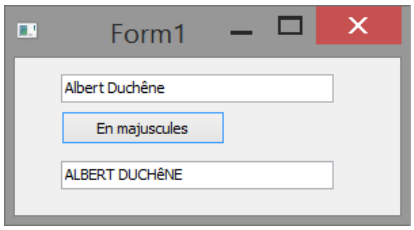
19 Communication TCP/IP avec DATASNAP

19.1 Communication simple avec une fonction distante



Voici une application client/serveur avec les composants **DataSnap.** Dans cet exemple l'application client met une chaîne quelconque en majuscules par appel d'un fonction distante accessible depuis le serveur. (dossier *excs_DataSnap_Function*)
Remarque : nous n'utilisons pas ici le Wizard proposé par Embarcadero pour créer le serveur.

19.1.1 Construction de la partie serveur avec VCL

Créer une application VCL de base enregistrée dans un nouveau dossier :
- Nom du projet : Ex01Serveur
- Nom de la fiche : Ex01ServeurA

Placer les trois composants suivants :

COMPOSANTS	NOM	PROPRIETES/EVENEMENTS	DESCRIPTION
TDSServer	DSServer1	AutoStart = True	
TDSServerClass	DSServerClass1	Server = DSServer1 OnGetClass = DSServerClass1GetClass	DSServerClass1GetClass est associée à l'événement OnGetClass décrit plus loin
TDSTCPServerTransport	DSTCPServerTransport1	Server = DSServer1 Port = 211	Il est possible de changer le port à votre convenance

TDSServer est le coeur logique de l'application serveur DataSnap avec les fonction de Marche/Arrêt
TDSServerClass représente une classe serveur. La propriété LifeCycle du composant TDSServerClass. a trois valeurs possibles :
- Server : une instance est créée par serveur d'exécution (singleton).
- Session : une instance est créée par connexion client active.
- Invocation : une nouvelle instance est créée pour chaque invocation d'un client (sans état).

TDSTCPServerTransport implémente un serveur TCP multithread écoutant les connexions client entrantes sur plusieurs threads.

Voici le code dans la partie serveur

```
. . .
type

{$METHODINFO ON}
  TServerMethods1 = class(TPersistent)
  public
    function EnMajuscules(Value: string): string;
  end;
{$METHODINFO OFF}

  TForm1 = class(TForm)
    DSServer1: TDSServer;
    DSServerClass1: TDSServerClass;
    DSTCPServerTransport1: TDSTCPServerTransport;
    procedure DSServerClass1GetClass(DSServerClass: TDSServerClass;
      var PersistentClass: TPersistentClass);
  end;
```

Description de la classe TServerMethods1 qui va être utilisée par le client

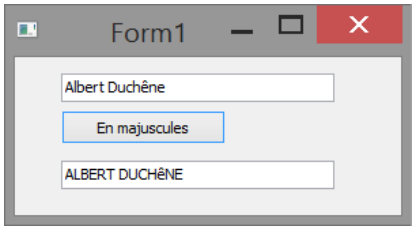
La directive $METHODINFO contrôle la génération de descripteurs de méthodes plus détaillés dans les RTTI pour les méthodes d'une interface; méthodes qui décrivent la manière dont les paramètres de la méthode doivent être transmis sur la pile et/ou dans les registres.

Les fonctions et procédures visibles du client doivent être en visibilité : **public**

. . . / . . .

procedure TForm1.DSServerClass1GetClass(DSServerClass: TDSServerClass;
  var PersistentClass: TPersistentClass);
begin
  PersistentClass := TServerMethods1;
end;
 	
function TServerMethods1.EnMajuscules(Value: string): string;
begin
  Result := UpperCase(Value) ;
end;
end.
	OnGetClass permet de spécifier la classe serveur. La définition est obligatoire. Toutes les méthodes publiques de cette classe peuvent être appelées par un client.
	
	Implémentation de la fonction qui sera utilisée par le client

19.1.2 Construction de la partie client



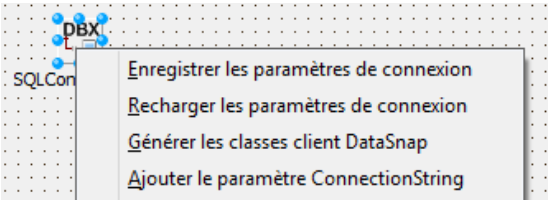
Voyons de manière plus détaillée la partie client.

La partie serveur doit être lancée pour pouvoir présenter une interface utilisable par le client

Outre les composants standards nous utilisons un composant **TSQLConnection** qui nous permet de décrire le pilote utilisé et ses paramètres. L'utilisation de ce composant nous indique que les mécanismes Client/Serveur sont similaires à ceux d'une requête SQL et de la réponse associée. Dans ce cas il ne s'agit pas de langage SQL à proprement parler.

COMPOSANT	NOM	PROPRIETES/EVENEMENTS	DESCRIPTION
TSQLConnection	SQLConnection1	Driver = 'DataSnap' Params.Strings	Rempli automatiquement Params.Strings en fonction du driver choisi Une fois le driver choisi il est possible de changer certain paramètres comme HostName=localhost Port=211 CommunicationProtocol=tcp/ip

Une fois ce composant placé en modifiant si besoin les paramètres comme le Port, nous allons générer automatiquement le code de la partie client. Le serveur doit être lancé. Un double-click droit sur le composant **TSQLConnection** ouvre une fenêtre de dialogue. Choisir "Générer les classes clients DataSnap". Le code généré dans une nouvelle unité est ensuite reporté dans l'unité de notre fiche



. . .
TMethodesDepuisServeur = class(TDSAdminClient)
public
function EnMajuscules(Value: string): string;
end;
 	
. . . / . . .
	Cette classe est dérivée de TDSAdminClient qui va gérer le tronc commun des fonction de **proxy**.
	
	Rappel : dans l'environnement particulier des réseaux, un **serveur proxy** est une procédure informatique **client/serveur** qui a pour fonction de relayer des requêtes entre une méthode cliente et une methode serveur (couches 5 à 7 du modèle OSI).

function TMethodesDepuisServeur.EnMajuscules(Value: string): string;
var
DBXCommand1: TDBXCommand;
begin
	TDBXCommand utilisé pour passer des commandes SQL peut aussi passer d'autre types de commande définis par **CommandType**

DBXCommand1 := FDBXConnection.CreateCommand; DBXCommand1.CommandType := TDBXCommandTypes.DSServerMethod; DBXCommand1.Text := 'TServerMethods1.EnMajuscules'; DBXCommand1.Prepare; DBXCommand1.Parameters[0].Value.SetWideString(Value); DBXCommand1.ExecuteUpdate; Result := DBXCommand1.Parameters[1].Value.GetWideString; FreeAndNil(DBXCommand1); end; . . . / . . .	**FDBXConnection** est décrite dans **TDSAdminClient** Nom de la fonction ou de la procédure à appeler Chargement du paramètre d'appel en WideString Appel de la procédure distante Récupération de la valeur de retour en WideString Destruction de l'instance

var
MDS1: TMethodesDepuisServeur;
begin
SQLConnection1.Open ;
MDS1 := TMethodesDepuisServeur.Create (SQLConnection1.DBXConnection) ;
Edit2.Text := MDS1.EnMajuscules(Edit1.Text);
MDS1.Free;
SQLConnection1.Close ;
end;
. . .
	Ouverture de la connection vers le serveur
	Création, utilisation d'une instance de **TmethodesDepuisServeur**
	
	Libération de l'instance après utilisation
	Fermeture de la connection vers le serveur