~~SaveColumnSettingsBehavior verze z 5.9.2019 pred prevzatymi zmenamy z Pricing repa~~

~~//using System;~~

~~//using System.Collections.Generic;~~

~~//using System.ComponentModel;~~

~~//using System.IO;~~

~~//using System.IO.IsolatedStorage;~~

~~//using System.Linq;~~

~~//using System.Runtime.CompilerServices;~~

~~//using System.Runtime.Serialization;~~

~~//using System.Windows;~~

~~//using System.Windows.Interactivity;~~

~~//using Mediaresearch.Framework.Gui.Telerik.ColumnHidingPopup;~~

~~//using Mediaresearch.Framework.Gui.Telerik.Resources;~~

~~//using Telerik.Windows.Controls;~~

~~//using Telerik.Windows.Controls.GridView;~~

~~//using Telerik.Windows.Data;~~

~~//namespace Mediaresearch.Framework.Gui.Telerik.Behaviors~~

~~//{~~

~~// public class SaveColumnSettingsBehavior : Behavior<RadGridView>, INotifyPropertyChanged, IBehaviorCreator~~

~~// {~~

~~// private RadGridView m\_grid;~~

~~// private string m\_gridViewNewRowText;~~

~~// private List<HideableColumn> m\_hideableColumns;~~

~~// private DataContractSerializer m\_serializer;~~

~~// public string GridViewNewRowText~~

~~// {~~

~~// get => m\_gridViewNewRowText;~~

~~// set~~

~~// {~~

~~// m\_gridViewNewRowText = value;~~

~~// OnPropertyChanged(nameof(GridViewNewRowText));~~

~~// }~~

~~// }~~

~~// public Behavior Create()~~

~~// {~~

~~// return new SaveColumnSettingsBehavior();~~

~~// }~~

~~// public event PropertyChangedEventHandler PropertyChanged;~~

~~// protected override void OnAttached()~~

~~// {~~

~~// base.OnAttached();~~

~~// m\_grid = AssociatedObject;~~

~~// if (m\_grid == null)~~

~~// {~~

~~// return;~~

~~// }~~

~~// m\_grid.ColumnWidthChanged += OnColumnWidthChanged;~~

~~// m\_grid.ColumnDisplayIndexChanged += OnColumnDisplayIndexChanged;~~

~~// m\_grid.Loaded += OnLoaded;~~

~~// m\_grid.DataLoaded += RefreshGridViewNewRow;~~

~~// m\_grid.SelectionChanged += RefreshGridViewNewRow;~~

~~// m\_grid.Filtered += RefreshGridViewNewRow;~~

~~// m\_grid.FilterOperatorsLoading += OnFilterOperatorsLoading;~~

~~// m\_serializer = new DataContractSerializer(typeof(GridColumnProperties));~~

~~// }~~

~~// private void OnFilterOperatorsLoading(object sender, FilterOperatorsLoadingEventArgs e)~~

~~// {~~

~~// e.DefaultOperator1 = FilterOperator.Contains;~~

~~// }~~

~~// private void RefreshGridViewNewRow(object sender, EventArgs e)~~

~~// {~~

~~// GridViewNewRowText = string.Format(Localisation.RowCountStatistics, m\_grid.Items.Count, m\_grid.SelectedItems.Count);~~

~~// }~~

~~// protected override void OnDetaching()~~

~~// {~~

~~// base.OnDetaching();~~

~~// m\_grid.ColumnWidthChanged -= OnColumnWidthChanged;~~

~~// m\_grid.ColumnDisplayIndexChanged -= OnColumnDisplayIndexChanged;~~

~~// m\_grid.Loaded -= OnLoaded;~~

~~// m\_grid.DataLoaded -= RefreshGridViewNewRow;~~

~~// m\_grid.SelectionChanged -= RefreshGridViewNewRow;~~

~~// m\_grid.Filtered -= RefreshGridViewNewRow;~~

~~// m\_grid.FilterOperatorsLoading -= OnFilterOperatorsLoading;~~

~~// }~~

~~// private void OnColumnWidthChanged(object source, ColumnWidthChangedEventArgs e)~~

~~// {~~

~~// SaveGridSettings();~~

~~// }~~

~~// private void OnColumnDisplayIndexChanged(object source, GridViewColumnEventArgs e)~~

~~// {~~

~~// SaveGridSettings();~~

~~// }~~

~~// private void OnColumnVisibilityChanged(object source, EventArgs e)~~

~~// {~~

~~// SaveGridSettings();~~

~~// }~~

~~// private void OnLoaded(object source, RoutedEventArgs e)~~

~~// {~~

~~// LoadGridSettings();~~

~~// }~~

~~// private void SaveGridSettings()~~

~~// {~~

~~// if (m\_grid?.Columns == null)~~

~~// {~~

~~// return;~~

~~// }~~

~~// var properties = new GridColumnProperties();~~

~~// foreach (var column in m\_grid.Columns)~~

~~// {~~

~~// if (column is GridViewDataColumn dataColumn)~~

~~// {~~

~~// if (properties.List.Any(c => c.DisplayIndex == column.DisplayIndex))~~

~~// {~~

~~// return;~~

~~// }~~

~~// var visibility = m\_hideableColumns?.Single(hc => hc.Key == column.UniqueName).IsVisible ?? dataColumn.IsVisible;~~

~~// var columnProperty = new GridColumnProperty~~

~~// {~~

~~// UniqueName = dataColumn.UniqueName,~~

~~// Width = dataColumn.ActualWidth,~~

~~// DisplayIndex = dataColumn.DisplayIndex,~~

~~// IsVisible = visibility~~

~~// };~~

~~// properties.Add(columnProperty);~~

~~// }~~

~~// }~~

~~// Save(m\_grid.Uid, properties);~~

~~// }~~

~~// private void Save(string Uid, GridColumnProperties properties)~~

~~// {~~

~~// try~~

~~// {~~

~~// using (var file = IsolatedStorageFile.GetUserStoreForDomain())~~

~~// {~~

~~// using (var stream = new IsolatedStorageFileStream(Uid, FileMode.Create, file))~~

~~// {~~

~~// m\_serializer.WriteObject(stream, properties);~~

~~// }~~

~~// }~~

~~// }~~

~~// catch (Exception ex)~~

~~// {~~

~~// MessageBox.Show($"Can not save Grid collumns settings: {ex.Message}");~~

~~// }~~

~~// }~~

~~// private void LoadGridSettings()~~

~~// {~~

~~// if (m\_grid?.Columns == null)~~

~~// {~~

~~// return;~~

~~// }~~

~~// var dc = m\_grid.DataContext as IHideableColumns;~~

~~// if (dc != null)~~

~~// {~~

~~// m\_hideableColumns = dc.HideableColumns;~~

~~// }~~

~~// var properties = Load(m\_grid.Uid);~~

~~// if (properties?.List == null)~~

~~// {~~

~~// return;~~

~~// }~~

~~// var columns = m\_grid.Columns.Cast<GridViewColumn>().ToList();~~

~~// foreach (var column in columns)~~

~~// {~~

~~// var columnProp = properties.List.SingleOrDefault(cp => cp.UniqueName == column.UniqueName);~~

~~// if (columnProp != null)~~

~~// {~~

~~// column.DisplayIndex = columnProp.DisplayIndex;~~

~~// column.Width = new GridViewLength(columnProp.Width);~~

~~// if (m\_hideableColumns == null)~~

~~// {~~

~~// column.IsVisible = columnProp.IsVisible;~~

~~// column.PropertyChanged += OnColumnVisibilityChanged;~~

~~// }~~

~~// else~~

~~// {~~

~~// m\_hideableColumns.Single(hc => hc.Key == column.UniqueName).IsVisible = columnProp.IsVisible;~~

~~// m\_hideableColumns.Single(hc => hc.Key == column.UniqueName).ColumnVisibilityChanged += OnColumnVisibilityChanged;~~

~~// }~~

~~// }~~

~~// else // sloupec je nový, musí se ponechat na své výchozí. Všechny následující se musí o 1 posunout~~

~~// {~~

~~// columns.ForEach(c =>~~

~~// {~~

~~// if (c.DisplayIndex > column.DisplayIndex && c.DisplayIndex < columns.Count - 1)~~

~~// {~~

~~// c.DisplayIndex++;~~

~~// }~~

~~// });~~

~~// properties.List.ForEach(c =>~~

~~// {~~

~~// if (c.DisplayIndex >= column.DisplayIndex)~~

~~// {~~

~~// c.DisplayIndex++;~~

~~// }~~

~~// });~~

~~// }~~

~~// }~~

~~// dc?.ShowHideColumnRequested(null);~~

~~// }~~

~~// private GridColumnProperties Load(string Uid)~~

~~// {~~

~~// try~~

~~// {~~

~~// using (var file = IsolatedStorageFile.GetUserStoreForDomain())~~

~~// {~~

~~// if (!file.FileExists(Uid))~~

~~// {~~

~~// return null;~~

~~// }~~

~~// using (var stream = new IsolatedStorageFileStream(Uid, FileMode.Open, file))~~

~~// {~~

~~// if (stream.Length > 0)~~

~~// {~~

~~// var loaded = (GridColumnProperties) m\_serializer.ReadObject(stream);~~

~~// return loaded;~~

~~// }~~

~~// }~~

~~// }~~

~~// }~~

~~// catch (Exception ex)~~

~~// {~~

~~// MessageBox.Show($"Can not load Grid collumns settings: {ex.Message}");~~

~~// }~~

~~// return null;~~

~~// }~~

~~// protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)~~

~~// {~~

~~// PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));~~

~~// }~~

~~// }~~

~~// [DataContract]~~

~~// internal class GridColumnProperty~~

~~// {~~

~~// [DataMember] public string Path { get; set; }~~

~~// [DataMember] public string UniqueName { get; set; }~~

~~// [DataMember] public object Header { get; set; }~~

~~// [DataMember] public double Width { get; set; }~~

~~// [DataMember] public int DisplayIndex { get; set; }~~

~~// [DataMember] public bool IsVisible { get; set; }~~

~~// }~~

~~// [DataContract]~~

~~// internal class GridColumnProperties~~

~~// {~~

~~// public GridColumnProperties()~~

~~// {~~

~~// List = new List<GridColumnProperty>();~~

~~// }~~

~~// [DataMember] public List<GridColumnProperty> List { get; set; }~~

~~// public void Add(GridColumnProperty gridColumnProperty)~~

~~// {~~

~~// List.Add(gridColumnProperty);~~

~~// }~~

~~// }~~

~~//}~~

**~~Tahle verze funguje:~~**

**~~using System;~~**

**~~using System.Collections.Generic;~~**

**~~using System.ComponentModel;~~**

**~~using System.IO;~~**

**~~using System.IO.IsolatedStorage;~~**

**~~using System.Linq;~~**

**~~using System.Runtime.CompilerServices;~~**

**~~using System.Runtime.Serialization;~~**

**~~using System.Windows;~~**

**~~using System.Windows.Interactivity;~~**

**~~using Mediaresearch.Framework.Gui.Telerik.ColumnHidingPopup;~~**

**~~using Mediaresearch.Framework.Gui.Telerik.Resources;~~**

**~~using Telerik.Windows.Controls;~~**

**~~using Telerik.Windows.Controls.GridView;~~**

**~~using Telerik.Windows.Data;~~**

**~~namespace Mediaresearch.Framework.Gui.Telerik.Behaviors~~**

**~~{~~**

**~~public class SaveColumnSettingsBehavior : Behavior<RadGridView>, INotifyPropertyChanged, IBehaviorCreator~~**

**~~{~~**

**~~private RadGridView m\_grid;~~**

**~~private string m\_gridViewNewRowText;~~**

**~~private List<HideableColumn> m\_hideableColumns;~~**

**~~private DataContractSerializer m\_serializer;~~**

**~~public string GridViewNewRowText~~**

**~~{~~**

**~~get => m\_gridViewNewRowText;~~**

**~~set~~**

**~~{~~**

**~~m\_gridViewNewRowText = value;~~**

**~~OnPropertyChanged(nameof(GridViewNewRowText));~~**

**~~}~~**

**~~}~~**

**~~public Behavior Create()~~**

**~~{~~**

**~~return new SaveColumnSettingsBehavior();~~**

**~~}~~**

**~~public event PropertyChangedEventHandler PropertyChanged;~~**

**~~protected override void OnAttached()~~**

**~~{~~**

**~~base.OnAttached();~~**

**~~m\_grid = AssociatedObject;~~**

**~~if (m\_grid == null)~~**

**~~{~~**

**~~return;~~**

**~~}~~**

**~~m\_grid.Loaded += OnLoaded;~~**

**~~m\_serializer = new DataContractSerializer(typeof(GridColumnProperties));~~**

**~~}~~**

**~~private void OnFilterOperatorsLoading(object sender, FilterOperatorsLoadingEventArgs e)~~**

**~~{~~**

**~~e.DefaultOperator1 = FilterOperator.Contains;~~**

**~~}~~**

**~~private void RefreshGridViewNewRow(object sender, EventArgs e)~~**

**~~{~~**

**~~GridViewNewRowText = string.Format(Localisation.RowCountStatistics, m\_grid.Items.Count, m\_grid.SelectedItems.Count);~~**

**~~}~~**

**~~protected override void OnDetaching()~~**

**~~{~~**

**~~base.OnDetaching();~~**

**~~m\_grid.ColumnWidthChanged -= OnColumnWidthChanged;~~**

**~~m\_grid.ColumnDisplayIndexChanged -= OnColumnDisplayIndexChanged;~~**

**~~m\_grid.Loaded -= OnLoaded;~~**

**~~m\_grid.DataLoaded -= RefreshGridViewNewRow;~~**

**~~m\_grid.SelectionChanged -= RefreshGridViewNewRow;~~**

**~~m\_grid.Filtered -= RefreshGridViewNewRow;~~**

**~~m\_grid.FilterOperatorsLoading -= OnFilterOperatorsLoading;~~**

**~~}~~**

**~~private void OnColumnWidthChanged(object source, ColumnWidthChangedEventArgs e)~~**

**~~{~~**

**~~SaveGridSettings();~~**

**~~}~~**

**~~private void OnColumnDisplayIndexChanged(object source, GridViewColumnEventArgs e)~~**

**~~{~~**

**~~SaveGridSettings();~~**

**~~}~~**

**~~private void OnColumnVisibilityChanged(object source, EventArgs e)~~**

**~~{~~**

**~~SaveGridSettings();~~**

**~~}~~**

**~~private void OnLoaded(object source, RoutedEventArgs e)~~**

**~~{~~**

**~~LoadGridSettings();~~**

**~~m\_grid.ColumnWidthChanged += OnColumnWidthChanged;~~**

**~~m\_grid.ColumnDisplayIndexChanged += OnColumnDisplayIndexChanged;~~**

**~~m\_grid.FilterOperatorsLoading += OnFilterOperatorsLoading;~~**

**~~m\_grid.DataLoaded += RefreshGridViewNewRow;~~**

**~~m\_grid.SelectionChanged += RefreshGridViewNewRow;~~**

**~~m\_grid.Filtered += RefreshGridViewNewRow;~~**

**~~}~~**

**~~private void SaveGridSettings()~~**

**~~{~~**

**~~if (m\_grid?.Columns == null)~~**

**~~{~~**

**~~return;~~**

**~~}~~**

**~~var properties = new GridColumnProperties();~~**

**~~foreach (var column in m\_grid.Columns)~~**

**~~{~~**

**~~if (column is GridViewDataColumn dataColumn)~~**

**~~{~~**

**~~if (properties.List.Any(c => c.DisplayIndex == column.DisplayIndex))~~**

**~~{~~**

**~~return;~~**

**~~}~~**

**~~var visibility = m\_hideableColumns?.Single(hc => hc.Key == column.UniqueName).IsVisible ?? dataColumn.IsVisible;~~**

**~~var columnProperty = new GridColumnProperty~~**

**~~{~~**

**~~UniqueName = dataColumn.UniqueName,~~**

**~~Width = dataColumn.ActualWidth,~~**

**~~DisplayIndex = dataColumn.DisplayIndex,~~**

**~~IsVisible = visibility~~**

**~~};~~**

**~~properties.Add(columnProperty);~~**

**~~}~~**

**~~}~~**

**~~Save(m\_grid.Uid, properties);~~**

**~~}~~**

**~~private void Save(string Uid, GridColumnProperties properties)~~**

**~~{~~**

**~~try~~**

**~~{~~**

**~~using (var file = IsolatedStorageFile.GetUserStoreForDomain())~~**

**~~{~~**

**~~using (var stream = new IsolatedStorageFileStream(Uid, FileMode.Create, file))~~**

**~~{~~**

**~~m\_serializer.WriteObject(stream, properties);~~**

**~~}~~**

**~~}~~**

**~~}~~**

**~~catch (Exception ex)~~**

**~~{~~**

**~~MessageBox.Show($"Can not save Grid collumns settings: {ex.Message}");~~**

**~~}~~**

**~~}~~**

**~~private void LoadGridSettings()~~**

**~~{~~**

**~~if (m\_grid?.Columns == null)~~**

**~~{~~**

**~~return;~~**

**~~}~~**

**~~var dc = m\_grid.DataContext as IHideableColumns;~~**

**~~if (dc != null)~~**

**~~{~~**

**~~m\_hideableColumns = dc.HideableColumns;~~**

**~~}~~**

**~~var properties = Load(m\_grid.Uid);~~**

**~~if (properties?.List == null)~~**

**~~{~~**

**~~return;~~**

**~~}~~**

**~~var columns = m\_grid.Columns.Cast<GridViewColumn>().ToList();~~**

**~~foreach (var column in columns)~~**

**~~{~~**

**~~Console.WriteLine($@"column.displayIndex = {column.DisplayIndex}");~~**

**~~}~~**

**~~foreach (var gridColumnProperty in properties.List.OrderByDescending(d => d.DisplayIndex).ToList())~~**

**~~{~~**

**~~var index = gridColumnProperty.DisplayIndex;~~**

**~~var column = columns.SingleOrDefault(d => d.UniqueName == gridColumnProperty.UniqueName);~~**

**~~column.DisplayIndex = index;~~**

**~~}~~**

**~~//foreach (var column in columns.OrderByDescending(d => d.DisplayIndex).ToList())~~**

**~~//{~~**

**~~// var columnProp = properties.List.SingleOrDefault(cp => cp.UniqueName == column.UniqueName);~~**

**~~// if (columnProp != null)~~**

**~~// {~~**

**~~// //m\_grid.ReorderColumns(column.DisplayIndex, columnProp.DisplayIndex);~~**

**~~// column.DisplayIndex = columnProp.DisplayIndex;~~**

**~~// column.Width = new GridViewLength(columnProp.Width);~~**

**~~// if (m\_hideableColumns == null)~~**

**~~// {~~**

**~~// column.IsVisible = columnProp.IsVisible;~~**

**~~// column.PropertyChanged += OnColumnVisibilityChanged;~~**

**~~// }~~**

**~~// else~~**

**~~// {~~**

**~~// m\_hideableColumns.Single(hc => hc.Key == column.UniqueName).IsVisible = columnProp.IsVisible;~~**

**~~// m\_hideableColumns.Single(hc => hc.Key == column.UniqueName).ColumnVisibilityChanged += OnColumnVisibilityChanged;~~**

**~~// }~~**

**~~// }~~**

**~~// else // sloupec je nový, musí se ponechat na své výchozí. Všechny následující se musí o 1 posunout~~**

**~~// {~~**

**~~// columns.ForEach(c =>~~**

**~~// {~~**

**~~// if (c.DisplayIndex > column.DisplayIndex && c.DisplayIndex < columns.Count - 1)~~**

**~~// {~~**

**~~// c.DisplayIndex++;~~**

**~~// }~~**

**~~// });~~**

**~~// properties.List.ForEach(c =>~~**

**~~// {~~**

**~~// if (c.DisplayIndex >= column.DisplayIndex)~~**

**~~// {~~**

**~~// c.DisplayIndex++;~~**

**~~// }~~**

**~~// });~~**

**~~// }~~**

**~~//}~~**

**~~dc?.ShowHideColumnRequested(null);~~**

**~~}~~**

**~~private GridColumnProperties Load(string Uid)~~**

**~~{~~**

**~~try~~**

**~~{~~**

**~~using (var file = IsolatedStorageFile.GetUserStoreForDomain())~~**

**~~{~~**

**~~if (!file.FileExists(Uid))~~**

**~~{~~**

**~~return null;~~**

**~~}~~**

**~~using (var stream = new IsolatedStorageFileStream(Uid, FileMode.Open, file))~~**

**~~{~~**

**~~if (stream.Length > 0)~~**

**~~{~~**

**~~var loaded = (GridColumnProperties) m\_serializer.ReadObject(stream);~~**

**~~return loaded;~~**

**~~}~~**

**~~}~~**

**~~}~~**

**~~}~~**

**~~catch (Exception ex)~~**

**~~{~~**

**~~MessageBox.Show($"Can not load Grid collumns settings: {ex.Message}");~~**

**~~}~~**

**~~return null;~~**

**~~}~~**

**~~protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)~~**

**~~{~~**

**~~PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));~~**

**~~}~~**

**~~}~~**

**~~[DataContract]~~**

**~~internal class GridColumnProperty~~**

**~~{~~**

**~~[DataMember] public string Path { get; set; }~~**

**~~[DataMember] public string UniqueName { get; set; }~~**

**~~[DataMember] public object Header { get; set; }~~**

**~~[DataMember] public double Width { get; set; }~~**

**~~[DataMember] public int DisplayIndex { get; set; }~~**

**~~[DataMember] public bool IsVisible { get; set; }~~**

**~~}~~**

**~~[DataContract]~~**

**~~internal class GridColumnProperties~~**

**~~{~~**

**~~public GridColumnProperties()~~**

**~~{~~**

**~~List = new List<GridColumnProperty>();~~**

**~~}~~**

**~~[DataMember] public List<GridColumnProperty> List { get; set; }~~**

**~~public void Add(GridColumnProperty gridColumnProperty)~~**

**~~{~~**

**~~List.Add(gridColumnProperty);~~**

**~~}~~**

**~~}~~**

**~~}~~**

Posledni verze , funguje I pridavani novych sloupcu:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.IO;

using System.IO.IsolatedStorage;

using System.Linq;

using System.Reflection;

using System.Runtime.CompilerServices;

using System.Runtime.Serialization;

using System.Windows;

using System.Windows.Interactivity;

using log4net;

using Mediaresearch.Framework.Gui.Telerik.ColumnHidingPopup;

using Mediaresearch.Framework.Gui.Telerik.Resources;

using Telerik.Windows.Controls;

using Telerik.Windows.Controls.GridView;

using Telerik.Windows.Data;

namespace Mediaresearch.Framework.Gui.Telerik.Behaviors

{

public class SaveColumnSettingsBehavior : Behavior<RadGridView>, INotifyPropertyChanged, IBehaviorCreator

{

private static readonly ILog m\_log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private RadGridView m\_grid;

private string m\_gridViewNewRowText;

private List<HideableColumn> m\_hideableColumns;

private DataContractSerializer m\_serializer;

public string GridViewNewRowText

{

get => m\_gridViewNewRowText;

set

{

m\_gridViewNewRowText = value;

OnPropertyChanged(nameof(GridViewNewRowText));

}

}

public Behavior Create()

{

return new SaveColumnSettingsBehavior();

}

public event PropertyChangedEventHandler PropertyChanged;

protected override void OnAttached()

{

base.OnAttached();

m\_grid = AssociatedObject;

if (m\_grid == null)

{

return;

}

m\_grid.Loaded += OnLoaded;

m\_serializer = new DataContractSerializer(typeof(GridColumnProperties));

}

private void OnFilterOperatorsLoading(object sender, FilterOperatorsLoadingEventArgs e)

{

e.DefaultOperator1 = FilterOperator.Contains;

}

private void RefreshGridViewNewRow(object sender, EventArgs e)

{

GridViewNewRowText = string.Format(Localisation.RowCountStatistics, m\_grid.Items.Count, m\_grid.SelectedItems.Count);

}

protected override void OnDetaching()

{

base.OnDetaching();

m\_grid.ColumnWidthChanged -= OnColumnWidthChanged;

m\_grid.ColumnDisplayIndexChanged -= OnColumnDisplayIndexChanged;

m\_grid.Loaded -= OnLoaded;

m\_grid.DataLoaded -= RefreshGridViewNewRow;

m\_grid.SelectionChanged -= RefreshGridViewNewRow;

m\_grid.Filtered -= RefreshGridViewNewRow;

m\_grid.FilterOperatorsLoading -= OnFilterOperatorsLoading;

}

private void OnColumnWidthChanged(object source, ColumnWidthChangedEventArgs e)

{

SaveGridSettings();

}

private void OnColumnDisplayIndexChanged(object source, GridViewColumnEventArgs e)

{

SaveGridSettings();

}

private void OnColumnVisibilityChanged(object source, EventArgs e)

{

SaveGridSettings();

}

private void OnLoaded(object source, RoutedEventArgs e)

{

LoadGridSettings();

m\_grid.ColumnWidthChanged += OnColumnWidthChanged;

m\_grid.ColumnDisplayIndexChanged += OnColumnDisplayIndexChanged;

m\_grid.FilterOperatorsLoading += OnFilterOperatorsLoading;

m\_grid.DataLoaded += RefreshGridViewNewRow;

m\_grid.SelectionChanged += RefreshGridViewNewRow;

m\_grid.Filtered += RefreshGridViewNewRow;

}

private void SaveGridSettings()

{

if (m\_grid?.Columns == null)

{

return;

}

var gridColumns = m\_grid.Columns.Cast<GridViewColumn>().ToList();

if (AnyUnNamedColumns(gridColumns))

{

m\_log.Error($"Can not save column settings for grid {m\_grid.Uid}. Some column has no Header neither UniqueName");

return;

}

var properties = new GridColumnProperties();

foreach (var column in m\_grid.Columns)

{

if (column is GridViewDataColumn dataColumn)

{

if (properties.List.Any(c => c.DisplayIndex == column.DisplayIndex))

{

return;

}

var visibility = m\_hideableColumns?.Single(hc => hc.Key == column.UniqueName).IsVisible ?? dataColumn.IsVisible;

var columnProperty = new GridColumnProperty

{

UniqueName = dataColumn.UniqueName,

Width = dataColumn.ActualWidth,

DisplayIndex = dataColumn.DisplayIndex,

IsVisible = visibility

};

if (properties.Contains(columnProperty.UniqueName, columnProperty.Header?.ToString()))

{

m\_log.Error($"Can not save settings for column header: '{(string) columnProperty.Header}' / UniqueName: '{columnProperty.UniqueName}' is not unique. Grid UID: '{m\_grid.Uid}'.");

}

else

{

properties.Add(columnProperty);

}

}

else

{

throw new NotSupportedException($"Column of type {column.GetType()} is not supported!");

}

}

Save(m\_grid.Uid, properties);

}

private void Save(string Uid, GridColumnProperties properties)

{

m\_log.Debug($"Saving columns settings for grid Uid: '{Uid}'...");

try

{

using (var file = IsolatedStorageFile.GetUserStoreForDomain())

{

using (var stream = new IsolatedStorageFileStream(Uid, FileMode.Create, file))

{

m\_serializer.WriteObject(stream, properties);

}

}

}

catch (Exception ex)

{

m\_log.Error($"Can not save Grid collumns settings: {ex.Message}");

}

}

private void LoadGridSettings()

{

if (m\_grid?.Columns == null)

{

return;

}

var dc = m\_grid.DataContext as IHideableColumns;

if (dc != null)

{

m\_hideableColumns = dc.HideableColumns;

}

var properties = Load(m\_grid.Uid);

if (properties?.List == null)

{

return;

}

var columns = m\_grid.Columns.Cast<GridViewColumn>().ToList();

if (AnyUnNamedColumns(columns))

{

m\_log.Error($"Can not load column settings for grid {m\_grid.Uid}. Some column has no Header neither UniqueName");

return;

}

foreach (var col in columns)

{

var colProp = properties.List.SingleOrDefault(cp => !string.IsNullOrEmpty(col.UniqueName) ? col.UniqueName.Equals(cp.UniqueName) : col.Header.ToString().Equals(cp.Header.ToString()));

if (colProp == null)

{

m\_log.Debug($"Adding new columnProperty: '{col.Header}' / UniqueName:'{col.UniqueName}'");

var columnProperty = new GridColumnProperty

{

UniqueName = col.UniqueName,

Width = col.ActualWidth,

DisplayIndex = col.DisplayIndex,

IsVisible = col.IsVisible

};

properties.List.Where(d => d.DisplayIndex >= columnProperty.DisplayIndex).ToList().ForEach(x => x.DisplayIndex++);

properties.List.Add(columnProperty);

}

}

foreach (var columnProp in properties.List.OrderByDescending(d => d.DisplayIndex).ToList())

{

m\_log.Debug($"Searching for loaded values of column header: '{columnProp.Header}' / UniqueName:'{columnProp.UniqueName}'");

try

{

var column = columns.SingleOrDefault(c => !string.IsNullOrEmpty(c.UniqueName) ? columnProp.UniqueName.Equals(c.UniqueName) : columnProp.Header.ToString().Equals(c.Header.ToString()));

if (column != null)

{

m\_log.Debug("Setting values to grid column.");

column.DisplayIndex = columnProp.DisplayIndex;

column.Width = new GridViewLength(columnProp.Width);

if (m\_hideableColumns == null)

{

column.IsVisible = columnProp.IsVisible;

column.PropertyChanged += OnColumnVisibilityChanged;

}

else

{

m\_hideableColumns.Single(hc => hc.Key == column.UniqueName).IsVisible = columnProp.IsVisible;

m\_hideableColumns.Single(hc => hc.Key == column.UniqueName).ColumnVisibilityChanged += OnColumnVisibilityChanged;

}

}

m\_log.Debug("Column finished.");

}

catch (Exception e)

{

m\_log.Error($"Could not set column property values for column UniqueName: '{columnProp.UniqueName}': {e.Message}");

}

}

dc?.ShowHideColumnRequested(null);

}

private GridColumnProperties Load(string Uid)

{

m\_log.Debug($"Searching file to load columns settings for grid Uid: {Uid}");

try

{

using (var file = IsolatedStorageFile.GetUserStoreForDomain())

{

if (!file.FileExists(Uid))

{

m\_log.Debug("No file found. Default values will be used.");

return null;

}

using (var stream = new IsolatedStorageFileStream(Uid, FileMode.Open, file))

{

m\_log.Debug($"Reading data from file {stream?.GetType()?.GetField("m\_FullPath", BindingFlags.Instance | BindingFlags.NonPublic)?.GetValue(stream)}...");

if (stream.Length > 0)

{

var loaded = (GridColumnProperties) m\_serializer.ReadObject(stream);

m\_log.Debug("Data loaded.");

return loaded;

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Can not load Grid collumns settings: {ex.Message}");

}

return null;

}

private bool AnyUnNamedColumns(List<GridViewColumn> columns)

{

return columns.Any(x => string.IsNullOrEmpty(x.UniqueName) && string.IsNullOrEmpty(x.Header?.ToString()));

}

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

[DataContract]

internal class GridColumnProperty

{

[DataMember] public string Path { get; set; }

[DataMember] public string UniqueName { get; set; }

[DataMember] public object Header { get; set; }

[DataMember] public double Width { get; set; }

[DataMember] public int DisplayIndex { get; set; }

[DataMember] public bool IsVisible { get; set; }

}

[DataContract]

internal class GridColumnProperties

{

public GridColumnProperties()

{

List = new List<GridColumnProperty>();

}

[DataMember] public List<GridColumnProperty> List { get; set; }

public void Add(GridColumnProperty gridColumnProperty)

{

List.Add(gridColumnProperty);

}

public bool Contains(string uniqueName, string header)

{

return List.Any(x => (x.UniqueName == null || x.UniqueName.Equals(uniqueName)) && (x.Header == null || x.Header.Equals(header)));

}

}

}