr.Close();
```

Exemple généré



Exemple de lecture

```
using System.Xml;
using System.Text;
...
XmlTextReader rdr = new XmlTextReader(@"c:\Pat.xml");
    while (!rdr.EOF)
    {
            rdr.Read();
            if (rdr.NodeType.ToString()== "Text")
            // Affiche "Dupond", et "Paris"
                    MessageBox.Show (rdr.Value);
    }
```

Classe XmlDocument

- Représentation mémoire sous forme d'arbre
- On charge un document avec la méthode *Load(nom_fichier)* ou *LoadXml(chaîne_contenant_flux)*
- La propriété *ChildNodes* donne accès à la collection de nœuds du document et retourne un objet de type *XmlNodes* … qui possède la méthode *ChildNodes*
- La propriété *Attributes* donne accès à la collection des attributs du document et retourne un objet de type *XmlAttribute*
- La méthode *WriteXml(nom_fichier)* de l'objet *DataSet* permet de créer un document XML à partir du DataSet .
- La méthode *ReadXml(nom_fichier)* de l'objet *DataSet* permet de charger un *DataSet* à partir d'un document XML.

Exemple de document Xml

```
using System.Xml;
...
XmlDocument doc;
doc = new XmlDocument();
doc.Load (@"c:\Pat.xml");
MessageBox.Show (doc.ChildNodes[1].Name);
 // Affiche "Identité"
MessageBox.Show (doc.ChildNodes[1].ChildNodes[0].InnerText);
 // Affiche "Dupond"
MessageBox.Show (doc.ChildNodes[1]. ChildNodes[1].Attributes.Value);
     // Affiche "Domicile"
```

Construction de l'exemple 1

- Nouveau projet sur le modèle "Application Windows"
- Création sur la fenêtre principale (FrmPrincipale) :
  - d'un bouton appelé *bLancement*
  - d'un bouton appelé *bArret*
  - d'un timer Windows appelé *Chrono*
- Codage
- Génération

Code de l'exemple 1

- Clause USING

using System;

using System.Drawing;

using System.Collections;

using System.ComponentModel;

using System.Windows.Forms;

using System.Data;

using System.Data.SqlClient;

- Dans public class FrmPrincipale

public int i;

- Dans private void bLancement_Click

Chrono.Interval = 5000;

Chrono.Start();

i=1;

Code de l'exemple 1 (suite)

- Dans private void bArret_Click

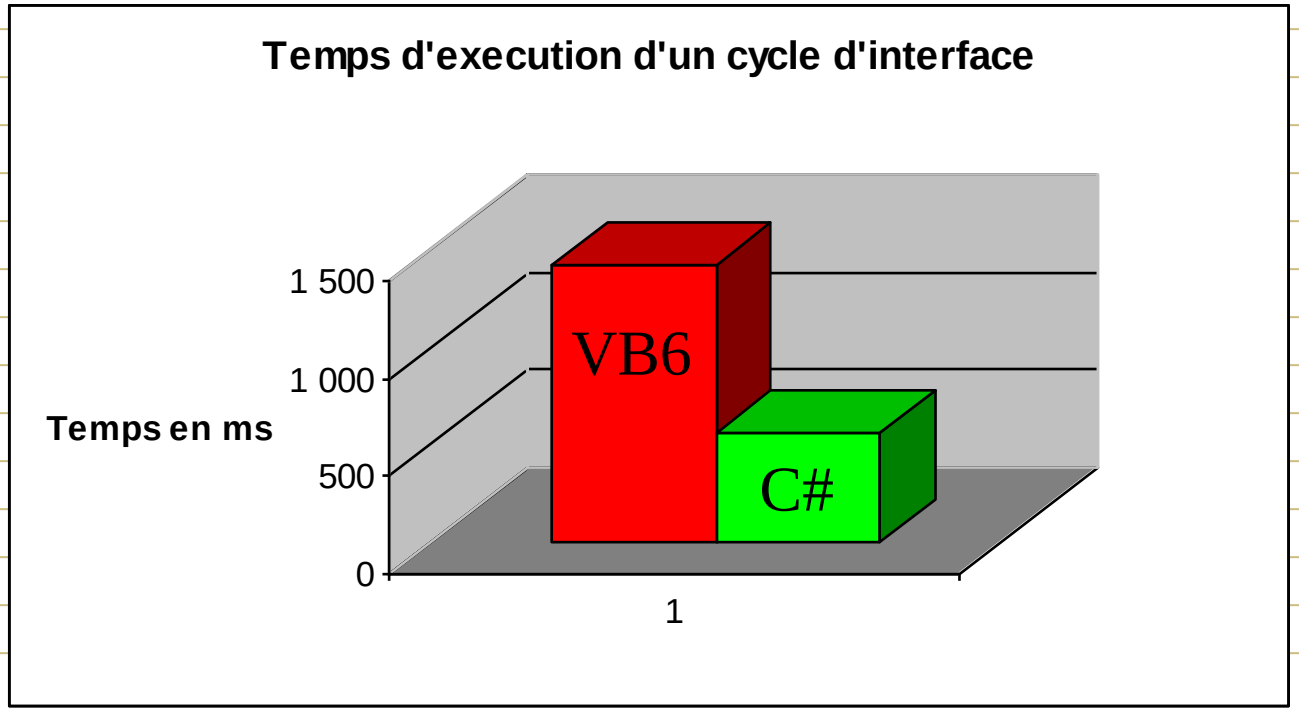
```
Chrono.Stop();
```

- Dans private void Chrono_Tick

```
SqlConnection objConnexion;
string strChaine =
        "Database=PRIMASERVICES;Server=Sancy;uid=id9;pwd=ghc00";
        objConnexion = new SqlConnection(strChaine);
objConnexion.Open();
SqlDataAdapter da;
da = new SqlDataAdapter("select * from idt_patient", objConnexion);
//chargement Dataset
DataSet ds = new DataSet();
da.Fill(ds, "t");
objConnexion.Close();
ds.WriteXml(@"c:\temp\CS_Exemple1_" + i + ".xml");
  i++;
```

C# : 3 fois plus rapide que VB

- La passerelle a été développée avec :
  - C# avec ADO.net (un cycle = 500 ms)
  - VB6 avec ADO 2.5 (un cycle = 1.5 s)
  - Delphi 5 avec BDE (un cycle = plusieurs mn)



Exemple 2 :
Passerelle en service NT

service Windows,

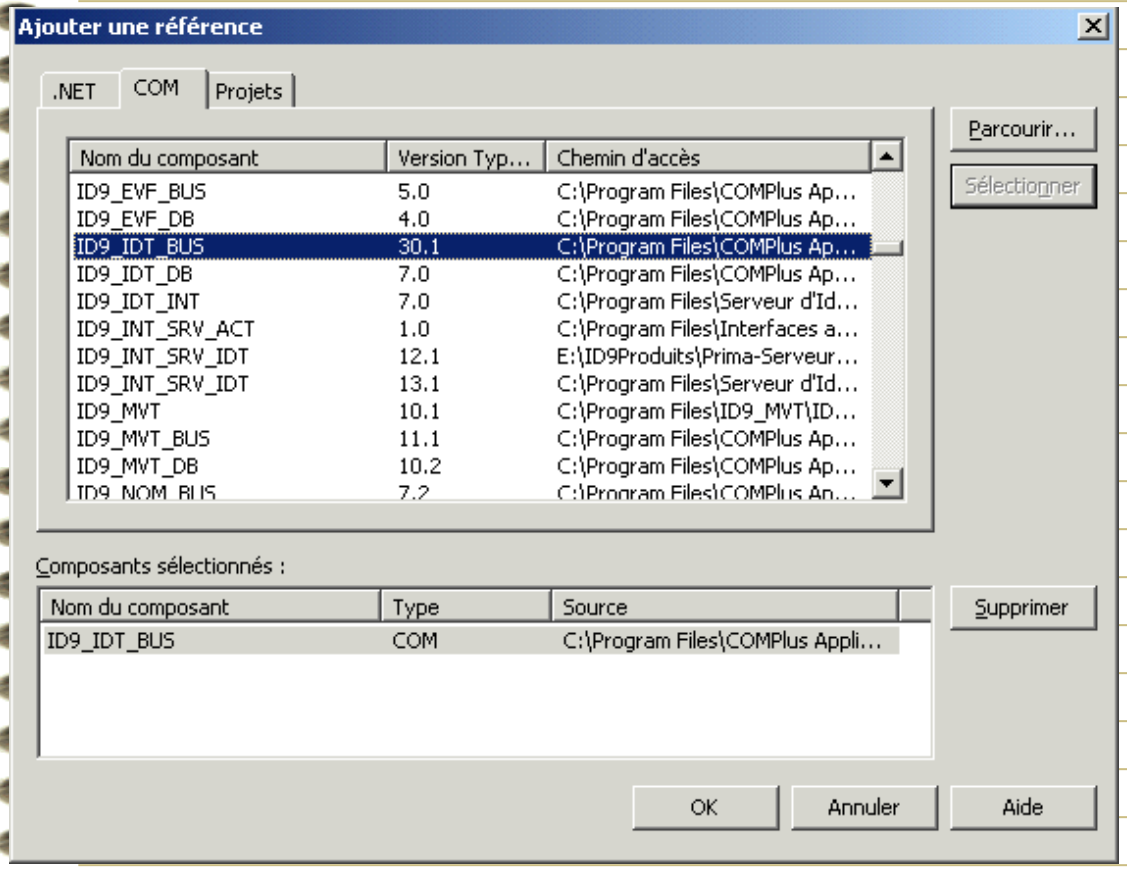
connexion aux serveurs (DCOM)

Présentation Exemple 2

- Toutes les 5 minutes, la passerelle va interroger le serveur d'identité (BUS) et chercher les informations relatives au patients provisoires. Ces informations sont enregistrées sur le disque dans le repertoire c:\temp sous un format XML dont le nom est incrémenté à chaque itération.

Utilisation de COM

Référence au composant COM



- Choisir le composant dans le menu "*Projet*", "*Ajouter une référence*", onglet "*COM*"
- Cliquer sur le bouton "*Sélectionner*"

Appel du composant

```
// Déclaration de l'objet
ID9_IDT_BUS._cls_identite idtObj ;
// Instanciation
idtObj = new ID9_IDT_BUS.cls_identiteClass();
// Accès au méthode
MessageBox.Show(idtObj.GetVersionBUS());
```

Service Windows

Écriture d'un service Windows

- Créer un nouveau projet avec le modèle "Service Windows"
- Faire glisser les composants (ex: Timer) sur le "Design" du service
- Double cliquer sur le concepteur("Design" pour coder :
  - l'action sur début de service (*void OnStart()*)
  - l'action sur fin de service (*void OnStop()*)

Déploiement

Principe

- Un projet de type "*Projets de configuration et de déploiement*" est ajouté à la solution contenant le projet à installer
- Un service Windows doit être installé pour se retrouver dans la liste du "Gestionnaire du contrôle des services"
- Le .*Net Framework* n'est pas installé par le setup. Il faut distribuer et lancer *Dotnetfx.exe*

Étapes principales

1/ Dans la solution du projet à installer ,"*ajouter un nouveau projet*" de type "*Projets de configuration et de déploiement*" avec le modèle "*Projet de configuration*"

2/ Sur la propriété "*ProductName*" de ce projet entrer le nom qui apparaîtra dans "*Ajout/Suppression de programmes*"

3/ Sur le nœud "*Dossier d'application*", choisir "*Action*", "*Ajouter*", "*Sortie de Projet*"

4/ Dans la liste déroulante choisir le projet à déployer et le groupe "*Sortie principale*" dans la liste

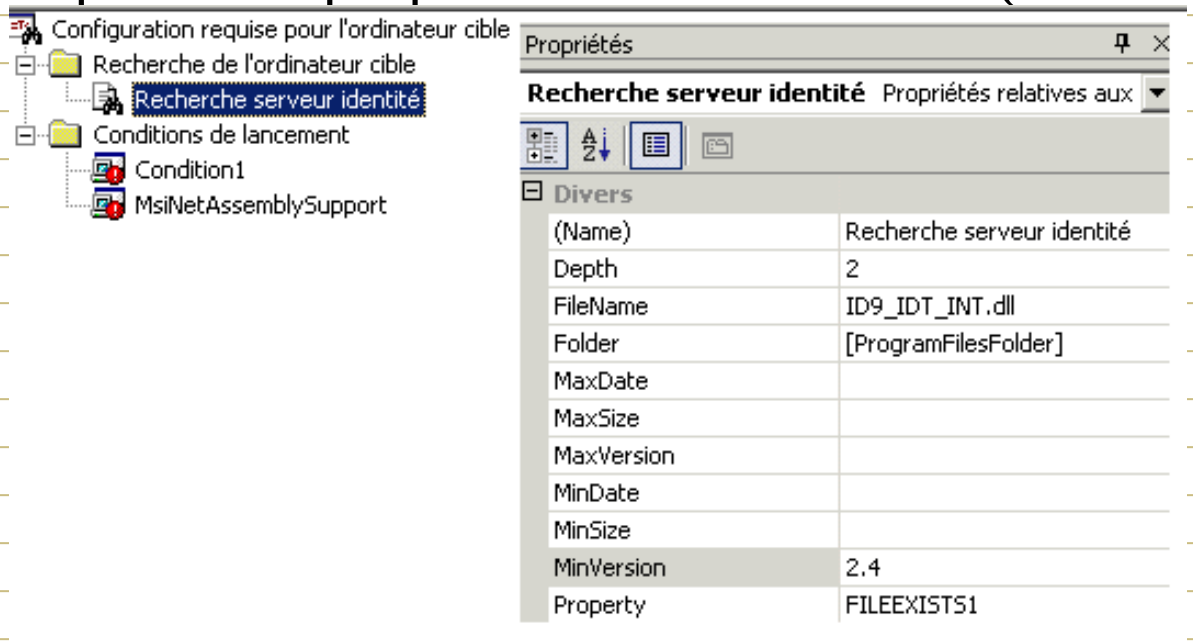
5/ Générer le projet

Créer des raccourcis

- Placer un raccourci sur le bureau de l'utilisateur
  - sélectionner le nœud "Sortie principale de .." dans "systèmes de fichiers"
  - clic droit, "Créer un raccourci vers"
  - le faire glisser vers le dossier "Bureau de l'utilisateur"
- Créer une arborescence du menu "Démarrer"
  - sélectionner le nœud "Menu Démarrer" et créer l'arborescence par clic droit
  - placer le raccourci comme précédemment

Tests de conditions d'installation

- "*Affichage*", "*Éditeur*", "*Conditions de lancements*", "*configuration requise pour l'ordinateur cible*"
- sur clic droit "*Ajout d'une condition de lancement de fichier*"
- compléter les propriétés de la recherche (comme suit)



- sur "Condition1", compléter la propriété "Message"

Compléments sur le déploiement

- Ajout d'une clé de registres
  - "*Affichage*", "*Éditeur*", "*Registre*"
- Boîte de dialogue personnalisée
  - "*Affichage*", "*Éditeur*", "*Interface utilisateur*"
- Création de répertoires
  - "*Affichage*", "*Éditeur*","*Système de fichiers*"
- Ajout de fichiers
  - Sur clic droit de "*Système de fichiers*", faire "*Ajouter*", "*Fichiers*"

Cas des services Windows

Dans le **projet du service**, il faut créer, avant génération, des "**programmes d'installation**" :

- cliquer sur "*Ajouter le programme d'installation*" dans la fenêtre propriété du concepteur de service
- les propriétés du composant "serviceInstaller" ainsi créé permet d'affecter un mode de démarrage.

Dans le **projet de déploiement**, il faut, avant génération, créer une "**action personnalisée**"

- dans l'explorateur de solution, clic droit sur le projet de déploiement puis "*Afficher*" et "*Actions personnalisées*"
- sélectionner le nœud "*Actions personnalisées*", clic droit et "*Ajouter une action personnalisée*"
- sélectionner la sortie principale du service dans "*Dossier d'application*"

Construction de l'exemple 2

- Nouveau projet "Service Windows"
- Création d'un timer serveur *Chrono*
- Ajouter une référence à ID9_IDT_BUS
- Codage
- Création des programmes d'installation
- Génération du projet service
- Création du projet de déploiement avec action personnalisée
- Génération du projet déploiement
- Installation

Code de l'exemple 2

- Clause USING

using System;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.ServiceProcess;

using System.Xml;

- Dans public class Service1

public int i;

- Dans protected override void OnStart

Chrono.Interval = 5000;

Chrono.Start();

i=1;

Code de l'exemple 2 (suite)

- Dans protected override void OnStop

```
Chrono.Stop();
```

- Dans private void Chrono_Tick

```
ID9_IDT_BUS._cls_identite idtObj = new ID9_IDT_BUS.cls_identiteClass();
XmlDocument doc;
doc = new XmlDocument();
        doc.LoadXml(idtObj.GetPatientByNomPrenomWithXML("","","%","%",
"01/01/1800","%","%","%","%",0));
i++;
doc.Save (@"c:\Temp\PatProv" + i + ".xml");
```

Compléments

Contrôle d'un service

Écriture dans le journal des évènements

Déclenchement sur réception de fichiers

Contrôler un service depuis une application Windows

❶ – Créer sur le concepteur un "ServiceController" (boîte à outils, "Composants")

❷ – instancier le. Par exemple si son nom est "sc":
ServiceController sc = new ServiceController();

❸ – utiliser les méthodes ou les attributs :

sc.Start();

sc.Refresh();

MessageBox.Show(sc.Status.ToString());

Écrire dans le journal des évènements

❶ – Créer sur le concepteur un "EventLog" (boîte à outils, "Composants") et le nommé (exemple : *ev*)

❷ – tester l'existence de la source et créer si besoin :

```
if (!System.Diagnostics.EventLog.SourceExists("Interface Test")
{System.Diagnostics.EventLog.CreateEventSource("Interface
  Test","Journal INT");}
```

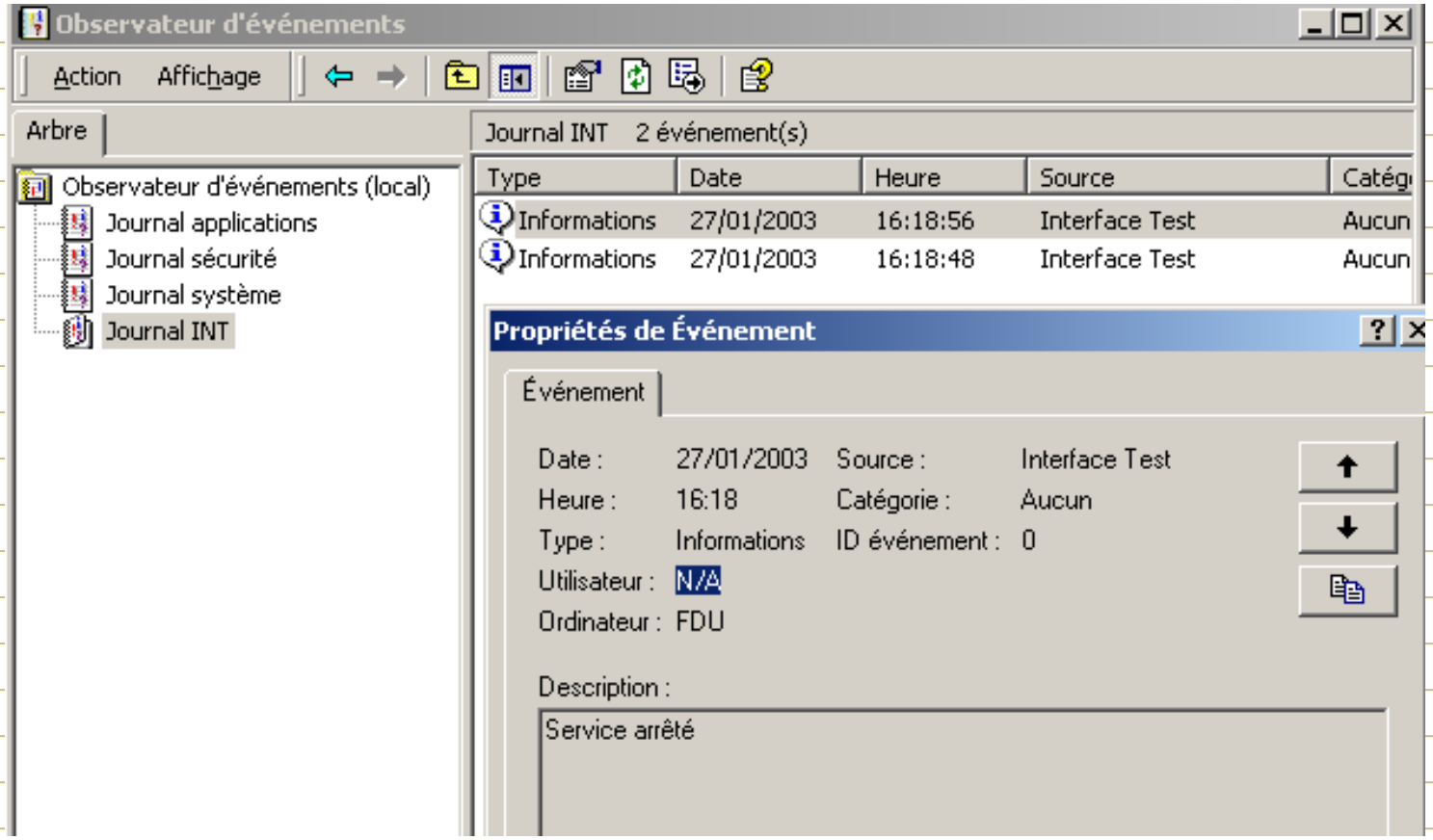
❸ – déclarer la source et le journal

```
ev.Source="Interface Test";
ev.Log="Journal INT";
```

❹ – utiliser la méthode *WriteEntry* pour écrire

```
ev.WriteEntry("Service démarré");
```

Exemple d'écriture de Log



Exemple de notification sur System.IO

```
System.IO.FileSystemWatcher FS  ; // déclaration

FS = new System.IO.FileSystemWatcher();// instanciation

FS.Path= @"C:\temp"; // répertoire à surveiller

FS.EnableRaisingEvents=true; // début surveillance

// exécution de la méthode Modif  en cas de modification d'un fichier
FS.Changed += new FileSystemEventHandler(Modif);

// execution de la méthode Crée en cas de création d'un fichier
FS.Created += new FileSystemEventHandler(Crée);

// exécution de la méthode Renom  en cas de renommage d'un fichier
FS.Renamed += new RenamedEventHandler(Renom);
```

Exemple de notification sur System.IO (suite)

```
public void Modif(object source, FileSystemEventArgs e)
{MessageBox.Show("Un fichier vient d'être modifié dans " + e.FullPath);
MessageBox.Show("Il s'agit du fichier " + e.Name);}

public void Crée(object source, FileSystemEventArgs e)
{MessageBox.Show("Un fichier vient d'être créé dans " + e.FullPath);
MessageBox.Show("Il s'agit du fichier " + e.Name);   }

public void Renom(object source, RenamedEventArgs e)
{MessageBox.Show("Un fichier vient d'être renommé dans " + e.FullPath);
MessageBox.Show( e.OldName + "a été  renommé en " + e.Name);}
```

Principaux évènements

- Changed Se produit lorsqu'un fichier ou un répertoire du Path spécifié est modifié.
- Created Se produit lorsqu'un fichier ou un répertoire du Path spécifié est créé.
- Deleted Se produit lorsqu'un fichier ou un répertoire du Path spécifié est supprimé.
- Error Se produit lorsque la mémoire tampon interne déborde.
- Renamed Se produit lorsqu'un fichier ou un répertoire du Path spécifié est renommé.