



Programming tutorial for PSG & C#

Programming Solutions Group

Note:
To change the product logo for your own print manual or PDF,
click "Tools > Manual Designer" and modify the print manual
template.

Title page 1

Use this page to introduce the product

by Programming Solutions Group

This is "Title Page 1" - you may use this page to introduce your product, show title, author, copyright, company logos, etc.

This page intentionally starts on an odd page, so that it is on the right half of an open book from the readers point of view. This is the reason why the previous page was blank (the previous page is the back side of the cover)

Programming tutorial for PSG & C#

Programming Solutions Group

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: October 2012 in (whereever you are located)

Publisher

...enter name...

Managing Editor

...enter name...

Technical Editors

...enter name...

...enter name...

Cover Designer

...enter name...

Team Coordinator

...enter name...

Production

...enter name...

Special thanks to:

All the people who contributed to this document, to mum and dad and grandpa, to my sisters and brothers and mothers in law, to our secretary Kathrin, to the graphic artist who created this great product logo on the cover page (sorry, don't remember your name at the moment but you did a great work), to the pizza service down the street (your daily Capricciosas saved our lives), to the copy shop where this document will be duplicated, and and and...

Last not least, we want to thank EC Software who wrote this great help tool called HELP & MANUAL which printed this document.

Table of Contents

Foreword	1
Part I Introduction	3
1 Welcome topic	3
2 C# Demo project	4
Part II Northwind modules	7
1 Overview	7
2 PSG programming model	8
3 Web services	10
C#	11
VFP	12
4 The application	13
5 Customers module - tutorial	15
Database and web services	17
Create project WinForms	17
Main form file	18
First loading form	19
Adding controls to form	20
View / Lookup	22
Run the module	22
Final	22
6 Orders module - tutorial	23
Database and web services	24
Create project WinForms	25
Main form file	25
First loading form	26
Adding controls to form	28
View / Lookup	30
Run the module	31
Final	31
Part III Using DataGridView	33
Part IV Other model documents	37
Index	0

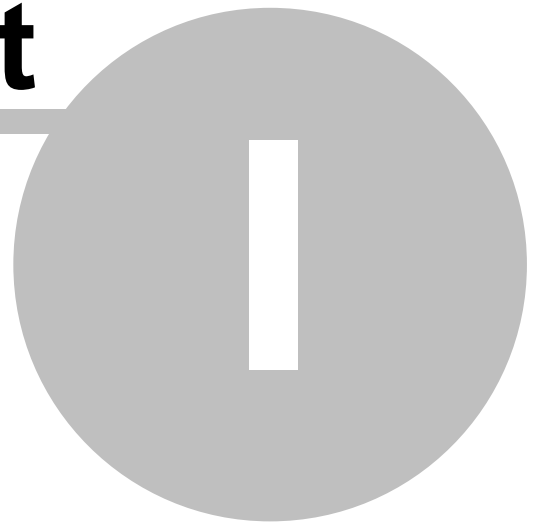
Foreword

This is just another title page
placed between table of contents
and topics

Top Level Intro

This page is printed before a
new top-level chapter starts

Part



1 Introduction

PSG programming tutorial for C# programmers.

Based on the C# demo project on PSG platform, this document presents a step by step tutorial, on how to create an application module using basic classes and templates of PSG. Programming client/server database applications, is now easier than ever, using your programming language of choice.

References:

- PSG demo projects.
- PSG server programming help file
- PSG client C# programming help file

Requirements:

- download and install the demo project on PSG for C# (www.psgsdk.com)
 - PSG Server manager
 - PSG Demo Project server
 - PSG Client
- C# programming language
- The Demo project includes NET-Reports engine
 - for Excel reports, Ms.Office is required
 - the Ms. Query from Ms. Office CD should be installed for Excel PivotTable reports (it is not installed by default with Ms.Office by some Ms. Office distributions)

The demo project was built in fifteen working hours during a week time. The time needed for the database documentation was also short, considering we used Northwind. We will present a step by step walk-through of the "Orders" module in the demo project. The source code is available for the entire project.

1.1 Welcome topic

PSG could be used with different programming languages such as C#,Java and VFP. The client PSG application is fully compatible with Windows Vista and Windows 7 OS.

PSG attributes for a modern architecture:

- a 4-tier client/server architecture for database applications
- base classes and templates
- database server independent platform
- high security communication protocol (HTTPS communication protocol and support for client hardware keys by default)
- short development and implementation time
- REST based web service client/server communications
- users management
- automatic update of client application
- simple help system
- international toolkit
- integrated reporting and data analysis tools
- server side licenses management

Programming Intranet and Internet database applications are the PSG platform's main task.

Please check the documentation included with the server and the PSG C# programming manual for more information.

1.2 C# Demo project

The demo project for C# on PSG was built based on a modified version of Northwind database. Northwind is a small and less complex database - being used as a demo database for many years, there is a chance that many programmers already know it, giving us the reason to use it for the demo and tutorial.

The differences compared to the original version of Northwind are found in the structure of some tables.

All tables that used to have a numeric auto increment primary key, were changed to a twelve character primary key (and all references)

All original data can be found in the database, the DATE field values were changed to actual period, so that they can be easily used in reports.

Two fields were added to each table:

- sid - code of the last user who modified the record
- mdl – date/ time of the last update of the record

There are three databases servers to choose from:

- Ms. SQL database server
- PostgreSQL
- Sqlite 3.0 (embedded database server component)

The application is working in the same manner with all of them (Sqlite 3.0 can be used with fewer users only).

By default, the server is installed with the Sqlite 3.0 database, as there is no installation required for the database.

The PostgreSQL and Ms. SQL databases backup is in the DB_back directory. To change the database, just install the databases from the backup and modify the server connection strings as described below:

There are two connection strings to be modified:

- server main connection (server configuration program)
- [reports management](#) - connections

Samples of connection strings:

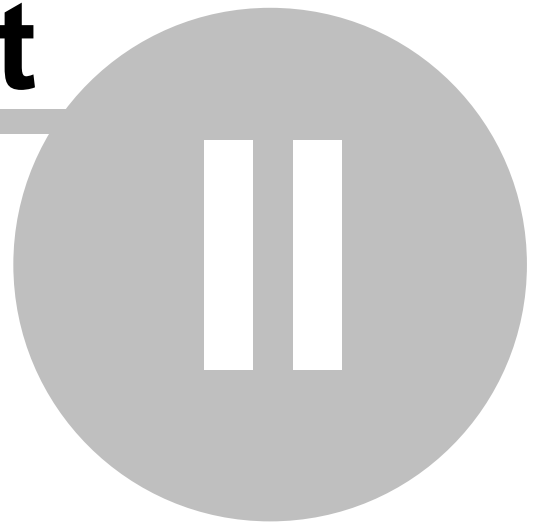
PostgreSql	Driver={PostgreSQL UNICODE};Server=localhost;Database=northwind;Port=5432;Uid=postgres;Pwd=123456;SSL=No;
MS.Sql	Driver={SQL Server};Server=ADRIAN-PC\SQLEXPRESS;Database=northwind_psg;Uid=sa;Pwd=123456;

Sqlite 3.0 DRIVER=SQLite3 ODBC
Driver;Database=C:\projects\wwb\db3\psg\northwind.db3;LongNames=0;Timeout=1000
;NoTXN=0;SyncPragma=NORMAL;StepAPI=0;

Top Level Intro

This page is printed before a
new top-level chapter starts

Part



2 Northwind modules

The Northwind database is simple, offering the basic functionality for a sales recording application.

The Demo application is built around the Northwind database, covering the interfaces for all lookup tables and the main module which is recording the sales orders.

On top of that the NET-Reports engine has some nice reports and data analysis.

2.1 Overview

The 'Orders' module:

- Orders list form

The screenshot shows the 'Orders - Browser' application window. It features a toolbar with 'Hide filters / settings', 'Add', 'Modify', and 'Delete' buttons. On the left, there is a 'Filter' section with a 'Search' and 'Reset' button, and a table for setting filters for 'Customer', 'Employee', 'Order date', 'Ship date', 'Freight (min)', 'Freight (max)', and 'Ship address'. The main area displays a table of 16 orders. The table has columns for 'Customer', 'Employee', 'Order date', 'Ship date', 'Freight', and 'Ship address'. The 5th order is highlighted in blue.

	Customer	Employee	Order date	Ship date	Freight	Ship address
1	Frankenversand	test 1	01.05.2012	03.05.2012	77,00	bbb
2	Cactus Comidas para lle...	Davolio 34	27.12.2011	05.01.2012	78,60	test 11
3	Königlich Essen	Davolio 34	26.12.2011	05.01.2012	2,33	Garden d
4	Drachenblut Delikatessen		25.12.2011	05.01.2012	78,80	
5	Mere Pailarde	test 1	23.12.2011	01.01.2012	16,85	Berliner F
6	Folies gourmandes	Davolio 34	23.12.2011	31.12.2011	100,60	Asda
7	test 1	Davolio 34	22.12.2011	19.01.2012	34,00	as
8	Gourmet Lanchonetes		22.12.2011	26.12.2011	55,00	rrr
9	Queen Cozinha	3	19.12.2011	23.12.2011	110,87	Alameda
10	Hanari Carnes		18.12.2011	19.12.2011	124,98	Rua do P
11	Magazzini Alimentari Ri...		18.12.2011	22.12.2011	70,09	Via Ludov
12	GROSELLA-Restaurante	Davolio 34	18.12.2011	24.12.2011	1,51	5ª Ave. Lo
13	Wartian Herkku		17.12.2011	19.12.2011	73,16	Torikatu 3
14	Cactus Comidas para lle...	Dodsworth	17.12.2011	22.12.2011	1,10	Cerrito 33
15	Berglunds snabbköp	3	17.12.2011	19.12.2011		
16	Berglunds snabbköp		16.12.2011	24.12.2011	6,79	Berguvsve

- Order details form

The screenshot shows a web application window titled "Order". It contains several input fields for customer, shipper, and shipping information, a table of products, and a total calculation.

Customer Information:

- Customer: Folies gourmandes
- Employee: Davolio 34
- Order date: 23.12.2011
- Require date: (empty)
- Ship date: 31.12.2011

Shipper Information:

- Shipper: United Package 34
- Ship name: aSAS
- Ship address: Asda
- Freight: 100.60

Shipping Information:

- Ship city: Lille
- Ship region: Not Specified
- Ship postal code: 590008
- Ship country: France

Products Table:

	Product	Unit price	Quantity	Discount	Data
▶	Camaron Ti...	62,5	30	0	
	Steeleye Stout	18	15	0	
	Scottish Lon...	12,5	18	0	
	Vegie-spread	2	2	2	

Total: 95

Tables used by orders:

orders	orders list
orderdetails	order details
customers	list of customers
employees	list of employees
products	list of products
shippers	list of shippers

We need to create the PSG web services and the presented user interfaces.

2.2 PSG programming model

The PSG platform offers a multi tier programming model.
The next picture offers a quick view of the platform's main components.

The Application - all modules that interact with the end user, the application user interface.

PSG Client - Offers the main screen and menu for the application and the communication object PSGCON.

PSG Server - communication and application server

User object - for each user there is a user object instantiated on PSG server

Data connection - the connection to the database server, one connection for each user

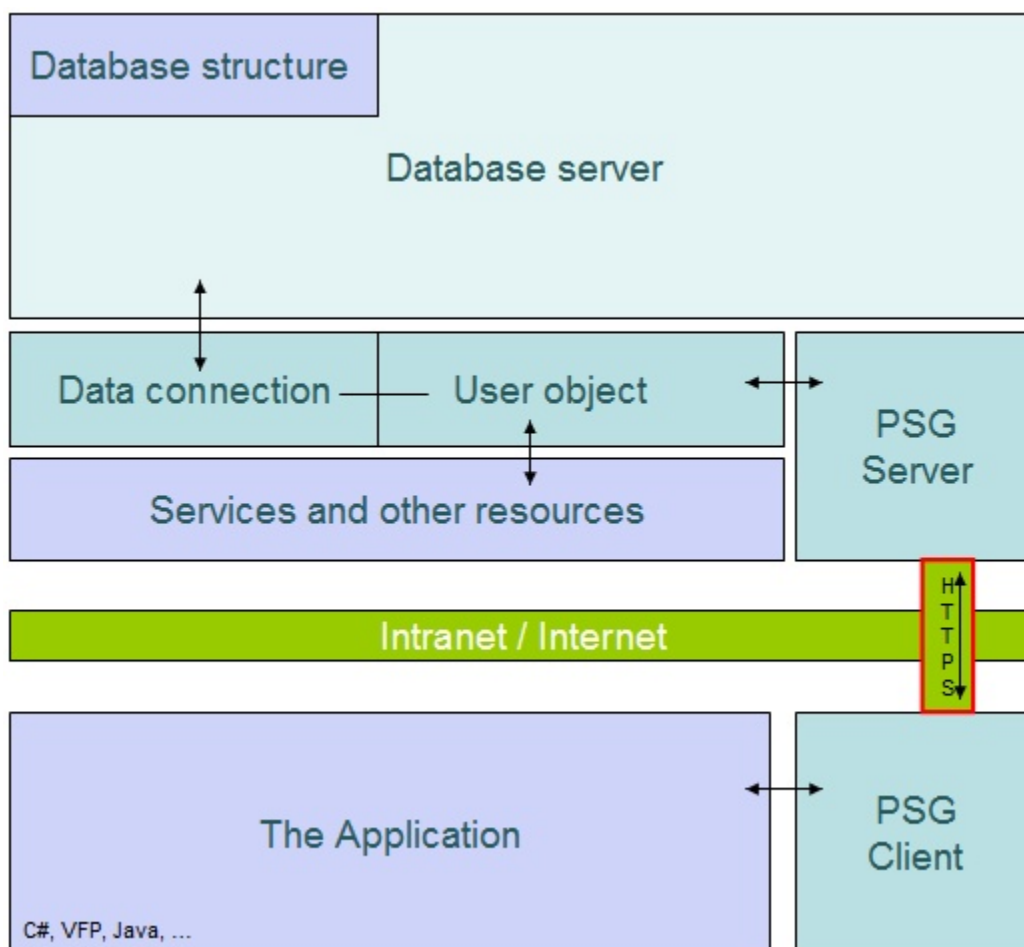
Services and other resources - requests sent to the server are based on web services, the service model is REST.

There is a big difference between REST and SOAP web services, while SOAP services are objects which should be instantiated by the client, the REST model is simpler and deals with requests and responses. The REST model is more reliable on heterogeneous networks and offers better scalability - it doesn't need WSDL XML files and there is nothing to be registered or instantiated on the server OS, the response is very fast compared with SOAP model.

Database server and database structure - these are not part of the PSG platform. They are represented just to complete the picture, any relational database server like Ms.Sql, PostgreSQL, Oracle, Sybase, MySQL and others could be used, along with the database itself.

To simplify the concept, we must consider the PSG server and the PSG client, as one application tier. Considering this, the model works as a four tier. As a programmer of a PSG application, you can interact with three tiers : the application, services and the database structure/database triggers (if used).

The business logic of the application should be balanced to the platform levels as needed and it's better to be used for services or database triggers.



To create a PSG based application you will need a database first.

Programming with PSG:

- programming PSG web services
- the application user interface modules
- the database

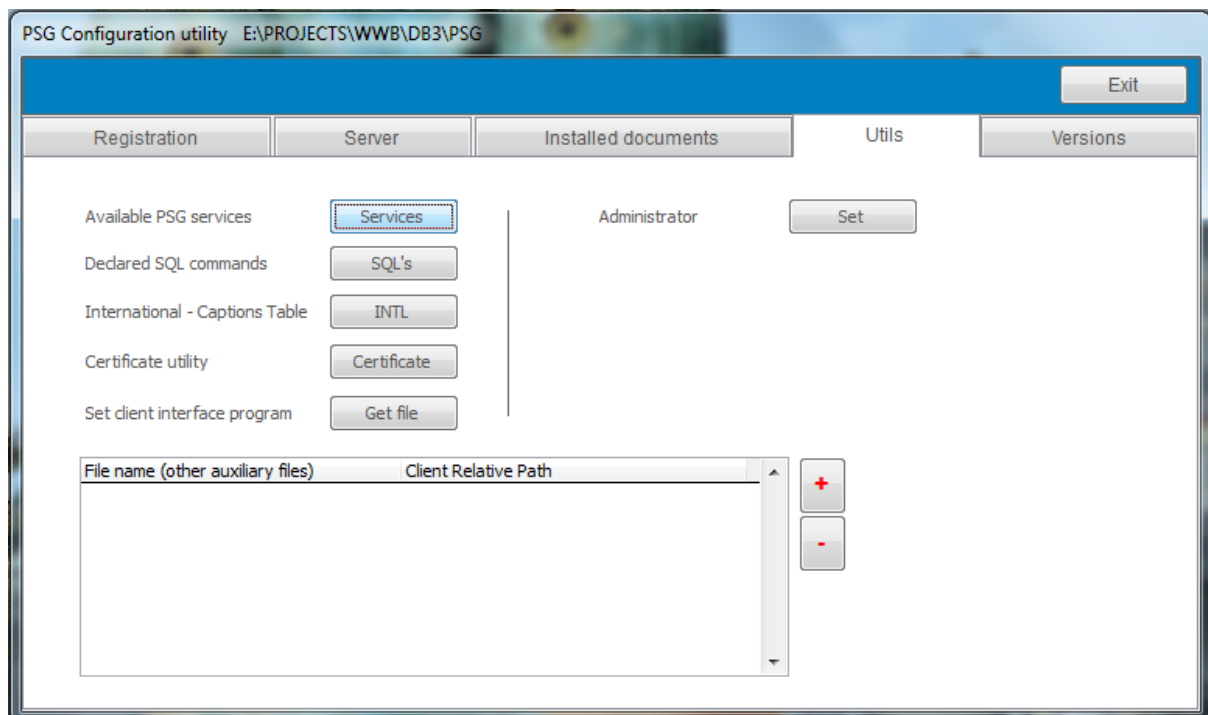
Next, we will focus on the tutorial topic, so please check the documentation for more details.

2.3 Web services

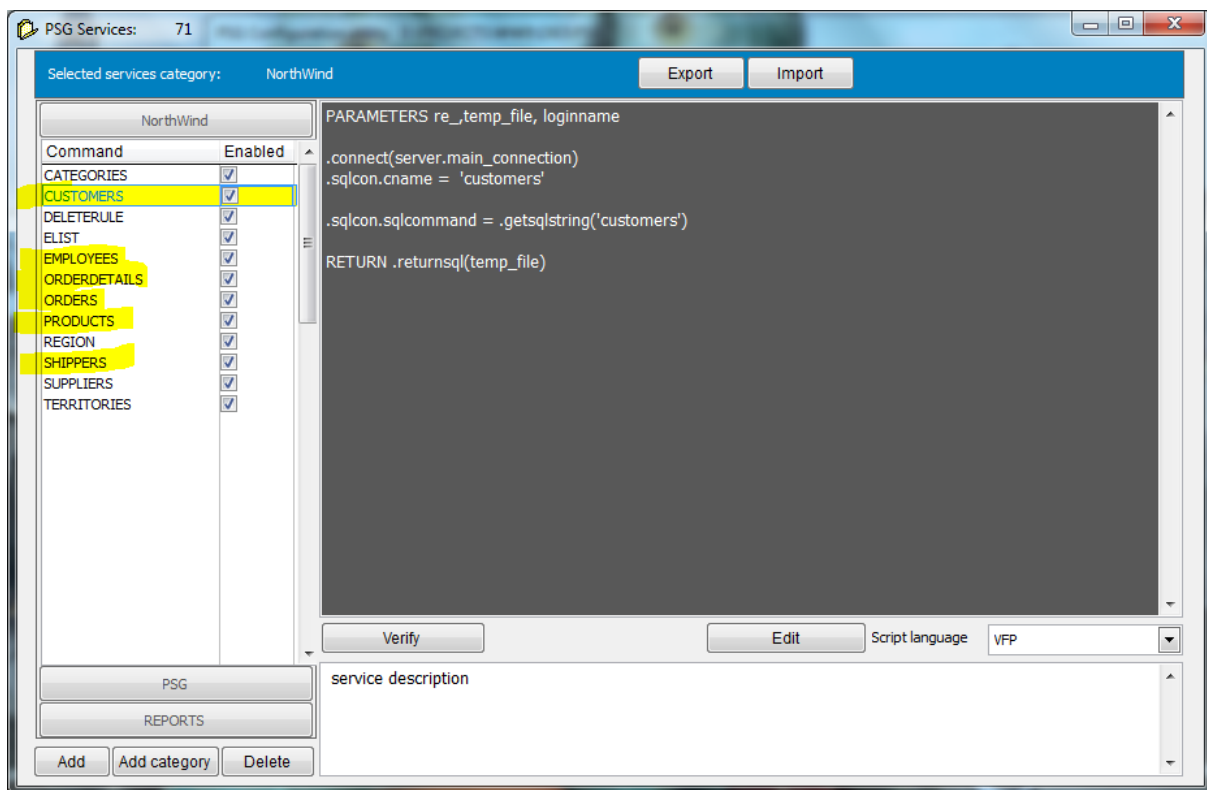
The PSG web services created for the "Orders" module, are used for data requests only.

For each required table, we need a web service that prepares the data on the server, for being transferred.

Use the server configuration utility to open the services manager.



Service editor and used services for the ORDERS module:



2.3.1 C#

Sample of "CUSTOMERS" service:

```
public string testcs(object[] param)
{
    sqlcon.cname = "customers";
    sqlcon.sqlcommand = "select * from customers";
    sqlcon.sqlexec(server.main_connection);

    return sqlcon.result;
}
```

This service is used to return the list of customers without any filter. The service is used by another one called "DATA", the DATA service being used to request temporary tables from the server, for local use on the client side - please also check the DATA service presented into the server programming help manual.

The web services editor uses a category list of services: Northwind, PSG, Reports respectively

application services, base services (PSG), NET-Reports reporting engine services (REPORTS). Services could be written using VFP, C#, VBscript, Javascript. Services written in different languages could coexist without interference. The service name is unique, regardless of the category name - the category is here just to help organize the services and it has no influence on the run time.

Coming back to "CUSTOMERS" service - here it is explained line by line:

`object[] param`

Parameters sent to the server.

`sqlcon.cname = "customers";`

set the returned cursor name

`sqlcon.sqlcommand = "select * from customers";`

set the SQL string to be sent to the database server.

`sqlcon.sqlexec(server.main_connection);`

execute the SQL statement

`return sqlcon.result;`

returns the cursor

To create a service, just copy and paste from another one and modify the code to fit the new service.

Services could be used for a lot of purposes.

By using services, a communication between client and server could be established as the response from server, or it could be a client side command (if implemented).

Please check the documentation of GETDATA into the client programming help file, for more details.

Some of the services here were created for other application modules and are being used for ORDERS too.

2.3.2 VFP

Sample of "CUSTOMERS" service:

`PARAMETERS re_,temp_file, loginname`

`.connect(server.main_connection)`

`.sqlcon.cname = 'customers'`

`.sqlcon.sqlcommand = .getsqlstring('customers')`

`RETURN .returnsql(temp_file)`

This service is used to return the list of customers without any filter.

The service is used by another one called "DATA", while the DATA service is used to request temporary tables from the server. For client side local use, please also check the DATA service, presented into the server programming help manual.

The web services editor uses a category list of services: Northwind, PSG, respectively Reports application services, base services (PSG), NET-Reports reporting engine services (REPORTS). Services could be written using VFP, C#, VBscript, Javascript. Services written in different languages could coexist without interference. The service name is unique, regardless of the category name - the category is here just to help organize the services and it has no influence on run time.

Coming back to "CUSTOMERS" service - here it is explained line by line:

PARAMETERS `re_`, `temp_file`, `loginname`

`re_` - is here for backward compatibility, the parameters were used into PSG 1.0, but some applications still use it.

`temp_file` - name of the archive which should be sent back to the client.

`loginname` - username, added automatically to each service request by the user object accepted method.

If there was any parameter sent to the server, it should be between '`re_`' and '`temp_file`'.

`.connect(server.main_connection)`

check if the used ODBC connection, is pointed to the main database, if not a connection to the database is established.

`.sqlcon.cname = 'customers'`

set the returned cursor name

`.sqlcon.sqlcommand = .getsqlstring('customers')`

set the SQL string to be sent to the database server. To have it more easily managed, there is the possibility to store the command. 'GETSQLSTRING' is used to retrieve the command. Use "SQL's" from "Utils" page of the server configuration, to access stored SQL's.

RETURN `.returnsql(temp_file)`

RETURNSQL - execute the SQL statement and returns the cursor

To create a service, just copy and paste from another one and modify the code to fit the new service.

Services could be used for a lot of purposes.

By using services, a communication between client and server could be established as the response from the server could be a client side executed command (if implemented).

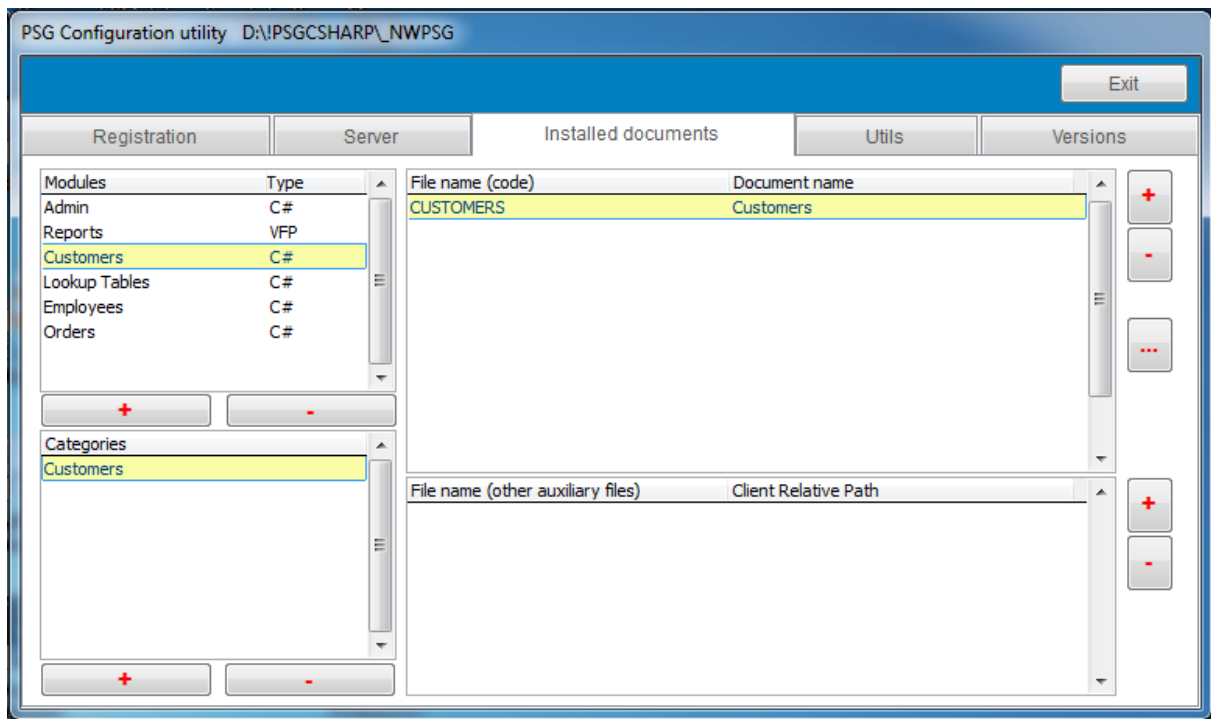
Please check the documentation of GETDATA into client programming help file for more details.

Some of the services here were created for others application modules and are used for ORDERS too.

2.4 The application

The client application offers a basic user interface that loads the application menu and initial modules, such as the report engine. The main application also embeds the communication object PSGCON. By using the PSGCON, the client could send a request to the server. The server will respond with data or a client side executed command, while a kind of device communication could be established using web services and local commands. For simple applications like the one presented here, only basic communications should be handled automatically, base classes and templates being available for each control which needs to communicate with the server for insertion, update or delete.

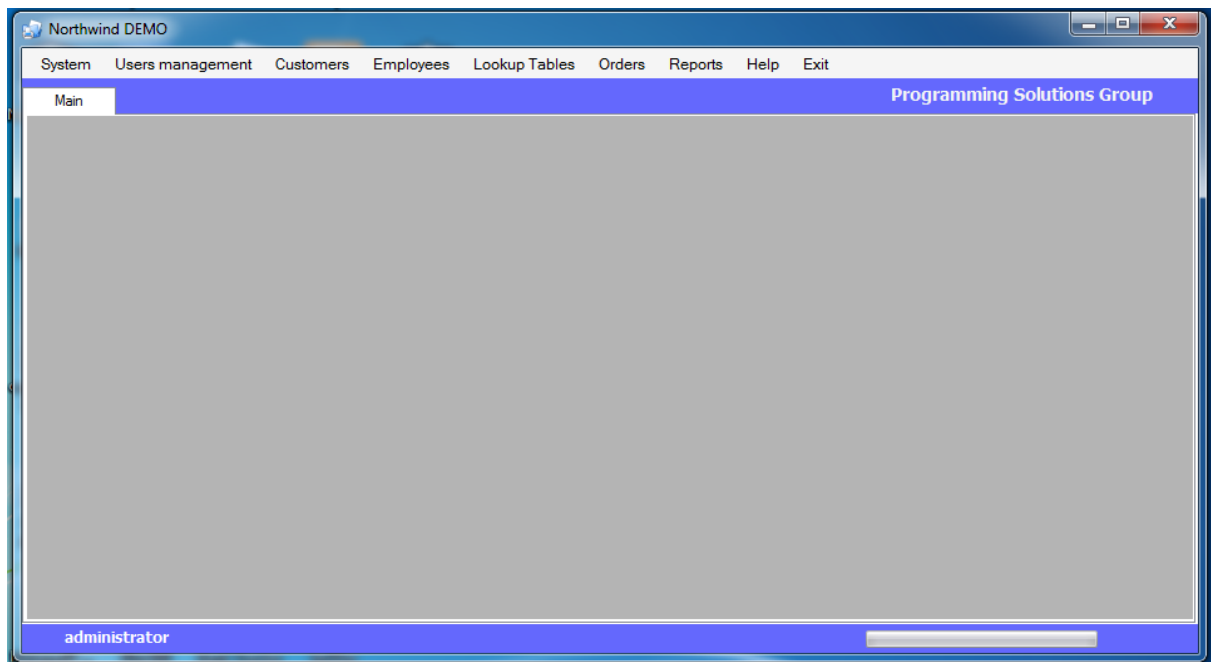
The Orders module is initiated from the main menu of the application. All used modules should be registered on the server, by using the server configuration utility. The registration is being stored into the PSG structures and no operating system registration is needed. By registering the module, the application menu is also created.



One module should be registered, in respect to that category and finally the document. After that, the class library file should be added to the document record.

C# - the document could be a DLL. C# interface can also load VFP exe documents.

Each module has categories and documents. Finally the module will translate into a menu item and the documents into menu pads under an item. Categories are here just for the better management of added documents.



System, Admin, Edit, Help and Exit, are automatically added to the main menu.
F1 for help - a web based help is opened into a small web browser window based on IE controls.

2.5 Customers module - tutorial

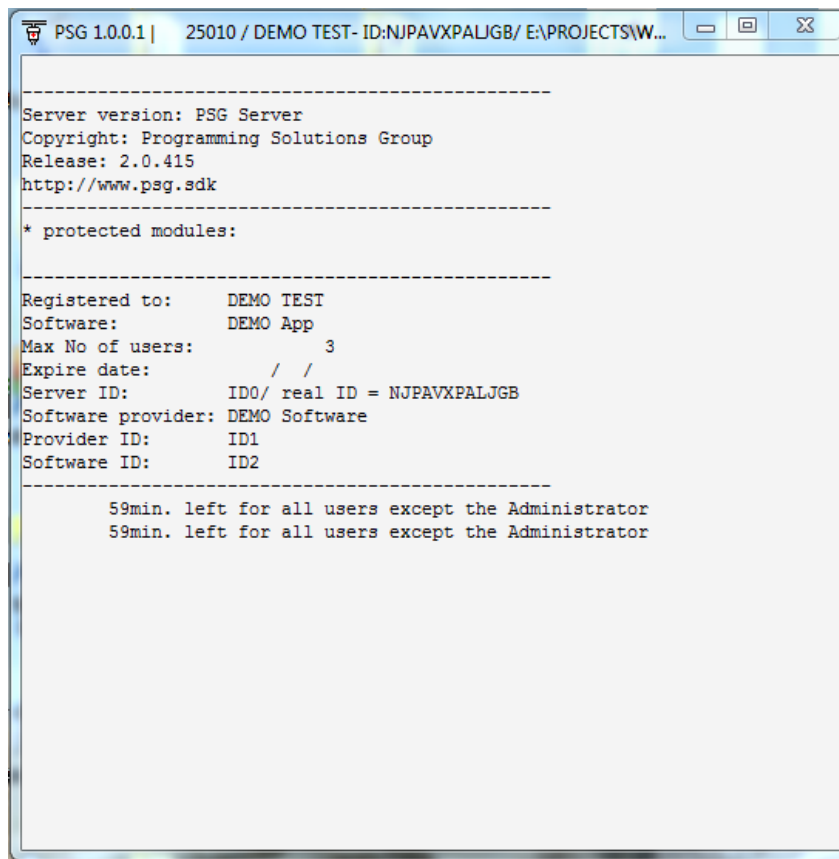
Creating a PSG document in C # is simple (structured steps) and quick (20-60 min).

Prepare the working place:

An instance of PSG server should be opened. It is better to have the PSG DEMO for C# where the Northwind demo application is used.

The PSG server can run as a service in production and it is preferable to have it run as an application for development.

Open the psg.exe in the server director. By default, the server has a demo license that allows three users for one hour, after that only the administrator can use the server.



```
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
```

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

The steps used to create a document:

1. [Creating tables in the database](#)
2. [Adding web services](#) (application config.exe)
3. Create project WinForms
 - Add the dataset (main table of document) : CustomersBE.xsd
 - Add the main form with all those controls: Customers.cs
 - Add global variables, constructors and methods based
 - Set specific properties of PSG controls

2.5.1 Database and web services

a) Scripts (PostgreSQL) for creating customer tables:

```
CREATE TABLE customers
(
  customerid character(50) NOT NULL,
  companyname character(40),
  contactname character(30),
  contacttitle character(30),
  address character(60),
  city character(15),
  region character(50),
  postalcode character(10),
  country character(15),
  phone character(24),
  fax character(24),
  CONSTRAINT p PRIMARY KEY (customerid)
)
```

b) Web services:

Details to create web services (click [here](#) for more).

```
PARAMETERS re_id, temp_file, loginname

.connect(server.main_connection)
.sqlcon.cname = 'customers'
.sqlcon.sqlcommand = "Select * from customers where customerid = ?m.id or ?m.id = 'null' "

RETURN .returnsql(temp_file)
```

2.5.2 Create project WinForms

Open Microsoft Visual Studio
Choose the new project -> Visual c# -> Windows -> Windows Forms Application.
Create a new project named "Customers". Must check the option: "Create directory for solution".

Rename file Form1 in Customers (rename class Form1 in Customers).
Add new file dataset (CustomersBE.xsd , BE- business entity), this will be the main table of the project.

Add DataTable "Customers".

Get the "psgbase.exe" file from our site (Downloads -> and then add reference PsgBase.

2.5.3 Main form file

- a) Add `using PsgBase;` in main form "Customers.cs".

Use base class "Base". If your project has details, you must use base class "BaseDetails".

```
public partial class Customers : Base
{
    ...
}
```

- b) The main form must have the "EditorClass" attribute and depending on the case, you must use one of the: NoView, ViewGrid, ViewTree, ViewTreeList attributes.

You should use the NoView attribute when your document doesn'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 Customers : Base
```

- c) Add global variables:

```
#region GlobalVariables

CustomersBE.CustomersDataTable table = new CustomersBE.CustomersDataTable();
CustomersBE.CustomersRow rowA = null;

#endregion GlobalVariables
```

- d) Add constructors:

```
#region Constructors

public Customers()
{
    InitializeComponent();
}
```

```

    }

    public Customers(ViewGrid _view) :base(_view)
    {
        InitializeComponent();
    }

    public Customers(LookupGrid _lookup) :base(_lookup)
    {
        InitializeComponent();
    }

    #endregion Constructors

```

2.5.4 First loading form

Open form Orders.cs

a) Add context:

```

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

    context.PsgFieldKey = "customerid";
    context.PsgFieldDisplay = "companyname";
    context.PsgCommand = "customers";
    context.PsgServerTableName = "customers";
}

```

b) Get tables from server:

```

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

    AddTable("customers", ID);
    GetTables(this);
}

```

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

Please check [web services](#) for details.

The GETDATA object is dealing with server communication - please check the SDK help file for more information.

c) Use data from server:

PsgData() is triggered after the data archive was loaded from the server and needs to be used

```

public override void PsgData(DataTable dt)

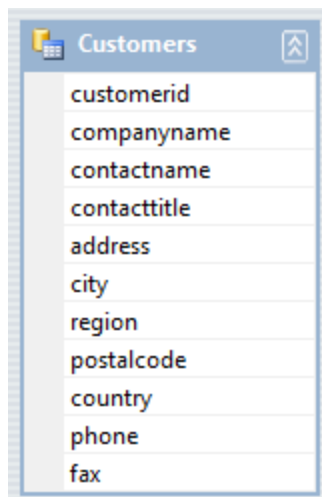
```

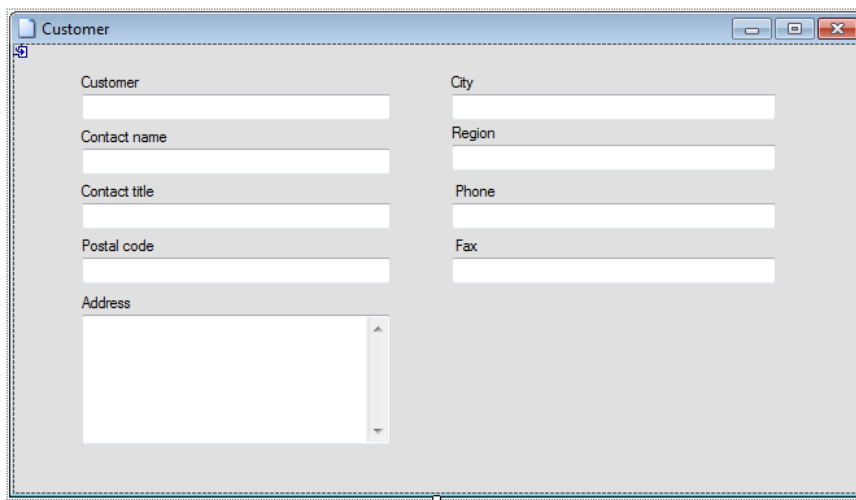
```
        {  
            base.PsgData(table);  
            switch (this.alias_name.ToUpper())  
            {  
                case "CUSTOMERS":  
                    table = (CustomersBE.CustomersDataTable)PsgFillTableCurrent(dt,  
table);  
                    rowA = (CustomersBE.CustomersRow)table.Rows[0];  
                    break;  
            }  
        }  
    }
```

2.5.5 Adding controls to form

Add columns (CustomersBE.xsd dataset) and controls (Customers.cs form).

- Open the 'CustomersBE.xsd' dataset and start adding columns to it.
- Open the 'Customers.cs' form and start adding controls to it.
It is a good idea not to add all controls at once:
 - add some controls
 - [publish](#) the dll file and open the client to check if it's working





Each control has PSG properties. We'll set the following properties in the designer mode:

PsgServerTableName - customers
PsgFieldKey – customersid
PsgFieldName - contactname
PsgFieldType – C

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

```
public override void PsgBind()
{
    BindEdit(edCustomer, table.companynameColumn);
    BindEdit(edContactNam, table.contactnameColumn);
    BindEdit(edContactTitle, table.contacttitleColumn);
    BindEdit(edAddress, table.addressColumn);
    BindEdit(edCity, table.cityColumn);
    BindEdit(edRegion, table.regionColumn);
    BindEdit(edPostalCode, table.postalcodeColumn);
    BindEdit(edPhone, table.phoneColumn);
    BindEdit(edFax, table.faxColumn);

    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

2.5.6 View / Lookup

Each document can have one form of view / lookup.

Columns and filters have settings in the code source file of the document - method [PsgView / PsgLookup](#)

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

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

    this.ColumnsView(table.CustomerColumn, "Customer");
    this.ColumnsView(table.EmployeeColumn, "Employee");

    this.FiltersView(table.CustomerColumn, "Customer");
}

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

    this.ColumnsLookup(table.CustomerColumn, "Customer");
    this.ColumnsLookup(table.EmployeeColumn, "Employee");

    this.FiltersLookup(table.CustomerColumn, "Customer");
}
```

2.5.7 Run the module

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

The module must be registered on the server (for details click [here](#))

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

2.5.8 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, so please check the psg sdk.com .

Other articles and information could be found on the website.

2.6 Orders module - tutorial

It is the same as the previous " Customers " module.

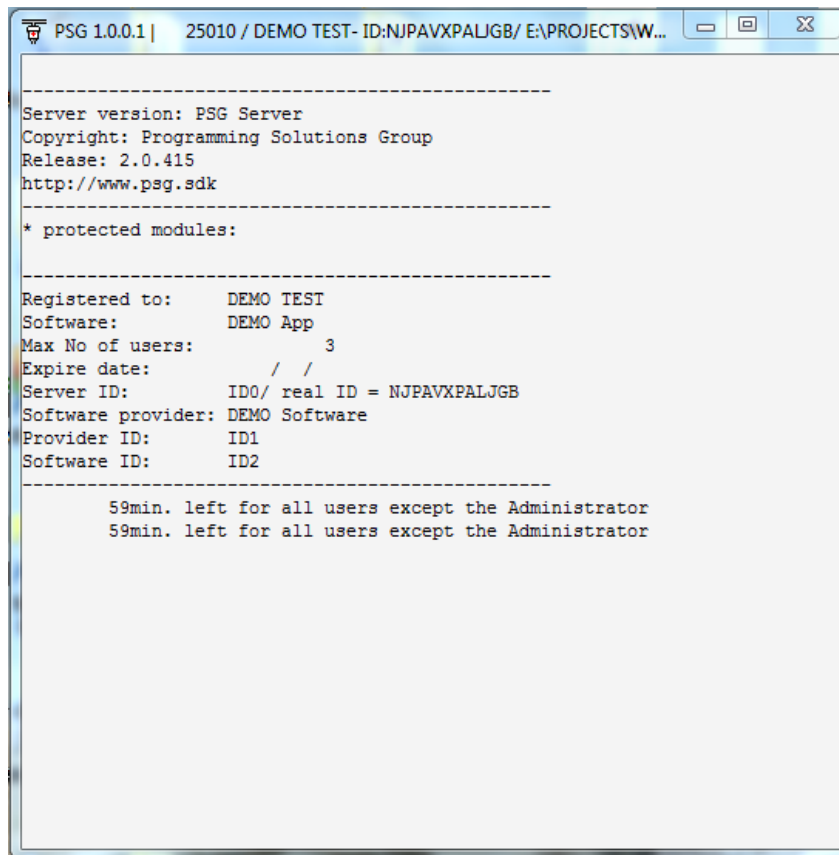
Creating a PSG document in C # is simple (structured steps) and quick (20-60 min).

Prepare the working place:

A instance of PSG server should be opened – it is better to have the PSG DEMO for C# where the Northwind demo application is used.

The PSG server can run as a service in production - preferable to have it run as an application for development.

Open the psg.exe in the server directory. By default, the server has a demo license that allows three users for one hour. After that, only the administrator can use the server.

A screenshot of a Windows application window titled "PSG 1.0.0.1 | 25010 / DEMO TEST- ID:NJPAVXPALJGB/ E:\PROJECTS\W...". The window contains a text area with the following text:

```
-----
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
```

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

The steps used to create a document:

1. [Creating tables in the database](#)
2. [Adding web services](#) (application config.exe)

3. Create project WinForms

- Add the dataset (main table of document) : OrdersBE.xsd
- Add the main form with all those controls: Orders.cs
- Add global variables, constructors and methods based
- Set specific properties of PSG controls

2.6.1 Database and web services

a) Scripts (PostgreSQL) for create tables orders and orderdetails:

```
CREATE TABLE orders
(
  orderid character(12) NOT NULL,
  customerid character(12),
  employeeid character(12),
  orderdate date,
  requiredate date,
  shippeddate date,
  shipperid character(12),
  freight numeric(12,2),
  shipname character(40),
  shipaddress character(60),
  shipcity character(15),
  shipregion character(15),
  shippostalcode character(10),
  shipcountry character(15),
  sid character(10),
  mdl timestamp without time zone,
  CONSTRAINT orderid PRIMARY KEY (orderid)
)
```

```
CREATE TABLE orderdetails
(
  detailid character(12) NOT NULL,
  orderid character(12),
  productid character(12),
  quantity integer,
  discount double precision,
  unitprice double precision,
  sid character(10),
  mdl timestamp without time zone,
  CONSTRAINT detailid PRIMARY KEY (detailid)
)
```

b) Web services:

Details for creating a web service (click [here](#) for more).

```
PARAMETERS re_, id, temp_file, loginname
.connect(server.main_connection)
.sqlcon.cname = 'orders'
.sqlcon.sqlcommand = .getsqlstring('orders')
RETURN .returnsql(temp_file)
```

```
PARAMETERS re_id, temp_file, loginname
.connect(server.main_connection)
.sqlcon.cname = 'orderdetails'
.sqlcon.sqlcommand = "select * from orderdetails where orderid = ?m.id"
RETURN .returnsql(temp_file)
```

In Sql's list (config application) we must have the SQL statement "orders":

```
Select o.*, c.companyname as customer, e.lastname as employee, s.companyname as shipper from orders o
left join customers c on o.customerid = c.customerid left join employees e on o.employeeid = e.employeeid left
join shippers s on o.shipperid = s.shipperid where orderid = ?m.id or ?m.id = 'null'
```

2.6.2 Create project WinForms

Open Microsoft Visual Studio
 Choose the new project -> Visual c# -> Windows -> Windows Forms Application.
 Create a new project named "Orders". Must check the option named: "Create directory for solution".

Add a new file dataset (OrdersBE.xsd), this will be main table of project.
 Add DataTable "Orders".

Get the "psgbase.exe" file from our site and then add the PsgBase reference.

2.6.3 Main form file

a) Add `using PsgBase;` in the main form "Orders.cs".

Use base class "BaseDetails". If your project does not have details, you should use the Base class.

```
public partial class Orders : BaseDetails
{
    ...
}
```

b) The main form must have the "EditorClass" attribute and depending on the case, you must use one of the: NoView, ViewGrid, ViewTree, ViewTreeList attributes.

You should use the NoView attribute when your document doesn'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 Orders : BaseDetails
```

c) Add global variables:

```
#region GlobalVariables

OrdersBE.OrdersDataTable table = new OrdersBE.OrdersDataTable();
OrdersBE.OrdersRow rowA = null;

#endregion GlobalVariables
```

d) Add constructors:

```
#region Constructors

public Orders()
{
    InitializeComponent();
}

public Orders(ViewGrid _view) :base(_view)
{
    InitializeComponent();
}

public Orders(LookupGrid _lookup) :base(_lookup)
{
    InitializeComponent();
}

#endregion Constructors
```

2.6.4 First loading form

Open form Orders.cs

a) Add context:

```

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

    context.PsgFieldKey = "orderid";
    context.PsgFieldDisplay = "customer";
    context.PsgCommand = "orders";
    context.PsgServerTableName = "orders";
    context.PsgColumnSort = "orderdate";
    context.PsgSortOrder = SortOrder.Descending;
}

```

b) Get tables from server:

```

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

    AddTable("orders", ID);
    AddTable("employees", Utils.Null);
    AddTable("customers", Utils.Null);
    AddTable("shippers", Utils.Null);
    AddTable("products", Utils.Null);
    AddTable("orderdetails", ID);
    GetTables(this);
}

```

For each requested table, a corresponding web service should exist on the server. Please check [web services](#) for details.

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

c) Use data from server:

PsgData() is triggered after the data archive was loaded from the server and needs to be used

```

public override void PsgData(DataTable dt)
{
    base.PsgData(table);
    switch (this.alias_name.ToUpper())
    {
        case "CUSTOMERS":
            cbCustomer.psgCMTable = this.Intf.TableServer;
            cbCustomer.DataSource = cbCustomer.psgCMTable;
            cbCustomer.DisplayMember = "companyname";
            cbCustomer.ValueMember = "customerid";
            break;
        case "EMPLOYEES":
            cbEmployee.psgCMTable = this.Intf.TableServer;
            cbEmployee.DataSource = cbEmployee.psgCMTable;
            cbEmployee.DisplayMember = "lastname";
            cbEmployee.ValueMember = "employeeid";

```

```
        break;
    case "SHIPPERS":
        cbShipper.psgCMTable = this.Intf.TableServer;
        cbShipper.DataSource = cbShipper.psgCMTable;
        cbShipper.DisplayMember = "companyname";
        cbShipper.ValueMember = "shipperid";
        break;
    case "ORDERDETAILS":
        gridDetaili.DataSource = this.Intf.TableServer;
        tableDServer = this.Intf.TableServer;
        break;
    case "PRODUCTS":
        dtProducts = this.Intf.TableServer;
        productid.DataSource = dtProducts;
        productid.DisplayMember = "productname";
        productid.ValueMember = "productid";
        break;
    case "ORDERS":
        table = (OrdersBE.OrdersDataTable)PsgFillTableCurrent(dt, table);
        rowA = (OrdersBE.OrdersRow)table.Rows[0];
        break;
    }
}
```

2.6.5 Adding controls to form

Add columns (OrdersBE.xsd dataset) and controls (Orders.cs form).

- Open the 'OrdersBE.xsd' dataset and start adding columns to it.
- Open the 'Orders.cs' form and start adding controls to it.
It is a good idea not to add all controls at once:
 - add some controls
 - [publish](#) the .dll file and open the client to check if it's working

The image displays two screenshots from a Visual Studio application. The top screenshot shows a list of fields for the 'Orders' table, including OrderID, CustomerID, Customer, EmployeeID, Employee, OrderDate, RequireDate, ShippedDate, ShipperID, Shipper, Freight, ShipName, ShipAddress, ShipCity, ShipRegion, ShipPostalCode, and ShipCountry. The bottom screenshot shows the 'Order' form in the application, which includes dropdowns for Customer and Employee, date pickers for Orders date, Require date, and Shipped date, and text boxes for Shipper, Ship name, Ship address, Freight, Ship city, Ship region, Ship postal code, and Ship country. Below the form is a table with columns Product, Unit price, Quantity, and Discount, and a Total field at the bottom.

Each control has PSG properties. We'll set the properties in the designer mode:

PsgServerTableName - orders
PsgFieldKey - orderid
PsgFieldName - customerid
PsgFieldType - C

OBSERVATION: The following properties are required for the control type - ComboEdit:

PsgFileName - customer.dll
PsgCMTableName – customers
PsgCMFieldKey - customerid
PsgCMFieldDisplay – companyname

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

```
public override void PsgBind()
{
    BindCombo(cbCustomer, table.CustomerColumn, table.CustomerIDColumn);
    BindCombo(cbEmployee, table.EmployeeColumn, table.EmployeeIDColumn);
    BindCombo(cbShipper, table.ShipperColumn, table.ShipperIDColumn);
    BindEdit(edFreight, table.FreightColumn);
    BindEdit(edShipName, table.ShipNameColumn);
    BindEdit(edShipAddress, table.ShipAddressColumn);
    BindEdit(edShipCity, table.ShipCityColumn);
    BindEdit(edShipRegion, table.ShipRegionColumn);
    BindEdit(edShipPostalCode, table.ShipPostalCodeColumn);
    BindEdit(edShipCountry, table.ShipCountryColumn);
    BindData(dtOrderDate, table.OrderDateColumn);
    BindData(dtRequireDate, table.RequireDateColumn);
    BindData(dtShippedDate, table.ShippedDateColumn);

    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

2.6.6 View / Lookup

Each document can have one form of view / lookup.

Columns and filters have the settings in the code source file of document - method [PsgView / PsgLookup](#)

When we run a document, its first form is the view form, then we can use standard commands: add,

modify, delete.

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

    this.ColumnsView(table.CustomerColumn, "Customer");
    this.ColumnsView(table.EmployeeColumn, "Employee");

    this.FiltersView(table.CustomerColumn, "Customer");
}

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

    this.ColumnsLookup(table.CustomerColumn, "Customer", "customers",
"companyname");
    this.ColumnsLookup(table.EmployeeColumn, "Employee");

    this.FiltersLookup(table.CustomerColumn, "Customer");
}
```

2.6.7 Run the module

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

The module must be registered on the server (for details click [here](#))

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

2.6.8 Final

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

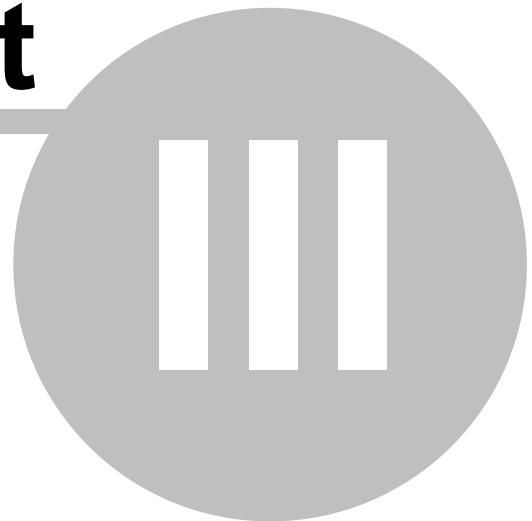
Help and technical support from PSGSDK is available by email, so please check the psg-sdk.com .

Other articles and information can be found on the website.

Top Level Intro

This page is printed before a
new top-level chapter starts

Part



3 Using DataGridView

We have two solutions when it comes to using the psgDataGridView control:

- I. Columns are generated
- II. Columns are defined

I. Columns are generated

In this situation we should follow the next steps:

1. Declare a variable of the DataTable type:

```
DataTable dtProducts = new DataTable();
```

2. Populate this variable and control datasource psgDataGridView in the PsgData method

```
case "PRODUCTS":  
    dtProducts = this.intf.TableServer;  
    dgvProducts.DataSource = dtProducts;  
    break;
```

3. Populate properties of control dgvProducts

- PsgFieldKey : productid
- PsgServerTableName : products

In this moment, all the modification over the grid will be sent to the database server.

Events for the add and delete commands on the grid

```
private void btAdd_Click(object sender, EventArgs e)  
{  
    DataRow row = dtProducts.NewRow();  
    row["productid"] = Sys2015;  
  
    dtProducts.Rows.Add(row);  
    dgvProducts.DataSource = dtProducts;  
  
    //send row on database server  
    psgInsert(row, "products", false);  
}  
  
private void btDelete_Click(object sender, EventArgs e)  
{  
    if (Utils.ConfirmDelete())  
    {  
        DataRow[] rows = dtProducts.Select(" id = '" + dgvProducts.PsgValueKey + "'");  
        if (rows == null && rows.Length == 0)  
            return;  
  
        dtProducts.Rows.Remove(rows[0]);  
        dgvProducts.DataSource = dtProducts;  
        psgDelete(dgvProducts.PsgServerTableName, dgvProducts.PsgValueKey, false);  
    }  
}
```

II. Columns are defined

In this situation we should follow the next steps:

1. Declare a variable of the DataTable type:

```
DataTable dtProducts = new DataTable();
```

2. Set the psgDataGridView control so that it won't generate columns (Load event)

```
dgvProducts.AutoGenerateColumns = false;
```

3. Populate this variable and control datasource psgDataGridView in the PsgData method

```
case "PRODUCTS":
    dtProducts = this.intf.TableServer;
    dgvProducts.DataSource = dtProducts;
    break;
```

3. Populate the control properties dgvProducts

```
- PsgFieldKey : productid
- PsgServerTableName : products
```

4. Define then columns of the grid. Columns must have the "DataPropertyName" property, set to one column from the database. We need one column to be the primary key (productid) with the "DataPropertyName" property set as "productid".

At this moment all the modifications on the the grid will be sent to the database server.

Events for the add and delete commands on the grid

```
private void btAdd_Click(object sender, EventArgs e)
{
    DataRow row = dtProducts.NewRow();
    row["productid"] = Sys2015;

    dtProducts.Rows.Add(row);
    dgvProducts.DataSource = dtProducts;

    //send row on database server
    psgInsert(row, "products", false);
}

private void btDelete_Click(object sender, EventArgs e)
{
    if (Utils.ConfirmDelete())
    {
        DataRow[] rows = dtProducts.Select(" id = '" + dgvProducts.PsgValueKey + "'");
        if (rows == null && rows.Length == 0)
            return;

        dtProducts.Rows.Remove(rows[0]);
        dgvProducts.DataSource = dtProducts;
        psgDelete(dgvProducts.PsgServerTableName, dgvProducts.PsgValueKey, false);
    }
}
```

```
}  
}
```

Top Level Intro

This page is printed before a
new top-level chapter starts

Part



IV

4 Other model documents

We can have documents without view / lookup or without loading data with standard PSG methods.

1. Document doesn't need a view and the data is being loaded using standard PSG methods
2. Document doesn't need a view and the data is being loaded without using standard PSG methods

The document doesn't need a view and **data is being loaded using the standard PSG methods** (PsgStart, PsgData, PsgBind)

- the NoView attribute is used in the editor form
- use the methods: PsgStart, PsgData, PsgBind for data loading

```
namespace Entity
{
    [EditorClass, NoView]
    public partial class Entity : Base
    {
        DataTable table = null;

        public Entity()
        {
            InitializeComponent();
        }

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

            context.PsgFieldKey = "entityid";
            context.PsgCommand = "entity";
        }

        public override void PsgStart()
        {
            base.PsgStart();
            AddTable("entity", ID);
            GetTables(this);
        }

        public override void PsgData(DataTable dt)
        {
            base.PsgData(dt);
            switch (this.alias_name.ToUpper())
            {
                case "ENTITY":
                    table = this.Intf.TableServer;
                    grid.DataSource = table;
                    break;
            }
        }

        public override void PsgBind()
        {
            base.PsgBind();
            ...
        }
    }
}
```

```
}
}
```

The document doesn't need a view and the **data is being loaded without using standard PSG methods** PsgStart, PsgData, PsgBind

- do not use the view attribute in editor form
- use AddTable and GetTables to call the data from the server
- use the "Accept" method to receive data

```
namespace Entity
{
    [EditorClass]
    public partial class Entity1 : Base
    {
        DataTable dt = null;

        public Entity()
        {
            InitializeComponent();
        }

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

            context.PsgFieldKey = "entityid";
            context.PsgCommand = "entity";
        }

        private void psgButton1_Click(object sender, EventArgs e)
        {
            base.PsgStart();
            AddTable("entity");
            GetTables(this);
        }

        public override int Accept(string result_command)
        {
            base.Accept(result_command);

            switch (result_command.Split('#')[0])
            {
                case "DOWNLOADDONE":
                    dt = this.Intf.TableServer;
                    grid.DataSource = dt;
                    break;
            }

            return 1;
        }
    }
}
```

Endnotes 2... (after index)

Back Cover