

Px Framework v.1.33.01 for Delphi Programers to Asp.Net application

Px Framework v.1.33 for ASP.NET

Components for fast work of a database

TABLE OF CONTENTS

1. Basic components of Px Framework (Px Controls), a brief description of the programming and work with them

- 1.1. **PxWebQuery** – component for working with databases, the basic description
- 1.2. **PxWebQuery** - set up of the PxWebQuery component relations with the other tables
- 1.3. **PxSuperGrid** – component for the direct display of data in the table
- 1.4. **PxEdit** – component for data editing, similar to the TextBox component
- 1.5. **PxComboBox** – component for selecting data from a list, similar to the DropDownList component
- 1.6. **PxCheckBox** – component for checking the value (Check / UnCheck value)
- 1.7. **PxFlyComboBox** – set of the consecutively linked comboboxes, suitable for the work with structured data (for example: selection of category and subcategory)
- 1.8. **PxGreatRepeater** – component for entering data with repeating structure, maximum number of values is limited
- 1.9. **PxDBNavigator** – the component for the work with the PxWebQuery components, row cursor movement, etc.
- 1.10. **PxJSDatePicker** – component for the date entry, based on the JavaScript
- 1.11. **PxDatePicker** – component of the award date
- 1.12. **PxMemo** – component used for the direct data editing, the extension of the PxEdit component
- 1.13. **PxLabel** – component for data display
- 1.14. **PxCheckBoxList** – component to view and select values from the list
- 1.15. **PxRadioButtonList** – component for view and selection of a value from the list
- 1.16. **PxChart** - the component for displaying and working with charts
- 1.17. **PxFilterView** - visual component for filtering the table data contents in the PxWebQuery component
- 1.18. **PxUploader** - component for uploading binary and text files to the server
- 1.19. **PxLogin** – component for authorization and logging into the application

2. Installing Px Framework

- 2.1. Installation Px Framework on the client computer
- 2.2. Installing the Px Framework on the server

3. Detailed description of programming using the Px Framework components

3.1. The PxWebQuery – component for working with the Oracle database, the detailed description

- 3.1.1. Data loading from the Oracle database by means of the PxWebQuery components
- 3.1.2. Program inserting of a new row into the database by means of the PxWebQuery component
- 3.1.3. Program editing of the existing row in the database by means of the PxWebQuery component
- 3.1.4. Program deleting of the existing row in the database by means of the PxWebQuery component
- 3.1.5. Loading values from the PxWebQuery component via the while cycle
- 3.1.6. Row search in the PxWebQuery component according to the value entered and the

- name of the column where the search shall be carried out
- 3.1.7. Setting of the cursor position in the row in the PxWebQuery component
- 3.1.8. Selection of several rows that match the selection criteria
- 3.1.9. Events of the PxWebQuery component
- 3.1.10. Creation of a new dynamic column (field) in the PxWebQuery component table
- 3.1.11. The "ReOpen" procedure of the PxWebQuery component and data re-load into the PxWebQuery component
- 3.1.12. Validation, checking of the entered values by means of the PxWebQuery component and other visual components (PxEdit, PxComboBox, etc.)
- 3.1.13. Finding the current version of the Px Framework
- 3.1.14. Setting the language mutation of the Px Framework
- 3.1.15. Other AddParam... methods of the PxWebQuery component
 - 3.1.15.A. Method AddParamLink components PxWebQuery
 - 3.1.15.B. AddParamDynamicField method of the PxWebQuery component
- 3.1.16. Special GetValueFromKey function of the PxWebQuery component
- 3.1.17. Three ways of using the PxWebQuery component

3.2. Work of styles, cascading style sheets (CSS)

- 3.2.1. PxEdit – Work of styles, cascading style sheets (CSS)
- 3.2.2. PxJSDatePicker – Work of styles, cascading style sheets (CSS)
- 3.2.3. PxGreatRepeater – Work of styles, cascading style sheets (CSS)

4.1 Specification Px Framework deployment under different database platforms (Oracle, MS SQL, MySQL, FireBird, InterBase)

- 4.1.1. Specification Px Framework deployment under Oracle database
- 4.1.2. Specification Px Framework deployment under MS SQL database
- 4.1.3. Specification Px Framework deployment under Mysql database
- 4.1.4. Specification Px Framework deployment under FireBird database
- 4.1.5. Specification Px Framework deployment under InterBase database
- 4.1.6. Specification Px Framework deployed generally

5.1 Auxiliary components Px Frameworks

- 5.1.1. Component to retrieve data from database - SQLCoreQuery

1.1 PxWebQuery – component for working with databases, the basic description

The *PxWebQuery* component is a component that allows fast working with a database, loading data from the database, their display and editing (line input, editing and deleting) or searching and selection of data. The *PxWebQuery* component contains a *DataTable* object to which the data, defined in the *SQL command* are loaded.

The *PxWebQuery* component is defined in the *.aspx file as follows:

```
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>
```

If you want the Px Framework components to be visible in the aspx files, you have to define the "Prx" prefix in the header, as follows:

```
<%@ Register TagPrefix="Prx" Namespace="PxControls" Assembly="PxControls" %>
```

After this you can operate with the *PxWebQuery* component in the *.aspx.cs file as follows. First, enter the *ConnectionString* to the component and then the *SQL command*:

```
wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";  
wquAdresar.SQLSelect = "select idadresar, name, address, create_date from Adresar";
```

Then use the "Open" command to load data from the database to the component, and now you can connect it to the *PxSuperGrid* component, which displays the data retrieved.

Example No.1.

adresar.aspx.cs file:

```
protected void Page_Load(object sender, EventArgs e)  
{  
    if ((!IsPostBack)&&(wquAdresar.Active==false))  
    {  
        wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";  
        wquAdresar.SQLSelect = "select idadresar , name, address, create_date from Adresar";  
  
        wquAdresar.Open();  
    }  
    grdAdresar.PxDataSource = wquAdresar;  
    grdAdresar.DataBind();  
}
```

adresar.aspx file:

```
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>  
  
<Prx:PxSuperGrid ID="grdAdresar" runat="server"  
    PageSize="4"  
    PxEditFormName=""  
    PxInfoFormName=""
```

```
PxVisibleButtons="SIDE0">
</Px:PxSuperGrid>
```

By connecting to the *PxSuperGrid*, the *PxWebQuery* component automatically provides such functions as data entry, editing, deleting, sorting and paging.

In order to input new lines via the *PxWebQuery*, it is necessary to create a sequence from where the *PxWebQuery* component can download unique IDs for its primary key. Sequence name shall be created as follows.

SEQ_“primary key column name “

Therefore, when creating new tables, do not call the primary key column 'ID', but ID + "table name".


In this example, we also want data entry to the database to be functional, so let's create the sequence by using the SQL command:

```
CREATE SEQUENCE SEQ_IDADRESAR
START WITH 1
INCREMENT BY 1
MAXVALUE 1E18
NOMINVALUE
NOORDER
NOCACHE
NOCYCLE;
```

Attention !!! On other database platforms the sequence is created in a different way, for more information see Part Four "Specification of the Px Framework usage on different database platforms (Oracle, MS SQL, MySQL, Firebird, InterBase)..

...	ID	Persons Name	Juridical Form	Address
	2	Ján Kolenička	Juridical person	THK 33, Banská Bystrica, 974 11
✓✗	3	Karel Čapek	Juridical person	Šalgotarianska 22
	4	Helena Mociková	Juridical person	Rudná 2
			Physical person	

Page 1 [2]

...	ID	Persons Name	Address	Juridical Form	Date of Establishment
 	2	Ján Kolenička	THK 33, Banská Bystrica, 974 11	Juridical person	21. 9. 2010 0:00:00
 	3	Karel Čapek	Šalgotarianska 22	Juridical person	14. 10. 2010 0:00:00
 	4	Helena Mociková	Rudná 2	Juridical person	29. 7. 2010 0:00:00
ADD NEW ROW...					

Page 1 [2]

1.2. PxWebQuery - set up of the PxWebQuery component relations with the other tables

1.2.1. AddParamKey – relating the simple classifier to the main table

AddParamKey serves for connection of the table and the classifier. The *AddParamKey* method is the *PxWebQuery* component method. According to the Px Framework philosophy as much information as possible is to be defined on the *PxWebQuery* component. This actually saves the programming and writing in case of the *PxSuperGrid*, *PxEdit*, *PxComboBox* and other components. All these components borrow data and settings from the *PxWebQuery* component. Therefore, the *PxWebQuery* component has a number of procedures and methods, which are used for defining various display methods and relations between the tables.

So the *AddParamKey* method is the simplest method for defining the relation between the main table and the classifier.

This definition can be written to the *Adresar.aspx.cs* file.

Example No.1.

```
protected void Page_Load(object sender, EventArgs e)
{
    if ((!IsPostBack)&&(wquAdresar.Active==false))
    {
        string sSQLText1 = "select IDPravForm as Key, Name as Value from PravForm Order By Value";

        wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";
        wquAdresar.SQLSelect = "select idadresar , name, address, idpravnaforma, create_date from Adresar";

        wquAdresar.AddParamKey("idpravnaforma", "Key", "Value", "Value", "Key", sSQLText1);

        wquAdresar.Open();

        wquAdresar.Columns["idadresar"].Caption = "ID";
        wquAdresar.Columns["name"].Caption = "Persons Name";
        wquAdresar.Columns["idpravnaforma"].Caption = "Juridical Form";
        wquAdresar.Columns["address"].Caption = "Address";
        wquAdresar.Columns["create_date"].Caption = "Date of Establishment";
    }
    else
    {
        grdAdresar.PxDataSource = wquAdresar;
        grdAdresar.DataBind();
    }
}
```

Description of the parameters of the *AddParamKey* method:

C# syntax:

```
public void AddParamKey(string aFieldName, string aFieldNameKey, string aFieldNameValue,
    string aFieldToView, string aFieldToDB,
    string aSQLText);
```

Description of the parameters:

aFieldName – name of the column in the table to which the *AddParamKey* method is applied

aFieldNameKey – name of the classifier's column, which represents the classifier's key (the name of the column can be obtained from the SQL command, which is entered to the *aSQLText* parameter)

aFieldNameValue – name of the classifier's column, which represents the classifier's value (the name of the column can be obtained from the SQL command, which is entered to the *aSQLText* parameter)














aFieldToView – is used to enter the name of the column which value shall be displayed in the Grid

(*PxSuperGrid*)














aFieldToDB - is used to enter the name of the column which value shall be put in the database

aSQLText – is used to define the SQL command for loading the classifier, individual columns are to be appropriately named as Key and Value, for better clarity

The *PxSuperGrid* component before applying the *AddParamKey* method:

...	ID	Persons Name	Address	Juridical Form	Date of Establishment
  	2	Ján Kolenička	THK 33, Banská Bystrica, 974 11	1	21. 9. 2010 0:00:00
  	3	Karel Čapek	Šalgotarianska 22	1	14. 10. 2010 0:00:00
  	4	Helena Mociková	Rudná 2	1	29. 7. 2010 0:00:00
  	35	Jurij Brezan	Moskovská 2	1	22. 10. 2010 0:00:00
 ADD NEW ROW...					
Page 1 [2]					

The *PxSuperGrid* component after applying the *AddParamKey* method:

...	ID	Persons Name	Address	Juridical Form	Date of Establishment
  	2	Ján Kolenička	THK 33, Banská Bystrica, 974 11	Juridical person	21. 9. 2010 0:00:00
  	3	Karel Čapek	Šalgotarianska 22	Juridical person	14. 10. 2010 0:00:00
  	4	Helena Mociková	Rudná 2	Juridical person	29. 7. 2010 0:00:00
  	35	Jurij Brezan	Moskovská 2	Juridical person	22. 10. 2010 0:00:00
 ADD NEW ROW...					
Page 1 [2]					

After the *AddParamKey* method was applied to the „idpravnaforma“ column, the key in this table was replaced by the value from the classifier defined by the SQL command in the *aSQLText* parameter in the *AddParamKey* method. The example contains more details on definition of the SQL command, which was used for replacing the key integral value by the text value from the classifier.

Another functionality of the PxSuperGrid component in case of the *AddParamKey* parameter entry enables automatic replacement of the *TextBox* component by the *DropDownList* component during the editing operation in the grid. The picture below shows that the column, to which the *AddParamKey* parameter is applied, is replaced by the *DropDownList*, with the pre-loaded values, which can be chosen and which are, after the confirmation, automatically saved to the database.

...	ID	Persons Name	Juridical Form	Address
	2	Ján Kolenička	Juridical person	THK 33, Banská Bystrica, 974 11
✓ ✕	3	Karel Čapek	<div> Juridical person ▼ Juridical person Physical person </div>	Šalgotarianska 22
	4	Helena Mociková		Hudná 2

Page 1 [2]

Other parameters such as *AddParamWebQuery*, *AddParamFlyComboBox*, *AddParamGreatRepeater*, etc. are described in the Part II of this reference manual.

1.2.2. AddParamWebQuery – relating the simple classifier to the main table

AddParamWebQuery method is similar to the AddParamKey method, where the PxWebQuery component serves as the source of data. AddParamWebQuery method is suitable in case the classifier contains data that are changing dynamically. If the classifier contains data, which do not change for a long time, it is suitable to use the AddParamKey method for relating the main table to the classifier.

Description of the parameters of the *AddParamWebQuery* method :

C# syntax:

```
public void AddParamWebQuery(string aFieldName,
                             string aFieldNameKey, string aFieldNameValue,
                             string aFieldToView, string aFieldToDB,
                             string aPxWebQueryName)
```

Description of the parameters:

aFieldName – name of the column in the table to which the AddParamWebQuery is applied

aFieldNameKey – name of the classifier's column, which represents the classifier's key (the name of the column can be obtained from the SQL command, which is entered to the PxWebQuery component, from which the classifier is loaded)

aFieldNameValue – name of the classifier's column, which represents the classifier's value (the name of the column can be obtained from the SQL command, which is entered to the PxWebQuery component, from which the classifier is loaded)

aFieldToView – is used to enter the name of the column which value shall be displayed in the Grid (*PxSuperGrid*)

aFieldToDB - is used to enter the name of the column which value shall be put in the database




aPxWebQueryName – name for the PxWebQuery component where the classifier is loaded via the SQL command, individual columns are to be appropriately named as „Key“ and „Value“, for better clarity.

```
wquAdresar.AddParamWebQuery("idpravnaforma", "Key", "Value", "Value", "Key", "wquPravForm");
```

1.3. PxSuperGrid – component for the direct display of data in the table

PxSuperGrid component serves for the direct display of data in the table by means of the PxWebQuery components. Using the PxSuperGrid components it can operate data as follows: You can insert new rows, edit or delete already existing rows. Further, the component enables automatic sorting and paging.

Functional description of the PxSuperGrid buttons :

-  – this button is used to edit an existing row (E)
-  – this button is used to delete an existing row (D)
-  – this button is used for selection, row selection (S)
- this button is used to open the information form, it is mainly used to view details of the row

 **ADD NEW ROW...** - this button is used to add a new row

...	ID	Persons Name
 	2	Ján Kolenička
 	3	Karel Čapek
 ADD NEW ROW...		
Page 1 [2] [3]		
   		

The picture below displays editing by means of the PxSuperGrid component.

...	ID	Persons Name	Address
	2	Ján Kolenička	THK 33, Banská Bystrica, 974 11
 	<input type="text" value="3"/>	<input type="text" value="Karel Čapek"/>	<input type="text" value="Šalgotarianska 22"/>
Page 1 [2] [3]			

The following example shows how to use the PxSuperGrid. First, load data from the to the PxWebQuery component, which is opened by the “Open()“ command and then bind this component, connect it to the PxSuperGrid component.

Example No.2.

adresar.aspx.cs file:

```
protected void Page_Load(object sender, EventArgs e)
{
    if ((!IsPostBack)&&(wquAdresar.Active==false))
    {
        wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";
        wquAdresar.SQLSelect = @"select idadresar , name, address, create_date from
        Adresar";

        wquAdresar.Open();
    }
    grdAdresar.PxDataSource = wquAdresar;
    grdAdresar.DataBind();
}
```

adresar.aspx file:

```
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>

<Prx:PxSuperGrid ID="grdAdresar" runat="server"
    PageSize="4"
    PxVisibleButtons="SIDE0">
</Prx:PxSuperGrid>
```

To change the caption in the PxSuperGrid(Title) table, write the following code in the *.aspx.cs file, assigned to the PxWebQuery component, property Caption.

```
wquAdresar.Columns["NAME"].Caption = "Persons Name";
```

To change the display of buttons in the *PxSuperGrid* (make them visible or hide them), use the *PxVisibleButtons* property of the *PxSuperGrid* component.

```
grdAdresar.PxVisibleButtons="SIDE0";
```

The further example describes the legend according to which the individual buttons are displayed. If there is any letter there, the relevant button is displayed in the PxSuperGrid.

```
S – select, row selection
I – insert or enter a new row
E – editing an existing row
D – deleting an existing row
O - open information form
```

If you do not want to edit or insert a new row directly in the *PxSuperGrid* component, type the path or the name of the form where the editing shall be carried out to the *PxEditFormName* property.

```
grdAdresar.PxEditFormName="AddAdresar.aspx";
```





This property can also be defined in the Adresar.aspx file in the body of the PxSuperGrid's definition.

```
<Prx:PxSuperGrid ID="grdAdresar" runat="server"
    PageSize="4"
    PxEditFormName="AddAdresar.aspx"
    PxInfoFormName="InfoAdresar.aspx"
    PxVisibleButtons="SIDE0">
</Prx:PxSuperGrid>
```

Pressing the "edit" or "add new row" button in the PxSuperGridu component shall open the AddAdresar.aspx form. The AddAdresar.aspx form can be combined from the PxEdit and PxComboBox editing components.

For more details see the description of the *PxEdit* and *PxComboBox* components below.

Description of the PxSuperGrid buttons:

-  – this button is used to edit an existing row
-  – this button is used to delete an existing row
-  – this button is used for selection, row selection
-  – this button is used to open the information form, it is mainly used to view details of the row



- this button is used to add a new row

Similarly, the form that will display only information about the row, that will show detail of the row, can be defined.

```
PxInfoFormName="InfoAdresar.aspx";
```

The InfoAdresar.aspx form, can be combined from the PxLabel components.

To display the selected row in the PxSuperGrid component, type the following line to the PxSuperGrid component:

```
<SelectedItemStyle BackColor="#ffddff" />
```

Thus the whole PxSuperGrid component definition entry in the Adresar.aspx file shall be as follows:

```

<Prx:PxSuperGrid ID="grdAdresar" runat="server"
    PageSize="4"
    PxEditFormName="AddAdresar.aspx"
    PxInfoFormName="InfoAdresar.aspx"
    PxVisibleButtons="SIDE0">
    <SelectedItemStyle BackColor="#ffddff" />
</Prx:PxSuperGrid>

```

To reduce the number of columns displayed in the Grid, use the VisibleFields property of the PxSuperGrid component. See the example below:

```
grdAdresar.VisibleFields = "idadresar;name;idpravnaforma;address;miesto;telefon";
```

Names of the columns, separated by a semicolon, that are to be displayed in the PxSuperGrid component were assigned to the VisibleFields property. If no value is assigned to the VisibleFields property, all columns (fields) shall be displayed in the PxSuperGrid component. Of course all columns in the PxSuperGrid component shall displayed exactly in the same order in which they were entered into the VisibleFields property. The VisibleFields property enables also to define the length of the column (by typing a colon after each column and the number of pixels that shall define the length of the column). See the example below:

```
grdAdresar.VisibleFields = "idadresar:50;name;idpravnaforma;address;miesto:100;telefon:150";
```

For the proper functioning of the VisibleFields property it shall be defined before connecting and binding to the PxSuperGrid component.

```

grdAdresar.VisibleFields = "idadresar;name;idpravnaforma;address;miesto;telefon";
grdAdresar.PxDataSource = wquAdresar;
grdAdresar.DataBind();

```

If you do not want to edit data in the PxSuperGrid component, set the ReadOnly property to „True“. See the example below:

```
grdAdresar.ReadOnly = True;
```











The PxSuperGrid component supports the view of the number of lines in the lower right corner. This display can be activated through the *VisibleRecordCount* property when it is set to “true”. Text can be changed through the property *CaptionRecordCount*.

```

grdAdresar.VisibleRecordCount = true;
grdAdresar.CaptionRecordCount = "Number of Records:";

```







The display of the number of rows in the PxSuperGrid component you can see in the picture below:

...	ID	Juridical form	Short name
 	1	Právnická osoba	PO
 	2	Physical Person	PO
 	3	Juridical Person	JP
 ADD NEW ROW...			
Page 1		Number of records: 3	
   			

If we want to enter the title of the PxSuperGrid component column (ToolTip), we can do this through the property *Header.ToolTip*. If we want to change the column name, we can do this through the property *Header.Text*. If we want to disable sorting for that column, we can use the *Header.NoSort* property and set it to “true”. Similarly, through the *Header.ReadOnly* property, we can disable editing the column in the PxSuperGrid component. Assignment of these properties must be done before the PxWebQuery component is assigned to the PxDataSource property.

```
grdAdresar.Header["name"].ToolTip = "Zadajte právnu formu !!!";
grdAdresar.Header["name"].NoSort = true;
grdAdresar.Header["name"].Text = "Právna Forma";
grdAdresar.Header["ID"].ReadOnly = true;
```

This picture shows an example of the assignment of the above mentioned properties.

...	ID	Juridical form	Short name
 	1	Právnická osoba	PO
	2	Physical Person	PO
	3	Juridical Person	JP
Page 1		Number of records: 3	
   			

The PxSuperGrid component contains the DateFormat property, by which we can control the date display in columns of the DateTime type in the PxSuperGrid component. If we want to show also the time next to the date in the grid, we use the mask „d.M.yyyy H:mm:ss“, which we enter into the DateFormat property.

```
grdTyp_Uloziska.DateFormat = "d.M.yyyy H:mm:ss";
```

If we want to display only date, we use the mask „dd.MM.yyyy“. If we want to edit the date in the grid, we better use the mask „dd.MM.yyyy“, because the automatic validation would stop data storing if we did not enter the correct date also with zeros. The example of data entering is shown below.

An example of the mask and then the input value in order to save the value correctly and avoid stopping the storing process through automatic validation.

```
"d.M.yyyy H:mm:ss"    -> 1.1.2011 0:00:00
"dd.MM.yyyy H:mm:ss" -> 01.01.2011 0:00:00
"dd.MM.yyyy"          -> 1.1.2011
```

The PxSuperGrid component (as well as PxEdit, PxComboBox, PxFlyComboBox, PxJSDatePicker, etc.) contains and supports automatic and complementary validation.

If the table column linked to the PxSuperGrid component is of the **string** type, validation of the string length is automatically carried out, if it is of the **int** type, validation operation to check whether the integer has been entered is carried out, in case of the **date** type, validation operation to check whether the valid date has been entered is carried out, etc. For more information about the automatic validation, see the Part II of this reference manual.

The picture below shows an example where we have entered a string which length exceeded the length of string defined in the database. Then we have entered an invalid date.



When move the cursor on the mouse over the icon with an exclamation mark, a tooltip displays an error message of the error that stopped deposition. In this case a correct date has not been entered.

During the automatic validation and control of the length string, validation may not work quite right.

It depends on the code page that is set in the database. If you use the database code page with UTF8 the characters with diacritics in the Oracle database occupy the 2 characters and some special characters to 3 characters in the database. Therefore, you must set the property of the CharacterCode component in PxWebQuery to "UTF8" so that the validation of the string length work properly. In the following we list costants of the CharacterCode property for the proper functionality of validation of the string length. The CharacterCode property is default set to ASCII.

N.	Constant CharacterCode
1	ASCII
2	UTF8
3	UTF16
4	UTF32
5	UTF7

If we set the code page to UTF8 in the database we should set it to the CharacterCode property before opening the PxWebQuery component. See examples below:

```
wquAdresar.CharacterCode = "UTF8";
```

1.4. PxEEdit – component for data editing, similar to the TextBox component

The PxEEdit component allows automatic editing of the table row column, which is loaded in the PxWebQuery component. Create the AddAdresar.aspx file, and type the following definition there:

AddAdresar.aspx file:

```
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>

<asp:Label ID="lblAddEditPanel" Font-Size="18pt" Font-Bold="true" runat="server"
    Text="Label"></asp:Label>
<br /><br />

<Prx:PxEdit ID="edtIDADRESAR" runat="server" AddTableRow="True" TableBegin="True" >
</Prx:PxEdit>
<Prx:PxEdit ID="edtName" runat="server" AddTableRow="True" ></Prx:PxEdit>
<Prx:PxEdit ID="edtADDRESS" runat="server" AddTableRow="True" ></Prx:PxEdit>
<Prx:PxComboBox ID="cmbPravnaForma" runat="server" AddTableRow="True" ></Prx:PxComboBox>
<Prx:PxJSDatePicker ID="edtCREATE_DATE" runat="server" AddTableRow="True" TableEnd="True">
</Prx:PxJSDatePicker>
<br />
<asp:Button ID="btnOK" runat="server" Text="Ok" Width="60" onclick="ButtonOk_Click" />
<asp:Button ID="btnStorno" runat="server" Text="Storno" Width="60" onclick="ButtonStorno_Click" />
```

In the AddAdresar.aspx.cs file, define linking of individual PxEEdit components to the PxWebQuery component and its linking to the specific table column. Then, this file contains definitions of the "Ok" and "Cancel" buttons methods. In case of the "Ok" button all changes in the PxEEdit components are confirmed by the Post() command, in case of the "Cancel" button, all changes are cancelled by the Cancel() command.

If both commands are successfully executed, the GoBackPreviousPage() command shall bring back the Adresar.aspx form. If data saving operation was not successful, the original form shall remain open and the reason of the saving operation failure shall be stated near the particular PxEEdit component. The PxEEdit component also supports automatic validation. For more information about the automatic validation see Part II of this reference manual.

AddAdresar.aspx.cs file:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (wquAdresar.StateQuery == PxWebQuery.eStateQuery.sqlInsert)
        lblAddEditPanel.Text = "Add new address";
    else lblAddEditPanel.Text = "Edit address";

    edtIDADRESAR.PxDataSource = wquAdresar;
    edtIDADRESAR.FieldName = "IDADRESAR";

    edtName.PxDataSource = wquAdresar;
    edtName.FieldName = "NAME";
```

```

edtADDRESS.PxDataSource = wquAdresar;
edtADDRESS.FieldName = "ADDRESS";

cmbPravnaForma.PxDataSource = wquAdresar;
cmbPravnaForma.FieldName = "idpravnaforma";

edtCREATE_DATE.PxDataSource = wquAdresar;
edtCREATE_DATE.FieldName = "CREATE_DATE";
}


protected void ButtonOk_Click(object sender, EventArgs e)
{
    wquAdresar.Post();
    wquAdresar.GoBackPreviousPage();
}

protected void ButtonStorno_Click(object sender, EventArgs e)
{
    wquAdresar.Cancel();
    wquAdresar.GoBackPreviousPage();
}

```

The resulting form for table row editing shall be as follows:

Edit Address


ID	<input type="text" value="35"/>
Persons Name	<input type="text" value="Jurij Brezan"/>
Address	<input type="text" value="Moskovská 2"/>
Juridical Form	<input type="text" value="Juridical person"/>
Date of Establishment	<input type="text" value="22.10.2010"/> 
<input type="button" value="Ok"/> <input type="button" value="Storno"/>	

To call the AddAdresar.aspx form, assign for the PxSuperGrid component in the Adresar.aspx form, the PxEditFormName property, name of the form, where data for the selected row shall be edited.

```

<Prx:PxSuperGrid ID="grdAdresar" runat="server"
    PageSize="4"
    PxEditFormName="AddAdresar.aspx"
    PxInfoFormName="InfoAdresar.aspx"
    PxVisibleButtons="SIDE0">
    <SelectedItemStyle BackColor="#ffddff" />
</Prx:PxSuperGrid>

```

Pressing the "edit"  button in the PxSuperGrid component opens the AddAdresar.aspx form. It is possible to enter new values or change old values in there, to save these values, press the "Ok", button.

The PxEdit component (as well as PxComboBox, PxFlyComboBox, PxJSDatePicker, etc.) contains

automatic validation.

If the table column linked to the PxEdit component is of the **string** type, validation of the string length is automatically carried out, if it is of the **int** type, validation operation to check whether the integer has been entered is carried out, in case of the **date** type, validation operation to check whether the valid date has been entered is carried out, etc.. For more information about the automatic validation, see the Part II of this reference manual.

The picture below shows an example where we have entered a string which length exceeded the length of string defined in the database. Then we have entered an invalid date.

Edit Address

The screenshot shows a web form titled "Edit Address". It contains several input fields: "Main text" with the value "35", "Persons Name" with "Jurij Brezan aaaaaa", "Address" with "Moskovská 2", "Juridical Form" with a dropdown menu showing "Juridical person", and "Date of Establishment" with "22.10.2010a". To the right of the "Persons Name" and "Date of Establishment" fields, there are red error messages: "* You have exceeded the allowed length of string, you can specify the maximum 50 characters !!!" and "* You have not entered the correct date !!!" respectively. At the bottom of the form, there are two buttons: "Ok" and "Storno".

Positioning and alignment of the PxEdit elements:

Entering the PxEdit components (one after another) to the AddAdresar.aspx file, causes asymmetrical raggedness of the elements. See the picture below:

Edit Address

The screenshot shows a web form titled "Edit Address". The elements are not aligned to the left, causing an asymmetrical, ragged appearance. The fields are: "Main text" with "4", "Persons" (a label), "Name" with "Helena Mociková", "Address" with "Rudná 2", "Juridical Form" with a dropdown menu showing "Juridical person", and "Date of Establishment" with "29.07.2010". At the bottom, there are two buttons: "Ok" and "Storno".

This asymmetrical raggedness can be corrected in the following way:

Define the AddTableRow property for each element and set the property to the „True“ value;

```
<Prx:PxEdit ID="edtIDADRESAR" runat="server" AddTableRow="True" ></Prx:PxEdit>
```

Each PxEdit component consists of the following elements in this order:

Label + TextBox + Label but Button

This switched on property enables to insert a table row between these components. In practice it look like this:

`<tr><td>Label + <td>TextBox +<td> Label but Button<td>`

At the first element set the TableBegin property to „True“. In practice, it looks like this:

```
<Prx:PxEdit ID="edtIDADRESAR" runat="server" AddTableRow="True" TableBegin="True" ></Prx:PxEdit>
```

At the last PxEdit component of the queue set the TableEnd property to „True“. In practice, it looks like this:

```
<Prx:PxEdit ID="edtCREATE_DATE" runat="server" AddTableRow="True" TableEnd="True" ></Prx:PxEdit>
```

It can be applied in this manner to the whole code, which shall look like the example in the picture below:

```
<Prx:PxEdit ID="edtIDADRESAR" runat="server" AddTableRow="True" TableBegin="True" ></Prx:PxEdit>
<Prx:PxEdit ID="edtName" runat="server" AddTableRow="True" ></Prx:PxEdit>
<Prx:PxEdit ID="edtADDRESS" runat="server" AddTableRow="True" ></Prx:PxEdit>
<Prx:PxComboBox ID="cmbPravnaForma" runat="server" AddTableRow="True" ></Prx:PxComboBox>
<Prx:PxJSDatePicker ID="edtCREATE_DATE" runat="server" AddTableRow="True" TableEnd="True" >
</Prx:PxJSDatePicker>
```

If the AddTableRow, TableBegin and TableEnd properties are set correctly, the elements shall be sorted and aligned correctly, see the picture below:

Edit Address

ID	<input type="text" value="35"/>
Persons Name	<input type="text" value="Jurij Brezan"/>
Address	<input type="text" value="Moskovská 2"/>
Juridical Form	<input type="text" value="Juridical person"/>
Date of Establishment	<input type="text" value="22.10.2010"/>
<input type="button" value="Ok"/> <input type="button" value="Storno"/>	

PxEdit component has a yet another property to align its content. Property AddTdToLabel, AddOnlyTdToLabel you serve to align the component elements PxEdit to HTML tables, just like property AddTableRow, TableBegin and TableEnd.

Each PxEdit component consists of the following elements in this order:

Label + TextBox + Label alebo Button

If the property AddTdToLabel set to True (AddTableRow property must be set to False), then you have the following elements components PxEdit table elements are inserted as follows:

<tr><td>Label + <td>TextBox +Label alebo Button

If the property `AddOnlyTdToLabel` set to `True` (`AddTableRow`, `AddTdToLabel` property must be set to `False`), then you have the following elements `PxEdit` components inserted table `<td>` element as follows:

```
Label + <td>TextBox
```

Property `AddTdToLabel`, `AddOnlyTdToLabel` are also implemented in other components, such as `PxComboBox`, `PxMemo`, `PxJSDatePicker`, `PxCheckBox`, `PxCheckBoxList` and under.

Caption name or names of the columns in the Grid (title) can be defined centrally, during the

definition of the `PxWebQuery` component, where the value in square brackets is the name of the column:

```
wquAdresar.Columns["NAME"].Caption = "Persons Name";
```

Changing the `PxEdit` component (Label) caption:

Besides central definition, the `PxEdit` component caption can be changed in the following way:

```
edtIDADRESAR.Caption = "Main text";
```

Changing the At Post caption of the `PxEdit` (Label) component:

The `PxEdit` component contains another caption, which is located after the `TextBox` component. That captions can be accessed via the `RCaption` property.

```
edtIDADRESAR.RCaption = "(Primary Key tbl Adresar)";
```

The final shape of the source code shall be then as follows:

```
edtIDADRESAR.PxDataSource = wquAdresar;  
edtIDADRESAR.FieldName = "IDADRESAR";  
edtIDADRESAR.Caption = "Main text";  
edtIDADRESAR.RCaption = "(Primary Key tbl Adresar)";
```

The picture below shows the resulting image:

Edit Address

Main text 35 (Primary Key tbl Adresar)

Persons Name Jurij Brezan

Address Moskovská 2

Juridical Form Juridical person

Date of Establishment 22.10.2010

Ok Storno

1.5. PxComboBox – component for selecting data from a list, similar to the DropDownList component

The PxComboBox component works similarly to or has got similar properties as the PxEdit component. Therefore there is no need in their detailed description, for the details please refer to the previous chapter about the PxEdit component. Things that have been described in detail in case of the PxEdit component shall only be mentioned here.

The PxComboBox component enables automatic editing of the table row column, which is loaded in the PxWebQuery component. Create the AddAdresar.aspx file, and type the following definition of the PxComboBox component there:

AddAdresar.aspx file:

```
<Prx:PxComboBox ID="cmbPravnaForma" runat="server" AddTableRow="True" ></Prx:PxComboBox>
```

In the AddAdresar.aspx.cs file, define linking of the PxComboBox component to the PxWebQuery component and its linking to the specific table column.

AddAdresar.aspx.cs file:

```
protected void Page_Load(object sender, EventArgs e)
{
    cmbPravnaForma.PxDataSource = wquAdresar;
    cmbPravnaForma.FieldName = "idpravnaforma";
}
```

The PxComboBox supports automatic loading of the list to its body, and automatic saving of the selected value to the database, during execution of the Post() command at the PxWebQuery component.

For automatic loading of the data to the PxComboBox component list, it is necessary to apply the AddParamKey parameter or the AddParamWebQuery parameter to the PxWebQuery component, to which the *PxComboBox* component is linked.

The AddParamKey serves for linking the table to the classifier. The *AddParamKey* method is the *PxWebQuery* component method. According to the Px Framework philosophy as much information as possible is to be defined on the *PxWebQuery* component. This actually saves the programming and writing in case of the *PxSuperGrid*, *PxEdit*, *PxComboBox* and other components. All these components borrow data and settings from the *PxWebQuery* component. Therefore, the *PxWebQuery* component has a number of procedures and methods, which are used for defining various display methods and relations between the tables.

So the *AddParamKey* method is the simplest method for defining the relation between the main

table and the classifier.

This definition can be written to the Adresar.aspx.cs file.

Example No.1.

```
protected void Page_Load(object sender, EventArgs e)
{
    if ((!IsPostBack)&&(wquAdresar.Active==false))
    {
        string sSQLText1 = "select IDPravForm as Key, Name as Value from PravForm Order By Value";

        wquAdresar.ConnectionString = "User Id=database01;Password=aa;Data Source=xe;";
        wquAdresar.SQLSelect = "select idadresar , name, address, idpravnaforma, create_date from Adresar";

        wquAdresar.AddParamKey("idpravnaforma", "Key", "Value", "Value", "Key", sSQLText1);

        wquAdresar.Open();

        wquAdresar.Columns["idadresar"].Caption = "ID";
        wquAdresar.Columns["name"].Caption = "Persons Name";
        wquAdresar.Columns["idpravnaforma"].Caption = "Juridical Form";
        wquAdresar.Columns["address"].Caption = "Address";
        wquAdresar.Columns["create_date"].Caption = "Date of Establishment";
    }
    else
    {
        {
            grdAdresar.PxDataSource = wquAdresar;
            grdAdresar.DataBind();
        }
    }
}
```

Description of the parameters of the *AddParamKey* method:

C# syntax:

```
public void AddParamKey(string aFieldName, string aFieldNameKey, string aFieldNameValue,
    string aFieldToView, string aFieldToDB,
    string aSQLText);
```

Description of the parameters:

aFieldName – name of the column in the table to which the AddParamKey method is applied
aFieldNameKey – name of the classifier's column, which represents the classifier's key (the name of the column can be obtained from the SQL command, which is entered to the **aSQLText** parameter)
aFieldNameValue – name of the classifier's column, which represents the classifier's value (the name of the column can be obtained from the SQL command, which is entered to the **aSQLText** parameter)
aFieldToView – is used to enter the name of the column which value shall be displayed in the Grid (*PxSuperGrid*)
aFieldToDB - is used to enter the name of the column which value shall be put to the database
aSQLText – is used to define the SQL command for loading the classifier, individual columns are to be appropriately named as Key and Value, for better clarity

After applying the AddParamKey parameter to the PxWebQuery component and linking the PxComboBox component, this component shall automatically download data during the loading process and automatically save the selected value to the database during the saving process.

Edit Address

The screenshot shows a form titled "Edit Address". It has the following fields and controls:

- ID**: A text box containing the value "35".
- Persons Name**: A text box containing the value "Jurij Brezan".
- Address**: A text box containing the value "Moskovská 2".
- Juridical Form**: A dropdown menu with a blue arrow. The visible options are "Juridical person" (highlighted in blue) and "Physical person".
- Date of Establishment**: A date picker control.
- Buttons**: Two buttons at the bottom, "Ok" and "Storno".

The AddTableRow, TableBegin and TableEnd properties, which are described in detail in the section dealing with the PxEdit component, are used once again for the alignment of the components.

Caption name or titles of the columns in the Grid can be defined centrally, during the definition of the PxWebQuery component, where the value in square brackets is the name of the column:

```
wquAdresar.Columns["NAME"].Caption = "Persons Name";
```

Changing the PxComboBox (Label) caption:

Besides central definition, the PxComboBox component caption can be changed in the following way:

```
cmbPravnaForma.Caption = "Juridical form";
```

Changing the At Post caption of the PxComboBox (Label) component:

The PxComboBox component contains another caption, which is located after the TextBox component. That captions can be accessed via the *RCaption* property.

```
cmbPravnaForma.RCaption = "(Text)";
```

The final shape of the source code shall be then as follows:

```
cmbPravnaForma.PxDataSource = wquAdresar;  
cmbPravnaForma.FieldName = "IDADRESAR";  
cmbPravnaForma.Caption = "Juridical Form 2";  
cmbPravnaForma.RCaption = "(Text)";
```

The picture below shows the resulting image:

Edit Address

ID	<input type="text" value="35"/>
Persons Name	<input type="text" value="Jurij Brezan"/>
Address	<input type="text" value="Moskovská 2"/>
Juridical Form 2	<input type="text" value="Juridical person"/> (Text)
Date of Establishment	<input type="text" value="22.10.2010"/>
<input type="button" value="Ok"/> <input type="button" value="Storno"/>	

If while saving the PxComboBox component no value is selected and the original text "Choose Value" is left, then null value for this column shall be saved in the database. If the "Choose Value" text is not suitable for you, then change it via the **ChooseValueText** property. See the example below:

```
cmbPravnaForma.ChooseValueText = "Choose Value...";  
cmbPravnaForma.PxDataSource = wquAdresar;  
cmbPravnaForma.FieldName = "idpravnaforma";
```

The PxComboBox component then would be as follows:

Juridical Form	<input type="text" value="Choose value"/>
	<input type="text" value="Choose value"/>
	<input type="text" value="Juridical person"/>
	<input type="text" value="Physical person"/>

1.6. PxCheckBox – component for checking the value (Check / UnCheck value)

The PxCheckBox component works similarly to or has got similar properties as the PxEdit component. Therefore there is no need in their detailed description, for the details please refer to the chapter about the PxEdit component. Things that have been described in detail in case of the PxEdit component shall only be mentioned here.

The PxCheckBox component enables automatic editing of the table row column, which is loaded in the PxWebQuery component. Create the AddAdresar.aspx file, and type the following definition of the PxCheckBox component there:

AddAdresar.aspx file:

```
<Prx:PxCheckBox ID="chkInvalid_Adress" runat="server" AddTableRow="True" ></Prx:PxCheckBox>
```

In the AddAdresar.aspx.cs file, define linking of the PxCheckBox component to the PxWebQuery component and its linking to the specific table column.

AddAdresar.aspx.cs file:

```
protected void Page_Load(object sender, EventArgs e)
{
    chkInvalid_Address.PxDataSource = wquAdresar;
    chkInvalid_Address.FieldName = "Invalid_Address";
}
```

The PxCheckBox component supports automatic loading of value from the PxWebQuery component and automatic saving of the entered value to the database, while executing the Post() command at the PxWebQuery component. Thus, as the PxCheckBox component allows to select only two states, i.e. True or False, it is possible to specify at the PxWebQuery component, the way these states shall be displayed and saved to the database. The parameter with the AddParamCheck parameter method is used for such definition. The AddParamCheck method is the PxWebQuery component method.

Description of the parameters of the *AddParamCheck* method:

C# syntax:

```
public void AddParamCheck(string aFieldName,
    string aCheckedValue, string aUnCheckedValue,
    bool aEditReadOnly)
```

Description of the parameters:

aFieldName – name of the column in the table to which the AddParamCheck method is applied

aCheckedValue – String value that shall replace the „True“ value, during selection in the PxCheckBox component

aUnCheckedValue – String value that shall replace the „False“ value, during selection in the PxCheckBox component

aEditReadOnly - If the grid is in the „edit“ state and the value entered is „True“ the editing operation shall be allowed

After applying the AddParamCheck parameter to the PxWebQuery component and linking the PxCheckBox component, this component can automatically interpret data loaded from the database during the loading process and automatically save the defined string value to the database during the saving process.

In this, the definition of the AddParamCheck parameter shall be as follows:

```
wquAdresar.AddParamCheck("invalid_adress", "A", "N", false);
```

This definition can be written to the Adresar.aspx.cs file, and it shall be as follows:

Example 05_PxCheckBox.

```
protected void Page_Load(object sender, EventArgs e)
{
    if ((!IsPostBack)&&(wquAdresar.Active==false))
    {
        string sSQLText1 = "select IDPravForm as Key, Name as Value from PravForm Order By Value";
    }
}
```



```

wquAdresar.ConnectionString = "User Id=database01;Password=aa;Data Source=xe;";
wquAdresar.SQLSelect = "select idadresar , name, address, idpravnaforma, create_date, invalid_address from
                        Adresar02";

wquAdresar.AddParamKey("idpravnaforma", "Key", "Value", "Value", "Key", sSQLText1);
wquAdresar.AddParamCheck("invalid_address", "A", "N", false);
wquAdresar.Open();







wquAdresar.Columns["idadresar"].Caption = "ID";
wquAdresar.Columns["name"].Caption = "Persons Name";
wquAdresar.Columns["idpravnaforma"].Caption = "Juridical Form";
wquAdresar.Columns["address"].Caption = "Address";
wquAdresar.Columns["create_date"].Caption = "Date of Establishment";
wquAdresar.Columns["invalid_address"].Caption = "Invalid address";

}
else
{
}
grdAdresar.PxDataSource = wquAdresar;
grdAdresar.DataBind();
}

```

After entering or applying of AddParamCheck parameter to the PxWebQuery component, the PxSuperGride component shall display values , instead of the „True“ value the "A" value shall be displayed and instead of the „False“ value the "N" value shall be displayed. Entry to the database shall be carried out in the similar way.

For more details please refer to the image below:

...	ID	Persons Name	Address	Juridical Form	Date of Establishment	Invalid address
  	14	Katarina Kolníková	Brezno	Physical person	2. 7. 2010 0:00:00	A
  	15	Jurij Brezan	Leningrad	Juridical person	1. 9. 2010 0:00:00	N
 ADD NEW ROW...						
Page 1 [2] [3]						

The AddAdresar.aspx form shall look like as follows:

Edit Address

ID
 Persons Name
 Address
 Juridical Form
 Date of Establishment
☐ Invalid address 2

The AddTableRow, TableBegin and TableEnd properties, which are described in detail in the section dealing with the PxEdit component, are used once again for the alignment of the component.

Caption name or titles of the columns in the Grid can be defined centrally, during the definition of the PxWebQuery component, where the value in square brackets is the name of the column:

```
wquAdresar.Columns["invalid_address"].Caption = "Invalid address";
```

Changing the PxCheckBox (Label) caption:

Besides central definition, the PxCheckBox component caption can be changed in the following way:

```
chkInvalid_Adress.Text = " Invalid address 2";
```

If you want to change text in the PxCheckBox component, enter a string to the Text property.

The final shape of the source code shall be then as follows:

```
chkInvalid_Adress.PxDataSource = wquAdresar;  
chkInvalid_Adress.FieldName = wquAdresar.Columns["Invalid_Adress"].ToString();  
chkInvalid_Adress.Text = " Invalid address 2";
```

The picture below shows the resulting image:

Edit Address

ID	<input type="text" value="15"/>
Persons Name	<input type="text" value="Jurij Brezan"/>
Address	<input type="text" value="Leningrad"/>
Juridical Form	<input type="text" value="Juridical person"/>
Date of Establishment	<input type="text" value="01.09.2010"/>
	<input type="checkbox"/> Invalid address 2

1.7. PxFlyComboBox – set of the consecutively linked comboboxes, suitable for the work with structured data (for example: selection of category and subcategory)

The PxFlyComboBox component serves for value selection in case of structured lists, it is mostly used when an object is integrated into a category and subcategory. The PxFlyComboBox component may consist of several mutually related comboboxes (DropDownList components).

There can be an optional number of such comboboxes – from 2 to n. See the picture below:

Region	District	Municipality
Trenčiansky	Nové Mesto nad Váhom	Vaďovce

The picture shows selection of the name of the municipality. Since there are nearly 3000 municipalities in Slovakia, it would have been difficult to choose from such list. The PxFlyComboBox component shall save the labour here. When the particular region is selected, only districts filtered for the given region shall be displayed in the combobox „district“. When the certain district is chosen, only municipalities filtered for the given district shall be displayed in the combobox.

However it is quite difficult to program such thin, but the PxFramework provides quick and cheap solution.

For the correct functioning of the PxFlyComboBox component, it is necessary to apply the AddParamFlyComboBox parameter to the given PxWebQuery component, which shall display or work with such structured data.

The definition of the AddParamFlyComboBox parameter is as follows:

```
wquAdresar.AddParamFlyComboBox("miesto", "idkraj;idokres;idobec",  
                                "idkraj;idokres;idobec", "kraj;okres;obec",  
                                "Obec;Okres", "wquObec", "mvStandard");
```

Description of the parameters of the *AddParamFlyComboBox* method:

C# syntax:

```
public void AddParamFlyComboBox(string aFieldName, string aKeyFields,  
                                string aKeyDetailFields, string aValueFields,  
                                string aFieldToView, string aPxWebQueryName,  
                                string aModeToView)
```

Description of the parameters:

aFieldName – Name of the column, which is created as a new column in the table and to which the AddParamFlyComboBox method is applied and which is then linked to the PxFlyComboBox component (this column name must be unique, no other table column with such name shall exist).

aKeyFields – String value to which keys of the main table columns are defined , to which during editing via the PxFlyComboBox the IDs are entered, in our case these are the IdKraj, IdOkres and IdObec. These columns are entered consequently in the way they are to be displayed in the PxFlyComboBox component. Individual columns are separated by a semicolon.

aKeyDetailFields – String value to which keys of the classifier table columns are defined, from which during editing via the PxFlyComboBox component the IDs are loaded and entered to the main table. In our case these are the IdKraj, IdOkres and IdObec columns. These columns are entered consequently in the way they are to be displayed in the PxFlyComboBox component. Individual columns are separated by a semicolon.

aValueFields - String value to which values of the classifier table columns are defined, which are displayed in the PxFlyComboBox component during editing via this component. In our case these are the IdKraj, IdOkres and IdObec columns. These columns are entered consequently in the way they are to be displayed in the PxFlyComboBox component. Individual columns are separated by a semicolon.

aFieldToView – String value, which defines the names of the columns that are displayed in PxSuperGrid component, under the aFieldName column.

Summary or listing of these columns may not be complete. These columns are entered consequently in the way they are to be displayed in the PxSuperGrid component. Individual columns are separated by a semicolon.

aPxWebQueryName – Name of the PxWebQuery component, where the classifier of Region (Kraj), District(Okres) and Municipality(Obec) is loaded. The classifier is formed by merging of three classifiers, in our case they are as follows: Ckraj, Cokres, CObec.

aModeToView – This string defines data display in the PxSuperGrid component, in our case of the column defined in the aFieldName entry. The existing display modes are as follows: **mvValueBR**, and **mvNameAndBR** **mvNormal**.

If the "Obec;Okres" string is entered to the FieldToView entry and the **mvNormal** display mode is selected, the display in the PxSuperGrid component shall look like this:

Municipality, District
Brezno, Brezno
Vad'ovce, Nové Mesto nad Váhom
Kuzmice, Topolčany

If the "Obec;Okres" string is entered to the FieldToView entry and the **mvValueBR** display mode is selected, the display in the PxSuperGrid component shall look like this:

Municipality, District
Brezno Brezno
Vad'ovce Nové Mesto nad Váhom
Kuzmice Topolčany










If the "Obec;Okres" string is entered to the FieldToView entry and the **mvNameAndBR** display mode is selected, the display in the PxSuperGrid component shall look like this:

Municipality, District
Municipality: Brezno District: Brezno
Municipality: Vad'ovce District: Nové Mesto nad Váhom
Municipality: Kuzmice District: Topolčany

If the "Obec;Okres;Kraj" string is entered to the FieldToView entry and the **mvNameAndBR** display mode is selected, the display in the PxSuperGrid component shall look like this:

Municipality, District	
Municipality:	Brezno
District:	Brezno
Region:	Banskobystrický
Municipality:	Vaňovce
District:	Nové Mesto nad Váhom
Region:	Trenčiansky
Municipality:	Kuzmice
District:	Topoľčany
Region:	Nitriansky

Total PxSuperGrid component display shall be as follows:

...	ID	Persons Name	Juridical form	Street	Municipality, District
  	17	Peter Sekularik	Physical person	Šalgotarianska 22	Municipality: Brezno District: Brezno Region: Banskobystrický
  	18	Helena Modiková	Physical person	THK 331	Municipality: Vaňovce District: Nové Mesto nad Váhom Region: Trenčiansky
  	19	Peter Savinon	Juridical person	Piešťanská 27	Municipality: Kuzmice District: Topoľčany Region: Nitriansky
 ADD NEW ROW...					
Page 1 [2]					

But let us return to a particular programming with the PxFlyComboBox component, our example of Databank Names and Companies.

To apply the AddParamFlyComboBox parameter to the wquAdresar component, it is necessary to create new PxWebQuery component, which shall be called wquObec.

Add this new component to the Adresar.aspx file and the definition shall look like this:

```
<Prx:PxWebQuery ID="wquObec" runat="server" Value="wquObec"/>
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>
```

Don't forget to define the Obec component before the wquAdresar component.

Then in the Adresar.aspx.cs file load data to the wquObec component, and then link this component to the wquAdresar component via the AddParamFlyComboBox parameter. During definition of the wquObec component, create a special SQL select, which shall combine the following three tables: Ckraj, Cokres, CObec.

This SQL command shall be as follows:

```
select COBEC.ID_OBEC as IDOBEC, COBEC.NAZOV_OBEC as OBEC,
       COBEC.ID_OKRES as IDOKRES, COKRES.NAZOV_OKRES as OKRES,
       COBEC.ID_KRAJ as IDKRAJ, CKRAJ.NAZOV_KRAJ as KRAJ
from
```

COBEC,COKRES,CKRAJ
 where COBEC.ID_OKRES=COKRES.ID_OKRES and
 COBEC.ID_KRAJ=CKRAJ.ID_KRAJ

The table resulting from this SQL command shall look like the one in the picture below:

IDOBEC	OBEC	IDOKRES	OKRES	IDKRAJ	KRAJ
1	Vadovce	19	Nové Mesto nad Váhom	3	Trenciansky
2	Veľká Hradná	24	Trencín	3	Trenciansky
3	Veľké Bierovce	24	Trencín	3	Trenciansky
4	Višnové	19	Nové Mesto nad Váhom	3	Trenciansky
5	Trnava	15	Trnava	2	Trnavský
6	Bínovce	15	Trnava	2	Trnavský
7	Bohdanovce n...	15	Trnava	2	Trnavský

The PxFlyComboBox component can work with such table fairly well.

The next image contains listing of the Adresar.aspx.cs file source code for filling data of the wquObec component and its linking to the wquAdresar component via the AddParamFlyComboBox parameter.

```
protected void Page_Load(object sender, EventArgs e)
{
    if ((!IsPostBack) && (wquObec.Active == false))
    {
        wquObec.ConnectionString = "User Id=database01;Password=aa;Data Source=xe;";
        wquObec.SQLSelect = @"select COBEC.ID_OBEC as IDOBEC, COBEC.NAZOV_OBEC as OBEC,
            COBEC.ID_OKRES as IDOKRES, COKRES.NAZOV_OKRES as OKRES,
            COBEC.ID_KRAJ as IDKRAJ,CKRAJ.NAZOV_KRAJ as KRAJ
            from
                COBEC,COKRES,CKRAJ
            where COBEC.ID_OKRES=COKRES.ID_OKRES and
                COBEC.ID_KRAJ=CKRAJ.ID_KRAJ";

        wquObec.Open();

        wquObec.Columns["IDOBEC"].Caption = "Id Municipality";
        wquObec.Columns["OBEC"].Caption = "Municipality";
        wquObec.Columns["IDOKRES"].Caption = "Id District";
        wquObec.Columns["OKRES"].Caption = "District";
        wquObec.Columns["IDKRAJ"].Caption = "Id Region";
        wquObec.Columns["KRAJ"].Caption = "Region";
    }

    if ((!IsPostBack)&&(wquAdresar.Active==false))
    {
        string sSQLText1 = "select IDPravForm as Key, Name as Value from PravForm Order By Value";

        wquAdresar.ConnectionString = "User Id=database01;Password=aa;Data Source=xe;";
        wquAdresar.SQLSelect = "select idadresar , name, address, idpravnaforma, create_date, invalid_address, idkraj, idokres, idobec from Adresar03";

        wquAdresar.AddParamKey("idpravnaforma", "Key", "Value", "Value", "Key", sSQLText1);
        wquAdresar.AddParamCheck("invalid_address", "A", "N", false);
        wquAdresar.AddParamFlyComboBox("miesto", "idkraj;idokres;idobec",
            "idkraj;idokres;idobec", "kraj;okres;obec",
            "Obec;Okres;Kraj", "wquObec", "mvNameAndBR");
    }
}
```



```

wquAdresar.Open();

wquAdresar.Columns["idadresar"].Caption = "ID";
wquAdresar.Columns["name"].Caption = "Persons Name";
wquAdresar.Columns["idpravnaforma"].Caption = "Juridical form";
wquAdresar.Columns["address"].Caption = "Street";
wquAdresar.Columns["create_date"].Caption = "Date of Establishment";
wquAdresar.Columns["invalid_adress"].Caption = "Invalid address";
wquAdresar.Columns["miesto"].Caption = "Municipality, District";
wquAdresar.Columns["idobec"].Caption = "Id Municipality";
wquAdresar.Columns["idokres"].Caption = "Id District";
wquAdresar.Columns["idkraj"].Caption = "Id Region";

}
else
{
}
grdAdresar.PxDataSource = wquAdresar;
grdAdresar.DataBind();
}

```

If you want to use the PxFlyComboBox component for editing in the AddAdresar.aspx form, it shall be defined in this file as follows:

```

<Prx:PxWebQuery ID="wquObec" runat="server" Value="wquObec"/>
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>

<Prx:PxEdit ID="edtIDADRESAR" runat="server" AddTableRow="True" TableBegin="True" ></Prx:PxEdit>
<Prx:PxEdit ID="edtName" runat="server" AddTableRow="True" ></Prx:PxEdit>
<Prx:PxComboBox ID="cmbPravnaForma" runat="server" AddTableRow="True" ></Prx:PxComboBox>
<Prx:PxJSDatePicker ID="edtCREATE_DATE" runat="server" AddTableRow="True" ></Prx:PxJSDatePicker>
<Prx:PxCheckBox ID="chkInvalid_Adress" runat="server" AddTableRow="True" ></Prx:PxCheckBox>
<Prx:PxEdit ID="edtADDRESS" runat="server" AddTableRow="True" TableEnd="True"></Prx:PxEdit>
<Prx:PxFlyComboBox ID="fcmBmiesto" runat="server"></Prx:PxFlyComboBox>

```

Link the PxFlyComboBox component to the wquAdresar component in the AddAdresar.aspx.cs file as follows:

```

fcmBmiesto.PxDataSource = wquAdresar;
fcmBmiesto.FieldName = "miesto";

```

The PxFlyComboBox component does not support alignments and AddTableRow, TableBegin TableEnd properties, which are supported by other components like PxEit, etc.

During the definition in the AddAdresar.aspx file do not forget to define besides the PxFlyComboBox component line, the wquObec component, because in other case, the PxFlyComboBox component shall not be displayed in the AddAdresar.aspx form.

```

<Prx:PxWebQuery ID="wquObec" runat="server" Value="wquObec"/>
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>

```

If everything is set up correctly, the resulting AddAdresar.aspx form shall look like this:

Edit Address

Titles in the Grid and captions in the PxFlyComboBox component, can be defined centrally during the definition of the wquObec component:

```
wquObec.Columns["IDOBEC"].Caption = "Id Municipality";
wquObec.Columns["OBEC"].Caption = "Municipality";
wquObec.Columns["IDOKRES"].Caption = "Id District";
wquObec.Columns["OKRES"].Caption = "District";
wquObec.Columns["IDKRAJ"].Caption = "Id Region";
wquObec.Columns["KRAJ"].Caption = "Region";
```

If while saving the PxComboBox component no value is selected and the original text "Choose Value" is left, then null value for this column shall be saved in the database. If the "Choose Value" text is not suitable for you, change it via the **ChooseValueText** property. See the example below:

```
fcmbMiesto.ChooseValueText = "Choose Value";
fcmbMiesto.PxDataSource = wquAdresar;
fcmbMiesto.FieldName = "miesto";
```

The PxFlyComboBox component shall look like as follows:

The PxFlyComboBox component contains a function through which we can find the set value of individual comboboxes. This method is called *GetValue* function.

Description of the parameters of the GetValue function:

C# syntax:

```
public string GetValue(string aFieldName);
```

Description of the parameters:

FieldName – name of the column which belongs to a given combobox in the PxFlyComboBox component. Usually these are the names of the columns that were specified in the AddParamFlyComboBox parameter, namely those which have been entered into the expression **aKeyFields**, **aKeyDetailFields** or **aValueFields**.

Example of use of the **GetValue** function:

```
string sGetVal = fcmbMiesto.GetValue("IdOkres");
```

In this example the GetValue function returns the value of the set district, which is set in the combobox, which is selected from a list of the districts.

1.8. PxGreatRepeater – component for entering data with repeating structure, maximum number of values is limited

The PxGreatRepeater component is used to input data with cyclically repeating structure. Simple example of repeating data is a person who has got a telephone number or several phone numbers. The picture below shows the PxGreatRepeater control element for entering the phone number:

The screenshot shows a single instance of the PxGreatRepeater control. It has a light blue header with a '+' button on the left and the title 'Telephone' in the center. Below the header, there are two input fields: 'Telephone' with the value '02 / 1212221' and 'Notice' with the value 'phone home'. A '-' button is located to the right of the input fields.

Of course it is possible to enter more phone numbers using the PxGreatRepeater control element, see the picture below:

The screenshot shows three instances of the PxGreatRepeater control stacked vertically. Each instance has a light blue header with a '+' button on the left and the title 'Telephone' in the center. Below the header, there are two input fields: 'Telephone' and 'Notice'. The first instance has 'Telephone' with '02 / 1212221' and 'Notice' with 'phone home'. The second instance has 'Telephone' with '0905 148 277' and 'Notice' with 'mobile'. The third instance has 'Telephone' with '045/ 56446121' and 'Notice' with 'Peter home'. A '-' button is located to the right of each set of input fields.

Press the "+" button to add new boxes for entering the phone number, press the "-" button to delete the existing numbers.

The example below shows how it works in practice. Add the following columns to the directory table:

```
Tel1 VARCHAR2(50),  
Tel_Pozn1 VARCHAR2(50),  
Tel2 VARCHAR2(50),  
Tel_Pozn2 VARCHAR2(50),  
Tel3 VARCHAR2(50),  
Tel_Pozn3 VARCHAR2(50)
```

As you can see we have entered the phone entry (tel1, tel2, tel3) three times to the Adresar table. Thus, during entering the phone number, and its note we will be limited, we will be able to enter only three phone numbers with a note. Entering and deleting, can be carried out by means of the "+" and "-" buttons.

To link the PxGreatRepeater component to the PxWebQuery component, apply the *AddParamGreatWebQuery* parameter to the PxWebQuery component.

For the correct functioning of the PxGreatRepeater component apply the *AddParamGreatWebQuery* parameter to the PxWebQuery component, which shall display or operate with such structured data. The *AddParamGreatWebQuery* parameter definition is as follows:

```
wquAdresar.AddParamGreatWebQuery("Telefon", "Tel1;Tel_Pozn1", "Tel1 - Tel_Pozn1", 3, "mvNormal");
```

Description of the parameters of the *AddParamGreatWebQuery* method:

C# syntax:

```
public void AddParamGreatWebQuery(string aFieldName,  
                                   string aEditAllFieldNames,  
                                   string aExpresionToView,  
                                   Int16 aMaxRepeatCount,  
                                   string aModeToView)
```

Description of the parameters:

aFieldName – Name of the column that is created as a new column in the table, to which the *AddParamGreatWebQuery* method is applied, and to which the PxGreatRepeater component is linked (this column name must be unique, no other table column with such name shall exist).

aEditAllFieldNames – Lists all names of the columns to be edited in the PxGreatRepeater component in exactly the same order they shall be listed in the PxGreatRepeater component. The individual columns are separated by a semicolon. The names of the columns shall be composed of columns which are cyclically repeated, i.e. Tel1 .. to .. Tel3. The name of the column that starts with the number 1 shall always be entered.

aExpresionToView – Lists all names of the columns to be displayed in the Grid. These names may be separated by a comma or a hyphen, which shall appear during the display in the Grid (PxSuperGrid).

aMaxRepeatCount – This value defines the maximum number of phone numbers which can be entered via the PxGreatRepeater component. If the contains tel1, tel2, tel3 columns then this variable shall equate to 3.

aModeToView – This string defines data display in the PxSuperGrid component, in our case of the column

defined in the aFieldName entry. The existing display modes are as follows: **mvBreakHR**, **mvLineNumber** a **mvNormal**.

If the "Tell - Tel_Pozn1" string is entered to the ExpressionToView entry and the **mvNormal** display mode is selected , the display in the PxSuperGrid component shall look like this:

Telephone
02 / 1212221 - phone home 0905 148 277 - mobile
045/55646 - phone home

If the "Tell - Tel_Pozn1" string is entered to the ExpressionToView entry and the **mvBreakHR**, display mode is selected , the display in the PxSuperGrid component shall look like this:

Telephone
02 / 1212221 - phone home
0905 148 277 - mobile
045/55646 - phone home

If the "Tell - Tel_Pozn1" string is entered to the ExpressionToView entry and the **mvLineNumber** display mode is selected, the display in the PxSuperGrid component shall look like this:

Telephone
1. 02 / 1212221 - phone home 2. 0905 148 277 - mobile
1. 045/55646 - phone home

To reduce the number of columns displayed in the grid use the VisibleFields property of the PxSuperGrid component. See the picture below:

```
grdAdresar.VisibleFields = "name;idpravnaforma;address;miesto;telefon";
```

Assign to the VisibleFields property the names of the columns separated by a semicolon which are to be displayed in the PxSuperGrid component. If no value is assigned in the VisibleFields property all columns (fields) shall be displayed in the PxSuperGrid component. Of course all columns in the PxSuperGrid component shall be displayed in exactly the same order they have been entered to the VisibleFields property.








The VisibleFields property can also be used to define the length of the column (by typing a colon after each column and the number of pixels that shall define the length of the column). See the example below:

```
grdAdresar.VisibleFields = "idadresar:50;name;idpravnaforma;address;miesto:100;telefon:150";
```

For the proper functioning of the VisibleFields property it shall be defined before linking and binding to the PxWebQuery component.

```
grdAdresar.VisibleFields = "name;idpravnaforma;address;miesto;telefon";
grdAdresar.PxDataSource = wquAdresar;
grdAdresar.DataBind();
```

The resulting PxSuperGrid shall be as follows:

...	ID	Persons Name	Juridical Form	Street	Municipality, District	Telephone
  	22	Juraj Peter Tarnoci	Juridical person	Fatrnska 22	Municipality: Ďubákovo District: Poltár Region: Banskobystrický	1. 02 / 1212221 - phone home 2. 0905 148 277 - mobile
  	23	Helena Modková	Physical person	Trade 281	Municipality: Jesenské District: Levice Region: Nitriansky	1. 045/55646 - phone home
 ADD NEW ROW...						
Page 1 [2] [3]						

The next image contains listing of the Adresar.aspx.cs file source code for application of the AddParamGreatWebQuery parameter to the wquAdresar component. The example shows entering of two parameters, i.e. telephone and fax numbers:

```
if ((!IsPostBack)&&(wquAdresar.Active==false))
{
    string sSQLText1 = "select IDPravForm as Key, Name as Value from PravForm Order By Value";

    wquAdresar.ConnectString = "User Id=database01;Password=aa;Data Source=x";
    wquAdresar.SQLSelect = @"select idadresar , name, address, idpravnaforma,
        create_date, invalid_adress, idkraj, idokres,
        idobec, tel1, tel_pozn1, tel2, tel_pozn2,
        tel3, tel_pozn3, fax1, fax_pozn1,
        fax2, fax_pozn2 from Adresar04";

    wquAdresar.AddParamKey("idpravnaforma", "Key", "Value", "Value", "Key", sSQLText1);
    wquAdresar.AddParamCheck("invalid_adress", "A", "N", false);
    wquAdresar.AddParamFlyComboBox("miesto", "idkraj;idokres;idobec", "idkraj;idokres;idobec",
        "kraj;okres;obec",
        "Obec;Okres;Kraj", "wquObec", "mvNameAndBR");

    wquAdresar.AddParamGreatWebQuery("Telefon", "Tel1;Tel_Pozn1", "Tel1 - Tel_Pozn1", 3,
        "mvLineNumber");
```

```

wquAdresar.AddParamGreatWebQuery("Fax", "Fax1;Fax_Pozn1", "Fax1 - Fax_Pozn1", 2, "");

wquAdresar.Open();

wquAdresar.Columns["idadresar"].Caption = "ID";
wquAdresar.Columns["name"].Caption = "Persons Name";
wquAdresar.Columns["idpravnaforma"].Caption = "Juridical Form";
wquAdresar.Columns["address"].Caption = "Street";
wquAdresar.Columns["create_date"].Caption = "Date of Establishment";
wquAdresar.Columns["invalid_address"].Caption = "Invalid address";
wquAdresar.Columns["miesto"].Caption = "Municipality, District";

wquAdresar.Columns["tel1"].Caption = "Telephone";
wquAdresar.Columns["tel_pozn1"].Caption = "Notice";
wquAdresar.Columns["tel2"].Caption = "Telephone";
wquAdresar.Columns["tel_pozn2"].Caption = "Notice";
wquAdresar.Columns["tel3"].Caption = "Telephone";
wquAdresar.Columns["tel_pozn3"].Caption = "Notice";

wquAdresar.Columns["fax1"].Caption = "Fax";
wquAdresar.Columns["fax_pozn1"].Caption = "Notice";
wquAdresar.Columns["fax2"].Caption = "Fax";
wquAdresar.Columns["fax_pozn2"].Caption = "Notice";

wquAdresar.Columns["telefon"].Caption = "Telephone";

}
else
{
}
grdAdresar.VisibleFields = "name;idpravnaforma;address;miesto;telefon";
grdAdresar.PxDataSource = wquAdresar;
grdAdresar.DataBind();
}

```

If you want to use the PxGreatRepeater component for editing in the AddAdresar.aspx form, it shall be defined in this file as follows:

```

<Prx:PxWebQuery ID="wquObec" runat="server" Value="wquObec"/>
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>

<Prx:PxEdit ID="edtIDADRESAR" runat="server" AddTableRow="True" TableBegin="True" ></Prx:PxEdit>
<Prx:PxEdit ID="edtName" runat="server" AddTableRow="True" ></Prx:PxEdit>
<Prx:PxComboBox ID="cmbPravnaForma" runat="server" AddTableRow="True" ></Prx:PxComboBox>
<Prx:PxJSDatePicker ID="edtCREATE_DATE" runat="server" AddTableRow="True" ></Prx:PxJSDatePicker>
<Prx:PxCheckBox ID="chkInvalid_Address" runat="server" AddTableRow="True" ></Prx:PxCheckBox>
<Prx:PxEdit ID="edtADDRESS" runat="server" AddTableRow="True" TableEnd="True" ></Prx:PxEdit>
<Prx:PxFlyComboBox ID="fcmBmiesto" runat="server"></Prx:PxFlyComboBox>
<br />
<Prx:PxGreatRepeater ID="gtrTelefon" runat="server"></Prx:PxGreatRepeater>
<br />
<Prx:PxGreatRepeater ID="gtrFax" runat="server"></Prx:PxGreatRepeater>

```

Link the PxGreatRepeater component to the wquAdresar component in the AddAdresar.aspx.cs file, this linking shall as follows:

```

gtrTelefon.PxDataSource = wquAdresar;
gtrTelefon.FieldName = "telefon";

gtrFax.PxDataSource = wquAdresar;
gtrFax.FieldName = "fax";

```

The PxFlyComboBox component does not support alignments and AddTableRow, TableBegin, TableEnd properties, which are supported by other components like PxEdit, etc., that is why it is necessary to use the
 tag.

Titles in the Grid (Title) and captions in the PxGreatRepeater component can be defined centrally, during the definition of the wquAdresar component:

```

wquAdresar.Columns["tel1"].Caption = "Telephone";
wquAdresar.Columns["tel_pozn1"].Caption = "Notice";
wquAdresar.Columns["tel2"].Caption = "Telephone";
wquAdresar.Columns["tel_pozn2"].Caption = "Notice";
wquAdresar.Columns["tel3"].Caption = "Telephone";
wquAdresar.Columns["tel_pozn3"].Caption = "Notice";

wquAdresar.Columns["fax1"].Caption = "Fax";
wquAdresar.Columns["fax_pozn1"].Caption = "Notice";
wquAdresar.Columns["fax2"].Caption = "Fax";
wquAdresar.Columns["fax_pozn2"].Caption = "Notice";

wquAdresar.Columns["telefon"].Caption = "Telephone";

```

If everything is set up correctly, the resulting AddAdresar.aspx form shall look like this:

Edit Address

ID	<input type="text" value="23"/>	
Persons Name	<input type="text" value="Helena Mociková"/>	
Juridical Form	<input type="text" value="Physical person"/>	
Date of Establishment	<input type="text" value="02.07.2010"/>	
	<input type="checkbox"/> Invalid address	
Street	<input type="text" value="Trade 281"/>	
Region	District	Municipality
<input type="text" value="Nitriansky"/>	<input type="text" value="Levice"/>	<input type="text" value="Jesenské"/>
<div> <div>+</div> <div>Telephone</div> </div>		
<div> <div>Telephone</div> <div><input type="text" value="045/55646"/></div> <div>-</div> </div>		
<div> <div>Notice</div> <div><input type="text" value="phone home"/></div> </div>		
<div> <div>+</div> <div>Fax</div> </div>		
<div> <div>Fax</div> <div><input type="text" value="041/645464"/></div> <div>-</div> </div>		
<div> <div>Notice</div> <div><input type="text" value="fax"/></div> </div>		
<div> <div>Ok</div> <div>Storno</div> </div>		

The PxComboBox, PxFlyComboBox, PxJSDatePicker are integrated in the PxGreatRepeater component and when a certain parameter (AddParamKey, AddParamWebQuery, AddParamFlyComboBox etc.) is applied to the given column in the PxWebQuery component, individual components are automatically activated in the PxGreatRepeater component. For more information please refer to the picture below:

The PxGreatRepeater component contains a function through which we can find the value set for each component that contains the PxGreatRepeater component. This method is called **GetValue**.

Description of the parameters of the GetValue function:

C# syntax:

```
public string GetValue(string aFieldName, Int32 aNumberRepeaterPanel);
```

Description of the parameters:

aFieldName – name of the column which belongs to the given component that is contained in the PxGreatRepeater component. Usually these are the names of the columns that were specified in the **AddParamGreatWebQuery** parameters, namely those which have been entered into the expression **aEditAllFieldNames**. In the PxFlyComboBox component, located in PxGreatRepeater component, there is an overload of the aFieldName variable in a way that firstly the FieldName is entered, which defines the PxFlyComboBox component and then enter the name of the combobox field, separated by a semicolon. More examples: "Miestol;IdOkres1"

aNumberRepeaterPanel – number of panel from which a value should be loaded. Number of the first panel is 0. Maximum number of panels is limited by variable **aMaxRepeatCount** in **AddParamGreatWebQuery** parameter.

Example of use of the **GetValue** method:

```
string sGetVal = gtrTelefon.GetValue("Tel1", 0);
```


In this example the *GetValue* function returns the value set in "Tell" in the first panel.

Example of use of the method **GetValue** and loading of a value from the PxFlyComboBox component, located in the PxGreatRepeater component:

```
string sGetVal = gtrMiesto.GetValue("Miestol;IDOBEC1", 0);
```

In this example the *GetValue* function returns us the value of the district set in the PxFlyComboBox component, which is inserted in the PxGreatRepeater component.

1.9. PxDbNavigator – the component for the work with the PxWebQuery components, row cursor movement, etc.

The PxDbNavigator component serves for moving or changing the cursor position and acts usually as a compliment to the PxSuperGrid component. The PxDbNavigator component is connected to the PxWebQuery component, and after the corresponding button is pressed, the carried-out change is displayed in the PxSuperGrid component, by moving the cursor, or by some other operation. The PxDbNavigator component looks like as follows:



If you want to work with the PxDbNavigator component, you have to define it in the Adresar.aspx file as follows:

```
<Px:PxDbNavigator ID="dbnAdresar" runat="server" />
```

Link the PxDbNavigator component to the PxWebQuery component in the Adresar.aspx.cs file as follows:

```
dbnAdresar.PxDataSource = wquAdresar;
```

Buttons display can be controlled via the *PxVisibleButtons* property in the following way:










```
dbnAdresar.PxVisibleButtons = "FRNLIEDPC";
```

The following value list shows the legend, according to which the individual buttons are displayed, if the specific letter is added to the *PxVisibleButtons* property, the corresponding button matching the this letter shall be displayed in the PxDbNavigator component:

F – (First) move the cursor to the first row in the component PxWebQuery
R – (Prior) move the cursor to the previous line in component PxWebQuery
N – (Next) move the cursor to the next line in component PxWebQuery
L – (Last) move the cursor to the last line in component PxWebQuery
I – (Insert) changing components in a state PxWebQuery entering a new row
E – (Edit) PxWebQuery changing components in a state of editing, change the actual row
D – (Delete) delete the actual row in component PxWebQuery
P – (Post) save changes after editing data or insert new row components PxWebQuery

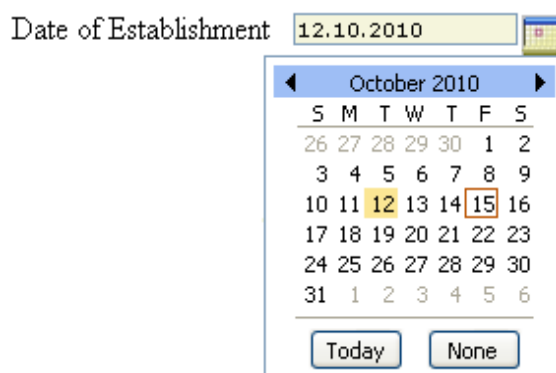
C – (Cancel) cancel the changes after editing data or insert new row components PxWebQuery

The next value list displays buttons, corresponding to the individual operations and a specific letter, which is entered to the PxVisibleButtons property, and the given button shall be displayed:

	- Operation „First“, the letter „F“ is added to the PxVisibleButtons property
	- Operation „Prior“, the letter „R“ is added to the PxVisibleButtons property
	- Operation „Next“, the letter „N“ is added to the PxVisibleButtons property
	- Operation „Last“, the letter „L“ is added to the PxVisibleButtons property
	- Operation „Insert“, the letter „I“ is added to the PxVisibleButtons property
	- Operation „Edit“, the letter „E“ is added to the PxVisibleButtons property
	- Operation „Delete“, the letter „D“ is added to the PxVisibleButtons property
	- Operation „Post“, the letter „P“ is added to the PxVisibleButtons property
	- Operation „Cancel“, the letter „C“ is added to the PxVisibleButtons property

1.10. PxJSDatePicker – component for the date entry, based on the JavaScript

The PxJSDatePicker component is a component of the calendar based on JavaScript. This component is used for the date selection in the folder menu, selection of current or selected date, or eventually selection of the manual date entry. The picture below displays the PxJSDatePicker component's calendar:



If you want to work with the PxJSDatePicker component, you have to define it in the AddAdresar.aspx file as follows:

```
<Px:PxJSDatePicker ID="edtCREATE_DATE" runat="server" AddTableRow="True"></Px:PxJSDatePicker>
```

Link the PxJSDatePicker component to the PxWebQuery component in the AddAdresar.aspx.cs file as follows:


```
edtCREATE_DATE.PxDataSource = wquAdresar;  
edtCREATE_DATE.FieldName = "CREATE_DATE";
```

The alignment of the component is carried once again via the *AddTableRow*, *TableBegin* and *TableEnd* properties. For their see their detailed description please refer to the section which deals with the PxEdit component.

If we wanted to change the date format, we can do it through property *dateFormat*, more examples below:

```
edtCREATE_DATE.DateFormat = "d.M.yyyy hh:mm:ss";
```

Entering of this format shall be reflected in the PxJSDatePicker component in the following way:

Date of Establishment 

1.11. PxDatePicker – component of the award date

The PxDatePicker component has been removed from the Px Framework v.1.08.17. That is why we recommend you to use the PxJSDatePicker component, which replaces this component.

1.12. PxMemo – component used for the direct data editing, the extension of the PxEdit component

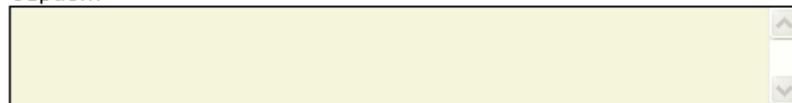
The PxMemo component works similarly to or has got similar properties as the PxEdit component. Therefore there is no need in its detailed description, for the details please refer to the chapter about the PxEdit component.

The PxMemo component is similar to the PxEdit component. The following TextBox definition which is part of the PxMemo component is added to the "OnInit"part:

```
Width = 400;  
Height = 50;  
TextMode = TextBoxMode.MultiLine;
```

The picture below displays the PxMemo component:

Caption:



If you want to work with the PxMemo component, you have to define it in the AddAdresar.aspx file as follows:

```
<Prx:PxMemo ID="memText" runat="server" AddTableRow="True" ></Prx:PxMemo>
```

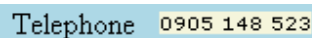
Link the PxMemo component to the PxWebQuery component in the AddAdresar.aspx.cs file as follow:

```
memText.PxDataSource = wquAdresar;  
memText.FieldName = wquAdresar.Columns["TEXT"].ToString();
```

The alignment of the component is carried once again via the AddTableRow, TableBegin and TableEnd properties. For their see their detailed description please refer to the section which deals with the PxEdit component.

1.13. PxLabel – component for data display

The PxLabel component used for data display via the PxWebQuery component. Please, see the picture below:

A screenshot of a web application showing a light blue rectangular box. Inside the box, the word 'Telephone' is followed by the number '0905 148 523'.

The PxLabel component consists of two labels as shown below:

Label + Label

The first label serves as a caption, to the second label the value is entered or in case of linking is automatically loaded from the PxWebQuery component.

If you want to work with the PxLabel component, you have to define it in the InfoAdresar.aspx file as follows:

```
<Prx:PxLabel ID="lblText" runat="server" AddTableRow="True" ></Prx:PxLabel>
```

Link the PxLabel component to the PxWebQuery component in the InfoAdresar.aspx.cs file as follows:

```
lblText.PxDataSource = wquAdresar;  
lblText.FieldName = "TEXT";
```

The alignment of the component is carried once again via the AddTableRow, TableBegin and TableEnd properties. For their see their detailed description please refer to the section which deals with the PxEdit component.

Captions or Grid Titles can be defined centrally, during the definition of the PxWebQuery component, where the value in brackets is the column name:

```
wquAdresar.Columns["Text"].Caption = "Text";
```

Changing captions of the PxLabel component:

Except for the central definition changing of the PxLabel component caption can be carried out in the following way:

```
lblText.Caption = "Text";
```

In the next case contains example with the PxLabel component. Create a new "InfoAdresar" form, as the result, two files "InfoAdresar.aspx" and "InfoAdresar.aspx.cs" shall be created. Prior to work with the newly created form, add properties to the PxSuperGrid component in the existing "Adresar.aspx" form as follows:

```
PxInfoFormName="InfoAdresar.aspx"
```

The PxSuperGrid component definition then shall look like this:

```
<Prx:PxSuperGrid ID="grdAdresar" runat="server"
    PageSize="4"
    PxEditFormName="AddAdresar.aspx"
    PxInfoFormName="InfoAdresar.aspx"
    PxVisibleButtons="SIDE0">
    <SelectedItemStyle BackColor="#ffddff" />
</Prx:PxSuperGrid>
```

The PxInfoFormName shall ensure that clicking the corresponding button in the PxSuperGrid component shall load the "InfoAdresar.aspx" form. After clicking that button, the cursor in the PxWebQuery component shall be set automatically. Then just link the PxLabel components in the "InfoAdresar.aspx" form and the current row of the PxWebQuery component shall be displayed in the PxLabel components.

Thus, the entire form definition in the "InfoAdresar.aspx" file shall look like this:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="InfoAdresar.aspx.cs" Inherits="InfoAdresar" %>

<%@ Register TagPrefix="Prx" Namespace="PxControls" Assembly="PxControls" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <Prx:PxWebQuery ID="wquObec" runat="server" Value="wquObec"/>
        <Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>
    </div>
    </form>
</body>
</html>
```

```

<asp:Label ID="lblAddEditPanel" Font-Size="18pt" Font-Bold="true" runat="server" Text="Label"></asp:Label>
<br /><br />

<Prx:PxLabel ID="lblIDADRESAR" runat="server" AddTableRow="True" TableBegin="True" ></Prx:PxLabel>
<Prx:PxLabel ID="lblName" runat="server" AddTableRow="True" ></Prx:PxLabel>
<Prx:PxLabel ID="lblPravnaForma" runat="server" AddTableRow="True" ></Prx:PxLabel>
<Prx:PxLabel ID="lblCREATE_DATE" runat="server" AddTableRow="True" ></Prx:PxLabel>
<Prx:PxLabel ID="lblInvalid_Adress" runat="server" AddTableRow="True" ></Prx:PxLabel>
<Prx:PxLabel ID="lblADDRESS" runat="server" AddTableRow="True" TableEnd="True"></Prx:PxLabel>
<Prx:PxFlyComboBox ID="fcmBmiesto" runat="server"></Prx:PxFlyComboBox>
<br />
<Prx:PxGreatRepeater ID="gtrTelefon" runat="server"></Prx:PxGreatRepeater>
<br />
<Prx:PxGreatRepeater ID="gtrFax" runat="server"></Prx:PxGreatRepeater>

<br />
<br />

<asp:Button ID="btnClose" runat="server" Text="Zavriet" onclick="ButtonClose_Click" />

</div>
</form>
</body>
</html>

```

Link the PxLabel components to the PxWebQuery component in the "InfoAdresar.aspx.cs" file as follows:

```

protected void Page_Load(object sender, EventArgs e)
{
    lblAddEditPanel.Text = "Detail Address";

    lblIDADRESAR.PxDataSource = wquAdresar;
    lblIDADRESAR.FieldName = "IDADRESAR";

    lblName.PxDataSource = wquAdresar;
    lblName.FieldName = "NAME";

    lblADDRESS.PxDataSource = wquAdresar;
    lblADDRESS.FieldName = "ADDRESS";

    lblPravnaForma.PxDataSource = wquAdresar;
    lblPravnaForma.FieldName = "idpravnaforma";

    lblCREATE_DATE.PxDataSource = wquAdresar;
    lblCREATE_DATE.FieldName = "CREATE_DATE";

    lblInvalid_Adress.PxDataSource = wquAdresar;
    lblInvalid_Adress.FieldName = "Invalid_Adress";

    fcmBmiesto.PxDataSource = wquAdresar;
    fcmBmiesto.FieldName = "miesto";

    gtrTelefon.PxDataSource = wquAdresar;
    gtrTelefon.FieldName = "Telefon";

    gtrFax.PxDataSource = wquAdresar;
    gtrFax.FieldName = "Fax";
}

```

```

}

protected void ButtonClose_Click(object sender, EventArgs e)
{
    this.Response.Redirect("~/Adresar.aspx");
}

```

1.14. PxCheckBoxList – component to view and select values from the list

The PxCheckBoxList component has many similar properties such as PxEdit component. Therefore, it makes no sense to describe them in details, you can read the chapter on the PxEdit component.

Things that were described in detail when dealing with the PxEdit component, we will pass here only by telegraph, briefly.

With the PxCheckBoxList component we can automatically edit a column of a row in the table, which is loaded in PxWebQuery component. We create a file AddAdresar.aspx, and write the following definition of PxCheckBoxList component into the file:

AddAdresar.aspx file:

```
<Prx:PxCheckBoxList ID="chlCar" runat="server" AddTableRow="True" />
```

In AddAdresar.aspx.cs file we define the connection PxCheckBoxListu components with the PxWebQuery component, and its link to a specific table column.

AddAdresar.aspx.cs file:

```

protected void Page_Load(object sender, EventArgs e)
{
    chlCar.PxDataSource = wquAdresar;
    chlCar.FieldName = "Z_COLOR";
}

```

The PxCheckBoxList component supports automatic loading of value from the PxWebQuery component, and automatic storage of selected value into the database after execution of the Post() command on the PxWebQuery component.

Because the PxCheckBoxList component enables to select a single item from the list it is possible to select (enter) on the PxWebQuery component how these conditions are displayed and entered into the database. For the definition the parameter named method *AddParamCheckList* is used.

The *AddParamCheckList* method is a method of PxWebQuery component.

Description of the parameters of the *AddParamCheckList* method:

C# syntax:

```
public void AddParamCheckList(string aFieldName, string aFieldNameKey, string  
aFieldNameValue, string aFieldToView, string aFieldToDB, string  
aPxWebQueryName)
```

```
public void AddParamCheckList(string aFieldName, string aFieldNameKey, string  
aFieldNameValue, string aPxWebQueryName)
```

Description of the parameters:

aFieldName – name of a column in the table on which method *AddParamCheckList* is applied. The best way is to assign the size varchar (500) to this column in the database.

aFieldNameKey – name of the code list column that represents the key of the code list (the name of this column can be detected from the SQL command that is entered into the *PxWebQuery* component in which the code list is loaded)

aFieldNameValue – name of the code list column that represents the value of the code list column (the name of this column can be detected from the SQL command that is entered into *PxWebQuery* component in which the code list is loaded)

aFieldToView – the name of the column which value will be displayed will be displayed in the Grid (*PxSuperGrid*) is entered here when selecting multiple items, these items are displayed separated by commas in the Memo component (TextBox).

aFieldToDB – the name of the column which value will be entered into the database is entered here when selecting multiple items, these items are entered one after the other and are separated by commas.

aPxWebQueryName – name of the *PxWebQuery* component in which the command code list is loaded via SQL, it is suitable to properly identify each column as Key and Value, for greater clarity

Example of *AddAdresar.aspx.cs* file in the *Page_Load* method:

```
wquAkcia.AddParamCheckList("Z_COLOR", "IDCOLOR", "COLOR_NAME",  
"wquListOfColor");
```

After the imposition of the parameter *AddParamCheckList* to the *PxWebQuery* component and connection of the component *PxCheckBoxList*, this component can automatically load data from the parameter and the selected value is automatically saved into the database.



The *AddTableRow*, *TableBegin* and *TableEnd* properties, which are described in detail in the section dealing with the *PxEdit* component, are used once again for the alignment of the components.

Caption name or titles of the columns in the Grid (Title) can be defined centrally during the definition of the *PxWebQuery* component, where the value in square brackets is the name of the column:

```
wquAdresar.Columns["COLOR_NAME"].Caption = "Color Name";
```

Changing the *PxCheckBoxList* (Label) caption:

Besides central definition, the *PxCheckBoxList* component caption can be changed in the following way:


```
chlCar.Caption = "Color Name 2";
```

The `PxCheckBoxList` component has *OnlyCheckBoxList* property, that enables to display the `PxCheckBoxList` component in two ways, if it is not in the state of editing.

If the value *True* is entered into the property *OnlyCheckBoxList* the component is displayed as a `CheckBoxList` as in the state of editing.

If the value *False* is entered into the property *OnlyCheckBoxList* the component is displayed as Memo (TextBox), and selected individual values are entered in a row and separated by a comma as shown in the picture below.



```
chlCar.OnlyCheckBoxList = false;
```

Through the *SeparatorSumText* property of the component `PxCheckBoxList` we can control visual display of the values separator when it is displayed in the Memo component (TextBox) and entered into the database.

The default value is comma `,` or you can type a semicolon `;`.

```
chlCar.SeparatorSumText = ",";
```

The `PxCheckBoxList` component allows filtering of displayed content of the code list through the *Filter* property. The names of the code list columns (fields) of the `PxWebQuery` component are used in the *Filter* property.

```
chlCar.Filter = "CurrentState=1";
```

After entering this command only those items that have in the code list column of the `PxWebQuery` titled "CurrentState" given value "1" are displayed in the `PxCheckBoxList` component.

1.15. `PxRadioButtonList` – component for view and selection of a value from the list

The *PxRadioButtonList* component has many similar properties as the `PxEdit` component.

Therefore, it makes no sense to describe them in details, you can read the chapter on `PxEdit` component.

Things that were described in detail when dealing with the `PxEdit` component, we will pass here only by telegraph, briefly.

With the help of the *PxRadioButtonList* component we can automatically edit the column of the row in the table which is stretched in the `PxWebQuery` component.

We create `AddAdresar.aspx` file and the following definition of the *PxRadioButtonList* component into it:

AddAdresar.aspx file:

```
<Px:PxRadioButtonList ID="rblCar" runat="server" AddTableRow="false"
CaptionOnTop="True" Width="150" />
```

In the file AddAdresar.aspx.cs we define the connection of the *PxRadioButtonList* component with the *PxWebQuery* component , and its link to a specific column in the table.

AddAdresar.aspx.cs file:

```
protected void Page_Load(object sender, EventArgs e)
{
    rblCar.PxDataSource = wquAdresar;
    rblCar.FieldName = "Z_COLOR";
}
```

The *PxRadioButtonList* component supports automatic loading of the value from the parameter of the *PxWebQuery* component and automatic storage of the selected value into the database when the command *Post()* on the *PxWebQuery* component is executed.

The *PxRadioButtonList* component enables selection of only one item from the list, therefore it is possible to specify on the *PxWebQuery* component how these conditions are displayed and entered into the database.

The parameter with the name of the method *AddParamRadioList* is used for definition.

The *AddParamRadioList* method is a method of the *PxWebQuery* component.

Description of the parameters of the *AddParamRadioList* method:

C# syntax:

```
public void AddParamRadioList(string aFieldName, string aDataKey, string
aDataValue, string aFieldToView, string aFieldToDB)
```

Description of the parameters:

aFieldName – name of a column in the table on which method *AddParamRadioList* is applied.

aDataKey – name of the column in which a string of keys separated by semicolons is entered, in this case is "1;2"

aDataValue – name of the column into which a string of values is entered separated by semicolons in the same order as the key values have been entered. In this case it is „Red;Blue“. This text will be displayed also in individual buttons in the *PxRadioButtonList* component

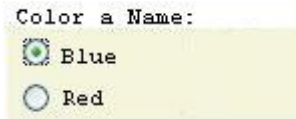
aFieldToView – the name of the column which value will be displayed will be displayed in the Grid (*PxSuperGrid*), in this case the text "Value" and "Key".

aFieldToDB – the name of the column which value will be entered into the database is entered here, in this case the text "Value" and "Key".

Example of entry in the AddAdresar.aspx.cs file in the Page_Load method:

```
wquAkcja.AddParamRadioList("Z_COLOR", "1;2", "Red;Blue", "Value", "Key");
```

After the imposition of the parameter *AddParamCheckList* to the *PxWebQuery* component and connection of the component *PxCheckBoxList*, this component can automatically load data from the parameter and the selected value is automatically saved into the database.



The AddTableRow, TableBegin and TableEnd properties, which are described in detail in the section dealing with the PxEdit component, are used once again for the alignment of the components.

Caption name or titles of the columns in the Grid (Title) can be defined centrally, during the definition of the PxWebQuery component, where the value in square brackets is the name of the column:

```
wquAdresar.Columns["Z_COLOR"].Caption =
    "Color a Name:";
```

Changing the PxRadioButtonList (Label) caption:

Besides central definition, the PxRadioButtonList component caption can be changed in the following way:

```
rblCar.Caption = "Color a Name 2";
```

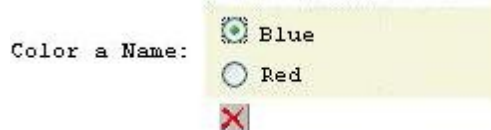
If we want to place the caption of the PxRadioButtonList component over the component body (as it is shown in the image above), we use the property *CaptionOnTop*, the example below:

```
rblCar.CaptionOnTop = true;
```

The PxRadioButtonList component was also equipped with a reset button that is activated by setting the ThreeState property to "true". The reset button is used to reset the contents of the PxRadioButtonList component.

```
rblCar.ThreeState = true;
```

The PxRadioButtonList component with the reset button is shown in the picture below.



1.16. PxChart - the component for displaying and working with charts

The PxChart component is a component of the Px Framework, which enables you to create attractive 3D and 2D charts for your web applications. The PxChart component can be load data directly from the PxWebQuery component, or directly from the XML file. The PxChart contains a wide scale of charts, such as a traditional Bar Graph, Line, Area, Pie, and so on. For the correct functioning of the PxChart component you need to have the Flash Player installed

in your web browser.

If you want to work with the PxChart component, you have to define it in the *.aspx file as follows:

```
<Prx:PxChart ID="Chart" runat="server"></Prx:PxChart>
```

If you want the PxChart component to be visible in the aspx files, define the "Prx" prefix related to the assembly of the PxChart component in the header of the file. Do it as follows:

```
<%@ Register TagPrefix="Prx" Namespace="PxChart" Assembly="PxChart" %>
```

Then you can work with the PxChart component in the *.aspx.cs file as follows. First, load to the PxWebQuery component data from the database, and then link it to the PxChart component. At first link the PxWebQuery component to the PxChart component via the *PxDataSource* property. Then, define through the *FieldName* and *FieldValue* properties of the PxChart component linking of the PxWebQuery component columns to individual axes, X and Y of the PxChart component. More in the example below:

```
Chart.PxDataSource = wquStatistics;  
Chart.FieldName = "name";  
Chart.FieldValue = "count";
```

Charts description can be defined via the *Caption* and *SubCaption* property. More in the example below:

```
Chart.Caption = "Statistics of genres ";  
Chart.SubCaption = "A summary of genres for all CD titles ";
```

Similarly chart type can be defined via the *TypeChart* property. Individual chart types are defined in the enumerative type *PxChart.enumTypeCharts*. More in the example below:

```
Chart.TypeChart = PxChart.PxChart.enumTypeCharts.tcColumn3D;
```

The whole source code of the example that was discussed here, together with data loading to the PxWebQuery component and linking to the PxChart component and selecting the chart type, is shown below in the attached list of the source code.

```
protected void Page_Load(object sender, EventArgs e)  
{  
    if (wquStatistics.Active == false)  
    {  
        wquStatistics.ConnectionString = AppConst.ConnectionString;  
        wquStatistics.SQLSelect = @"select CdTitul.IdZaner as idzaner,COUNT(*) as count,  
                                         Zaner.Name as name from CdTitul, Zaner  
                                         where CdTitul.IDZaner=Zaner.IdZaner  
                                         group by CdTitul.IDZaner";  
    }  
}
```

```

wquStatistics.Open();

wquStatistics.Refresh = true;
wquStatistics.IntervalQueryRefresh = 30;

wquStatistics.Columns["IdRecNum"].Caption = "Id";
wquStatistics.Columns["Pocet"].Caption = "Count genres";
wquStatistics.Columns["name"].Caption = "Name of genres";
wquStatistics.Columns["idzaner"].Caption = "Count genres II.";
}
Chart.PxDataSource = wquStatistics;
Chart.FieldName = "name";
Chart.FieldValue = "count";

Chart.Caption = "Statistics of genres";
Chart.SubCaption = "A summary of genres for all CD titles";
//Chart.Width = 800;
//Chart.Height = 800;

Chart.TypeChart = PxChart.PxChart.enumTypeCharts.tcColumn3D;
Chart.AddHeader("canvasBgColor", "F6DFD9");
Chart.AddHeader("divlinecolor", "F47E00");
}

```

The AddHeader method, enables to enter to the PxChart component various elements with the definition of value that can affect form and shape of the output graph.

Description of the parameters of the AddHeader method:

C # syntax:

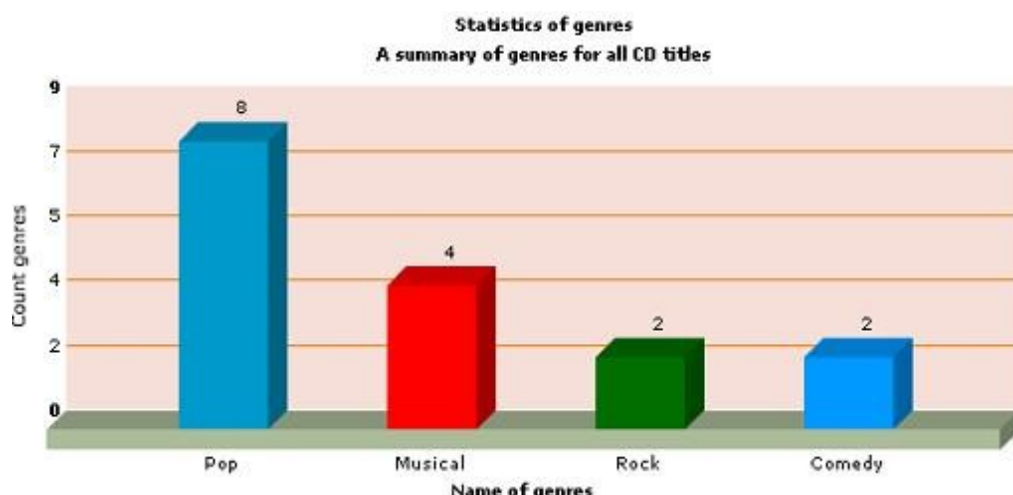
```
public void AddHeader(string Tag, string Value)
```

Description of the parameters:

Tag – enter the tag name to this variable

Value – enter the tag value to this variable

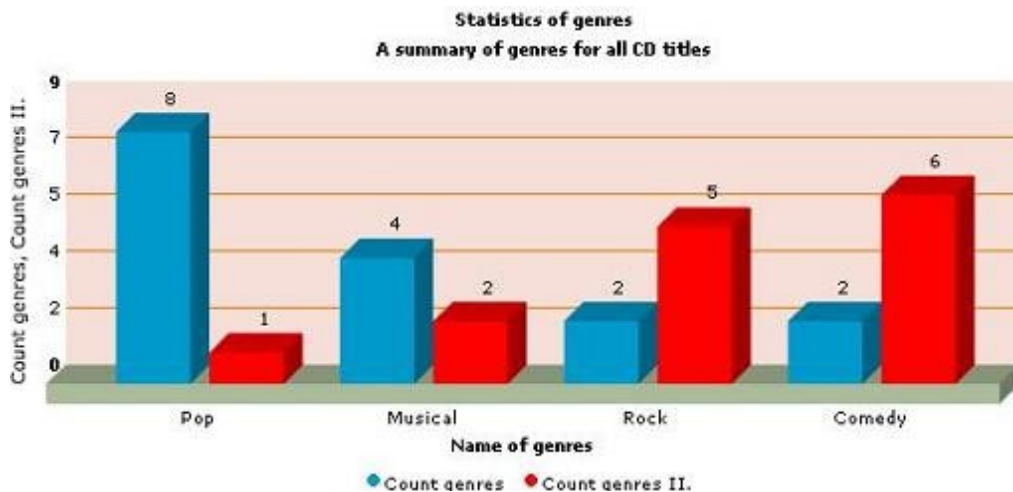
Picture of the resulting graph looks like this, see the image below:



If you want to work with the series chart, i.e. more charts next to each other, you can do it via the *FieldValue* property of the PxChart component. Carry out a so-called overload of the *FieldValue* property by consequently entering more column names from the PxWebQuery component, which are separated by a semicolon. Yet of course you should choose a serial chart in TypeChart property, for example tcMSColumn3D. Each serial chart starts with the tcMS ... prefix.
More in the example below:

```
Chart.PxDataSource = wquStatistics;
Chart.FieldName = "name";
Chart.FieldValue = "count;idzane";
Chart.TypeChart = PxChart.PxChart.enumTypeCharts.tcMSColumn3D;
```

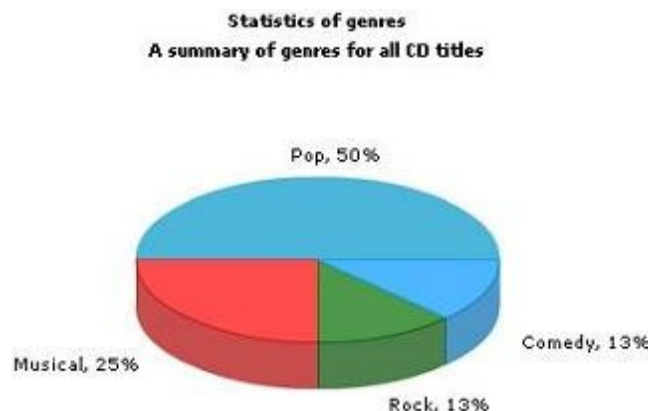
Picture of the resulting graph looks like this, see the image below:



If you want to create a pie-chart, you can do it by entering the “tcPie3D” the chart type to the *TypeChart* property. More in the example below:

```
Chart.PxDataSource = wquStatistics;
Chart.FieldName = "name";
Chart.FieldValue = "count";
Chart.TypeChart = PxChart.PxChart.enumTypeCharts.tcPie3D;
```

Picture of the resulting graph looks like this, in the example below:



If you want to load data to the PxChart component from the xml file and not from the PxWebQuery component, you can do so by means of the *ChartXML* property. In case you load the data to the PxChart component via the *ChartXML* property, all properties of the PxChart component will be blocked, except for the *TypeChart* property, where the chart type is defined. The picture below contains an example of the XML file model:

```
<graph caption='Statistics of genres' subCaption='A summary of genres for all CD titles'
showPercentageInLabel='1' pieSliceDepth='25' decimalPrecision='0' showNames='1'
pieRadius='100' >
  <set name='Pop' value='8' color='0099CC' />
  <set name='Muzikal' value='4' color='FF0000' />
  <set name='Rock' value='2' color='006F00' />
  <set name='Komédia' value='2' color='0099FF' />
</graph>
```

In the <graph> section enter elements which describe more closely the resulting graph. They are for example the following tags: *Caption*, *Subcaption*, *showNames* and so on. In the <set> section enter name and value of the output graph. The PxChart component is an encapsulation of the free FusionCharts v.2.0 component. For more detailed description of individual elements that effect chart display and description of output XML file creation please refer to the documentation for the FusionCharts v.2.0 components, or go to our web:

<http://www.proxima-soft.sk/data/FusionChartsFree/Contents/menu.html>

1.17. PxFilterView - visual component for filtering the table data contents in the PxWebQuery component

The PxFilterView component enables filtering table data contents in the PxWebQuery component. If you want to work with the PxFilterView component, you have to define it in the *.aspx file as follows:

```
<Prx:PxFilterView ID="filterCdTitul" runat="server" />
```

Parameter, selection criteria are entered to the PxFilterView component by means of the AddParamFilter method. The PxFilterView component is dynamically compiled according to these parameters. More in the picture below:

Juridical Form:	Juridical Person ▼
Region:	Banskobystrický ▼
District:	Brezno ▼
Municipality:	Hronec ▼
Specialization:	Choose value ▼
Search string:	pe sk

Enter to thus depicted PxFilterView component, the selection criteria from which the resulting SQLScript shall be compiled, which in turn shall be assigned to the PxWebQuery component.

For the PxFilterView to be depicted in such form as is the picture, specify the following parameters:

```
filterSposOsoba.AddParamFilter("IDTypOsoba", "<b>Juridical Form:</b>");  
filterSposOsoba.AddParamFilter("Miesto", "<b>Region:</b>", "IDKraj1");  
filterSposOsoba.AddParamFilter("Miesto", "<b>District:</b>", "IDOKRES1");  
filterSposOsoba.AddParamFilter("Miesto", "<b>Municipality:</b>", "IDOBEC1");  
filterSposOsoba.AddParamFilter("Specializacia", "<b>Specialization:</b>", "SPECIALIZATION1");  
filterSposOsoba.AddParamFilter("FIRST_NAME;LAST_NAME", "<b>Search string:</b>");  
  
filterSposOsoba.OrigSQLSelect = "select * from SposOsoba";
```

Definition of the AddParamFilter method:

```
public void AddParamFilter(string aFieldName, string aCaption)
```

```
public void AddParamFilter(string aFieldName, string aCaption, string aParentFieldName)
```

```
public void AddParamFilter(string aFieldName, string aCaption, string aParentFieldName, string aChooseValueText)
```

Description of the parameters:

aFieldName – Name of the column in the table of the PxWebQuery component according to which table data contents of the PxWebQuery component shall be filtered

aCaption – row caption PxFilterView component

aParentFieldName – This column is filled in only if the AddParamFlyComboBox or AddParamGreatWebQuery parameter is applied to the column defined in the aFieldName parameter. It is necessary to bear in mind that if AddParamFlyComboBox or AddParamGreatWebQuery parameters are applied to the aFieldName, this field, or column becomes a merge column, that is it can encompass several fields.

Such field can be called a child, and the columns that are merged - Parent. In the aParentFieldName parameter enter a specified name of the parent field according to which table data contents of the PxWebQuery component shall be filtered

aChooseValueText – here, enter the text that shall be displayed in the PxFilterView component at the particular display line, as far as the selection shall be carried out in the list (ComboBox). The text shall replace the "Choose value" text . For more information please refer to the picture above.

The PxFilterView component works so that in case of choosing from the list or typing some text and confirming it by the "Enter button", it generates its own SQL command. Since it is linked to the particular PxWebQuery component, this new SQL command is assigned to the PxWebQuery component and the reopen of the PxWebQuery component is carried out after each change. Thus, when changing the selection criteria of the PxFilterView component, the data contents of the PxWebQuery table are also changed.

If the PxWebQuery component contains columns of the string type to which none of the parameters is applied (AddParamKey, AddParamWebQuery etc..), these columns can be merged or separated by a semicolon during definition of the parameters. In case of searching the entered term is searched for in both of the defined columns. More in the example below:

```
filterSposOsoba.AddParamFilter("FIRST_NAME;LAST_NAME", "<b>Search string:</b>");
```

Then if you enter a word in the "Search string" line of the PxFILTERVIEW filter, this word shall be searched for in both columns, thus filtering the final shape of the table.

Are you interested in the criteria according to which criteria the filtering part of the PxFILTERVIEW component formed? The TextBox component where the text is entered as the selection criterion is displayed in the edit part for the columns to which none of the parameters is applied. If any of the AddParamKey, AddParamWebQuery, etc. parameters is applied to the given column, the TextBox component is replaced by the DropDownList component with a filled list contents of the component. The PxFILTERVIEW component can read the contents of tables and their linking (individual parameters), and thus compile the resulting panel of the PxFILTERVIEW component.

The PxFILTERVIEW component, in addition to traditional input parameters through *AddParamFilter()*, also supports other input parameters that are added to the resulting SQL query, which can serve the complex requirements for filtration through the PxFILTERVIEW component. Additional parameters are entered via the *RebuildSqlSelect()* method.

Description of the RebuildSqlSelect method:

C# syntax:

```
public void RebuildSqlSelect(string AddSelect, string AddFrom, string AddWhere)
```

Description of the parameters:

AddSelect – the string to be added to the section between **select** and **from** of the SQL command

AddFrom – the string to be added to the section between **from** and **where** of the SQL command

AddWhere – the string to be added to the section after **where** of the SQL command

The method RebuildSqlSelect() has a memory effect, i.e. the PxFILTERVIEW component remembers recently entered state, and at every change of the PxFILTERVIEW component these entered and remembered parameters are compiled to the resulting SQL statement, which is then written to the PxWebQuery component and the procedure ReOpen() of this component is invoked.

If we want to delete these parameters from the PxFILTERVIEW component, we have to call the method RebuildSqlSelect() again with empty parameters.

Here in the next example we list the use of the RebuildSqlSelect() method.

The file *.aspx in definition of the components PxFILTERVIEW we entered the next filter element that is a part of the filter PxFILTERVIEW component.

```
<Prx:PxFILTERVIEW ID="filterSposOsoba" runat="server" TableEnd="false" />
  <tr><td></td><td>
    <asp:CheckBox ID="chkFyzickaOsoba" Text="Show only physical person"
      runat="server" Checked="false" AutoPostBack="true"
      oncheckedchanged="chkFyzickaOsoba_CheckedChanged" />
  </td></tr>
</table>
```

The next element is the CheckBox component. If we checked this component, only natural person is filtered from the list. This component has AutoPostBack set to true, which means that in case of the change of its condition, the method registered in the *oncheckedchanged* property is called. This method is called "chkFyzickaOsoba_CheckedChanged". The body of this method is as follows:

```
protected void chkFyzickaOsoba_CheckedChanged(object sender, EventArgs e)
{
    string AddWhere = String.Empty;
    if (chkFyzickaOsoba.Checked == true)
    {
        AddWhere = "IDTypOsoba=2";
    }
    filterSposOsoba.RebuildSqlSelect(String.Empty, String.Empty,
        AddWhere);
}
```

The part of the SQL command is entered into the *AddWhere* parameter that will allow us to choose only natural person from the list. This parameter is entered into the body of the the PxFILTERView component through the *RebuildSqlSelect()* method, by which a new SQL command is generated.

If the component CheckBox were unchecked, the method "chkFyzickaOsoba_CheckedChanged" is called again, where the parameter *AddWhere* would be empty, and the list with all persons would return.

You must notice the PxFILTERView component, which TableEnd property is set to „false“, and that allows us to insert the CheckBox component into a row in the table, finish the table with the tag "</ table>" and align it with the PxFILTERView component. The resulting image of the PxFILTERView component is shown below:

If we want to monitor the output SQL command that is generated in the body of the PxFILTERView component when it is changed, we can use the "MonitorSQLSelect" event in the PxFILTERView component.

It depends only on us whether we can correctly fill in all the variables that are entered via the AddParamFilter method. Usage of the PxFILTERView component can be found in the **510_CD_Titul_Px_Fram_Ajax** example which is, together with other examples, part of every installation package of the Px Framework.

1.18. PxUploader - component for uploading binary and text files to the server

The PxUploader component secures uploading of binary or text files from a client browser to the server. The PxUploader works with browsers like Windows Internet Explorer, Firefox, etc.. For the correct functioning of the PxUploader component you need to have the JavaScript enabled and the Flash Media Player installed in your web browser.

If you want to work with the PxUploader component, you have to define it in the *.aspx file as follows:

```
<Prx:PxUploader ID="Uploader" runat="server" />
```

If you want the PxUploader component to be visible in the aspx files, define the "Prx" prefix related to the assembly of the PxUploader component in the header of the file. Do it as follows:

```
<%@ Register TagPrefix="Prx" Namespace="PxUploader" Assembly="PxUploader" %>
```

Then you can work with the PxUploader component in the *.aspx.cs file as follows. First, use the *UploadDir* property to define in the PxUploader component the path to the directory where the files shall be stored. Then enter to the *IdUploadFile* property the file Id which shall be attached to the file name as a prefix for the better unique file identification. The *MultiFiles* property enables you to define whether one or several files can be selected for uploading to the server. More in the example below:

```
Uploader.UploadDir = @"uploads\data";  
Uploader.IdUploadFile = 7777;  
Uploader.MultiFiles = false;
```

The *FileDescription* and *FileExt* properties enable you to control definition of the filter in the OpenFileDialog form, where the file selection according to the file extension is carried out.

```
Uploader.FileDescription = "Image Files";  
Uploader.FileExt = "*.jpg;*.png;*.gif;*.bmp;*.jpeg;*.zip;*.rar";
```

The *SizeLimit* property serves for definition of the maximum file size in bytes, which can be uploaded to the server. The PxUploader component allows to upload files up 2GB to the server. For more information please refer to the example below, which with the current pre-set allows to upload a file with the maximum size of 10 MB:

```
Uploader.SizeLimit = 10485760;
```

The default file size that can be uploaded to the server, is set to 4MB in the IIS. This can be changed through the settings in the web.config file, under the <system.web> section in the *maxRequestLength* property. See the example below:

```
<system.web>  
  <httpRuntime executionTimeout="110" maxRequestLength="2097151"  
    requestLengthDiskThreshold="80" useFullyQualifiedRedirectUrl="false"  
    minFreeThreads="8" minLocalRequestFreeThreads="4" appRequestQueueLimit="5000"  
    enableKernelOutputCache="true" enableVersionHeader="true"  
    requireRootedSaveAsPath="true" enable="true" shutdownTimeout="90"  
    delayNotificationTimeout="5" waitChangeNotification="0" maxWaitChangeNotification="0"  
    enableHeaderChecking="true" sendCacheControlHeader="true"  
    apartmentThreading="false"/>  
</system.web>
```

If you want to upload files of 2GB to the server, in the IIS version 7, set the *maxAllowedContentLength* property in the <system.webServer> section of the web.config file. Of

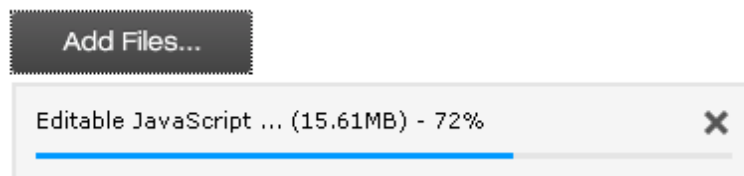
course, it is necessary to remember to adjust the *SizeLimit* property and set the 2GB size there. More in the example below:

```
<system.webServer>
  <security>
    <requestFiltering>
      <requestLimits maxAllowedContentLength="2097151000" />
    </requestFiltering>
  </security>
</system.webServer>
```

The *ButtonText* property allows to change the name of the button of the PxUploader component. More in the example below:

```
Uploader.ButtonText = "Add Files...";
```

The image of PxUploader component during the uploading of a file to the server looks like this, see picture below:



The PxUploader component events:

The PxUploader component generates two events. These are the CompleteUpload and the AllCompleteUpload events. The CompleteUpload event is called in after the complete file uploading to the server directory, which was specified in the *UploadDir* property. The called in method body also contains the uploaded variables such as file name and file path where the file has been saved. The AllCompleteUpload event is called in after the CompleteUpload event, usually when all files have been uploaded to the server. Definition of individual events, and their linking to the methods shall be described, for more information please refer to the example below:

```
protected void Page_Load(object sender, EventArgs e)
{
    Uploader.IdUploadFile = 7777;
    Uploader.UploadDir = @"uploads\data";
    Uploader.ButtonText = "Add Files...";
    Uploader.CompleteUpload += new
    PxUploader.CompleteUploadEventHandler(Uploader_CompleteUpload);
    Uploader.AllCompleteUpload += new
    PxUploader.AllCompleteUploadEventHandler(Uploader_AllCompleteUpload);
}

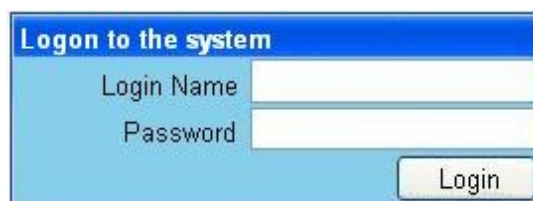
void Uploader_AllCompleteUpload(object sender, EventArgs e)
{
    string a = "";
}

void Uploader_CompleteUpload(object sender, string FileName, string Path, EventArgs e)
{
}
```

```
string a = "";  
}
```

1.19. PxLogin – component for authorization and logging into the application

The PxLogin component is used for authentication and logging into the application. The component consists of two TextBox into which you enter the user name and password, and the button for confirmation and login to the application as it is shown in the picture below:



We create a Login.aspx file and the following definition of the PxLogin component into it:

Login.aspx file:

```
<Prx:PxLogin ID="lgLogin" LoginFormName="Login.aspx"  
FirstFormAfterLogin="Adresar.aspx" runat="server" />
```

As it is seen from the definition of the PxLogin component two properties are defined at the same time with its definition. These properties are *LoginFormName* and *FirstFormAfterLogin*. We put the PxLogin component into everyone form, and after we have done it, the PxLogin component will make the guard and see if we are logged into the system. If you we are not logged in, the component PxLogin redirects us from the original form that is specified in the property *LoginFormName*, in this case it is the Login.aspx form. If we are logged in, the PxLogin component can determine it from its structure and we remain in its original form.

After we have entered name and password into each TextBox we confirm it with the button "Login" and the PxLogin component calls event "ValidateLogin". We assign the procedure where the event will be processed to this event of the PxLogin component in the body Page_Load method.

Login.aspx.cs file:

```
protected void Page_Load(object sender, EventArgs e)  
{  
    ...  
    lgLogin.ValidateLogin += new  
        ValidateLoginEventHandler(lgLogin_ValidateLogin);  
    ...  
}
```

The method `lgLogin_ValidateLogin` contains the input parameters, i.e. the name and the password

that was entered in the form of the PxLogin component. Besides these input parameters it includes output parameters and they are *AcceptLogin* and *ErrorMsg*.

Briefly, all authorization runs in the method **lgLogin_ValidateLogin** and if the username and password is correct, we write true into the parameter *AcceptLogin*. If it is not correct we write false. In case we have entered false we also write an error message into the *ErrorMsg* parameter.

In case the name and password are correct and comply with the authorization and we have entered true into the *AcceptLogin* parameter, the PxLogin component automatically redirects us to a form which name is entered in the *FirstFormAfterLogin* property.

In this case it is the Adresar.aspx form. See the example below:

```
void lgLogin_ValidateLogin(object sender, EventArgs e, string LoginName,
    string Passwd, ref bool AcceptLogin, ref string ErrorMsg)
{
    string GroupUsers = String.Empty;
    Int32 IdUser = -1;
    AcceptLogin = IsValidLogin(LoginName, Passwd, ref GroupUsers,
        ref IdUser);
    if (AcceptLogin == true)
    {
        lgLogin.GroupUsers = GroupUsers;
        lgLogin.IDUser = IdUser.ToString();
        Session["GroupUsers"] = GroupUsers.RemoveDiacritics();
    }
    else
    {
        ErrorMsg = "Nezadali ste správne meno alebo heslo !!!";
        lgLogin.GroupUsers = String.Empty;
        lgLogin.IDUser = "-1";
        Session["GroupUsers"] = null;
    }
}
```


2. Installing Px Framework

2.1 Installation Px Framework on the client computer

Create a new Web project in Visual Studio. Unpack the file PxControls_[DbType].zip. After unpacking the file, should we should find the directories "bin" and "images". The directory "bin" of our project, copying the files from the directory "PxControls \ bin". As in our project we have not a directory "bin", then create it, and copy the files in it "PxControls \ bin". Create another directory, and a directory entitled "images", and copy the files in it "PxControls\images".

All *. dll files in the directory "bin" add a reference via the "Add Reference ...".

If we want to Px Framework components were visible in the aspx file, we can define the header file "Prx" prefix. This will make the following:

```
<%@ Register TagPrefix="Prx" Namespace="PxControls" Assembly="PxControls" %>
```

2.2. Installing the Px Framework on the server

When you install Px Framework, we will behave similarly as for installation on the client computer. Read more in "Installing Px Framework on the client computer.

3. Detailed description of programming using the Px Framework components

3.1. The PxWebQuery – component for working with the Oracle database, the detailed description

3.1.1. Data loading from the Oracle database by means of the PxWebQuery components

The PxWebQuery component enables easy data loading from the database and facilitates working with thus loaded data.

In our example, the ConnectionString is linked to the ConnectString, and the SQL command relating to the data you want to load is linked to the SQLSelect property. Data loading is then executed by means of the "Open" command.

```
protected void Page_Load(object sender, EventArgs e)
{
    if ((!IsPostBack)&&(wquAdresar.Active==false))
    {
        wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=x;";
        wquAdresar.SQLSelect = "select idadresar , name, address, create_date from Adresar";

        wquAdresar.Open();
    }
}
```

3.1.2. Program inserting of a new row into the database by means of the PxWebQuery component

To insert new rows with the help of the PxWebQuery component you need to create a sequence, from which the PxWebQuery component shall load a unique ID for its primary key. The name of the sequence shall be created in the following way.

SEQ_“name of primary key column name “

Therefore, when you create new tables, do not name the primary key column as 'ID' only, but as ID + "table name".

In our example, for the data inserting into the database to work as well, create the sequence by the SQL command:

```
CREATE SEQUENCE SEQ_IDADRESAR
START WITH 1
INCREMENT BY 1
MAXVALUE 1E18
NOMINVALUE
NOORDER
NOCACHE
NOCYCLE;
```

Attention !!! Of course on other database platforms the sequence is created in a different way, for more details please refer to the Part Four of "Specification of using the Px Framework on different database platforms (Oracle, MS SQL, MySQL, Firebird, InterBase).

Therefore, when you enter the "wquAdresar.Insert ()" command, the PxWebQuery component automatically calls the sequence, which generates a new ID for the primary key . You do not have to worry about generation and assigning of a unique key to the field where the primary key ID is entered.

For more information, please refer to the example below:

```
wquAdresar.Insert();
wquAdresar.Fields["name"] = "Grof Monte Christo";
wquAdresar.Fields["idpravnaforma"] = "Physical Person";
wquAdresar.Fields["address"] = "Karlovarska 2";
wquAdresar.Fields["create_date"] = "04.07.2010";
wquAdresar.Fields["invalid_adress"] = "N";
wquAdresar.Post();
```

3.1.3. Program editing of the existing row in the database by means of the PxWebQuery component

You can edit and modify the existing row just like in the previous example, just replace the *Insert* command by the *Edit* command.

```
wquAdresar.Edit();
wquAdresar.Fields["name"] = "Juraj Peter Tarnoci";
wquAdresar.Fields["idpravnaforma"] = "Physical person";
wquAdresar.Fields["address"] = "Fatranska 22";
wquAdresar.Fields["create_date"] = "12.10.2010";
wquAdresar.Fields["invalid_adress"] = "N";
wquAdresar.Post();
```

If you know the ID of the row, which you want to edit, you can find it in the following way:

```
if (wquAdresar.FindByIdPK(6)!=-1)
    lblAdresar4.Text = wquAdresar.Fields["name"].ToString();
```

In our case, the number 6 is the column ID of the primary key row, which you are looking for and the search result (i.e. value of the column "name") is saved in the Label component.

3.1.4. Program deleting of the existing row in the database by means of the PxWebQuery component

Deleting of the existing row can be carried out via the *Delete* command of the PxWebQuery component. The FindByIdPK command enables you to find the row you want to delete, then use the *Delete* command to delete it.

```
if (wquAdresar.FindByIdPK(6)!=-1)
    wquAdresar.Delete();
```

3.1.5. Loading values from the PxWebQuery component via the while cycle

If you want to load some values for all rows in the PxWebQuery component, you can do it via the while cycle. This way of working with the PxWebQuery component is very similar to working with the TTable components in Delphi programming language. There are two possible ways of loading data. The first way is to download from the first line to the last. The second method of loading is from the last line to the first line.

Further here there is an extract of the code where the values are loaded from the first to the last row and the result is then assigned to the Label component.

```
wquAdresar.First();
string s=String.Empty;
while (!wquAdresar.Eof())
{
    s = s+", "+wquAdresar.Fields["name"].ToString();
    wquAdresar.Next();
}
lblAdresar.Text = s;
```

The other example contains an extract of the code where the values are loaded from the last to the first row and the result is then assigned to the Label component.

```
wquAdresar.Last();
s = String.Empty;
while (!wquAdresar.Bof())
{
    s = s + ", " + wquAdresar.Fields["name"].ToString();
    wquAdresar.Prior();
}
lblAdresar2.Text = s;
```

The `wquAdresar.Eof()` command returns true value if the `Next()` command cannot move us to the next following row. Similarly, the `wquAdresar.Bof()` command returns the true value if the `Prior()` command cannot move us to the previous row, because we are already on the first row.

3.1.6. Row search in the PxWebQuery component according to the value entered and the name of the column where the search shall be carried out

If you need to search in the table of the PxWebQuery component, you can do it via the *Find* command, see the example below:

```
if(wquAdresar.Find("%sko%", "name")!=-1)
    lblAdresar3.Text = wquAdresar.Fields["name"].ToString();
```

Definition of the *Find* command:

```
public Int32 Find(string SearchValue,string sFieldName);
```

Description of the *Find* command parameters:

The return value of Find command is ItemIndex (row index, which corresponds to the selection criteria) of the PxWebQuery component, if no row is found, the return value then is -1.

SearchValue – value, according to which the search is, in the column specified in the parameters sFieldName. SearchValue value, we can arrange for example. percent "%" functionality is similar to the SQL command "Like"
sFieldName – column name, which is the search string

Then, the PxWebQuery component enables search according to the primary key, please refer to the example below:

```
if (wquAdresar.FindByIdPK(6)!=-1)  
    lblAdresar4.Text = wquAdresar.Fields["name"].ToString();
```

The return value of the FindByIdPK command is ItemIndex (row index, which corresponds to the selection criteria) of PxWebQuery component, if no row is found, the return value then is -1. The valid value of the primary key is entered as a search parameter.

3.1.7. Setting of the cursor position in the row in the PxWebQuery and PxSuperGrid components

Setting of the cursor position in the PxSuperGrid component can be carried out by assigning the 0..RecordCount-1 value to the ItemIndex property of the PxWebQuery component to which the PxSuperGrid component is linked.

```
if (wquAdresar.FindByIdPK(6) != -1)  
    wquAdresar.ItemIndex = wquAdresar.FindByIdPK(6);
```

3.1.8. Selection of several rows that match the selection criteria

The PxWebQuery component allows also to select more than one row, if it is possible due to the selection criteria .

The wquAdresar.DataSource () command returns the DataTable object. The DataTable object supports selection, row selection. For more information please refer to the example below:

```
//Statement IDs lines that match selection criteria  
DataRow[] dr = null;  
dr = wquAdresar.DataSource().Select("name like '%a%'", "name");  
  
ArrayList aList = new ArrayList();  
Int32 iOrdValue = 0;
```

```

for (Int32 iCykl = 0; iCykl < dr.Length; iCykl++)
{
    //Here are performed DISTINCT
    if (!aList.Contains(dr[iCykl].ItemArray.GetValue(iOrdValue).ToString()))
    {
        aList.Add(dr[iCykl].ItemArray.GetValue(iOrdValue).ToString());
    }
}
string s = String.Empty;
for (Int32 iCykl2 = 0; iCykl2 < aList.Count; iCykl2++)
{
    s = s + ", " + aList[iCykl2].ToString();
}
lblAdresar5.Text = s;

```

This routine shall bring us the list of all IDs of the rows, which match the selection criteria, in this case, these are the rows, which contain the letter "a" in the text of "name" column.

3.1.9. Events of the PxWebQuery component

The PxWebQuery component contains events that are triggered by the change in state of the component. Each event of the PxWebQuery component can be defined in the "Page_Load" procedure of the form, where this event shall be triggered.

The definition of the events looks like as follows:

```

if ((!IsPostBack)&&(wquAdresar.Active==false))
{
    ....
    wquAdresar.AfterScroll += new PxControls.AfterScrollEventHandler(wquAdresar_AfterScroll);
    wquAdresar.Open();
    ....
}
else
{
    wquAdresar.AfterScroll += new PxControls.AfterScrollEventHandler(wquAdresar_AfterScroll);
}

```

Definition of the "wquAdresar_AfterScroll " procedure, which is called the " AfterScroll " event shall be as follows:

```

void wquAdresar_AfterScroll(object sender, EventArgs e)
{
    lblAdresar.Text = wquAdresar.Fields["name"].ToString();
}

```

Each change of the current row cursor, either in the PxWebQuery component or in the PxSuperGrid component, which is linked to the PxWebQuery component, triggers the "AfterScroll" event. The "Name" column value of the current row is registered in the Label component for the each event triggered.

The following table contains a summary of all events of the PxWebQuery component.

Event Name	Event Description
BeforeInsert	This event is triggered before the execution of the PxWebQuery.Insert () command;
AfterInsert	This event is triggered after the execution of the PxWebQuery.Insert () command;
BeforeEdit	This event is triggered before the execution of the PxWebQuery.Edit () command ;
AfterEdit	This event is triggered after the execution of the PxWebQuery.Edit () command;
BeforePost	This event is triggered before the execution PxWebQuery.Post () command;
AfterPost	This event is triggered after the execution of the PxWebQuery.Post () command;
BeforeValidPost	This event is triggered during the verification of the specified items before the actual data storing by Post () method;
BeforeOpen	This event is triggered before the execution of the PxWebQuery.Open () command;
AfterOpen	This event is triggered after the execution of the PxWebQuery.Open () command;
BeforeScroll	This event is triggered before the change of the row cursor position in the PxWebQuery component
AfterScroll	This event is triggered after the change of the row cursor position in the PxWebQuery component
BeforeDelete	This event is triggered before the execution of the PxWebQuery.Delete() command;
AfterDelete	This event is triggered after the execution of the PxWebQuery.Delete () command;
BeforeCancel	This event is triggered before the execution of the PxWebQuery.Cancel () command;
AfterCancel	This event is triggered after the execution of the PxWebQuery.Cancel () command;
StateChange	This event is triggered during the change in the state of the PxWebQuery component, usually after the execution of the Insert, Edit, Delete, Post, and other commands.
AfterCreateField	This event is triggered after the set up of all the table columns (fields) of the PxWebQuery component. This event is used mainly to create a new dynamic column (field) of the PxWebQuery component table.

3.1.10. Creation of a new dynamic column (field) in the PxWebQuery component table

If, during the work with the PxWebQuery component you need to create a new dynamic column, you can it via the "AfterCreateField" event.

```
if ((!IsPostBack)&&(wquAdresar.Active==false))
{
    ....
    wquAdresar.AfterCreateField += new AfterCreateFieldEventHandler(AfterCreateField_Adresar);
    wquAdresar.Open();
    ....
}
else
{
    wquAdresar.AfterCreateField += new AfterCreateFieldEventHandler(AfterCreateField_Adresar);
}
```

In the "AfterCreateField_Adresar" procedure, create a new column and then assign it a caption. When creating a column, it is necessary to enter such column name, which does not exist in the PxWebQuery component table.

```
protected void AfterCreateField_SposOsoba(object sender, EventArgs e)
{
    wquSposOsoba.DataSource().Columns.Add("FullName", Type.GetType("System.String"),
    "ISNULL(Last_Name+' ') + "+ISNULL(First_Name+' ') + "+ISNULL(' '+Titl1,")");
    wquSposOsoba.DataSource().Columns["FullName"].Caption="Person name";

    wquSposOsoba.DataSource().Columns.Add("FullNameAll", Type.GetType("System.String"),
    "ISNULL(Last_Name+' ') + "+ISNULL(First_Name+' ') + "+ISNULL(' '+Titl1,")");
    wquSposOsoba.DataSource().Columns["FullNameAll"].Caption="Last Name, First Name, Title";
}
```

The following functions can be used during the creation of the expression, during the creation of a dynamic column:

CONVERT

Description	Converts given expression to a specified .NET Framework Type.
Syntax	Convert(expression, type)
Argument	Expression -- The expression to convert. Type -- The .NET Framework type to which the value will be converted.

Example: myDataColumn.Expression="Convert(total, 'System.Int32')"

All conversions are valid with the following exceptions: Boolean can be coerced to and from Byte, SByte, Int16, Int32, Int64, UInt16, UInt32, UInt64, String and itself only. Char can be coerced to and from Int32, UInt32, String, and itself only. DateTime can be coerced to and from String and itself only. TimeSpan can be coerced to and from String and itself only.

LEN

Description	Gets the length of a string
Syntax	LEN(expression)
Argument	<i>Expression</i> -- The string to be evaluated.

Example: myDataColumn.Expression="Len(ItemName)"

ISNULL

Description	Checks an expression and either returns the checked expression or a replacement value.
Syntax	ISNULL(expression, replacementvalue)
Argument	Expression -- The expression to check. Replacementvalue -- If expression is a null reference (Nothing), replacementvalue is returned.

Example: myDataColumn.Expression="IsNull(price, -1)"

IIF

Description	Gets one of two values depending on the result of a logical expression.
Syntax	IIF(expr, truepart, falsepart)
Argument	expr -- The expression to evaluate. truepart -- The value to return if the expression is true. falsepart -- The value to return if the expression is false.

Example: myDataColumn.Expression = "IIF(total>1000, 'expensive', 'dear')"

TRIM

Description	Removes all leading and trailing blank characters like\r,\n,\t, ' '
Syntax	TRIM(expression)
Argument	<i>expression</i> -- The expression to trim.

SUBSTRING

Description	Gets a sub-string of a specified length, starting at a specified point in the string.
Syntax	SUBSTRING(expression, start, length)
Argumenty	expression -- The source string for the substring. start -- Integer that specifies where the substring begins. length -- Integer that specifies the length of the substring.

Example: myDataColumn.Expression = "SUBSTRING(phone, 7, 8)"

Note You can reset the Expression property by assigning it a null value or empty string. If a default value is set on the expression column, all previously filled rows are assigned the default value after the Expression property is reset.

3.1.11. The "ReOpen" procedure of the PxWebQuery component and data re-load into the PxWebQuery component

If you want to reload data to the PxWebQuery component, to carry-out the so-called "refresh", you do not have close and open the PxWebQuery component, you can do it via the "ReOpen ()" procedure. The "ReOpen ()" procedure is very fast, and it is recommended to use the procedure during the change of the SQL command. Please refer to the example below:

```
protected void Page_Load(object sender, EventArgs e)
{
    if ((!IsPostBack)&&(wquAdresar.Active==false))
    {
```

```

wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";
wquAdresar.SQLSelect = "select idadresar , name, address, create_date from Adresar";

wquAdresar.Open();
}
else
{
wquAdresar.SQLSelect = "select idadresar , name, address, create_date from Adresar
where name like 'a%'";
wquAdresar.ReOpen();
}
grdAdresar.PxDataSource = wquAdresar;
grdAdresar.DataBind();
}

```

Another case is the automatic refresh, which can be run in the background of the PxWebQuery component.

The PxWebQuery can perform during its operation an automatic background refresh and update of the table data contents.

This refresh can be turned on by setting the AutomaticRefresh property value to true.

It is possible to specify the interval value in seconds in the IntervalQueryRefresh property. After the expiration of the interval since the last loading or refreshing of the data, the new data refresh of the PxWebQuery component shall be carried out.

If no value is specified in the IntervalQueryRefresh property, the default value of 120 seconds shall be used, i.e. the automatic refresh shall be carried out every two minutes.

```

wquAdresar.Open();
wquAdresar.AutomaticRefresh = true;
wquAdresar.IntervalQueryRefresh = 180; //in seconds

```

3.1.12. Validation, checking of the entered values by means of the PxWebQuery component and other visual components (PxEdit, PxComboBox, etc.)

Through the PxEdit component (as well as PxComboBox, PxFlyComboBox, PxJSDatePicker, etc.) the PxWebQuery component contains and automatic validation. If the table column linked to the PxEdit component is of the **string** type, validation of the string length is automatically carried out, if it is of the **int** type, validation operation to check whether the integer has been entered is carried out, in case of the **date** type, validation operation to check whether the valid date has been entered is carried out, etc.

The picture below shows an example where we have entered a string which length exceeded the length of string defined in the database. Then we have entered an invalid date.

Linking the PxWebQuery and the PxEdit component causes the automatic validation. If the value specified in the PxEdit component does not correspond to the given type, it shall not pass the automatic validation, data storing shall be automatically stopped, and the PxEdit component shall report an error message stating the reason for which the data has not been stored.

Edit Address

Main text	<input type="text" value="35"/>	
Persons Name	<input type="text" value="Jurij Brezan aaaaaa"/>	* You have exceeded the allowed length of string, you can specify the maximum 50 characters !!!
Address	<input type="text" value="Moskovská 2"/>	
Juridical Form	<input type="text" value="Juridical person"/>	
Date of Establishment	<input type="text" value="22.10.2010a"/>	* You have not entered the correct date !!!

In addition to the automatic validation, there are additional validations, defined in the *PxWebQuery* component via the `AddParamValidation` parameter.

```
wquAdresar.AddParamValidation("name", "vtIsNotNull");
```

In this case, data storing shall not be carried out if the "Name" column is not filled out, and after pressing the "Ok" button the original form shall remain and the "Persons Name" component shall report the error message. Please refer to the picture below:

Add Address

ID	<input type="text" value="86"/>	
Persons Name	<input type="text"/>	* You did not name the person !!!
Juridical Form	<input type="text" value="Choose value"/>	

Similarly, if the "vtIsNotNull" validation parameter is applied to the "juridical form" column, the original form shall remain after pressing the "Ok" button until any value from the "juridical form" list is selected. The command specifying the juridical form validation parameter shall be as follows:

```
wquAdresar.AddParamValidation("idpravnaforma", "vtIsNotNull");
```

If no Juridical Form from the combobox is selected, the original form shall remain after the "Ok" button is pressed, and the reason why the form has not been saved shall be stated in red letters. Please refer to the picture below.

Add Address

ID	<input type="text" value="86"/>	
Persons Name	<input type="text"/>	* You did not name the person !!!
Juridical Form	<input type="text" value="Choose value"/>	* You did not value !!!

Definition of the `AddParamValidation` command:

```
public void AddParamValidation(string sFieldName, string sEnumValidationTest, string sRegularExpression, Int32 iMinMaxValue, string sErrorMsg)
```

```
public void AddParamValidation(string sFieldName, string sEnumValidationTest, Int32 iMinMaxValue)
```

```
public void AddParamValidation(string sFieldName, string sEnumValidationTest)
```

Description of the parameters of the AddParamValidation command:

sFieldName – field name , column name, to which this validation type shall be applied
sEnumValidationTest – validation type, control type that shall be carried out, description of individual fields shall be given below
sRegularExpression – expression for setting the validation via regular expressions
iMinMaxValue – minimal, maximal expression value at the specified integer field type
sErrorMsg – this expression serves for re-writing or entering of a new error message at this validation type

Description of individual validation types:

EnumValidationTest	Description of the validation type
vtIsNotNull	This type checks whether any value is specified, if none value is specified, the error message "No value is specified !!!" shall be sent
vtValidToEmail	This type checks whether valid e-mail address is specified
vtValidToMaxNumber	This type checks whether the smaller number than in the iMinMaxValue variable is specified, if not, the error message is sent
vtValidToMinNumber	This type checks whether the bigger number than in the iMinMaxValue variable is specified, if not, the error message is sent
vtValidToPSC	This type checks whether the valid zip code is specified, if not, the error message is sent
vtValidToRegularExpression	This type checks whether the specified value correspond to the set regular expression , if not, the error message is sent
vtValidToString	This type checks whether the specified string contains only letters, if not, the error message is sent
vtValidToTelefon	This type checks whether the valid phone number is specified, if not, the error message is sent
vtValidToURL	This type checks whether the valid URL value is specified, if not, the error message is sent
vtValidToCurrency	This type checks whether the valid currency type number is specified, if not, the error message is sent
vtValidToNumber	This type checks whether the whole specified number is valid, if not, the error message is sent
vtValidToFloat	This type checks whether the valid real number is specified, if not, the error message is sent
vtValidToDateTime	This type checks whether the specified expression stands for the valid date
vtValidToLengthString	This type checks is the length of the string value of the variable iMinMaxValue, if string length is greater than the length of the variable iMinMaxValue return an error message
vtIsUnique	This type checks, whether the value is unique, unique for that particular column

Each type of validation has the fixed error message assigned to it. If you want to change this error message in case its text is not satisfactory, you can do it in the following way:

```
wquAdresar.AddParamValidation("name", "vtIsNotNull", "", 0, "You did not name the person!!!");
```

Further, there is an example of validation using regular expressions.

This book is not intended to describe how to operate with regular expressions and how to compile them, for further information please refer to various web resources.

We have added another column to the "adresar" table. This column serves for entering the zip code. Define validation via the regular expression to the Adresar.aspx.cs file as shown below:

```
string sRegVyrzToPSC = @"^\d{5}$|^d{3}\s\d{2}$";  
wquAdresar.AddParamValidation("psc", "vtValidToRegularExpression", sRegVyrzToPSC, 0, "You have not entered a valid zipcode !!!");
```

The defined regular expression allows to enter a 5 digit number, either with a space after the third digit, with or without a gap. Whenever you want to carry out validation via the regular expression, you have to specify the "vtValidToRegularExpresion" validation type. If validation for the zip code item is defined in such a way, entering of an incorrect zip code shall stop saving of a form. The error message defined in the "ERRORMSG" expression shall appear after the zip code component, please refer to the picture below.

Edit Address

ID 71

Persons Name Karel Dotfuscator

Juridical Form Physical person

Date of Establishment 06.10.2010

☐ Invalid address

Street Šalgotarianska 27

Region Nitriansky District Nitra Municipality Dolné Lefantovce

Zip Code 976 8

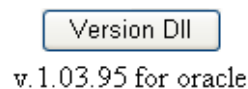
* You have not entered a valid zipcode !!!

3.1.13. Finding the current version of the Px Framework

Current version of the Px Framework can be found by running the following command on the PxWebQuery component:

```
protected void Button12_Click(object sender, EventArgs e)
{
    lblAdresar.Text = wquAdresar.Version;
}
```

The string of the current PxFramework version shall be assigned to the Label (lblAdresar) component, as is shown in the picture below:



Version Dll
v.1.03.95 for oracle

According to the extract you can identify whether you are using the Px Framework version 1.03.95, which runs on Oracle database.

3.1.14. Setting the language mutation of the Px Framework

The Px Framework supports currently the following language versions, see the listing below.

Culture Name	Culture Identifier	Language-Country/Region
sk-SK	0x041B	Slovak - Slovakia
cs-CZ	0x0405	Czech - Czech Republic
en-US	0x0409	English - United States

Language version, can be changed by typing the "Culture Name" string to the CurrentCulture property of the PxCulture static object. This shall be done immediately on startup of the application in each form, in the "Page_Load" procedure. Please refer to the source code listing below:

```
protected void Page_Load(object sender, EventArgs e)
{
    PxCulture.CurrentCulture = CultureInfo.CreateSpecificCulture("cs-CZ");
    ...
}
```

If you would like to choose the English version, you may do so as follows:

```
protected void Page_Load(object sender, EventArgs e)
{
    PxCulture.CurrentCulture = new CultureInfo("en-US");
    ...
}
```

Entering linguistic mutations affects such components as PxSuperGrid, PxComboBox, PxFlyComboBox, PxJSDatePicker and validation objects containing texts of the error messages. If you do not set the linguistic mutation, the Px Framework shall find it itself from the IIS server configuration, and shall use it as default for all pages.

3.1.15. Other AddParam... methods of the PxWebQuery component

3.1.15.A. Method AddParamLink components PxWebQuery

By means of the AddParamLink method you can create a link, a hyperlink to a file or some other webpage in the PxSuperGrid component.

Definition of the AddParamLink command:

```
public void AddParamLink(string aFieldName, string aFieldNameText, string aFieldNameFile,
                        string aPathFile, string aExtensionFile)
```

Description of the AddParamLink command parameters:

aFieldName – Name of the column, which is created as a new column in the table to which the AddParamLink method is applied (This column name must be unique, there shall not exist any other column in the table with the same name). Then let the PxSuperGrid component display the column with that name, which shall display the link further defined by us.

aFieldNameText – Name of the column containing the text, which shall be displayed as the link title

aFieldNameFile – Name of the column, which defines the name of a file or a page that shall be opened after you click on the link

aPathFile – Here the path to a file or page is defined, taking into account that aFieldNameFile column does not contain definition of the full path to the file or the page

aExtensionFile – Here the file extension is defined, unless it has not been defined in the aFieldNameFile column











From all of the parameters defined above, the link to the page defined by us shall be created in the cell of the PxSuperGrid component. In practice, application of the AddParamLink parameter to the PxWebQuery component is carried out as follows:

```
wquAdresar.AddParamLink("NameLink", "LinkText", "pdffilename", "../files_pdf/", ".pdf");
...
```

What goes on during the depiction of a specific cell of the PxSuperGrid component, when the AddParamLink parameter is applied to the PxWebQuery component? Link from the above mentioned parameters, is created, composed as follows:

```
<a href=[aPathFile][aFieldNameFile][aExtensionFile]> [aFieldNameText]</a>
```

Link in the PxSuperGrid component, defined via the AddParamLink method shall look like this:

...	ID	Názov	PDF File Name
  	253	C# 2005 programujeme profesionálne	801-122-122
  	254	ASP.NET konfigurácia a nastavenie	801-122-123
  	255	Začíname programovať v C#	801-122-124
 PRIDAŤ NOVÝ RIADOK...			
[...] [76] [77] [78] [79] [80] [81] [82] [83] [84] Page 85			

3.1.15.B. AddParamDynamicField method of the PxWebQuery component

By means of the AddParamDynamicField method, we can create dynamic content of the columns and data display in the cell of the PxSuperGrid component. This method fills the gap and enables dynamic definition of the PxSuperGrid component cell content.

Definition of the AddParamDynamicField command:

```
public void AddParamDynamicField(string aFieldName,
    ParamDynamicFieldEventHandler aMethodName)
```

Description of the AddParamLink command parameters:

aFieldName – Name of the column, which is created as a new column in the table to which the AddParamDynamicField method is applied (This column name must be unique, there shall not exist any other column in the table with the same name). Then let the PxSuperGrid component display the column with that name, which shall display the link further defined by us. Then the content that was generated by means of the aMethodName method shall be displayed in the PxSuperGrid component in this column with this name.

aMethodName – Name of the method that is always recalled before rendering cell content of the PxSuperGrid component and that generates content for its rendering. The type of this parameter is a delegate of the ParamDynamicFieldEventHandler, which is defined in the body of the PxWebQuery component.

The second parameter of the AddParamDynamicField method is a delegate, which is defined in the body of the PxWebQuery component. To this parameter we attach the method defined by us, in which we dynamically define the content of this particular cell, which shall be rendered in the PxSuperGrid component. The AddParamDynamicField is applied to the PxWebQuery component as follows:

```
wquSposOsoba.AddParamDynamicField("DynamicField", CreateDynamicField_Adresar);
```


The CreateDynamicField_Adresar method is defined in the following way:

```
protected void CreateDynamicField_Adresar(object sender, DataRow DataRow, bool SelectedRow, ref string Value)
{
    if (DataRow != null)
    {
        Value = "<img src='./foto/' + DataRow[\"fotoname\"] + \".jpg' alt='Foto' width=50 align='left'>";
        Value += "<br /> " + DataRow[\"Name\"].ToString() + ", <br /> " +
            DataRow[\"IDPravnaForma\"].ToString();
    }
    else
        Value = String.Empty;
}
```

It should be noted that the number of the CreateDynamicField_Adresar method parameters, and their type is determined by the definition of the delegate in the body of the PxWebQuery component. The delegate definition is as follows:

```
Delegate void ParamDynamicFieldEventHandler(object sender, DataRow DataRow, bool SelectedRow, ref string Value);
```

The CreateDynamicField_Adresar method has a DataRow as the second parameter, where the current row of the PxWebQuery component which shall be rendered is loaded. The parameter SelectedRow is a bool type, and is used us to identify the current line, that line where the cursor is set. If the row is cursor set to SelectedRow parameter is "true" if not, then it is "false".

The Value parameter is of the "ref" type, which means that it is a pointer, to which we define the return value, which shall be displayed in the corresponding cell of the PxSuperGrid component.

In the CreateDynamicField_Adresar method we have defined the image display, and then display of the name and legal form of the person. The „DataRow[\"fotoname\"]” object returns us the name of the photo and the „DataRow[\"Name\"]” object returns us the name of the person. The resulting display of the PxSuperGrid component is shown in the picture below:

...	ID	DynamicField 1	Dátum založenia	Obec, Okres
  	246	 Peter Dowel, Právnická osoba	17. 11. 2010 12:00:00	Obec: Vysoké Tatry Okres: Poprad Kraj: Prešovský
  	247	 Helena Mociková, Fyzická osoba	23. 11. 2010 12:00:00	Obec: Kuzmice Okres: Topoľčany Kraj: Nitriansky
  	248	 Dana Maeská, Fyzická osoba	4. 11. 2010 12:00:00	Obec: Čab Okres: Nitra Kraj: Nitriansky
  	249	 Juraj Fandly, Fyzická osoba	17. 11. 2010 12:00:00	Obec: Jesenské Okres: Levice Kraj: Nitriansky
 PRIDAŤ NOVÝ RIADOK...				
Page 1 [2]				

3.1.16. A special function GetValueFromStructKey of the PxWebQuery component

GetValueFromStructKey function is used to load the column values, if we know the key value that defines the line of the output value of the data structure, which was generated through a connected parameter AddParamKey or AddParamWebQuery.

A special GetValueFromStructKey function in PxWebQuery component can be used when the component PxWebQuery is open through the Open() command, and there are data stretched from the database.

Description of the function GetValueFromStructKey parameters:

C# syntax:

```
public string GetValueFromStructKey(string FieldName, string FieldNameIn,
    string ValueIn, string FieldNameOut);
```

Description of the parameters:

FieldName – name of the column in the table of PxWebQuery component, for which we imposed the AddParamKey parameter or AddParamWebQuery

FieldNameIn – name of the column in the table of the structure by which we find a row with a return value

ValueIn – the value that we find in the column **FieldNameIn** by which we find a row with a return value

FieldNameOut – the value of this column from searched row. It is a return value of the GetValueFromStructKey function.

Return – if the row with the ValueIn in the column FieldNameIn is not found, the return value is String.Empty. If this row is found, the function returns the value from the FieldNameOut column of the found row.

Briefly, the GetValueFromStructKey function is used to load a value from the table of the

PxWebQuery component structure created through the parameters AddParamKey or AddParamWebQuery, if we know the key of the table and the unique value of this key, which defines a line with an output value.

3.1.17. Three ways of using the PxWebQuery component

The PxWebQuery component allows us to use it in different ways. If it is used in a reasonable way, it can save the PC's performance and speed-up the application running. Below you can find description of three ways of using the PxWebQuery component, and their suitability for the given types of situations.

Uses of the PxWebQuery component:

1. The PxWebQuery component remains opened (without Close ());
2. The PxWebQuery component is closed via the Close ()command; (the so-called soft close)
3. The PxWebQuery component is closed via the Close(True) command; (the so-called hard close)

3.1.17.1. The PxWebQuery component remains opened (without Close ());

After loading data into the PxWebQuery component and its processing, the Close () command is not executed.

This way of using the component fastens greatly the application, data is stored in the memory and are not lost even after the termination of the page runtime cycle.

When the PxWebQuery component is used in this way, its data is usually lost after the end of the session of the given user. This method of using the PxWebQuery component is very suitable, it saves the database server and network, fastens the resulting web page generation.

The example below shows the correct usage of the PxWebQuery component in this case.

The PxWebQuery component is defined in the *.aspx file as follows:

```
<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>
```

After carrying out this operation you can work with the PxWebQuery component in the *.aspx.cs file as follows.

First, enter the ConnectionString to the component and then the SQL command. Then use the "Open" command to load data from the database to the component, and now you can link it to the PxSuperGrid component, which displays the data retrieved. After linking to the PxSuperGrid component, do not enter the Close () command, and data in the PxWebQuery component remain until the end of the user session.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (wquAdresar.Active==false)
    {
        wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";
        wquAdresar.SQLSelect = "select idadresar , name, address, create_date from Adresar";
    }
}
```

```

        wquAdresar.Open();
    }
    grdAdresar.PxDataSource = wquAdresar;
    grdAdresar.DataBind();
}

```

For the occasional refreshment of the PxWebQuery component data you can use the automatic refresh, which runs in the background of the PxWebQuery component. During its operation, the PxWebQuery component can perform an automatic background refresh and update of the table data contents.

This refresh can be turned on by setting the AutomaticRefresh property value to true.

It is possible to specify the interval value in seconds in the IntervalQueryRefresh property. After the expiration of the interval since the last loading or refreshing of the data, the new data refresh of the PxWebQuery component shall be carried out.

If no value is specified in the IntervalQueryRefresh property, the default value of 120 seconds shall be used, i.e. the automatic refresh shall be carried out every two minutes.

```

wquAdresar.Open();
wquAdresar.AutomaticRefresh = true;
wquAdresar.IntervalQueryRefresh = 180; //in seconds

```

3.1.17.2. The PxWebQuery component is closed via the Close() command; (the so-called soft close)

After loading data into the PxWebQuery component and its processing, the Close () or Close (false) command, the so called soft close, is executed, the component closure and the deletion of memory space at the end of the runtime cycle of an ASP.NET page.

In this case, data of the PxWebQuery component is erased (cleared down) after each runtime cycle of the page, and in case of any other subsequent web page startup the data is loaded once again into the PxWebQuery component via the select ... command. This method loads the database server, and slows down the resulting web page generation. The advantage of this method is that after each web page display the PxWebQuery component always contains current data from the database.

The example below shows the correct usage of the PxWebQuery component in this case.

The PxWebQuery component is defined in the *.aspx file as follows:

```

<Prx:PxWebQuery ID="wquAdresar" runat="server" Value="wquAdresar"/>

```

After carrying out this operation you can work with the PxWebQuery component in the *.aspx.cs file as follows.

First, enter the ConnectionString to the component and then the SQL command. Then use the "Open" command to load data from the database to the component, and now you can link it to the PxSuperGrid component, which displays the data retrieved. After linking to the PxSuperGrid component, type the Close () command, and data in the PxWebQuery component remain until the end of the runtime cycle of the generated web page.

```

protected void Page_Load(object sender, EventArgs e)
{
    if (wquAdresar.Active==false)
    {
        wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";
        wquAdresar.SQLSelect = "select idadresar , name, address, create_date from Adresar";

        wquAdresar.Open();
    }
    grdAdresar.PxDataSource = wquAdresar;
    grdAdresar.DataBind();

    wquAdresar.Close();
}

```

Instead of the wquAdresar.Close() command, you can also enter the wquAdresar.Close(False) command, which has the similar effect.

In this case, it is not recommended to use the automatic refresh of the PxWebQuery component.

3.1.17.3. The PxWebQuery component is closed via the Close(True) command;(the so-called hard close)

After loading data into the PxWebQuery component and its processing, the Close (True) command, the so-called hard close is executes. Closure of the component and the deletion of its memory space, takes place s as soon as the Close (True) is called.

This method is suitable in case of the dynamic creation of the PxWebQuery component when you need to retrieve quickly specific data for a single usage. In case of the dynamic creation of the PxWebQuery component, it is appropriate to terminate it via the Close(True); command. Dynamic creation of the PxWebQuery component and its termination is shown in the example below:

```

PxWebQuery wquAdresar = new PxWebQuery();

wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xe;";
wquAdresar.SQLSelect = "select idadresar , name, address, create_date from Adresar";
wquAdresar.Open();
.....
.....
wquAdresar.Close(True);

```

In this case, is not recommended to use the automatic refresh of the PxWebQuery component.

3.2. Work of styles, cascading style sheets (CSS)

3.2.1. PxEdit – Work of styles, cascading style sheets (CSS)

In this section, we describe how the component PxEdit of work styles, how to declare the style for all components, from which the components PxEdit. PxEdit component consists of the following components:

Label + TextBox + Label

Therefore, each of these components in the definition of the CSS, we need to declare separately. More definition of the cascade style file AddAdresar.aspx.

```
<style media="all" type="text/css">
    .textbox {
        // style definition for TextBox
    }
    .textbox-label {
        // style definition for Label
    }
    .textbox-rlabel {
        // style definition for Rlabel, or otherwise Right Label
    }
</style>
```

In defining the components PxEdit style, it should be noted that the TextBox component is the root, and since it can be deduced the names of classes for other components, which is composed of component PxEdit . If we declare a style for TextBox component, which is located in component PxEdit, we do so through the CssClass property to assign the string "textbox".

The style definition for captions PxEdit components, we have the CSS style definition to define the class of the following names. The left caption Taking the name of the root class in this case the "textbox" and link it from the chain, which is precisely defined and it is "-label" . Thus we got to the title of class, through which we can declare a style for the left caption , and the name is "textbox-label" . Similarly, it is the right caption, where the resulting class name would be "textbox-rlabel".

Name of components from which composition components PxEdit	How to create a name CssClass definition for individual components contained in the component declaration PxEdit for CSS styles	Description of what component it is
TextBox	CssClass	Editing window TextBox
Label	CssClass+"-label"	Left caption (Caption)
RLabel	CssClass+"-rlabel"	Right caption (Caption)

The file AddAdresar.aspx can specify style PxEdit component as follows:

```
<Prx:PxEdit ID="edtName" runat="server" CssClass="textbox" AddTableRow="True" ></Prx:PxEdit>
```

In component PxEdit, the component Label and RLabel class name for CssClass modifies so that the text contained in the property CssClass add more text "-label" or "-rlabel". So the final declaration of all styles for all components in component PxEdit CSS declaration will be as follows:

```
<head runat="server">
  <title>Add Adresar - Px Framework</title>
  <style media="all" type="text/css">
    .textbox {
      background: lightblue;
      color: red;
      font-size: larger;
      height: 19px;
      width: 150px;
      font-weight: bold;
      border:solid 1px #b0c4de;
    }
    .textbox-label {
      background: yellow;
      color: Blue;
      font-size: larger;
      font-weight: bold;
      border: 0;
    }
    .textbox-rlabel {
      background: lightblue;
      color: Red;
      font-size: larger;
      font-weight: bold;
      border: 0;
    }
  </style>
</head>
```

After the declaration of individual styles, as declared above, PxEdit component will look like this:

Edit Address

ID	<input type="text" value="5"/>
Persons Name	Jurij Brezan
Juridical Form	Physical persc ▼
Date of Establishment	07.07.2010 
	<input type="checkbox"/> Invalid address
Street	Šalgotariánska 2

3.2.2. PxJSDatePicker – work of styles, cascading style sheets (CSS)

In this section, we describe how the component PxJSDatePicker of work styles, how to declare the style for all components, from which the components PxJSDatePicker.

PxJSDatePicker component consists of the following components:

Label + TextBox + Image alebo Button +Label

Therefore, each of these components in the definition of the CSS, we need to declare separately. More definition of the cascade style in file AddAdresar.aspx.

```
<style media="all" type="text/css">
    .textbox {
        // style definition for TextBox
    }
    .textbox-label {
        // style definition for Label
    }
    .textbox-image {
        // style definition for Label
    }
    .textbox-button {
        // style definition for Button
    }
    .textbox-rlabel {
        // style definition for Rlabel, or otherwise Right Label
    }
</style>
```

The definition of style PxJSDatePicker components, it should be noted that the TextBox component is the root, and since it can be deduced class names for the other components that compose component PxEEdit.

If we declare a style for TextBox component, which is located in PxEEdit component, we can do so via the CssClass property to assign a string "textbox". To enable us to declare the style for captions PxJSDatePicker components, we have the definition of CSS style class defined in the following names. The left caption Taking the name of the root class in this case the "textbox" and link it from the chain, which is precisely defined and it is "-label". Thus we got to the title of class, through which we can declare a style for the left caption and the title is "textbox-label". Similarly, it is right caption, where the resulting class name would be "textbox-rlabel".

Name of components from which composition components PxJSDatePicker	How to create a name CssClass definition for individual components contained in the component declaration PxJSDatePicker for CSS styles	Description of what component it is
TextBox	CssClass	Editing window TextBox
Label	CssClass+“-label“	Left caption (Caption) Label
Image	CssClass+“-image“	Image Button calendars
Button	CssClass+“-button“	Default button calendars
RLabel	CssClass+“-rlabel“	Right caption (Caption) Label

The file AddAdresar.aspx can specify style PxJSDatePicker component as follows:

```
<Prx:PxJSDatePicker ID="edtCREATE_DATE" runat="server" CssClass="textbox" AddTableRow="True" >
</Prx:PxJSDatePicker>
```

In PxJSDatePicker component is the component label, Image, Button and RLabel class name for CssClass modifies so that the text contained in the CssClass property will add more text „-label“, „-image“, „-button“alebo „-rlabel“.

So the final declaration of all styles for all components in component PxJSDatePicker CSS declaration will be as follows:

```
<style media="all" type="text/css">
    .textbox {
        background: lightblue;
        color: red;
        font-size: larger;
        height: 19px;
        width: 150px;
        font-weight: bold;

        border:solid 1px #b0c4de;
    }
    .textbox-label {
        background: yellow;
        color: Blue;
        font-size: larger;
        font-weight: bold;
        border: 0;
    }
    .textbox-rlabel {
        background: lightblue;
        color: Red;
        font-size: larger;
        font-weight: bold;
        border: 0;
    }
    .textbox-caption {
        background: lime;
        color: blue;
        font-size: larger;
        font-weight: bold;
        border: 0;
    }
    .textbox-button-insert
    {
        width:20px;
        height:17px;
    }
    .textbox-button-delete
    {
        width:20px;
        height:17px;
    }
</style>
```

3.2.3. PxGreatRepeater – work of styles, cascading style sheets (CSS)

In this section, we describe how the component PxGreatRepeater of work styles, how to declare the style for all components, from which the components PxGreatRepeater. PxGreatRepeater component may be composed of the following components:

PxEdit
PxCombobox
PxFlyCombobox
PxLabel

Each of these components is composed of under component, and therefore for each of these under component in the definition of the CSS, we will declare a separate class. More definition of the cascade style file AddAdresar.aspx.

```
<style media="all" type="text/css">
    .textbox {
        // style definition for TextBox
    }
    .textbox-label {
        // style definition for Label
    }
    .textbox-image {
        // style definition for Label
    }
    .textbox-button {
        // style definition for Label
    }
    .textbox-rlabel {
        // style definition for Rlabel, or otherwise Right Label
    }
</style>
```

In addition, components of which component PxGreatRepeater dynamically linking, there are components that are directly related to the component of PxGreatRepeater directly create it. These are buttons for creating and deleting blocks of data, and caption PxGreatRepeater components. At the button "Insert" or "+" components PxGreatRepeater we can get through the class `CssClass+“-button-insert”`. At the button "Delete" or "-" components PxGreatRepeater we can get through the class `CssClass+“-button-delete”`. At the captions PxGreatRepeater components we can get through the class `CssClass+“-caption”`.

Then a dynamically generated element PxGreatRepeater components, we can get through the following CSS style definition.

```
<style media="all" type="text/css">
    .textbox-button-insert {
        // style definition for the button „Insert“, or „+“
    }
    .textbox-button-delete {
        // style definition for the button „Delete“, or „-“
    }
    .textbox-caption {
        // style definition for the caption
    }
</style>
```

```

    }
    .textbox-caption {
        // style definition for the Caption komponenty
        PxGreatRepeater
    }
</style>

```

The file AddAdresar.aspx can specify style PxGreatRepeater component as follows:

```

<br />
<Prx:PxGreatRepeater ID="grTelefon" runat="server" CssClass="textbox"></Prx:PxGreatRepeater>
<br />

```

So the final declaration of all styles for all components in component PxGreatRepeater CSS declaration will be as follows:

```

<style media="all" type="text/css">
    .textbox {
        background: lightblue;
        color: red;
        font-size: larger;
        height: 19px;
        width: 150px;
        font-weight: bold;

        border:solid 1px #b0c4de;
    }
    .textbox-label {
        background: yellow;
        color: Blue;
        font-size: larger;
        font-weight: bold;
        border: 0;
    }
    .textbox-rlabel {
        background: lightblue;
        color: Red;
        font-size: larger;
        font-weight: bold;
        border: 0;
    }
    .textbox-caption {
        background: lime;
        color: blue;
        font-size: larger;
        font-weight: bold;
        border: 0;
    }
    .textbox-button-insert
    {
        width:20px;
        height:17px;
    }
    .textbox-button-delete
    {
        width:20px;
        height:17px;
    }
</style>

```

After application of this style, form AddAdresar.aspx will look as follows:

Edit Address

ID	<input type="text" value="5"/>	
Persons Name	Jurij Brezan	
Juridical Form	Physical persc	
Date of Establishment	07.07.2010	
	<input type="checkbox"/> Invalid address	
Street	Šalgotariánska 22	
Region	District	Municipality
Košický	Rožňava	Čučma
Zip Code	974 11	

+

Telephone

Telephone

02/56456546

Notice

pevná linka

+

Fax

Fax

Notice

Ok

Storno

Other components Px Framework, have similar styles of work, so I will not describe further.

4.1. Specification Px Framework deployment under different database platforms (Oracle, MS SQL, MySQL, FireBird, InterBase)

When deployed on different database platforms the Px Framework appears solid as a whole, in short, that an application designed for Oracle database will be easy to migrate the instance on MS SQL database.

What will be adjusted, will consist of two things by modifying the source code and database structure.

These arrangements will cover the following two things:

1. Change `ConnectionString` to enter into `PxWebQuery` components for the database platform
2. Modify, covering the sequence defined for the database platform

The following models will be listed `ConnectionString` for the database platform, and also how to create a sequence (generators) to Px Framework of them worked correctly. Sequence (generators) are designed to create a new row in the table to define the unique ID for the primary key column.

4.1.1. Specification Px Framework deployment under Oracle database

To Px Framework to work correctly with Oracle database, we need to `PxWebQuery` components, enter the correct string `ConnectionString`. Below is an example:

```
wquAdresar.ConnectionString = "User Id=adr;Password=aa;Data Source=xex";
```

Furthermore, we have for each table into which to enter new lines to create a sequence that serves to create a new row in the table to define the unique ID for the primary key column.

In order to put through `PxWebQuery` new lines need to create a sequence, from which you download the component `PxWebQuery` unique ID for your primary key. Sequence name, you should create the following way.

SEQ_ "primary key column name "

Therefore, when you create new tables, do not name the primary key column 'ID', but as ID + "table name".

In our example, we worked well to insert data into a database, create sequence to SQL command:

```
CREATE SEQUENCE SEQ_IDADRESAR  
START WITH 1  
INCREMENT BY 1  
MAXVALUE 1E18  
NOMINVALUE  
NOORDER  
NOCACHE  
NOCYCLE;
```

4.1.2. Specification Px Framework deployment under MS SQL database

To Px Framework to work correctly with MS SQL database, we need to PxWebQuery components, enter the correct string ConnectionString. Below is an example:

```
wquAdresar.ConnectionString = "server=localhost\\SQLEXPRESS;Integrated Security=true;database=dbname;"
```

Furthermore, we have for each table into which to enter new lines to create a sequence that serves to create a new row in the table to define the unique ID for the primary key column.

In order to put through PxWebQuery new rows, you need to create a sequence, from which you download the component PxWebQuery unique ID for your primary key.

MS SQL Server does not support sequence was therefore chosen a solution that replaces the sequence. So if you want to work with Px framework available on MS SQL database, you must create a sequence as follows. This sequence will create the following two steps:

1. Create a table where the suspended last loaded value for each table
2. The creation of procedures that will manage, retrieve and increments the value of the table row

For us to function and input data into a database, SQL commands create table first and then procedure. Table Name and procedure Name should remain unchanged.

Table to generate sequence, created by entering the following SQL command:

```
--Creating SEQUENCE for the table ADRESAR A PRAVNA FORMA
--CREATION table where they will be saved VALUE SEQUENCE
Create table All_Sequences (
    SeqName nvarchar(255) primary key, -- name of the sequence
    Start int not null default(1), -- seed value
    Increment int not null default(1), -- incremental
    CurrVal int
)
GO
```

Create a procedure that uses Px Framework to retrieve a unique key value for the table created by entering the following SQL command:

```
--PROCEDURE FOR LOADING SEQUENCE
create procedure Get_Sequence
    @SeqName nvarchar(255),
    @result int OUTPUT,
    @start int = 1,
    @incr int = 1,
    @currvalue int = 1
as
begin
    declare @NewSeqVal int

    set NOCOUNT ON
    update All_Sequences
    set @NewSeqVal = CurrVal = CurrVal+Increment
    where SeqName = @SeqName

    if @@rowcount = 0 begin
```

```

if exists (
    select 1 from All_Sequences
    where SeqName = @SeqName )
begin
    print 'Sequence already exists.'
    return 1
end

if @start is null set @start = 1
if @incr is null set @incr = 1
set @currvalue = @start

insert into All_Sequences (SeqName, Start, Increment, CurrVal)
values (@SeqName, @start, @incr, @currvalue)

set @result = @currvalue
RETURN
end
set @result = @NewSeqVal
return
end

GO

```

Immediately after the creation of tables and procedures, we can work with Px Framework and insert new rows. As for the ID table, the table "All_Sequence" was not the establishment line, the row is created automatically when the first call. For each calling the increments the value of an up or the value of the Increment, All_Sequences entered in the table.

4.1.3. Specification Px Framework deployment under Mysql database

To Px Framework to work correctly with Mysql database, we need to PxWebQuery components, enter the correct string ConnectionString. Below is an example:

```
wquAdresar.ConnectionString = "Database=dbname;Data Source=localhost;User Id=root;Password=aaaaa;";
```

Furthermore, we have for each table into which to enter new lines to create a sequence that serves to create a new row in the table to define the unique ID for the primary key column.

In order to put through PxWebQuery new rows, you need to create a sequence, from which you download the component PxWebQuery unique ID for your primary key. MySQL Server does not support sequence was therefore chosen a solution that replaces the sequence. So if you want to work with Px framework available under the MySQL database, you must create a sequence as follows. This sequence will create the following two steps:

1. Create a table where the suspended last loaded value for each table
2. The creation of procedures that will manage, retrieve and increments the value of the table row

For us to function and input data into a database, SQL commands create table first and then procedure. Table Name and procedure Name should remain unchanged.

Table to generate sequence, created by entering the following SQL command:

```

--CREATE TABLE FOR SEQUENCE,
--Generation NEW ID FOR MYSQL
create table All_Sequence(

```

```
seq_name varchar(30),
start int,
increment int,
currval int,
flag char(1));
```

Create a procedure that uses Px Framework to retrieve a unique key value for the table created by entering the following SQL command:

```
--CREATE PROCEDURE SEQUENCE
CREATE DEFINER = 'root'@'localhost'
PROCEDURE Get_Sequence(
    in SeqName varchar(255),
    out NextVal int
)
NOT DETERMINISTIC
CONTAINS SQL
SQL SECURITY DEFINER
COMMENT "
begin

declare rowcount int;
declare v_flag char(1);

select Count(*) into rowcount from All_Sequence where seq_name=SeqName;
if rowcount=0 then
    insert into All_Sequence values(SeqName,1,1,1,'A');
    set NextVal = 1;
else
    set v_flag='N';
    while (v_flag='N') do
        select flag into v_flag from All_Sequence where seq_name=SeqName;
    end while;

    update All_Sequence set flag='N' where seq_name=SeqName;
    select currval+INCREMENT into NextVal from All_Sequence where seq_name=SeqName;
    update All_Sequence set flag='A',currval=NextVal where seq_name=SeqName;
end if;
end;
```

The section "CREATE DEFINER" procedures should still change your user name and hostname is running on MySQL database, in our case it is „root'@'localhost“.

The above procedure is made to compile without error on the MySQL server version 5.1.

Where should we did not pass the compilation of the above treatments, further indicates revised procedure, which should shrink the MySQL server version 5.0 .. and the earlier version, there are by definition delimiter.

```
DELIMITER $$

CREATE DEFINER = 'root'@'localhost'
PROCEDURE Get_Sequence(
    in SeqName varchar(255),
    out NextVal int
)
NOT DETERMINISTIC
CONTAINS SQL
SQL SECURITY DEFINER
COMMENT "
begin

declare rowcount int;
declare v_flag char(1);
```



```

select Count(*) into rowcount from All_Sequence where seq_name=SeqName;
if rowcount=0 then
  insert into All_Sequence values(SeqName,1,1,1,'A');
  set NextVal = 1;
else
  set v_flag='N';
  while (v_flag='N') do
    select flag into v_flag from All_Sequence where seq_name=SeqName;
  end while;

  update All_Sequence set flag='N' where seq_name=SeqName;
  select curval+INCREMENT into NextVal from All_Sequence where seq_name=SeqName;
  update All_Sequence set flag='A',curval=NextVal where seq_name=SeqName;
end if;
end$$

DELIMITER ;

```

Immediately after the creation of tables and procedures, we can work with Px Framework and insert new rows. As for the ID table, the table "All_Sequence" was not the establishment line, the row is created automatically when the first call. For each calling the increments the value of an up or the value of the Increment, entered in the table "All_Sequence".

4.1.4. Specification Px Framework deployment under FireBird database

To Px Framework to work correctly with FireBird database, we need to PxWebQuery components, enter the correct string ConnectionString. Below is an example:

```

wquAdresar.ConnectionString = "DataSource = localhost;Database =D:\\Program
Files\\Firebird\\DATA27.FDB;UserID = SYSDBA;Password = masterkey;";

```

Furthermore, we have for each table into which to enter new lines to create a sequence that serves to create a new row in the table to define the unique ID for the primary key column.

In order to put through PxWebQuery new lines, you need a generator, from which you download the component PxWebQuery unique ID for your primary key. Name generator, you should create the following way.

SEQ_“primary key column name“

Therefore, when you create new tables, do not name the primary key column 'ID', but as ID + "table name".

In our example, we worked well to insert data into a database, create a generator SQL command:

```

CREATE GENERATOR "SEQ_IDADRESAR";

```

4.1.5. Specification Px Framework deployment under InterBase database

To Px Framework to work correctly with InterBase database, we need to PxWebQuery components, enter the correct string ConnectionString. Below is an example:

```
wquAdresar.ConnectionString = "DataSource = localhost;Database  
=C:\\WINNT\\DATA27.GDB;UserID = SYSDBA;Password = masterkey;character  
set=WIN1250;"
```

If you work well with accents, it is necessary to specify the ConnectionString character set, as the provider for interbase, supplied with Px Framework get result error messages when entering diacritics and not defining the character set in the ConnectionString.

```
character set=WIN1250;
```

Furthermore, we have for each table into which to enter new lines to create a sequence that serves to create a new row in the table to define the unique ID for the primary key column.

In order to put through PxWebQuery new lines, you need a generator, from which you download the component PxWebQuery unique ID for your primary key. Name generator, you should create the following way.

SEQ_ "primary key column name"

Therefore, when you create new tables, do not name the primary key column 'ID', but as ID + "table name".

In our example, we worked well to insert data into a database, create a generator SQL command:

```
CREATE GENERATOR "SEQ_IDADRESAR";
```

4.1.6.Specification Px Framework deployed generally

Px Framework has been tested under the following database platforms:

Oracle server v.10.2.0.4.0.

Oracle server 10g Express Edition

MS SQL server v.10.0.2531.0 SP1

MS SQL server 2005

MySQL server v.5.5.25. Win32

MySQL server v.5.1.46. Win32

MySQL server v.5.0.51b

FireBird server v.2.5.1.

FireBird server v.2.0.5.

InterBase server WI-V6.5.0.28.

5.1 Auxiliary components Px Frameworku

5.1.1. Component to retrieve data from database - SQLCoreQuery

SQLCoreQuery component used for entering, editing, loading and deleting data in the database. Over the component SQLCoreQuery component PxWebQuery is built. SQLCoreQuery component, does not store any of its structures to sessions or ViewState objects, such as the components PxWebQuery. More examples below, as content can be downloaded from the database table to the grid.

```
protected void Button5_Click(object sender, EventArgs e)
{
    SQLCoreQuery quPravForm = new SQLCoreQuery();

    quPravForm.ConnectionString = "User Id=database01;Password=aa;Data Source=xe;";

    quPravForm.SQLText = "SELECT * FROM pravform";
    quPravForm.Open();

    StringBuilder sb = new StringBuilder();

    sb.Append("<TABLE border=1>");

    while (quPravForm.Read())
    {
        sb.Append("<TR>");
        for (int iCykl = 0; iCykl < quPravForm.FieldCount; iCykl++)
        {
            sb.Append("<TD>");
            sb.Append(quPravForm.Fields(iCykl).ToString());
        }
    }
    sb.Append("</TABLE>");

    LiteralControl tbl = new LiteralControl(sb.ToString());

    this.Page.Controls.Add(tbl);

    lblAdresar3.Text = sb.ToString();

    quPravForm.Close();
}
```

In order to be able to work with component SQLCoreQuery, we have to define the assembly clause uses the name "Component".

```
using Component;
```

Conclusion

Px Framework is a philosophy that is the heart PxWebQuery component applications (Core), which provides or contains within itself robust data definitions related tables, display some items defined in component PxSuperGrid, shift in the definition of data components as PxFlyCombobox, PxGreatRepeater and so on. and further validation of the data module to save them.

Component PxWebQuery lending data from its robustness, the other composite components such as components PxEdit, PxCombobox, PxCheckbox, PxFlyCombobox, PxGreatRepeater and so on., which working in well-defined component of diapason set PxWebQuery.

Px Framework, what the time of its duration is very young and therefore very pleased to welcome comments and suggestions, which could contribute to the enhancement component, working with them and their excellence.

All fans wish Px Framework, success in working with Px Framework available, and perseverance to penetrate into the mysteries of work and philosophy on which the Px Framework creation. Wish you much success of Px Framework Implementation Team.