**App Class – entry point in Silverlight application**

public partial class App

{

public App()

{

Startup += ApplicationStartup; // subscribe on start event

Exit += ApplicationExit; // subscribe on exit event

UnhandledException += ApplicationUnhandledException; // subscribe on unhadled exception

InitializeComponent();

}

public Database Database { get; internal set; }

// here we start

private void ApplicationStartup(object sender, StartupEventArgs e)

{

// take storage instance for application

using (var stor = IsolatedStorageFile.GetUserStoreForApplication())

{

// checking that Perst DB file exists and initializing it

if (stor.FileExists(DataGenerator.StorageName))

{

InitializePerstStorage();

}

}

// initializing root visual object for application

RootVisual = new MainPage { VerticalAlignment = VerticalAlignment.Stretch };

}

// finishing application and closing database

private void ApplicationExit(object sender, EventArgs e)

{

if (Database != null && Database.Storage != null)

Database.Storage.Close();

}

// global exception handler

private static void ApplicationUnhandledException(object sender, ApplicationUnhandledExceptionEventArgs e)

{

// Exiting in case we are in debugger

if (Debugger.IsAttached) return;

// mark that this exception is handled

e.Handled = true;

// calling error showing method in main GUI thread

Deployment.Current.Dispatcher.BeginInvoke(() => ReportErrorToDom(e));

}

// initializing Perst storage

internal void InitializePerstStorage()

{

// Creating Instance of Perst Storage

var storage = StorageFactory.Instance.CreateStorage();

// Initial Size set 512KB to fit in Silverlight Isolated Storage

storage.SetProperty("perst.file.extension.quantum", 512 \* 1024);

// Step of storage extension 256KB to have less fragmentation on disk storage.SetProperty("perst.extension.quantum", 256 \* 1024);

storage.Open(DataGenerator.StorageName, 0); // Open Storage

//Create Database wrapper over Perst Storage

Database = new Database(storage, false, true, new FullTextSearchHelper(storage));

//Turn off auto-index creation (defined manually)

Database.EnableAutoIndices = false;

}

// Show error in browser window

private static void ReportErrorToDom(ApplicationUnhandledExceptionEventArgs e)

{

// Create error message text

var errorMsg = e.ExceptionObject.Message + e.ExceptionObject.StackTrace.Replace('"', '\'').Replace("\r\n", @"\n");

// Show error by JavaScript window

HtmlPage.Window.Eval("throw new Error(\"Unhandled Error in Silverlight Application " + errorMsg + "\");");

}

}

**Main Control**

public partial class MainPage

{

// Cache of detail panels

private readonly Dictionary<DataGrid, DetailPanel> cacheDetailPanels = new Dictionary<DataGrid, DetailPanel>();

// creating object delaying showing data in detail panel

private readonly Delayer detailDelayer;

private object currentDataGrid; // reference on currently selected grid

public MainPage()

{

InitializeComponent();

SizeChanged += OnThisSizeChanged; // subscribe on resize event

// subscribe on MouseEnter event of all grids

gridContact.MouseEnter += GridOnMouseEnter;

gridLead.MouseEnter += GridOnMouseEnter;

gridActivity.MouseEnter += GridOnMouseEnter;

// subscribe GotFocus event of all grids

gridContact.GotFocus += DataGridGotFocus;

gridLead.GotFocus += DataGridGotFocus;

gridActivity.GotFocus += DataGridGotFocus;

// in case DataBase initialized – load data in contacts grid

if (Database != null)

gridContact.ItemsSource = Database.GetTable<Contact>().ToObservableCollection();

// tune delayer time for all grids

Delayer.DelayMilliseconds = 300;

// create delayer object for contacts grid

var contactDelayer = new Delayer();

// subscribe on event of contact selection change

gridContact.SelectionChanged += (sender, e) => contactDelayer.Action = RefreshLeads;

// create delayer object for leads grid

var leadDelayer = new Delayer();

// subscribe on event of lead selection change

gridLead.SelectionChanged += (sender, e) => leadDelayer.Action = RefreshActivities;

// check buttons state on selection changes in grids

gridContact.SelectionChanged += (sender, e) => CheckButtons();

gridLead.SelectionChanged += (sender, e) => CheckButtons();

CheckButtons();

detailDelayer = new Delayer();

}

private static Database Database

{

get { return ((App)Application.Current).Database; }

}

private bool IsSearchableState

{

get { return !tbSearch.IsEmpty; }

}

private static Storage Storage

{

get { return Database.Storage; }

}

// Add database record in DB, grid and details

private void AddItem(IPersistent obj, DataGrid dataGrid)

{

Database.AddRecord(obj); // Adding new record to Database

// Add object in grid

((IList)dataGrid.ItemsSource).Insert(0, obj);

// select new added object

dataGrid.SelectedItem = obj;

// show new object in details panel

ShowDetailPanel(dataGrid);

// focus first input in details

((DetailPanel)swDetail.Content).FocusFirstTextBox();

}

// clean search box

private void bClearSearch\_Click(object sender, RoutedEventArgs e)

{

tbSearch.Clear();

}

// create and add new Activity

private void bNewActivity\_Click(object sender, RoutedEventArgs e)

{

// get current selected lead

var currentLead = (Lead)gridLead.SelectedItem;

// create and add (in db and grid) new Activity instance

AddItem(new Activity { Lead = currentLead }, gridActivity);

}

// create and add new Contact object

private void bNewContact\_Click(object sender, RoutedEventArgs e)

{

AddItem(new Contact(), gridContact);

}

// create and add new Lead object

private void bNewLead\_Click(object sender, RoutedEventArgs e)

{

// get current selected contact

var currentContact = (Contact)gridContact.SelectedItem;

// create and add (in db and grid) new Lead instance

AddItem(new Lead { Contact = currentContact }, gridLead);

}

// check buttons state

private void CheckButtons()

{

// turn off «Clear», «New Contact» buttons in case DataBase not initialized

bClearDB.IsEnabled = Database != null;

bNewContact.IsEnabled = Database != null;

// turn off «New Lead» in case no contacts selected

bNewLead.IsEnabled = gridContact.SelectedItem != null;

// turn off «New Activity» in case no Leads selected

bNewActivity.IsEnabled = gridLead.SelectedItem != null;

}

// clear DB

private void ClearDBClick(object sender, RoutedEventArgs e)

{

// create cleanup dialog

var clearPopup = new ClearPopup();

// subscribe on dialog close event

clearPopup.Closed += (sender1, e1) =>

{

// in case Ok clicked reload contacts grid (others will follow automatically)

if (clearPopup.DialogResult == true)

RefreshContact();

// refresh buttons state

CheckButtons();

};

// show cleanup dialog

clearPopup.Show();

}

// implementing grid focus logic

private void DataGridGotFocus(object sender, RoutedEventArgs e)

{

// load details panel for selected item. We have to use if here to overcome Silverlight 3 known bug of duplicate event

if (currentDataGrid == sender)

ShowDetailPanel((DataGrid)sender);

}

// logic of changes in grid rows

private void DataGrid\_RowEditEnded(object sender, DataGridRowEditEndedEventArgs e)

{

// in case edit finished without commit cancel method

if (e.EditAction != DataGridEditAction.Commit) return;

// get a link on persistent object

var persistent = e.Row.DataContext as Base;

if (persistent == null) return;

persistent.Save(); // Saving Item to Storage

Storage.Commit(); // Commiting changes

}

// Load data in details panel with delay

private void DataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

// delegate for delayed load

detailDelayer.Action = () =>

{

// check that cache has required grid

if (!cacheDetailPanels.ContainsKey((DataGrid)sender)) return;

// set a target to display

cacheDetailPanels[(DataGrid)sender].Target =

((DataGrid)sender).SelectedItems;

};

}

// generate demo data

private void GenerateDBClick(object sender, RoutedEventArgs e)

{

// create dialog object

var generatorPopup = new GeneratorPopup();

// do actions after dialog close

generatorPopup.Closed += (sender1, e1) =>

{

// On ok refresh contacts grid (others will do the same automatically)

if (generatorPopup.DialogResult == true)

RefreshContact();

CheckButtons();

};

// show dialog

generatorPopup.Show();

}

// set current active grid on mouse enter

private void GridOnMouseEnter(object sender, MouseEventArgs e)

{

currentDataGrid = sender;

}

// refresh list of activities

private void RefreshActivities()

{

// don’t refresh list in case something typed in search box

if (IsSearchableState) return;

// reset current list of activities

gridActivity.ItemsSource = null;

IEnumerable<Activity> res = null;

// get data in case DB exists

if (Database != null)

{

if (gridLead.SelectedItem != null)

{

var leads = gridLead.SelectedItems.Cast<Lead>();

res = (from activity in Database.GetTable<Activity>()

// Load activities

where leads.Contains(activity.Lead)

// by selected lead

select activity);

}

if (res == null && gridContact.SelectedItem != null)

{

var contacts = gridContact.SelectedItems.Cast<Contact>();

res = (from activity in Database.GetTable<Activity>()

// Load activities

where activity.Lead != null && contacts.Contains(activity.Lead.Contact)

// by selected Contact

select activity);

}

}

if (res != null)

{

// convert results to proper collection type

var result = res.ToObservableCollection();

// set activity data source

gridActivity.ItemsSource = result;

}

else

{

gridActivity.ItemsSource = null;

}

}

// refresh contacts list

private void RefreshContact()

{

// don’t refresh list in case something typed in search box

if (IsSearchableState) return;

var contacts = Database != null ?

Database.GetTable<Contact>().ToObservableCollection() : // Reload all contacts

null;

gridContact.ItemsSource = contacts;

// refresh Leads

RefreshLeads();

}

// refresh Leads

private void RefreshLeads()

{

if (IsSearchableState) return;

var contacts = gridContact.SelectedItems.Cast<Contact>();

gridLead.ItemsSource = null;

var res = Database != null ?

(from lead in Database.GetTable<Lead>()

// Loading Leads

where contacts.Contains(lead.Contact)

// by selected Contact

select lead).ToObservableCollection() :

null;

gridLead.ItemsSource = res;

// refresh Activities

RefreshActivities();

}

// search by typed words

private void Search()

{

// exit in case DB not initialized

if (Database == null) return;

// Make full-text search in DB limited to 1000 items and 2 seconds

// without results sorting

var prefixes = Database.SearchPrefix(tbSearch.Text, 1000, 2000, false);

var contacts = new ObservableCollection<Contact>();

var leads = new ObservableCollection<Lead>();

var activities = new ObservableCollection<Activity>();

var arrayRes = new List<FullTextSearchHit>();

if (prefixes != null) arrayRes.AddRange(prefixes.Hits);

// sort out objects by found type

foreach (var hit in arrayRes)

{

if (hit.Document is Contact)

{

if (!contacts.Contains((Contact)hit.Document))

contacts.Add((Contact)hit.Document);

}

else if (hit.Document is Lead)

{

if (!leads.Contains((Lead)hit.Document))

leads.Add((Lead)hit.Document);

}

else if (hit.Document is Activity)

{

if (!activities.Contains((Activity)hit.Document))

activities.Add((Activity)hit.Document);

}

}

// set data to corresponding grids

gridContact.ItemsSource = contacts;

gridLead.ItemsSource = leads;

gridActivity.ItemsSource = activities;

}

// show details by specified grid

private void ShowDetailPanel(DataGrid dataGrid)

{

// skip method in case we already showing same thing

if (cacheDetailPanels.ContainsKey(dataGrid) && swDetail.Content == cacheDetailPanels[dataGrid]) return;

DetailPanel detail;

// get proper details panel for grid and creating new one in case of its absense in cache

if (!cacheDetailPanels.TryGetValue(dataGrid, out detail))

{

Type typeObj;

if (dataGrid == gridContact)

typeObj = typeof(Contact);

else if (dataGrid == gridLead)

typeObj = typeof(Lead);

else if (dataGrid == gridActivity)

typeObj = typeof(Activity);

else

throw new ArgumentOutOfRangeException("dataGrid");

// create new details panel and save in cache

cacheDetailPanels[dataGrid] = detail = new DetailPanel(typeObj, dataGrid);

}

// set new data source

if (detail.Target != dataGrid.SelectedItems)

detail.Target = dataGrid.SelectedItems;

// show details panel

swDetail.Content = detail;

}

// event handler for search box typing

private void tbSearch\_SearchChanged(object sender, EventArgs e)

{

// search in case there is something to search

if (!tbSearch.IsEmpty)

Search();

else

RefreshContact(); // reload contacts in case search box empty

}

}

**Details panel**

public class DetailPanel : StackPanel, INotifyPropertyChanged

{

// create drop-down controls cache

private readonly Dictionary<ItemsControl, Type> dropdowns = new Dictionary<ItemsControl, Type>();

// initial target list

private IList target;

public DetailPanel(Type typeObj, DataGrid dataGrid)

{

if (typeObj == null) throw new ArgumentNullException("typeObj");

if (dataGrid == null) throw new ArgumentNullException("dataGrid");

// type of shown object in details panel

TypeObj = typeObj;

// grid to take out details

DataGrid = dataGrid;

Init();

}

public DataGrid DataGrid { get; private set; }

public IList Target

{

get

{

if (target == null)

target = new List<object>();

return target;

}

set

{

// set preliminary targets list

SetTarget(value);

// reset state of Cover controls

ResetControls();

// find data object for connection with data source

EvaluateDataContext();

// tell system about changed properties

InvokePropertyChanged(new PropertyChangedEventArgs("Target"));

InvokePropertyChanged(new PropertyChangedEventArgs("Title"));

}

}

public string Title

{

get

{

if (Target == null) return "";

switch (Target.Count)

{

case 0:

return string.Format("No {0} Selected", TypeObj.Name);

case 1:

return string.Format("{0} Details", TypeObj.Name);

default:

return string.Format("{0} {1}s Details", Target.Count, TypeObj.Name);

}

}

}

public Type TypeObj { get; private set; }

#region INotifyPropertyChanged Members

public event PropertyChangedEventHandler PropertyChanged;

#endregion

// Utility method to split word by capital letters

private static string SeparateCapitalWords(IEnumerable<char> name)

{

var array = name.ToList();

var res = new List<char>();

foreach (var c in array)

{

if (res.Count > 1 && c >= 'A' && c <= 'Z')

res.Add(' ');

res.Add(c);

}

return new string(res.ToArray());

}

public void FocusFirstTextBox()

{

UpdateLayout();

// find first cover control and set his focus

foreach (var child in Children)

{

if (!(child is CoverControl)

|| !(((CoverControl)child).Control is TextBox)) continue;

((CoverControl)child).Control.Focus();

break;

}

}

// delete with confirmation selected objects

private void DeleteOnClick(object sender, RoutedEventArgs e)

{

// exit if no items selected

if (DataGrid.SelectedItems == null) return;

// confirmations

if (MessageBox.Show(

string.Format("Delete record - {0}?",

(DataGrid.SelectedItems.Count == 1

? DataGrid.SelectedItems[0]

: string.Format("{0} items", DataGrid.SelectedItems.Count))), "Delete",

MessageBoxButton.OKCancel) != MessageBoxResult.OK) return;

var selected = new ArrayList();

// prepare list for deletion

foreach (var item in DataGrid.SelectedItems)

selected.Add(item);

// Delete objects

foreach (var item in selected)

{

// removing objectgs from grid

((IList)DataGrid.ItemsSource).Remove(item);

((Persistent)item).Deallocate(); // Removing Deleted object from Database

}

((App)Application.Current).Database.Storage.Commit(); // Commiting Changes

}

// find data object for connection with data source

private void EvaluateDataContext()

{

Base context = null;

// in case there is only 1 target – that is the context

if (Target.Count == 1)

context = (Base)Target[0];

// in case there are many targets we need to prepare special object for multi-edit/multi-show

else if (Target.Count > 1)

{

var type = Target[0].GetType();

context = (Base)Activator.CreateInstance(type); // create instance by type

context.IsTemp = true; // mark object as temporary

}

// subscribe context on property change events

if (context is INotifyPropertyChanged)

{

((INotifyPropertyChanged)context).PropertyChanged -= OnPropertyChanged;

((INotifyPropertyChanged)context).PropertyChanged += OnPropertyChanged;

}

// unsubscribe old context

if (DataContext is INotifyPropertyChanged)

((INotifyPropertyChanged)DataContext).PropertyChanged -= OnPropertyChanged;

DataContext = null;

// in case context not empty reload drop-downs

if (context != null)

RefreshDropDowns();

else

ResetDropDowns();

DataContext = context;

// initialise context with common data in case it’s temporary

if (DataContext is Base && ((Base)DataContext).IsTemp)

IntersectProperties(context);

}

// initialize panel

private void Init()

{

var detail = this;

// iterate over properties and automatically create controls for them

foreach (var propertyInfo in TypeObj.GetProperties(BindingFlags.DeclaredOnly

| BindingFlags.Public | BindingFlags.Instance))

{

detail.Children.Add(new TextBlock { Text = SeparateCapitalWords(propertyInfo.Name) });

FrameworkElement element;

var binding = new Binding(propertyInfo.Name);

if (propertyInfo.CanWrite)

{

// bind writable properties

binding.Mode = BindingMode.TwoWay;

// bind date-time properties

if (typeof(DateTime).IsAssignableFrom(propertyInfo.PropertyType))

{

var dtp = new DatePicker();

dtp.SetBinding(DatePicker.SelectedDateProperty, binding);

element = new CoverControl { Control = dtp, Name = propertyInfo.Name };

}

// bind enumerations

else if (typeof(Enum).IsAssignableFrom(propertyInfo.PropertyType))

{

var cb = new ComboBox();

cb.SetBinding(Selector.SelectedItemProperty, binding);

var propertyInfo1 = propertyInfo;

Utilities.FillEnums(cb, propertyInfo1.PropertyType);

element = new CoverControl { Control = cb, Name = propertyInfo.Name };

}

// bind reference type properties

else if (typeof(Persistent).IsAssignableFrom(propertyInfo.PropertyType))

{

var cb = new ComboBox();

cb.SetBinding(Selector.SelectedItemProperty, binding);

var propertyInfo1 = propertyInfo;

dropdowns[cb] = propertyInfo1.PropertyType;

element = new CoverControl { Control = cb, Name = propertyInfo.Name };

}

// text properties binding

else

{

var tb = new TextBox();

tb.SetBinding(TextBox.TextProperty, binding);

element = new CoverControl { Control = tb, Name = propertyInfo.Name };

}

}

else

{

// bind one way read-only properties

binding.Mode = BindingMode.OneWay;

var tb = new TextBlock();

tb.SetBinding(TextBlock.TextProperty, binding);

element = tb;

}

detail.Children.Add(element);

}

var grid = new Grid { Margin = new Thickness(0, 50, 0, 0) };

grid.ColumnDefinitions.Add(new ColumnDefinition());

grid.ColumnDefinitions.Add(new ColumnDefinition());

grid.ColumnDefinitions.Add(new ColumnDefinition());

var b = new Button { Content = "Delete" };

b.Click += DeleteOnClick;

b.SetValue(Grid.ColumnProperty, 2);

grid.Children.Add(b);

detail.Children.Add(grid);

}

// find common value for same property of several selected objects

private void IntersectProperties(Base context)

{

// iterate by controls in panel

foreach (var c in Children)

{

var child = c as CoverControl;

if (child == null) continue;

// get properties by name

var propInfo = target[0].GetType().GetProperty(child.Name);

if (propInfo == null)

throw new ArgumentNullException(string.Format("Property with name {0} not found.", child.Name));

if (!propInfo.CanWrite) continue;

var firstObject = true;

object val = null;

// iterate by targets

foreach (var o in target)

{

var nextVal = propInfo.GetValue(o, null);

if (firstObject)

{

val = nextVal;

firstObject = false;

continue;

}

if (Equals(nextVal, val)) continue;

// in case values are different showing cover instead of value

child.IsShowingCover = true;

break;

}

// in case cover not set that means all values the same and it can be saved in temporary object

if (!child.IsShowingCover)

propInfo.SetValue(context, val, null);

}

}

private void InvokePropertyChanged(PropertyChangedEventArgs e)

{

var handler = PropertyChanged;

if (handler != null) handler(this, e);

}

private void OnPropertyChanged(object sender, PropertyChangedEventArgs e)

{

var baseSender = (Base)sender;

// find control with changed value

var coverControl =

(CoverControl)

(from child in Children

where child is CoverControl && ((CoverControl)child).Name == e.PropertyName

select child).First();

// hide control cover

coverControl.IsShowingCover = false;

// in case this is multiselection and object temporary we need to set value to all underlying objects, otherwise just saving object in DB

if (baseSender.IsTemp)

WriteValueToTargets(baseSender, e.PropertyName);

else

baseSender.Save();

}

private void RefreshDropDowns()

{

// refresh drop-down controls

foreach (var dd in dropdowns)

Utilities.FillObjects(dd.Key, dd.Value, Target);

}

// reset all controls

private void ResetControls()

{

foreach (var c in Children)

{

var child = c as CoverControl;

if (child == null) continue;

child.IsShowingCover = false;

child.AllowShowCover = Target.Count > 1;

}

}

// clear values in drop-down controls

private void ResetDropDowns()

{

foreach (var dropdown in dropdowns)

dropdown.Key.Items.Clear();

}

// save targets list

private void SetTarget(IList value)

{

target.Clear();

foreach (var o in value)

target.Add(o);

}

// save value of detail panel to multi-edited objects property

private void WriteValueToTargets(object sender, string paramName)

{

// get property by name

var pInfo = sender.GetType().GetProperty(paramName);

if (pInfo == null) return;

// get new value of property from changes temporary object

var val = pInfo.GetValue(sender, null);

// save value in each object

foreach (Base obj in target)

{

pInfo.SetValue(obj, val, null);

obj.Save();

}

}

}