

New entity in PSG framework

- Products entity -

Creating a PSG entity in C # is simple (structured steps) and quick (20-60 min).

Prepare the working place:

That contains: client application, server application and a visual studio solution.

A instance of PsgDataSet visual studio solution should be opened.

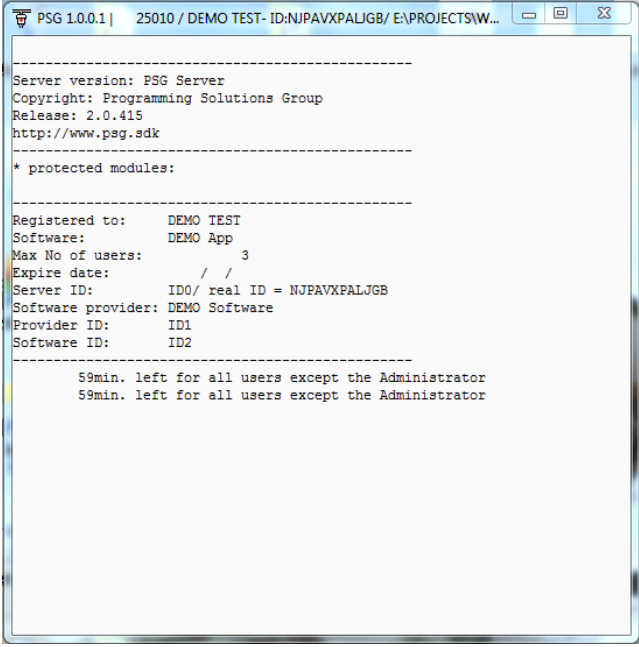
Additional resources you will need:

- PSG server manager
- PSG server (empty server)
- PSG Client

All of these are available for download on www.psgsdk.com

By default the server has a demo licence that allows tree users for one hour, after that only the administrator can use the server.

Messages from the connected user object are directed to the screen.



```
PSG 1.0.0.1 | 25010 / DEMO TEST- ID:NJPAVXPALJGB/ E:\PROJECTS\W...
-----
Server version: PSG Server
Copyright: Programming Solutions Group
Release: 2.0.415
http://www.psg.sdk
-----
* protected modules:
-----
Registered to: DEMO TEST
Software: DEMO App
Max No of users: 3
Expire date: / /
Server ID: ID0/ real ID = NJPAVXPALJGB
Software provider: DEMO Software
Provider ID: ID1
Software ID: ID2
-----
59min. left for all users except the Administrator
59min. left for all users except the Administrator
```

The steps used to create an entity:

1. Create tables in database (database)
2. Adding model, web services (Visual Studio solution - server)
3. Create project WinForms for entity (Visual Studio solution - client)
 - Add main form with all those controls: Products.cs
 - Add global variables, constructors and methods required by framework

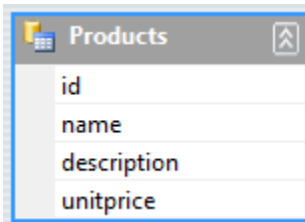
Database and web services

a) Scripts (PostgreSQL) for create tables products:

```
CREATE TABLE products
(
    id character varying(50) NOT NULL,
    name character varying(40),
    description character varying(5000),
    unitprice numeric(12,2),
    CONSTRAINT pp PRIMARY KEY (id)
)
```

b) Model and Web services:

In Model project add dataset ProductsDS.



In project PsgServices, folder 1.Services, class library Services.cs have web services.

In project PsgServices, folder 2.DataAccess add the class library "Products.cs".

Ex:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Model;
using System.Xml.Serialization;
using System.IO;
using System.Data.Odbc;

namespace PsgServer
{
    class Products
    {
        SqlConnection sqlcon;
        Server server;
```

```

        public Products()
        {
            server = Utils.server;
            sqlcon = server.sqlcon;
        }

        public void Load(object[] param, string outFileName)
        {}

        public void Save(string fileXmlEntity)
        {}

    }
}

```

In the Load method must write code for load data and return serialization file.
In the Save method must write code for save data from client application.

Ex Load:

```

public void Load(object[] param, string outFileName)
{
    if (Utils.StringEquals(param[0], Utils.Null))
    {
        DataTable dt = new DataTable("Products");

        string cmd = " Select * from products ";

        Utils.LoadView(sqlcon, cmd, ref dt);
        server.Serialize(dt, outFileName);
    }
    else
    {
        ProductsDS ds = new ProductsDS();

        string cmd = string.Format(" Select * from products where id = {0} ", param[0]);

        try
        {
            sqlcon.Open();
            Utils.LoadDataSet(sqlcon, ds, cmd, "Products");
        }
        catch (Exception ex)
        {
            server.smsg(ex.Message);
        }
        finally
        {
            sqlcon.Close();
        }

        server.Serialize<ProductsDS>(ds, outFileName);
    }
}

```

Ex Save:

```

public void Save(string fileXmlEntity)
{
    ProductsDS be = server.Deserialize<ProductsDS>(fileXmlEntity);

    try
    {
        sqlcon.Open();
        Utils.SaveDataSet(sqlcon, be, "Products", "products");
    }
    catch (Exception ex)
    {
        server.smsg(ex.Message);
    }
    finally

```

```

    {
        sqlcon.Close();
    }
}

```

Attention!

[Server needs Model.dll and PsgServices.dll](#)

Build project and copy files (Model.dll and PsgServices.dll) to csharpdll directory on the server.

[Client needs Model.dll](#)

The Model.dll library must be added on server (Utils - Other auxiliary files).

Attention!

Add command "products" type script language CSharp+. (config.exe application on the server Utils -> Services).

Create WinForms project

Open Microsoft Visual Studio.

Choose File -> New -> Project -> Visual C# -> Windows -> Windows Forms Application.

Name the project "Products". Option "Create directory for solution" has to be checked.

Open Project Properties and set Target Framework to ".Net Framework 4" (not Client Profile).

Add reference Model (project Class Library created on server)

Add reference PsgBase (get file "psgbase.exe" from our site [C# Classes & Templates / SDK](#))

Path: .\C# Framework

Add new Windows Form (Products.cs), this will be the main form.

Main form file

- Use base class "BaseEntity". If your project have details, you must use base class "BaseEntityDetails".

Add "using PsgBase;" in main form "Products.cs".

```

public partial class Products : BaseEntity
{
    ...
}

```

Main form must have "EditorClass" attribute and, depending on the case, you must use one of attributes: NoView, ViewGrid, ViewTree, ViewTreeList

b) You should use the NoView attribute when your document don't have a view, but data is loaded using standard psg methods ([PsgStart](#), [PsgData](#), [PsgBind](#))

If your document doesn't have a view and data is loaded without using standard psg methods, then you should not use the view attribute.

In our case we have:

```

[EditorClass, ViewGrid]
public partial class Products : BaseEntity

```

- Add global variables:

```
ProductsDS ds = new ProductsDS();
ProductsDS.ProductsDataTable table = new ProductsDS.ProductsDataTable();
ProductsDS.ProductsRow rowA = null;
```

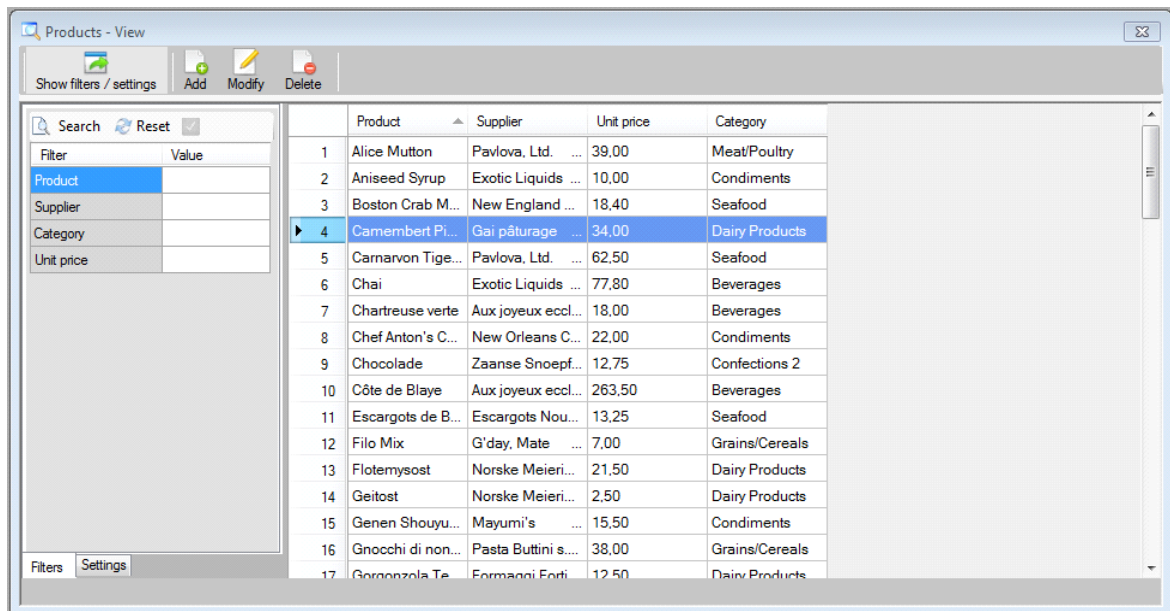
- Add constructors:

```
public Products()
{ InitializeComponent(); }
public Products (ViewGrid _view):base(_view)
{ InitializeComponent(); }
public Products (LookupGrid _lookup):base(_lookup)
{ InitializeComponent(); }
```

View / Lookup form

Each document can have one view and/or lookup (This form is generated by the framework).

Ex view:



Columns and filters of view / lookup can be customized by code in source file – method `PsgView / PsgLoopup. ()`

When we run a document, the view form is loaded, then we can use standard commands: add, modify, delete.

```
public override void PsgView()
{
    base.PsgView();

    this.ColumnsView(table.nameColumn, "Product");
    this.ColumnsView(table.unitpriceColumn, "Unit price");
}
```

```

        this.FiltersView(table.nameColumn, "Product");
        this.FiltersView(table.unitpriceColumn, "Unit price");
    }

    public override void PsgLookup()
    {
        base.PsgLookup();

        this.ColumnsLookup(table.nameColumn, "Product");
        this.ColumnsLookup(table.unitpriceColumn, "Unit price");

        this.FiltersLookup(table.nameColumn, "Product");
        this.FiltersLookup(table.unitpriceColumn, "Unit price");
    }

```

Editor form

Open form Products.cs

- Add context:

```

public override void PsgContext(IContext context)
{
    base.PsgContext(context);

    context.PsgFieldKey = "id";
    context.PsgFieldDisplay = "name";
    context.PsgCommand = "Product";
    context.PsgServerTableName = "product";
}

```

- Get tables from server:

```

public override void PsgStart()
{
    base.PsgStart();

    AddService("products", ID);
    RunServices(this);
}

```

For each requested table a corresponding web service should exist on server.

Please check web services tutorials for details - www.psgsdk.com/downloads.php.

The GETDATA object deals with server communication, please check the SDK help file for more information.

- Use data from server:

PsgData() is triggered after the data is loaded from the server and need to be use.

```

public override void PsgData(DataTable dt)
{
    base.PsgData(table);
    switch (this.alias_name.ToUpper())
    {
        case "PRODUCTS":
            this.BeforeSetMainData(table);

            ds.ReadXml(this.serializedDataFile);
            table = ds.Products;

            if (table.Rows.Count > 0)
                rowA = (ProductsDS.ProductsRow)table.Rows[0];
    }
}

```

```

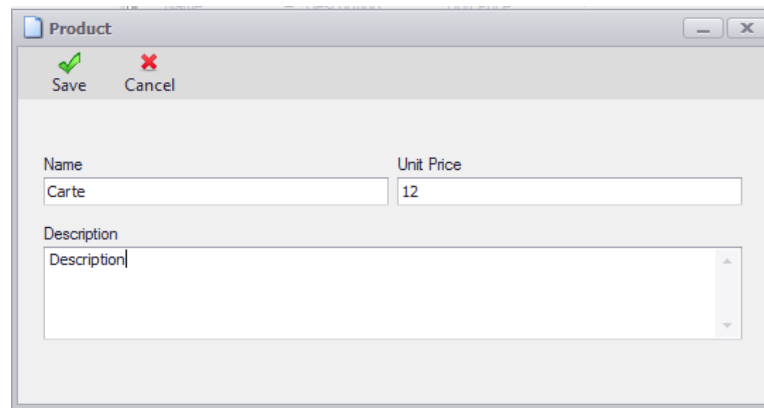
        this.AfterSetMainData(table);
        break;
    }
}

```

`this.serializedDataFile` – file received from the server

Adding controls (Windows Forms) to form

- Open the 'Products.cs' form and start to add controls on it.



After you have added all controls, you must make the binding with the dataset.

```

public override void PsgBind()
{
    BindEdit(edName, table.nameColumn);
    BindEdit(edDescription, table.descriptionColumn);
    BindEdit(edUnitPrice, table.unitpriceColumn);

    base.PsgBind();
}

```

Here we can use:

BindEdit - binding control psgEdit to field from dataset

BindCombo - binding controls psgCombo, psgComboEdit to fields from dataset

BindData - binding control psgDateTime to field from dataset

In order to save must use the method:

```

public override void PsgSaveEntity()
{
    base.BeforeSaveEntity();
    SaveDataSet("Products", table.DataSet);

    base.PsgSaveEntity();
}

```

Run the module

Build project in class library file (ex: products.dll).

The module (products.dll) must be registered on the server.

A user with administrator rights can allocate this module to any user.

Final

Feel free to improve the basic design of the demo project as you want.

Help and technical support from PSGSDK is available by email, please check the psgsdk.com.

Other articles and information could be found on the website.