Руководство по эксплуатации

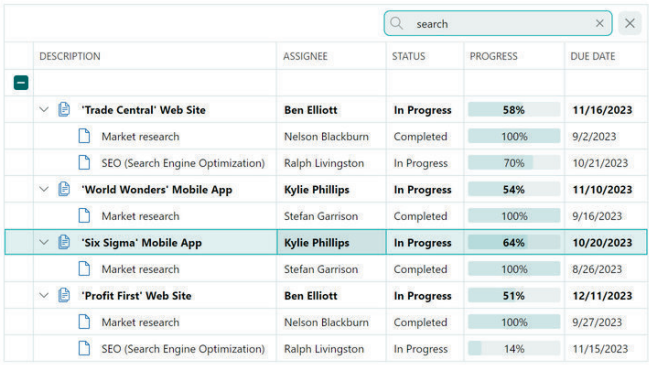
библиотека контролов для Avalonia UI

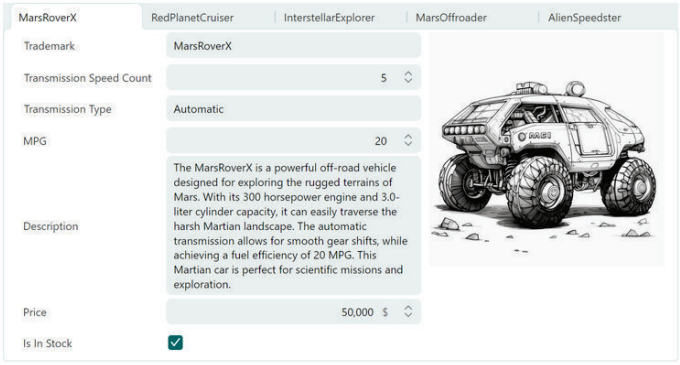
«EMX Controls 1.X»

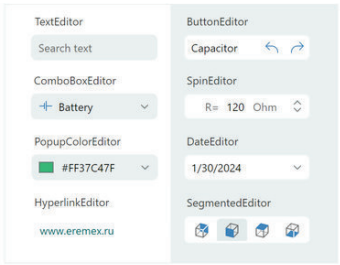
АО "ЭРЕМЕКС"
controls@eremex.com

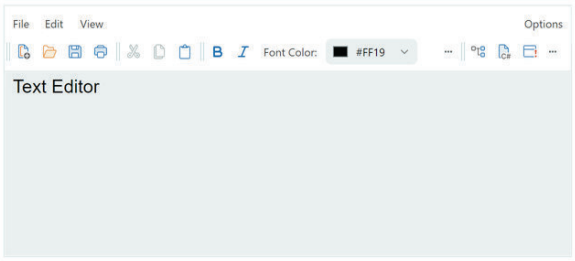
EMX Controls 1.X

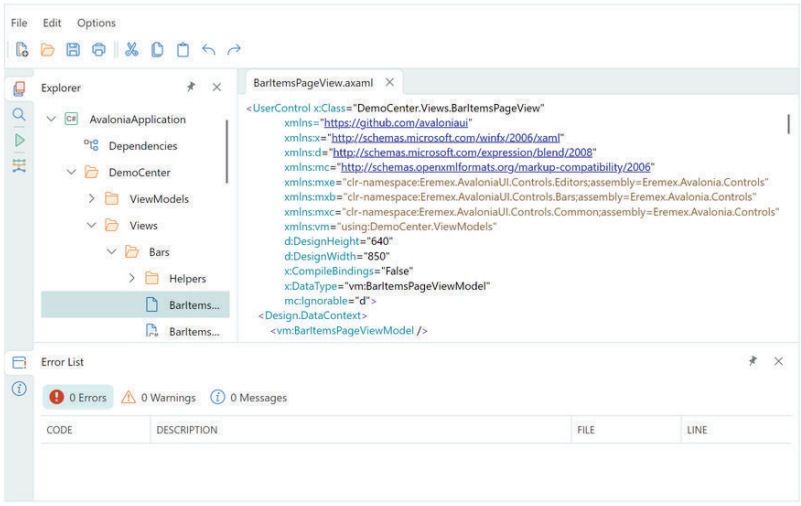
Библиотека контролов EMX Controls для Avalonia UI включает в себя многофункциональные контролы пользовательского интерфейса и библиотеки утилит для Avalonia framework, которые помогут вам создавать гибко настраиваемые кроссплатформенные приложения с улучшенным UX.











Контролы для отображения табличных данных

- DataGrid
- TreeList и TreeView
- PropertyGrid

Классический офисный пользовательский интерфейс

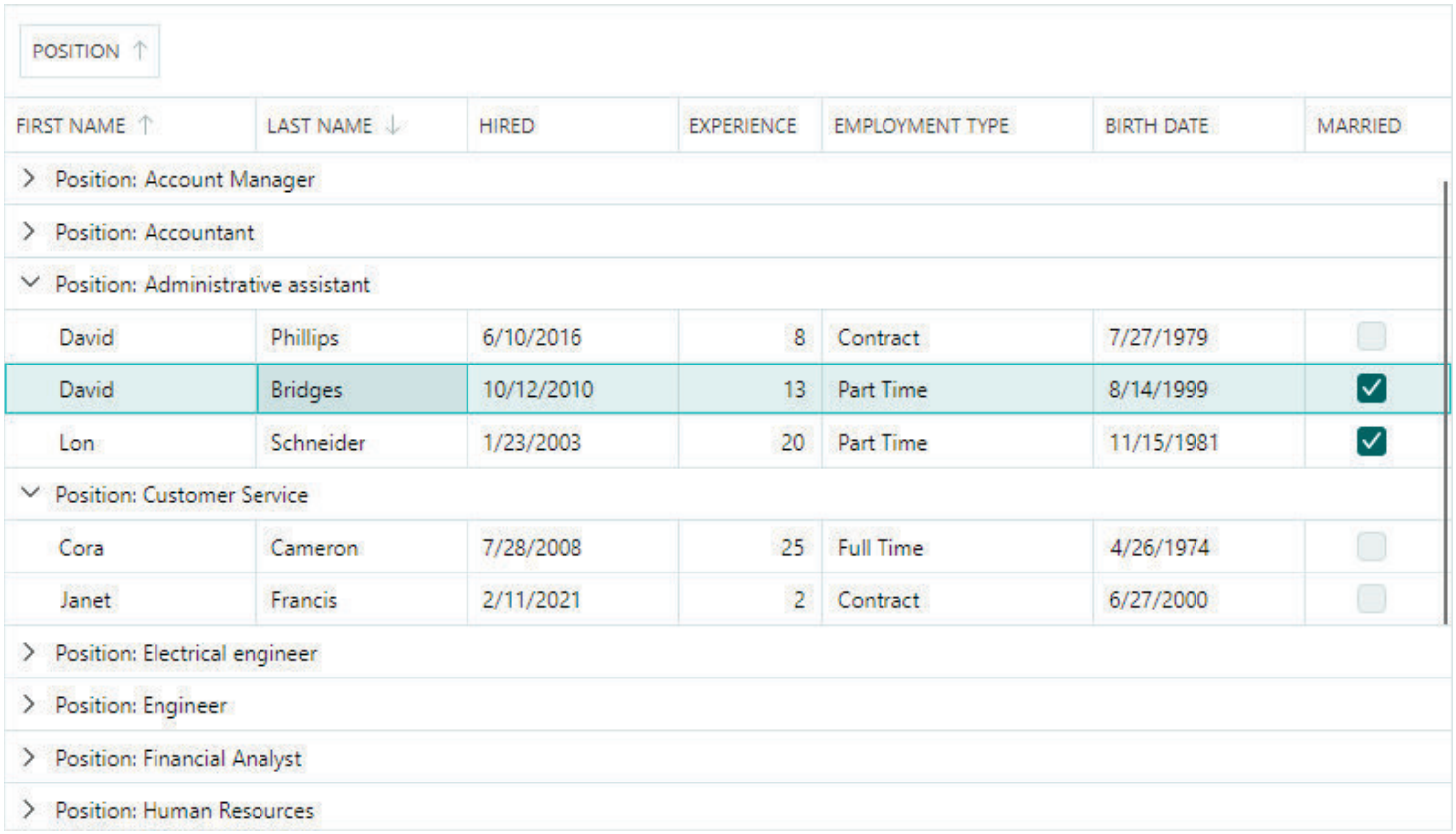
- Панели инструментов и меню
- Интерфейс Докинга

Редакторы и вспомогательные контролы

- Редакторы данных
- Вспомогательные контролы

DataGrid

`DataGridControl` позволяет отображать данные из источника объектов в виде строк и столбцов. Он обеспечивает широкие возможности формирования и редактирования данных. Пользователи могут переставлять столбцы, редактировать, сортировать, группировать и выполнять поиск данных.



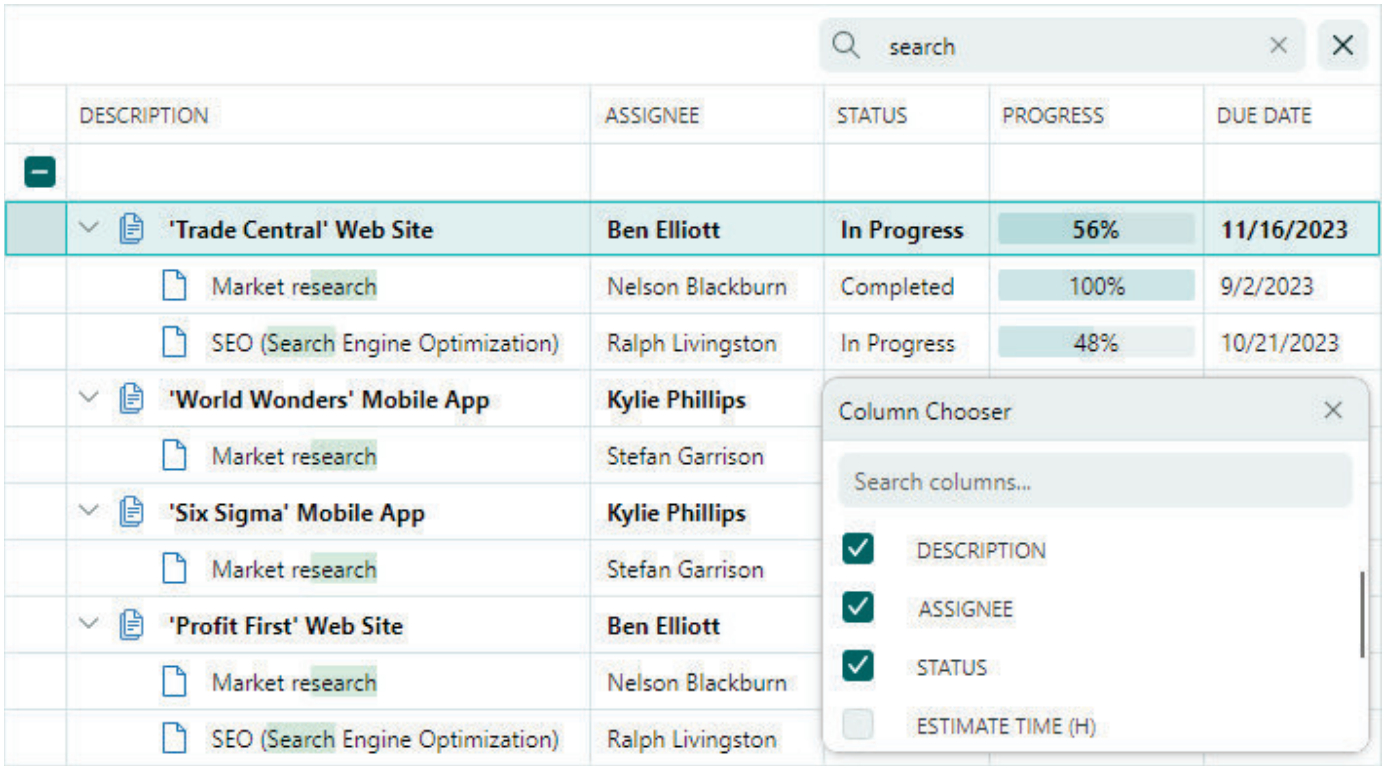
- Привязка данных — вы можете привязать контрол к IList, IBindingList, DataTable или любому источнику данных IEnumerable.
- Сортировка данных — позволяет сортировать данные по неограниченному количеству столбцов.
- Группировка данных — Функция группировки данных объединяет строки с одинаковыми значениями в столбцах в одни и те же группы данных. Вы можете сгруппировать данные контрола по нескольким столбцам.
- Стили — позволяют настраивать свойства внешнего вида элементов контрола в различных состояниях.
- Операции редактирования данных — пользователь может редактировать значения ячеек, если включено редактирование данных. Вы можете встроить Eremex и пользовательские редакторы в ячейки, чтобы редактировать и представлять значения ячеек определенным образом.

- Валидация данных — Механизм валидации помогает проверять вводимые пользователем значения и значения источника данных и показывать ошибки в ячейках.
- Встроенное и пользовательское контекстное меню.
- Несвязанные столбцы — Вы можете добавить несвязанные столбцы (те, которые не привязаны к полям источника данных) и заполнить их данными вручную, используя событие.
- Операции изменения размера и перемещения столбцов.
- Панель поиска — помогает пользователю быстро найти строки по содержащимся в них данным.
- Строка автоматической фильтрации — специальная строка, которая позволяет пользователю фильтровать данные по столбцам.
- Шаблоны заголовков столбцов – позволяют отображать пользовательское содержимое в заголовках столбцов, включая картинки.
- Выбор нескольких строк (выделение) — Вы можете включить режим выбора нескольких строк, чтобы позволить пользователю выбирать (выделять) несколько строк одновременно.
- Поддержка атрибутов Data Annotation — контрол DataGrid учитывает определенные атрибуты Data Annotation, применяемые к свойствам источника данных. Вы можете использовать атрибуты Data Annotation, чтобы указать пользовательскую видимость, позицию, состояние "только для чтения" и отображаемое имя для автоматически сгенерированных столбцов.

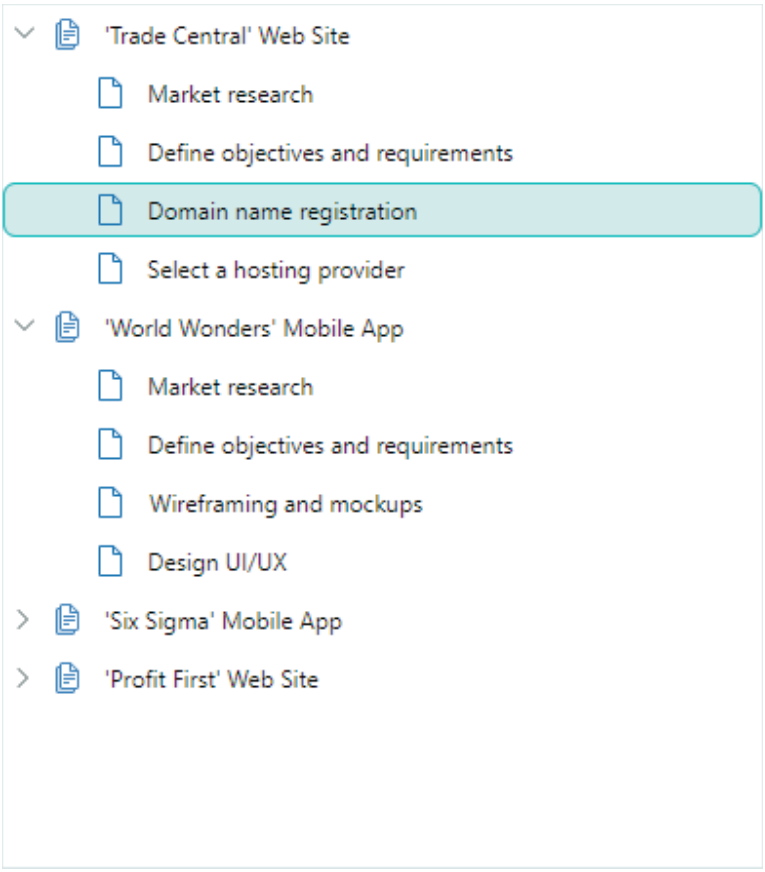
TreeList и TreeView

Библиотека контролов Eremex включает в себя два работающих с данными контрола для отображения иерархических данных в виде дерева - `TreeListControl` и `TreeViewControl`. Они отображают элементы источника данных в виде узлов (строк). Если узлу принадлежат дочерние узлы, вы можете развернуть его, чтобы отобразить следующий уровень иерархии узлов.

`TreeList` поддерживает несколько столбцов.



`TreeView` - это контрол с одним столбцом.



Контролы наследованы от одного предка, поэтому они имеют множество общих функций:

- Привязка данных — вы можете привязать контролы к самореферентным (плоским) и иерархическим источникам данных.
- Несвязанный режим — позволяет вручную создать структуру узла.
- Встроенные флажки узлов — позволяют выбирать отдельные узлы.
- Сортировка данных — Позволяет сортировать одноуровневые узлы в порядке возрастания или убывания. TreeList поддерживает сортировку данных по одному или нескольким столбцам.
- Картинки узлов — позволяют отображать пользовательские картинки перед значениями ячеек в столбце иерархии.
- Стили — позволяют настраивать свойства внешнего вида элементов контролов в различных состояниях.
- Панель поиска — помогает пользователю быстро найти узлы по содержащимся в них данным.
- Операции редактирования данных — пользователь может редактировать значения ячеек, если включено редактирование данных. Вы можете встроить Eremex и пользовательские редакторы в ячейки, чтобы редактировать и представлять значения ячеек определенным образом.
- Валидация данных — Механизм валидации помогает проверять вводимые пользователем значения и значения источника данных и показывать ошибки в ячейках.
- Встроенное и пользовательское контекстное меню.
- Поддержка атрибутов Data Annotation — контролы TreeList и TreeView учитывают определенные атрибуты Data Annotation, применяемые к свойствам источника данных. Вы можете использовать атрибуты Data Annotation, чтобы указать пользовательскую видимость,

позицию, состояние "только для чтения" и отображаемое имя для автоматически сгенерированных столбцов.

Специфичные для TreeList функции включают в себя:

- Несвязанные столбцы — Вы можете добавить несвязанные столбцы (те, которые не привязаны к полям источника данных) и заполнить их данными вручную, используя событие.
- Операции изменения размера и перемещения столбцов.
- Строка автоматической фильтрации — специальная строка, которая позволяет пользователю фильтровать данные по столбцам.
- Шаблоны заголовков столбцов – позволяют отображать пользовательское содержимое в заголовках столбцов, включая картинки.
- Выбор нескольких узлов (выделение) — Вы можете включить режим выбора нескольких узлов, чтобы позволить пользователю выбирать (выделять) несколько узлов одновременно.

Смотрите следующий раздел для получения дополнительной информации: Обзор контролов TreeList и TreeView .

Обзор контролов TreeList и TreeView

Контролы TreeList (класс `TreeListControl`) и TreeView (класс `TreeViewControl`) позволяют отображать иерархические данные в виде дерева, где узлы (записи) могут иметь другие узлы в качестве дочерних.

Как контролы `TreeListControl`, так и `TreeViewControl` являются потомками `TreeListControlBase`, и как таковые, они имеют много общих функций. Разница между контролами заключается в том, что TreeList поддерживает несколько столбцов данных — каждый узел (строка) отображает одну или несколько ячеек. TreeView может отображать только один столбец данных.

Привязка данных

Контролы TreeList и TreeView поддерживают два типа источников данных: самореферентные (плоские) и иерархические. Они отличаются тем, как записи хранят информацию об иерархии.

Помимо связанного режима, контролы могут функционировать в несвязанном режиме, в котором вы вручную добавляете узлы и, таким образом, создаете иерархию.

Контрол TreeList поддерживает несвязанные столбцы, которые вы можете использовать для отображения произвольных данных.

Дополнительные сведения см. в следующих разделах:

- Привязка данных
- Режим без привязки к данным
- Несвязанные столбцы

Узлы

Контролы TreeList и TreeView создают узлы, как только вы привязываете их к источнику данных. Каждый узел в контроле TreeView отображает одно значение. Узел контрола TreeList отображает значения для всех столбцов.

Контролы также поддерживают несвязанный режим, в котором вы можете создать иерархию узлов вручную.

Смотрите следующий раздел для получения дополнительной информации: Узлы .

Столбцы (TreeList)

Контрол TreeList отображает данные в табличной форме, в которой данные размещены в строках (узлах) и столбцах. В отличие от контрола TreeView, TreeList поддерживает несколько столбцов.

Смотрите следующий раздел о опциях настройки столбцов и операциях с столбцами: Столбцы .

Стили

Механизм стилей позволяет вам настраивать свойства внешнего вида визуальных элементов контролов в различных состояниях.

Смотрите следующий раздел для получения подробной информации о механизме стилизации: Стили .

Редактирование данных

Контролы TreeView и TreeList используют встроенные редакторы Eremex для отображения и редактирования значений ячеек по умолчанию. Вы можете явно назначить редакторы столбцам контролов, чтобы настроить способ отображения и редактирования значений ячеек.

Дополнительные сведения см. в следующих разделах:

- Редактирование данных
- Пользовательские редакторы в столбцах TreeView и TreeList

Поиск и фильтрация данных

Функции поиска и фильтрации данных позволяют вам и вашему пользователю находить узлы, содержащие определенные значения.

Вы можете найти более подробную информацию в следующем разделе: Поиск и фильтрация .

Сортировка

Сортировка данных позволяет контролам TreeList и TreeView размещать одноуровневые узлы в порядке возрастания или убывания. TreeList поддерживает сортировку данных по нескольким столбцам. Кроме того, пользователи могут сортировать данные в TreeList, используя действия мыши и контекстное меню заголовка столбца.

Смотрите следующий раздел для получения дополнительной информации: Сортировка данных .

Контекстные меню

Всплывающие меню отображаются, когда пользователь щелкает правой кнопкой мыши контролы. Смотрите следующий раздел, чтобы узнать, как настроить всплывающие меню по умолчанию и отображать пользовательские меню для определенных элементов пользовательского интерфейса: Контекстные меню .

Привязка данных

Как только контрол TreeList/TreeView привязан к источнику данных, он создает узлы (объекты `TreeListNode`) для записей источника данных. Узлы изначально пусты. Вам необходимо указать свойства записи данных (поля источника данных), значения которых отображаются в узлах.

Чтобы отобразить значения в узлах TreeList, создайте связанные или несвязанные столбцы в коллекции `TreeListControl.Columns`. Включите опцию `TreeListControl.AutoGenerateColumns`, чтобы автоматически генерировать отсутствующие столбцы для всех свойств/полей источника данных после привязки контрола. Смотрите разделы Столбцы и Несвязанные столбцы для получения дополнительной информации.

Контрол TreeView - это версия TreeList с одним столбцом. Используйте свойство `TreeViewControl.DataFieldName`, чтобы указать, какие значения отображать в его узлах в режиме привязки. Этот элемент определяет имя свойства/поля в источнике данных, данные которого отображаются контролом.

Свойство `ItemsSource` позволяет привязать контрол TreeList/TreeView к источнику данных, который содержит информацию о родительско-дочерних связях между записями. Контролы поддерживают два типа источников данных, которые отличаются способом кодирования информации об иерархии:

- Самореферентный (плоский) источник данных.
- Иерархический источник данных

Самореферентный (плоский) источник данных

Самореферентный источник данных - это плоская таблица, коллекция или список, в которых записи имеют два служебных свойства/поля, используемых для создания родительско-дочерних связей между записями:

- Ключевое поле — уникальный идентификатор записи.
- Поле родительского ключа — хранит значение ключевого поля родительской записи.

Как правило, и ключевое поле, и родительское ключевое поле имеют целочисленный тип данных. Родительский и дочерний объекты данных всегда имеют один и тот же тип данных.

Смотрите Привязка к самореферентному источнику данных для получения информации о том, как привязать контролы TreeList/TreeView к самореферентному (плоскому) источнику данных.

Общий API

- `ItemsSource` — Источник данных контрола.
- `KeyFieldName` — имя поля, в котором хранятся уникальные идентификаторы записей (значения ключевых полей).
- `ParentFieldName` — Имя поля, в котором хранится идентификатор (значение ключевого поля) родительской записи.
- `RootValue` — значение поля родительского ключа корневого узла.

API контрола TreeList

- `Columns` — Коллекция связанных и несвязанных столбцов TreeList.
- `AutoGenerateColumns` — Указывает, будет ли TreeList автоматически генерировать отсутствующие столбцы для публичных свойств/полей в источнике данных. Если коллекция `Columns` контрола уже содержит столбец, привязанный к определенному свойству/полю, автоматически не генерируется дополнительный столбец, привязанный к тому же свойству/полю.
- `AutoGenerateServiceColumns` — Указывает, будет ли TreeList автоматически генерировать столбцы, привязанные к ключевому полю и родительскому ключевому полю. Это свойство действует, если включена опция `AutoGenerateColumns`.

API контрола TreeView

- `DataFieldName` — название поля, данные которого отображаются в контроле.

Иерархический источник данных

В иерархическом источнике данных бизнес-объект (запись) имеет свойство, в котором хранится коллекция дочерних объектов данных. Родительский и дочерний объекты данных могут быть разных типов, но у них должен быть общий набор свойств, которые контролы TreeView/TreeList отображают в виде столбцов.

При привязке к иерархическому источнику данных TreeList и TreeView контролируют загрузку узлов по требованию: дочерние узлы динамически загружаются при развертывании родительского узла. Это накладывает ограничения на функции проверки узлов и фильтрации/поиска.

Установите для свойства `AllowDynamicDataLoading` значение `false`, чтобы загрузить все узлы одновременно, как только вы привяжете контрол к источнику данных.

Смотрите Привязка к иерархическим данным для получения дополнительной информации.

Общий API

- `ItemsSource` — Установите это свойство для объекта, содержащего данные (например, коллекцию корневых объектов), используемые для создания корневых узлов.
- `ChildrenSelector` — селектор, который возвращает дочерние объекты для каждого бизнес-объекта (записи). Используйте либо `ChildrenSelector`, либо `ChildrenFieldName`.
- `ChildrenFieldName` — имя свойства (поля), в котором хранятся дочерние объекты для каждого бизнес-объекта. Используйте либо `ChildrenSelector`, либо `ChildrenFieldName`.
- `HasChildrenFieldName` — Имя логического свойства, которое возвращает `true`, если у объекта есть дочерние объекты. Свойство `HasChildrenFieldName` позволяет контролу динамически определять видимость кнопок "Развернуть" ('+'). Используйте `HasChildrenFieldName` вместе со свойством `ChildrenFieldName`.

API контрола TreeView

- `DataFieldName` — название поля, данные которого отображаются в контроле.

Смотрите также

- Привязка к иерархическим данным
- Привязка к самореферентному источнику данных
- Несвязанные столбцы
- Режим без привязки к данным

Привязка к иерархическим данным

В типичном иерархическом источнике данных бизнес-объект имеет свойство, в котором хранится коллекция дочерних объектов данных. Два подхода позволяют вам предоставлять дочерние данные в контролы TreeList и TreeView в режиме иерархической привязки:

- Укажите путь к дочерним данным
- Укажите дочерний селектор данных

Родительский и дочерний объекты данных могут быть разных типов, но у них должен быть общий набор идентичных свойств, которые контрол TreeList отображает в виде столбцов.

Динамическая загрузка данных

При привязке к иерархическому источнику данных TreeList и TreeView контролируют загрузку узлов по требованию по умолчанию: дочерние узлы динамически загружаются при развертывании родительского узла.

Установите для свойства `AllowDynamicDataLoading` значение `false`, чтобы загрузить все узлы одновременно после привязки контрола к источнику данных.

Если вы используете динамическую загрузку узлов, контролы TreeList и TreeView не имеют доступа к узлам (и их нижележащим данным), которые не были загружены. Это накладывает следующие ограничения на функции проверки узлов и фильтрации/поиска:

- Когда вы проверяете родительский узел в рекурсивном режиме (см. `AllowRecursiveNodeChecking`), контрол проверяет этот узел вместе с загруженными в данный момент дочерними узлами. Контрол не проверяет дочерние узлы, которые не были загружены.
- Когда вы выполняете поиск данных во встроенном окне поиска или фильтруете данные с помощью строки автоматической фильтрации, контрол выполняет поиск данных только по загруженным в данный момент узлам.

Путь к дочерним данным

Один из способов предоставить дочерние данные в контрол TreeList/TreeView заключается в указании имени свойства, которое хранит дочерние данные в бизнес-объекте. Используйте свойство `ChildrenFieldName` для этой цели. Например, если бизнес-объект хранит дочерние данные в коллекции `Items`, установите для свойства `ChildrenFieldName` значение "*Items*".

Вам также необходимо присвоить свойству `HasChildrenFieldName` имя свойства, которое возвращает `true`, если бизнес-объект имеет дочерние данные, и `false`, в противном случае. Эта информация необходима для отображения или скрытия кнопок расширения узла. Если вы не укажете свойство `HasChildrenFieldName`, контрол отображает кнопки расширения для дочерних узлов до тех пор, пока пользователь не попытается развернуть эти узлы.

В следующем примере показано, как привязать контрол TreeList к данным. Объект *Employee* в примере содержит коллекцию *Subordinates*, которая содержит дочерние элементы. Свойство `ChildrenFieldName` указывает имя свойства (строка "*Subordinates*"), в котором хранятся дочерние данные.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
...
<mxtl:TreeListControl Grid.Column="0" Width="400" Name="treeList1" >
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn Name="colName" FieldName="Name" Header="Name"/>
        <mxtl:TreeListColumn Name="colBirthdate" FieldName="Birthdate" Header="Birthdate"/>
    </mxtl:TreeListControl.Columns>
</mxtl:TreeListControl>
```

```
using CommunityToolkit.Mvvm.ComponentModel;

Employee p1 = new Employee() { Name = "Mark Douglas", Birthdate=new DateTime(1990, 01, 5) };
Employee p2 = new Employee() { Name = "Mary Watson", Birthdate = new DateTime(1985, 12, 17) };
Employee p3 = new Employee() { Name = "Alex Wude", Birthdate = new DateTime(2000, 10, 7) };
Employee p4 = new Employee() { Name = "Sam Louis", Birthdate = new DateTime(1975, 8, 27) };
Employee p5 = new Employee() { Name = "Dan Miller", Birthdate = new DateTime(1981, 3, 6) };
p1.Subordinates.Add(p2);
p1.Subordinates.Add(p3);
p2.Subordinates.Add(p4);
p3.Subordinates.Add(p5);

ObservableCollection<Employee> employees = new ObservableCollection<Employee>() { p1 };

treeList1.ChildrenFieldName = "Subordinates";
treeList1.HasChildrenFieldName = "HasChildren";

treeList1.ItemsSource = employees;

public partial class Employee : ObservableObject
{
    [ObservableProperty]
```

```
    public string name="";

    [ObservableProperty]
    public DateTime? birthdate=null;

    public ObservableCollection<Employee> Subordinates { get; } = new();

    public bool HasChildren { get { return Subordinates.Count > 0; } }
}
```

Селектор дочерних данных

Другим подходом к предоставляет дочерние данные в режиме иерархической привязки является реализация *selector*. Селектор указывает, доступны ли дочерние данные в бизнес-объекте, и возвращает эти дочерние данные по запросу. Контрол TreeList/TreeView использует информацию, предоставленную селектором, для создания дочерних узлов и рисования кнопок расширения узлов.

Используйте свойство `ChildrenSelector`, чтобы назначить селектор вашему контролу. Селектор - это объект, который реализует интерфейс `ITreeListChildrenSelector`:

```
public interface ITreeListChildrenSelector
{
        // The method should return whether a business object has child data.
    // The control uses this information to display or hide node expand buttons.
    bool HasChildren(object item) => true;
    // The method should return child objects for a business object.
        IEnumerable? SelectChildren(object item);
}
```

В следующем примере создается селектор (*MyTreeListChildrenSelector*), который предоставляет информацию о дочерних данных для контрола TreeList.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
...
<Grid RowDefinitions="Auto, Auto, Auto, Auto" ColumnDefinitions="400, *">
    <Grid.Resources>
        <local:MyTreeListChildrenSelector x:Key="mySelector"/>
    </Grid.Resources>

    <mxtl:TreeListControl Grid.Column="0" Name="treeList2"
     ChildrenSelector="{StaticResource mySelector}">
        <mxtl:TreeListControl.Columns>
            <mxtl:TreeListColumn Name="colName1" FieldName="Name"/>
            <mxtl:TreeListColumn Name="colBirthdate1" FieldName="Birthdate"/>
        </mxtl:TreeListControl.Columns>
```

```
    </mxtl:TreeListControl>
</Grid>
```

```
using CommunityToolkit.Mvvm.ComponentModel;

Employee p1 = new Employee() { Name = "Mark Douglas", Birthdate=new DateTime(1990, 01, 5) };
Employee p2 = new Employee() { Name = "Mary Watson", Birthdate = new DateTime(1985, 12, 17) };
Employee p3 = new Employee() { Name = "Alex Wude", Birthdate = new DateTime(2000, 10, 7) };
Employee p4 = new Employee() { Name = "Sam Louis", Birthdate = new DateTime(1975, 8, 27) };
Employee p5 = new Employee() { Name = "Dan Miller", Birthdate = new DateTime(1981, 3, 6) };
p1.Subordinates.Add(p2);
p1.Subordinates.Add(p3);
p2.Subordinates.Add(p4);
p3.Subordinates.Add(p5);

ObservableCollection<Employee> employees = new ObservableCollection<Employee>() { p1 };

treeList2.ItemsSource = employees;

public class MyTreeListChildrenSelector : ITreeListChildrenSelector
{
    public bool HasChildren(object? item) => (item as Employee).HasChildren;
    public IEnumerable? SelectChildren(object? item) => (item as Employee).Subordinates;
}

public partial class Employee : ObservableObject
{
    [ObservableProperty]
    public string name="";

    [ObservableProperty]
    public DateTime? birthdate=null;

    public ObservableCollection<Employee> Subordinates = new();

    public bool HasChildren { get { return Subordinates.Count > 0; } }
}
```

Смотрите также

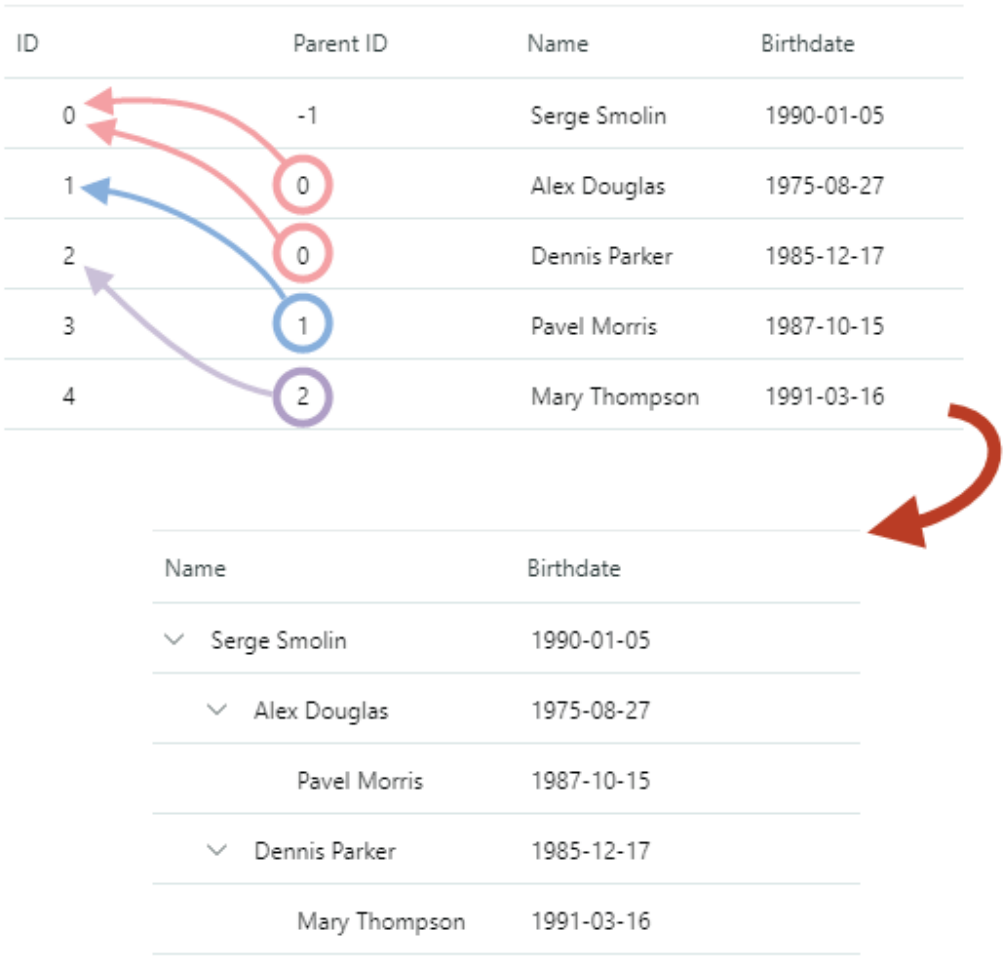
- Привязка данных
- Привязка к самореферентному источнику данных
- Столбцы
- Как создать контрол TreeList и привязать его к иерархическому источнику данных

Привязка к самореферентному источнику данных

Вы можете использовать самореферентный источник данных для кодирования иерархических связей между записями. Самореферентный источник данных - это плоский список или коллекция записей. Его записи имеют два служебных свойства, которые определяют отношения родитель-потомок:

- *Key field* — уникальный идентификатор записи (простого типа данных; например, Integer).
- *Parent key field* — *Key field* родительской записи.

На следующей картинке показан пример таблицы данных с самореферентными ссылками и контрола TreeList, который отображает эти данные после того, как свойства определенных свойств контрола установлены.



Все записи, которые будут отображаться как корневые узлы (на корневом уровне), должны иметь значение *Parent key field*, равное одному и тому же корневому значению. Корневое значение должно быть уникальным — оно не должно совпадать ни с одним ключевым полем в источнике данных.

Чтобы привязать контрол TreeList/TreeView к самореферентному источнику данных, выполните следующее:

- Убедитесь, что записи источника данных имеют два публичных свойства, которые указывают ключевое поле и *Parent key field*.
- Установите для свойства `TreeListControlBase.KeyFieldName` контрола значение "Имя ключевого поля".
- Установите для свойства `TreeListControlBase.ParentFieldName` контрола значение *Parent key field* name.
- Установите для свойства `TreeListControlBase.RootValue` контрола значение root, определенное для корневых записей в источнике данных.

Видимость служебных столбцов

TreeList по умолчанию не создает столбцы, привязанные к полям служебных ключей. Чтобы разрешить контролу автоматически создавать столбцы, привязанные к указанным *Key field* и *Parent key field*, установите для свойств `AutoGenerateColumns` и `AutoGenerateServiceColumns` значение `true`.

Пример

Следующий пример привязывает контрол TreeList к самореферентному источнику данных (коллекции записей *Employee*). Класс Employee определяет два свойства (*ID* и *ParentID*), которые определяют *Key field* и *Parent key field* записи соответственно. Для корневых записей *Parent key field* установлено значение **-1**, поэтому свойству `RootValue` контрола присвоено это значение.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
...
<mxtl:TreeListControl Grid.Column="0" Name="treeList2">
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn Name="colName1" FieldName="Name"/>
        <mxtl:TreeListColumn Name="colBirthdate1" FieldName="Birthdate" >
            <mxtl:TreeListColumn.EditorProperties>
                <mxe:TextEditorProperties DisplayFormatString="yyyy-MM-dd"/>
            </mxtl:TreeListColumn.EditorProperties>

        </mxtl:TreeListColumn>
    </mxtl:TreeListControl.Columns>
</mxtl:TreeListControl>
```

```
using CommunityToolkit.Mvvm.ComponentModel;

ObservableCollection<Employee> employees = new ObservableCollection<Employee>();
employees.Add(new Employee()
 { ID = 0, ParentID = -1, Name = "Serge Smolin", Birthdate = new DateTime(1990, 01, 5) });
employees.Add(new Employee()
 { ID = 1, ParentID = 0, Name = "Alex Douglas", Birthdate = new DateTime(1975, 8, 27) });
employees.Add(new Employee()
 { ID = 2, ParentID = 0, Name = "Dennis Parker", Birthdate = new DateTime(1985, 12, 17) });
employees.Add(new Employee()
 { ID = 3, ParentID = 1, Name = "Pavel Morris", Birthdate = new DateTime(1987, 10, 15) });
employees.Add(new Employee()
 { ID = 4, ParentID = 2, Name = "Mary Thompson", Birthdate = new DateTime(1991, 03, 16) });
employees.Add(new Employee()
 { ID = 5, ParentID = 3, Name = "Vera Liskina", Birthdate = new DateTime(1991, 04, 16) });

treeList2.KeyFieldName = "ID";
treeList2.ParentFieldName = "ParentID";
treeList2.RootValue = -1;

treeList2.ItemsSource = employees;
```

```
public partial class Employee : ObservableObject
{
    [ObservableProperty]
    public string name = "";

    [ObservableProperty]
    public DateTime? birthdate = null;

    public int ID { get; set; }

    public int ParentID { get; set; }

}
```

Смотрите также

- Привязка данных
- Привязка к иерархическим данным
- Столбцы
- Как создать контрол TreeView и привязать его к самореферентному источнику данных

Режим без привязки к данным

Контролы TreeList и TreeView поддерживают несвязанный режим, в котором вы можете вручную создать иерархию узлов.

Не инициализируйте свойство `ItemSource` контрола. В противном случае контрол переключается в режим привязки и, таким образом, запрещает вам добавлять узлы вручную.

Используйте свойство `TreeListControlBase.Nodes`, чтобы добавить корневые узлы к контролу. Для каждого узла вы можете использовать свойство `TreeListNode.Nodes` для добавления дочерних узлов.

Узел в контролах TreeList и TreeView инкапсулируется объектом `TreeListNode`. Его свойство `TreeListNode.Content` позволяет вам указать содержимое узла. Вы можете присвоить свойству `TreeListNode.Content` значение бизнес-объекта, публичные свойства которого предоставляют данные для столбцов контрола. Для контрола TreeView вы можете присвоить свойству `TreeListNode.Content` значение String object.

Если вы используете контрол TreeList, убедитесь, что коллекция `TreeListControl.Columns` контрола содержит столбцы, привязанные к определенным именам полей.

Следующий код XAML создает иерархическую структуру узлов в коллекции `TreeListControl.Nodes`. Свойство `Content` каждого узла инициализируется объектом *Person*, определенным в коде позади.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:local="clr-namespace:AvaloniaApplication1"
...
<mxtl:TreeListControl Grid.Column="3" Width="800" Name="treeListUnbound" HorizontalAlignment="Stretch">
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn Name="colFirstName" FieldName="FirstName"
         Header="First Name" Width="*"  AllowSorting="False"/>
        <mxtl:TreeListColumn Name="colLastName" FieldName="LastName" Header="Last Name" Width="*"/>
        <mxtl:TreeListColumn Name="colCity" FieldName="City" Header="City" Width="*" ReadOnly="True"/>
        <mxtl:TreeListColumn Name="colPhone" FieldName="Phone" Header="Phone" Width="*"/>
    </mxtl:TreeListControl.Columns>

    <mxtl:TreeListControl.Nodes>
        <mxtl:TreeListNode>
            <mxtl:TreeListNode.Content>
                <local:Person FirstName="Roman" LastName="Suponin" City="Saint Petersburg"
                 Phone="(7)724-347-47"/>
            </mxtl:TreeListNode.Content>
            <mxtl:TreeListNode.Nodes>
                <mxtl:TreeListNode >
                    <mxtl:TreeListNode.Content>
```

```
                        <local:Person FirstName="Ivan" LastName="Kovalev" City="Moscow"
                         Phone="(7)111-90-73"/>
                    </mxtl:TreeListNode.Content>
                    <mxtl:TreeListNode.Nodes>
                        <mxtl:TreeListNode>
                            <mxtl:TreeListNode.Content>
                                <local:Person FirstName="Artin" LastName="Tusk" City="Aksaray"
                                 Phone="(4)123-14-56"/>
                            </mxtl:TreeListNode.Content>
                        </mxtl:TreeListNode>
                        <mxtl:TreeListNode>
                            <mxtl:TreeListNode.Content>
                                <local:Person FirstName="George" LastName="Botkin" City="Hua Hin"
                                 Phone="(61)457-198-34"/>
                            </mxtl:TreeListNode.Content>
                            <mxtl:TreeListNode.Nodes>
                                <mxtl:TreeListNode>
                                    <mxtl:TreeListNode.Content>
                                        <local:Person FirstName="Lee" LastName="Wan" City="Shanghai"
                                         Phone="(56)335-57-89"/>
                                    </mxtl:TreeListNode.Content>
                                </mxtl:TreeListNode>
                            </mxtl:TreeListNode.Nodes>
                        </mxtl:TreeListNode>
                    </mxtl:TreeListNode.Nodes>
                </mxtl:TreeListNode>
            </mxtl:TreeListNode.Nodes>
        </mxtl:TreeListNode>
    </mxtl:TreeListControl.Nodes>
</mxtl:TreeListControl>
```

```
public partial class Person : ObservableObject
{
    public Person() { }
    public Person(string firstName, string lastName, string city, string phone)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.city = city;
        this.phone = phone;
    }

    [ObservableProperty]
    string firstName;
    [ObservableProperty]
    string lastName;
    [ObservableProperty]
    string city;
    [ObservableProperty]
    string phone;
}
```

Следующий пример создает структуру узла в коде позади. Он добавляет объект *node1* на корневой уровень TreeList. Другие узлы добавляются на вложенных уровнях.

```
TreeListNode node1 = new TreeListNode() {
 Content = new Person("Kim", "Magnus", "Doha", "(5)433-07-12") };
TreeListNode node11 = new TreeListNode() {
 Content = new Person("Patricia", "Rooney", "Cairo", "(5)450-39-49") };
TreeListNode node111 = new TreeListNode() {
 Content = new Person("Victor", "Boev", "Sarajevo", "(98)328-23-54") };
TreeListNode node12 = new TreeListNode() {
 Content = new Person("Vincent", "Novak", "Belgrad", "(476)487-598-465") };
treeListUnbound.Nodes.Add(node1);
node1.Nodes.Add(node11);
node11.Nodes.Add(node111);
node1.Nodes.Add(node12);
```

Особенности TreeView

Вы можете присвоить свойству `Content` узла TreeView значение бизнес-объекта или простой строки.

Следующий код показывает, как заполнить узлы TreeView простыми строками.

```
<mxtl:TreeViewControl Grid.Column="5" Width="400" Name="treeViewUnbound2">
    <mxtl:TreeViewControl.Nodes>
        <mxtl:TreeListNode Content="Russia">
            <mxtl:TreeListNode.Nodes>
                <mxtl:TreeListNode Content="Tula oblast">
                    <mxtl:TreeListNode.Nodes>
                        <mxtl:TreeListNode Content="Tula"/>
                        <mxtl:TreeListNode Content="Aleksin">
                            <mxtl:TreeListNode.Nodes>
                                <mxtl:TreeListNode Content="Pavlovo"/>
                                <mxtl:TreeListNode Content="Kolosovo"/>
                                <mxtl:TreeListNode Content="Shutilovo"/>
                            </mxtl:TreeListNode.Nodes>
                        </mxtl:TreeListNode>
                        <mxtl:TreeListNode Content="Belyov"/>
                        <mxtl:TreeListNode Content="Suvorov"/>
                    </mxtl:TreeListNode.Nodes>
                </mxtl:TreeListNode>
            </mxtl:TreeListNode.Nodes>

        </mxtl:TreeListNode>
    </mxtl:TreeViewControl.Nodes>
</mxtl:TreeViewControl>
```

Если вы назначаете бизнес-объект свойству `Content`, присвойте элементу `TreeViewControl.DataFieldName` имя свойства бизнес-объекта, которое предоставляет отображаемые значения для контрола, как показано в примере ниже.

```
TreeListNode node1 = new TreeListNode() {
    Content = new Person("Kim", "Magnus", "Doha", "(5)433-07-12") };
TreeListNode node11 = new TreeListNode() {
    Content = new Person("Patricia", "Rooney", "Cairo", "(5)450-39-49") };
TreeListNode node111 = new TreeListNode() {
    Content = new Person("Victor", "Boev", "Sarajevo", "(98)328-23-54") };
TreeListNode node12 = new TreeListNode() {
    Content = new Person("Vincent", "Novak", "Belgrad", "(476)487-598-465") };

treeViewUnbound1.DataFieldName = "LastName";
treeViewUnbound1.Nodes.Add(node1);
node1.Nodes.Add(node11);
node11.Nodes.Add(node111);
node1.Nodes.Add(node12);
```

Несвязанные столбцы (TreeList)

Если вам нужно отобразить пользовательскую информацию в столбце TreeList, вы можете создать несвязанный столбец. Этот столбец не привязан к полю в нижележащем источнике данных. Вы должны заполнить этот столбец данными вручную, используя событие `TreeListControl.CustomUnboundColumnData`.

Чтобы создать несвязанный столбец, выполните следующие действия:

- Создайте объект `TreeListColumn`.
- Установите для свойства `UnboundDataType` столбца значение типа данных, которые этот столбец предназначен для отображения.
- Установите для свойства `FieldName` столбца значение уникального имени поля.
- Добавьте столбец в коллекцию `TreeListControl.Columns`, используя метод `Add` или `Insert`. Вы также можете расположить столбец в позиции со свойством `TreeListColumn.VisibleIndex`.

Обратите внимание, что контрол не хранит и не кэширует данные для несвязанных столбцов. Он вызывает событие `CustomUnboundColumnData`, которое вам нужно обработать, чтобы указать данные для несвязанных столбцов.

Событие `CustomUnboundColumnData` срабатывает в двух случаях:

- Когда TreeList необходимо отобразить значение ячейки в несвязанном столбце. В этом случае параметр события `IsGettingData` возвращает значение `true`. Вам нужно присвоить значение параметру события `Value`.

- Когда пользователь изменяет данные в ячейках несвязанных столбцов. В этом случае параметр события `IsGettingData` возвращает значение `false`. Прочитайте параметр события `Value` и вручную кэшируйте его в своем хранилище для дальнейшего использования.

Пример 1

В следующем примере создается несвязанный столбец *Total*, доступный только для чтения, и обрабатывается событие *CustomUnboundColumnData* для вычисления значений столбца на основе значений других полей в соответствии с выражением: Total=UnitPrice*Количество. Обработчик событий проверяет параметр события `IsGettingData` и извлекает значения, если этот параметр равен `true`.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:sys="clr-namespace:System;assembly=System.Runtime"
...
<mxtl:TreeListControl Grid.Column="5" Width="400" Name="treeList1"
                      CustomUnboundColumnData="treeList1_CustomUnboundColumnData" >
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn Name="colUnitPrice" FieldName="UnitPrice" Header="Unit Price"
         Width="*"/>
        <mxtl:TreeListColumn Name="colQuantity" FieldName="Quantity" Header="Quantity"
         Width="*"/>
        <mxtl:TreeListColumn Name="colTotal" FieldName="Total" Header="Total"
         Width="*" ReadOnly="True"
        UnboundDataType="{x:Type sys:Decimal}"/>
    </mxtl:TreeListControl.Columns>
</mxtl:TreeListControl>
```

```
List<PurchaseRecord> list = new List<PurchaseRecord>();
list.Add(new PurchaseRecord() { UnitPrice = 1.3m, Quantity = 2 });
list.Add(new PurchaseRecord() { UnitPrice = 4m, Quantity = 1 });
list.Add(new PurchaseRecord() { UnitPrice = 10m, Quantity = 20 });
list.Add(new PurchaseRecord() { UnitPrice = 7m, Quantity = 12 });

treeList1.ItemsSource = list;

private void treeList1_CustomUnboundColumnData(object? sender, TreeListUnboundColumnDataEventArgs e)
{
    if (e.IsGettingData && e.Column.FieldName == "Total")
    {
        PurchaseRecord rec = e.Node.Content as PurchaseRecord;
        if (rec != null)
        {
            e.Value = rec.Quantity * rec.UnitPrice;
        }
    }
}

public partial class PurchaseRecord : ObservableObject
{
    [ObservableProperty]
    decimal unitPrice;
    [ObservableProperty]
    int quantity;
}
```

Пример 2

В следующем примере показано, как можно кэшировать данные, введенные пользователями, в несвязанных столбцах. В примере создается столбец *Data* и обрабатывается событие

`CustomUnboundColumnData`, чтобы предоставить данные в TreeList и сохранить данные, введенные пользователями.

```
treeList1.Columns.Add(new TreeListColumn() { FieldName = "UserData",
 UnboundDataType = typeof(string), Header = "Data" });
treeList1.CustomUnboundColumnData += treeList1_CustomUnboundColumnData;

Dictionary<int, string> cache = new Dictionary<int, string>();

private void treeList1_CustomUnboundColumnData(object? sender,
TreeListUnboundColumnDataEventArgs e)
{
    if (e.Column.FieldName != "UserData") return;
    if (e.IsGettingData)
    {
        if (cache.ContainsKey(e.Node.Id))
            e.Value = cache[e.Node.Id];
        else
            e.Value = cache[e.Node.Id] = "-empty-";
    }
    else
    {
        cache[e.Node.Id] = e.Value.ToString();
    }
}
```

Узлы

Класс `TreeListNode` инкапсулирует узел в контролах TreeList и TreeView. Узел TreeView отображает одно значение, в то время как узел TreeList может отображать несколько значений (по значению для каждого столбца).

Создание узлов и доступ к ним

В режиме привязки контролы TreeList и TreeView автоматически создают узлы для всех элементов источника данных. Вы можете получить доступ к созданным узлам, используя следующие свойства:

- `TreeListControlBase.Nodes` — Корневые узлы.
- `TreeListNode.Nodes` — Дочерние узлы узла.

Вам нужно создавать узлы вручную, когда контрол находится в Режим без привязки к данным .

Смотрите также: Поиск узлов .

Получение и установка значений узлов

Свойство `TreeListNode.Content` определяет нижележащий объект данных узла. Вы можете ввести значение этого свойства в свой бизнес-объект, а затем прочитать отдельные значения. В Режим без привязки к данным вы можете вручную назначить объект свойству `TreeListNode.Content`. Не назначайте объекты `TreeListNode.Content` в режиме привязки.

Чтобы получить и задать значения отдельных ячеек, вы можете использовать следующие методы:

- `GetCellValue` и `GetCellDisplayText`
- `SetCellValue`

Сфокусированный узел

Используйте свойство `TreeListControlBase.FocusedNode` для доступа к текущему сфокусированному узлу (узлу, который получает события клавиатуры). Чтобы получить объект данных (бизнес) сфокусированного узла, используйте унаследованное свойство `DataControlBase.FocusedItem`.

Событие `TreeListControlBase.FocusedNodeChanged` позволяет вам реагировать на перемещение фокуса между узлами.

Смотрите также: Выбор нескольких узлов (выделение) .

Картинки для узлов

Контролы TreeList и TreeView поддерживают картинки узлов. Эти картинки отображаются перед значениями ячеек в столбце иерархии (первый столбец, по умолчанию).

Установите для свойства `TreeListControlBase.ShowNodeImages` значение `true`, чтобы включить картинки узлов.

Следующие подходы позволяют вам предоставлять картинки узлов:

- Используйте свойство `TreeListControlBase.NodeImageFieldName`, чтобы указать свойство (поле) бизнес-объекта, которое возвращает картинку узла (объект `IImage`).
- Используйте свойство `TreeListControlBase.NodeImageSelector`, чтобы указать селектор (объект `ITreeListNodeImageSelector`), который возвращает картинки узлов для определенных узлов.
- В несвязанном режиме вы можете задать картинку узла, используя свойство `TreeListNode.Image`.

Пример

Следующий код отображает картинки для узлов, которые имеют определенное значение ячейки.

В примере создается объект `NodeImageSelector`, который возвращает картинки в соответствии со свойством *OnVacation* бизнес-объекта.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<Grid.Resources>
    <local:MyNodeImageSelector x:Key="myNodeImageSelector"/>
    ...
</Grid.Resources>

<mxtl:TreeListControl
    Grid.Column="0" Grid.Row="1" Name="treeList2"
    ChildrenSelector="{StaticResource mySelector}"
    ItemsSource="{Binding Employees}"
    NodeImageSelector="{StaticResource myNodeImageSelector}"
    ShowNodeImages="True"
    >
...
</mxtl:TreeListControl>
```

```
using Avalonia.Media.Imaging;
using Avalonia.Platform;

public class MyNodeImageSelector : ITreeListNodeImageSelector
{
```

```
    IImage onVacationImage;
    IImage defaultImage;

    public MyNodeImageSelector()
    {
        onVacationImage = new Bitmap(AssetLoader.Open(new Uri("avares://AvaloniaApp1/Assets/plane.png")));
        defaultImage = null;
    }
    public IImage SelectImage(TreeListNode node)
    {
        Employee row = node.Content as Employee;
        return row.OnVacation? onVacationImage: defaultImage;
    }
}
```

Встроенные флажки

Свойство `ShowNodeCheckBoxes` включает встроенные флажки для узлов. Они позволяют пользователю проверять (выбирать) отдельные узлы.

По умолчанию нажатые флажки имеют два состояния — нажатый и ненажатый. Если вы установите для свойства `AllowIndeterminateCheckState` значение `true`, нажатые флажки поддерживают три состояния — нажато, ненажатое и неопределенное.

Получение и настройка состояния нажатия для узла

Используйте свойство `TreeListNode.IsChecked` узла для считывания и указания состояния нажатия узла. Свойство `TreeListNode.IsChecked` имеет логический тип с возможностью обнуления, поэтому вы можете присвоить свойству `null` для переключения узла в неопределенное состояние.

Получение нажатых (выбранных) узлов

Используйте метод `GetAllCheckedNodes` для извлечения узлов с нажатым состоянием.

Вы также можете создать итератор для извлечения узлов, соответствующих пользовательским критериям.

Синхронизация состояний нажатия с источником данных

Используйте свойство `CheckBoxFieldName`, чтобы синхронизировать состояния нажатия узла с определенным полем источника данных. Это свойство определяет имя логического поля или поля источника логических данных с возможностью обнуления, в котором хранятся состояния нажатия для узлов.

Рекурсивная проверка

`AllowRecursiveNodeChecking` позволяет выполнять рекурсивную проверку узлов. В этом режиме дочерние узлы изменяют свое состояние нажатия, когда изменяется состояние нажатия родительского узла, и наоборот.

Выбор нескольких узлов (выделение)

Контролы TreeList и TreeView поддерживают режим выбора нескольких узлов, который позволяет вам и вашему пользователю выбирать (выделять) несколько узлов одновременно.

Установите для свойства `SelectionMode` значение `Multiple`, чтобы включить режим выбора нескольких узлов.

Выбор узлов с помощью мыши и клавиатуры

Пользователи могут выбирать несколько узлов с помощью мыши и клавиатуры. Им нужно щелкать по узлам, удерживая нажатой клавишу CTRL и/или SHIFT.

Работа с выборкой узлов в коде

Следующий API позволяет вам выбирать / отменять выбор узлов и определять, выбран ли узел:

- `SelectAll`
- `SelectNode`
- `SelectRange`
- `UnselectNode`
- `ClearSelection`
- `IsNodeSelected`

Чтобы получить выбранный узел, используйте следующие элементы:

- `GetSelectedNodes` — Возвращает коллекцию выбранных в данный момент объектов `TreeListNode`.
- `SelectedItems` — определяет набор объектов данных (бизнес-объектов), которые соответствуют выбранным узлам.

Обработайте событие `SelectionChanged`, чтобы отреагировать на изменения в выборе узла.

Вызов любого метода, который изменяет выбранное состояние узла, вызывает обновление контрола TreeList/TreeView и вызывает событие `SelectionChanged`.

Чтобы выполнить пакетные модификации выбранного узла и предотвратить ненужные обновления, вы можете обернуть код, который изменяет выбранные состояния узлов, парой методов `BeginSelection` и `EndSelection`. В этом случае контрол перерисовывает выделение, и событие `SelectionChanged` срабатывает после вызова метода `EndSelection`.

```
treeList1.SelectionMode = Eremex.AvaloniaUI.Controls.DataControl.RowSelectionMode.Multiple;
// Start a batch update of the node selection.
treeList1.BeginSelection();
treeList1.ClearSelection();
treeList1.SelectNode(node1);
treeList1.SelectNode(node2);
//...
// Finish the batch update.
treeList1.EndSelection();
```

Сфокусированный узел и выбранные узлы

Сфокусированный узел - это узел, который получает пользовательский ввод. В режиме одиночного выбора сфокусированный узел автоматически получает выбранное состояние. Вы можете использовать свойство `FocusedNode` и метод `GetSelectedNodes` для извлечения сфокусированного узла.

Выбран узел или нет, можно определить по выделению узла. Выделяются только выбранные узлы.

Сфокусированное и выбранное состояния различаются в режиме выбора нескольких узлов. В этом режиме щелчок по узлу фокусирует и выбирает этот узел одновременно. Пользователь, однако, может переключать выбранное состояние сфокусированного узла, используя следующие действия:

- Нажмите CTRL+ПРОБЕЛ.
- Щелкните по выделенному узлу, удерживая нажатой клавишу CTRL.

Когда вы выбираете узел в коде, этот узел не получает сфокусированное состояние в режиме множественного выбора, и наоборот.

Поиск узлов

Метод `TreeListControlBase.FindNode` позволяет вам находить узлы, соответствующие пользовательским критериям.

Следующий код позволяет выбрать несколько узлов и находит и выбирает два узла, которые содержат определенные значения в поле *Name*.

```
treeList1.SelectionMode = Eremex.AvaloniaUI.Controls.DataControl.RowSelectionMode.Multiple;
treeList1.ExpandAllNodes();
TreeListNode node1 = treeList1.FindNode(node => (node.Content as Employee).Name.Contains("Sam"));
TreeListNode node2 = treeList1.FindNode(node => (node.Content as Employee).Name.Contains("Dan"));
treeList1.ClearSelection();
treeList1.SelectNode(node1);
treeList1.SelectNode(node2);
```

Итерация по узлам

Вы можете создать итератор (объект `TreeListNodeIterator`) для рекурсивного перебора узлов и выполнения над ними операции.

Используйте один из следующих конструкторов для создания объекта `TreeListNodeIterator`:

```
// Recursively iterates through child nodes of the specified node, and their children.
public TreeListNodeIterator(TreeListNode? node, bool onlyExpanded = false)
```

```
// Recursively iterates through the specified nodes, and their children.
public TreeListNodeIterator(TreeListNodeCollection? nodes, bool onlyExpanded = false)
```

Параметр *onlyExpanded* указывает, следует ли выполнять итерацию по развернутым узлам или по развернутым и свернутым узлам.

```
foreach (var node in new TreeListNodeIterator(treeList1.Nodes))
{
    if (node != null)
    {
        //do smth
    }
}
```

Столбцы

Контрол TreeList поддерживает несколько столбцов данных, в то время как TreeView отображает значения в одном столбце.

Столбец в контроле TreeView

Вам не нужно создавать столбец в TreeView контроле. Вам нужно только установить свойство `TreeViewControl.DataFieldName`, чтобы указать имя поля источника данных или публичного свойства, значения которого отображаются контролом.

Столбцы в контроле TreeList

Столбцы TreeList инкапсулируются объектами `TreeListColumn`, которые являются потомками класса `ColumnBase`.

Контрол не создает автоматически столбцы после привязки контрола к источнику данных. Три подхода позволяют создавать столбцы:

- Создайте все столбцы TreeList вручную в коллекции `TreeList.Columns` (в XAML или code-behind). При таком подходе у вас есть доступ к созданным объектам столбца по имени в коде.
- Включите опцию `TreeListControl.AutoGenerateColumns`, чтобы автоматически генерировать отсутствующие столбцы после привязки контрола к источнику данных. Вы можете применить определенные атрибуты (из пространств имен `System.ComponentModel` и `System.ComponentModel.DataAnnotations`) к свойствам бизнес-объекта, чтобы управлять автоматической генерацией столбцов и настраивать свойства автоматически генерируемых столбцов (например, отображаемое имя столбца и порядок).
- Комбинация двух описанных выше подходов: создайте необходимые столбцы вручную в коллекции `TreeList.Columns`, а затем включите опцию `TreeListControl.AutoGenerateColumns`, чтобы делегировать генерацию других столбцов в TreeList.

В следующем примере создаются два столбца TreeList и настраивается формат отображения значений второго столбца:

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
```

```
<mxtl:TreeListControl Name="treeList1" >
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn Name="colName" FieldName="Name"/>
        <mxtl:TreeListColumn Name="colBirthdate" FieldName="Birthdate" >
            <mxtl:TreeListColumn.EditorProperties>
                <mxe:TextEditorProperties DisplayFormatString="yyyy-MM-dd"/>
            </mxtl:TreeListColumn.EditorProperties>
        </mxtl:TreeListColumn>
    </mxtl:TreeListControl.Columns>
</mxtl:TreeListControl>
```

Смотрите Автоматическая генерация столбцов для получения информации об автоматически сгенерированных столбцах.

Привязка столбцов к данным

Свойство `TreeListColumn.FieldName` позволяет привязать столбец к полю в таблице нижележащих данных или к публичному свойству бизнес-объекта. Как только столбец привязан, он извлекает значения из источника данных.

TreeList также позволяет вам создавать несвязанные столбцы, значения которых следует вводить вручную, с помощью события `TreeListControl.CustomUnboundColumnData`. Смотрите Несвязанные столбцы для получения дополнительной информации.

Не рекомендуется привязывать несколько столбцов TreeList к одному и тому же полю данных/свойству.

В следующем примере создаются столбцы TreeList и привязываются к ним данные.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Grid.Column="3" Width="200" Name="treeListUnbound" HorizontalAlignment="Stretch">
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn Name="colFirstName" FieldName="FirstName"
         Header="First Name" Width="*"  AllowSorting="False"/>
        <mxtl:TreeListColumn Name="colLastName" FieldName="LastName" Header="Last Name" Width="*"/>
        <mxtl:TreeListColumn Name="colCity" FieldName="City" Header="City" Width="*" ReadOnly="True"/>
        <mxtl:TreeListColumn Name="colPhone" FieldName="Phone" Header="Phone" Width="*"/>
    </mxtl:TreeListControl.Columns>
</mxtl:TreeListControl>
```

```
using Eremex.AvaloniaUI.Controls.TreeList;

TreeListColumn colFirstName = new TreeListColumn() { FieldName = "FirstName",
Header = "First Name", AllowSorting = false, Width= new GridLength(1, GridUnitType.Star) };
treeList1.Columns.Add(colFirstName);
```

Автоматическая генерация столбцов

Установите для свойства `AutoGenerateColumns` значение `true` (значение по умолчанию - `false`), чтобы включить автоматическую генерацию столбцов для свойств в источнике данных. Когда для `AutoGenerateColumns` установлено значение `true`, контрол TreeList извлекает публичные свойства из источника данных, генерирует столбцы и привязывает их к свойствам. Если коллекция `Columns` контрола уже содержит столбец, привязанный к определенному свойству/полю, автоматически не генерируется дополнительный столбец, привязанный к тому же свойству/полю.

События `AutoGeneratingColumn` и `AutoGeneratedColumns` позволяют настраивать автоматически сгенерированные столбцы. Событие `AutoGeneratingColumn` срабатывает, когда автоматически сгенерированный столбец собирается быть добавленным в коллекцию `Columns`. Событие позволяет вам предотвратить добавление столбца в коллекцию.

Событие `AutoGeneratedColumns` срабатывает после того, как все столбцы были автоматически сгенерированы.

Когда вы назначаете контролу другой источник данных, TreeList сначала удаляет столбцы, которые ранее были сгенерированы автоматически, а затем автоматически генерирует столбцы для нового источника данных.

Использование атрибутов для настройки свойств автоматически генерируемых столбцов

Вы можете применить определенные атрибуты (из пространств имен `System.ComponentModel` и `System.ComponentModel.DataAnnotations`) к свойствам бизнес-объекта (записи источника данных), чтобы настроить статус видимости, вид и свойства поведения для соответствующих автоматически сгенерированных столбцов TreeList. Поддерживаются следующие атрибуты:

Атрибут `Browsable`

Атрибут `System.ComponentModel.BrowsableAttribute` контролирует автоматическую генерацию столбца. Примените атрибут **Browsable(false)** к определенным свойствам, чтобы предотвратить автоматическое создание соответствующих столбцов.

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel;

public partial class MyBusinessObject : ObservableObject
{
    [ObservableProperty]
    [property: Browsable(false)]
    public int serviceId = "";
}
```

Атрибут `System.ComponentModel.BrowsableAttribute` эквивалентен использованию атрибута `System.ComponentModel.DataAnnotations.DisplayAttribute` с параметром `AutoGenerateField`.

Атрибут `Display`

`System.ComponentModel.DataAnnotations.DisplayAttribute` - это атрибут общего назначения, который контролирует автоматическую генерацию столбцов и свойства отображения автоматически сгенерированных столбцов. Атрибут имеет следующие параметры, поддерживаемые контролом TreeList:

- `AutoGenerateField` — Указывает, следует ли автоматически генерировать соответствующий столбец.
- `Order` — Указывает видимую позицию автоматически сгенерированного столбца (`ColumnBase.VisibleIndex`).
- `Name` — Указывает заголовок автоматически сгенерированного столбца (`ColumnBase.Header`).
- `ShortName` — эквивалент параметра `Name` .

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel.DataAnnotations;

public partial class MyBusinessObject : ObservableObject
{
    [ObservableProperty]
    [Display(Name = "Birth date", Order=2)]
    public DateTime? birthdate = null;
}
```

Атрибут `DisplayName`

Атрибут `System.ComponentModel.DisplayNameAttribute` позволяет вам инициализировать заголовок автоматически сгенерированного столбца (`ColumnBase.Header`).

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel;

public partial class MyBusinessObject : ObservableObject
{
    [ObservableProperty]
    [property: DisplayName("Birth date")]
    public DateTime? birthdate = null;
}
```

Атрибут `System.ComponentModel.DisplayNameAttribute` эквивалентен использованию атрибута `System.ComponentModel.DataAnnotations.DisplayAttribute` с параметром `Name` или `ShortName` .

Атрибут `Editable`

Атрибут `System.ComponentModel.EditableAttribute` , примененный к свойству, создает недоступный для редактирования столбец. Пользователи не могут открывать встроенные редакторы и, таким образом, выделять и копировать текст.

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel;

public partial class MyBusinessObject : ObservableObject
{
    [ObservableProperty]
    [property: Editable(false)]
    public int parentId = -1;
}
```

Атрибут `Readonly`

Атрибут `System.ComponentModel.ReadonlyAttribute`, применяемый к свойству, создает столбец, доступный только для чтения. Пользователи могут выделять и копировать текст в столбцах, доступных только для чтения, но не редактировать значения.

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel;

public partial class MyBusinessObject : ObservableObject
{
    [ObservableProperty]
    [property: ReadOnly(true)]
    public int id = -1;
}
```

Перемещение столбцов

Используйте свойство `TreeListColumn.VisibleIndex`, чтобы указать визуальную позицию столбца. Чтобы скрыть столбец, установите для его свойства `TreeListColumn.VisibleIndex` значение **-1** или установите для свойства `IsVisible` значение `false`.

Пользователь может переставлять столбцы по умолчанию. Следующие свойства позволяют запретить перемещение столбца:

- `TreeListControl.AllowColumnMoving` — Указывает, может ли пользователь перемещать любой столбец.
- `TreeListColumn.AllowMoving` — Указывает, может ли пользователь перемещать определенный столбец.

Изменение размера столбцов

Чтобы контролировать ширину столбца в TreeList, вы можете использовать следующие свойства:

- `TreeListColumn.Width` — Ширина столбца, указанная как значение `GridLength`.

- `TreeListColumn.MinWidth` — Минимальная ширина столбца.
- `TreeListColumn.MaxWidth` — Максимальная ширина столбца.

Свойство `Width` относится к типу `GridLength`. Это позволяет вам установить ширину столбца равной:

- Фиксированная ширина (количество пикселей).
- Взвешенная доля доступного пространства (обозначение *star*).
- Значение "Автоматически" — активирует автоматическое вычисление ширины столбца, подогнанное под заголовок столбца и видимые значения. Когда пользователь прокручивает контрол по вертикали, контрол может увеличить ширину столбца, подгоняя новые значения ячеек, появившиеся во время операции прокрутки.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Name="treeList1">
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn Name="colFirstName" FieldName="FirstName" Header="First Name" Width="*"/>
        <mxtl:TreeListColumn Name="colLastName" FieldName="LastName" Header="Last Name" Width="2*"/>
        <mxtl:TreeListColumn Name="colPhone" FieldName="Phone" Header="Phone" Width="*"/>
    </mxtl:TreeListControl.Columns>
</mxtl:TreeListControl>
```

Следующие свойства контролируют операции изменения размера столбцов, выполняемые пользователями.

- `TreeListControl.AllowColumnResizing` — Указывает, может ли пользователь изменять размер любого столбца.
- `TreeListColumn.AllowResizing` — Указывает, может ли пользователь изменять размер определенного столбца.

Заголовки столбцов

Заголовки столбцов TreeList отображаются на панели заголовков. Вы можете скрыть эту панель с помощью свойства `TreeListControl.ShowColumnHeaders`.

Высота панели автоматически регулируется в подогнанном под содержимое заголовков столбцов. Используйте свойство `HeaderPanelMinHeight`, чтобы ограничить минимальную высоту панели.

Заголовок столбца изначально отображает заголовок (текстовую метку), который является текстовым представлением свойства `ColumnBase.Header`. Если свойство `ColumnBase.Header` не задано, заголовок столбца генерируется из имени поля столбца (`ColumnBase.FieldName`).

Используйте свойство `ColumnBase.HeaderContentTemplate`, чтобы указать шаблон, используемый для отображения заголовка столбца. Шаблон позволяет отображать картинки и пользовательские контролы,

а также отображать текст произвольным способом.

Следующий код отображает картинку перед заголовком столбца. Выражение `<TextBlock Text="{Binding}">` отображает содержимое свойства `Header` столбца:

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Name="treeList" HeaderPanelMinHeight="50">
    <mxtl:TreeListColumn FieldName="Number" Header="Position" HeaderVerticalContentAlignment="Bottom">
        <mxtl:TreeListColumn.HeaderContentTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal">
                    <Image Source="/info24x24.png" Width="24" Height="24" Margin="0,0,5,0"></Image>
                    <TextBlock Text="{Binding}" VerticalAlignment="Center"/>
                </StackPanel>
            </DataTemplate>
        </mxtl:TreeListColumn.HeaderContentTemplate>
    </mxtl:TreeListColumn>
</mxtl:TreeListControl>
```

Используйте свойства `ColumnBase.HeaderHorizontalContentAlignment` и `ColumnBase.HeaderVerticalContentAlignment`, чтобы выровнять содержимое заголовка столбца по горизонтали и вертикали.

Сортировка столбцов

Пользователь может щелкнуть заголовок столбца или использовать контекстное меню заголовка столбца, чтобы отсортировать TreeList по столбцу. Смотрите Сортировка данных для получения дополнительной информации.

Значения в столбцах

Чтобы узнать, как извлекать значения ячеек для определенных узлов, смотрите Узлы .

Смотрите также

- Несвязанные столбцы

Редактирование данных

Встроенные редакторы Eremex, используемые по умолчанию

Если вы явно не указываете редакторы ячеек, контролы TreeList и TreeView используют встроенные редакторы Eremex для отображения и редактирования значений ячеек распространенных типов данных:

- Логические значения — `CheckEditor`
- Двойные значения — `SpinEditor`
- Значения перечисления — `ComboBoxEditor`
- Свойства с атрибутом `TypeConverter`, метод `TypeConverter.GetStandardValuesSupported` которого возвращает `true` — `ComboBoxEditor`
- Другие значения — `TextEditor`

Вы можете динамически получать доступ к экземплярам встроенных редакторов Eremex и изменять их, когда эти редакторы активированы. Смотрите раздел Доступ к активному встроенному редактору Eremex для получения более подробной информации.

Назначение встроенных редакторов Eremex

Контролы TreeList и TreeView позволяют вам явно назначать встроенные редакторы Eremex ячейкам (столбцам), чтобы переопределить дефолтное назначение редактора, или настраивать редакторы ячеек в XAML или code-behind. Используйте для этой цели следующие свойства:

- Контрол TreeView : `TreeViewControl.EditorProperties`

  Контрол TreeView отображает один столбец данных. Таким образом, свойство `TreeViewControl.EditorProperties` определяет встроенный редактор, используемый для редактирования ячеек этого столбца.

- Контрол TreeList: `TreeListColumn.EditorProperties`

  Каждый столбец в контроле TreeList может иметь собственный встроенный редактор. Создайте столбец TreeList (объект `TreeListColumn`) в коллекции `TreeListControl.Columns` и установите редактор столбца, используя свойство `TreeListColumn.EditorProperties`.

Вы можете присвоить свойству `EditorProperties` следующие объекты, которые определяют тип встроенного редактора (все эти объекты являются потомками `BaseEditorProperties`):

- `ButtonEditorProperties` — Содержит свойства, специфичные для контрола `ButtonEditor`.

- `CheckEditorProperties` — Содержит свойства, специфичные для контрола `CheckEditor`.
- `ColorEditorProperties` — Содержит свойства, специфичные для контрола `ColorEditor`.
- `ComboBoxEditorProperties` — Содержит свойства, специфичные для контрола `ComboBoxEditor`.
- `HyperlinkEditorProperties` — Содержит свойства, специфичные для контрола `HyperlinkEditor`.
- `PopupColorEditorProperties` — Содержит свойства, специфичные для контрола `PopupColorEditor`.
- `PopupEditorProperties` — Содержит свойства, специфичные для контрола `PopupEditor`.
- `SegmentedEditorProperties` — Содержит свойства, специфичные для контрола `SegmentedEditor`.
- `SpinEditorProperties` — Содержит свойства, специфичные для контрола `SpinEditor`.
- `TextEditorProperties` — Содержит свойства, специфичные для контрола `TextEditor`.

Предположим, что вы присвоили свойству `EditorProperties` значение объекта `SpinEditorProperties`. В режиме отображения (редактирование ячеек не активно) контрол TreeList/TreeView использует свойства объекта `SpinEditorProperties` для эмуляции встроенных редакторов SpinEditor в ячейках целевого столбца. Никакой реальный SpinEditor не создается до тех пор, пока в ячейке не начнется операция редактирования. Когда пользователь запускает редактирование ячейки, контрол TreeList/TreeView создает реальный редактор SpinEditor, встроенный в сфокусированную ячейку. После завершения операции редактирования контрол уничтожает реальный SpinEditor и начинает эмулировать SpinEditor в этой ячейке. Смотрите Доступ к активному встроенному редактору Eremex , чтобы узнать, как получить доступ к реальному редактору ячеек.

Пример - Как использовать ButtonEditor в качестве встроенного редактора в столбце TreeList

Следующий код присваивает встроенному редактору `ButtonEditor` столбец TreeList.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl.Columns>
    <mxtl:TreeListColumn Header="Name" FieldName="Name">
        <mxtl:TreeListColumn.EditorProperties>
            <mxe:ButtonEditorProperties>
                <mxe:ButtonEditorProperties.Buttons>
                    <mxe:ButtonSettings Content="Clear"
                     Command="{Binding $parent[mxtl:CellControl].DataControl.DataContext.ClearValueCommand}"/>
                </mxe:ButtonEditorProperties.Buttons>
            </mxe:ButtonEditorProperties>
        </mxtl:TreeListColumn.EditorProperties>
    </mxtl:TreeListColumn>
</mxtl:TreeListControl.Columns>
```

Пример - Как использовать ComboBoxEditor в качестве встроенного редактора в TreeView

Следующий код назначает встроенному редактору `ComboBoxEditor` TreeView.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeViewControl.EditorProperties>
    <mxe:ComboBoxEditorProperties ItemsSource="{Binding Families}"/>
</mxtl:TreeViewControl.EditorProperties>
```

Назначение встраиваемых редакторов Eremex с помощью шаблонов

Контролы TreeList и TreeView позволяют использовать следующие свойства шаблона для назначения столбцам встроенных редакторов Eremex.:

- `TreeListColumn.CellTemplate`
- `TreeViewControl.CellTemplate`

Убедитесь, что для редактора Eremex, определенного в `DataTemplate`, свойству `x:Name` присвоено значение **"PART_Editor"**. В этом случае контрол TreeList/TreeView автоматически привязывает свойство редактора `EditorValue` к полю строки. Кроме того, контрол начинает поддерживать свойства внешнего вида встроенного редактора (видимость границ и цвет отображаемого текста в активном и неактивном состояниях).

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
...
<mxtl:TreeListColumn Header="Phone" FieldName="Phone">
    <mxtl:TreeListColumn.CellTemplate>
        <DataTemplate>
            <mxe:ButtonEditor x:Name="PART_Editor">
                <mxe:ButtonEditor.Buttons>
                    <mxe:ButtonSettings Content="..."/>
                </mxe:ButtonEditor.Buttons>
            </mxe:ButtonEditor>
        </DataTemplate>
    </mxtl:TreeListColumn.CellTemplate>
</mxtl:TreeListColumn>
```

Пользовательские редакторы

Вы можете указать пользовательские редакторы для ячеек в контролах TreeList и TreeView, используя следующие подходы:

- Назначьте редактор непосредственно определенному столбцу.
- Динамически назначайте редакторы столбцам на основе типа данных нижележащего объекта столбца. Этот метод применим к контролу TreeList.

Смотрите раздел Пользовательские редакторы для получения дополнительной информации.

Получение и установка значений ячеек

Контролы TreeList и TreeView предоставляют следующий API для получения и установки значений ячеек:

- `TreeListControl.GetCellDisplayText`
- `TreeListControl.GetCellValue`
- `TreeListControl.SetCellValue`

- `TreeViewControl.GetCellDisplayText`
- `TreeViewControl.GetCellValue`
- `TreeViewControl.SetCellValue`

Доступ к активному встроенному редактору Eremex

Когда встроенные редакторы Eremex назначаются столбцам TreeList/TreeView (неявно или явно с использованием свойства и шаблонов `EditorSettings`), контролы эмулируют указанные встроенные редакторы в ячейках столбцов в режиме отображения (когда редактирование ячеек не активно). На данный момент реального встроенного редактора не существует. Эмуляция встроенных редакторов в ячейках в режиме отображения повышает производительность приложения.

Когда пользователь начинает редактировать ячейку, контрол создает реальный редактор встроенного текста. Вы можете использовать свойство `ActiveEditor` контрола TreeList/TreeView для доступа к реальному экземпляру редактора Eremex в данный момент. Когда ячейка теряет фокус, реальный редактор уничтожается, а свойство `ActiveEditor` возвращает значение `null`.

Обработайте событие `ShowingEditor`, чтобы получать уведомления при запуске операций редактирования в ячейках. Вы можете безопасно получить доступ к свойству `ActiveEditor` в вашем обработчике событий `ShowingEditor`.

Как отключить активацию редактора

Чтобы предотвратить активацию редактора ячеек в определенных случаях, вы можете обработать событие `ShowingEditor` и установить для параметра обработчика событий `Cancel` значение `true`.

Пользовательские редакторы в ячейках TreeList и TreeView

Контролы TreeList и TreeView по умолчанию используют встроенные редакторы Eremex. Смотрите Редактирование данных для получения информации о том, как назначить встроенные редакторы по умолчанию и получить к ним доступ.

В этом разделе показано, как назначить пользовательские редакторы контролам. Доступны следующие два подхода:

- Назначить пользовательский редактор столбцу TreeList и TreeView напрямую
- Динамически назначать редакторов на основе типа данных столбца TreeList

Назначение пользовательского редактора непосредственно столбцу в контроле TreeList и Tree View

Столбцу TreeList или TreeView можно назначить встроенный редактор, создав `DataTemplate`. Используйте свойства `TreeListColumn.CellTemplate` и `TreeViewControl.CellTemplate` для этой цели следующим образом.

- Создайте объект `DataTemplate` с редактором, определенным внутри шаблона.
- Присвойте `DataTemplate` свойству `CellTemplate`.
- При необходимости явно привяжите редактор к значению столбца.

В следующем примере показан код XAML, который устанавливает свойство `CellTemplate` столбца TreeList для объекта `TextBox`:

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListColumn Header="Phone" FieldName="Phone">
    <mxtl:TreeListColumn.CellTemplate>
        <DataTemplate>
            <TextBox Text="{Binding Value}"/>
        </DataTemplate>
```

```
    </mxtl:TreeListColumn.CellTemplate>
</mxtl:TreeListColumn>
```

Неявная привязка данных для редакторов Eremex

Если вы используете редактор Eremex внутри `DataTemplate`, вы можете опустить явную привязку данных к значению столбца для редактора. Убедитесь, что в редакторе Eremex для свойства `x:Name` установлено значение **"PART_Editor"**. В этом случае контрол TreeList/TreeView автоматически привязывает свойство редактора `EditorValue` к значению ячейки столбца. Кроме того, контрол начинает поддерживать свойства внешнего вида встроенного редактора (видимость границ и цвет отображаемого текста в активном и неактивном состояниях).

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
...
<mxtl:TreeListColumn Header="Phone" FieldName="Phone">
    <mxtl:TreeListColumn.CellTemplate>
        <DataTemplate>
            <mxe:ButtonEditor x:Name="PART_Editor">
                <mxe:ButtonEditor.Buttons>
                    <mxe:ButtonSettings Content="..."/>
                </mxe:ButtonEditor.Buttons>
            </mxe:ButtonEditor>
        </DataTemplate>
    </mxtl:TreeListColumn.CellTemplate>
</mxtl:TreeListColumn>
```

Явная привязка данных

Явная привязка данных требуется в следующих случаях:

- Вы используете редактор, который не является редактором Eremex.

  > **СОВЕТ**
  >
  > Редакторы Eremex - это контролы, производные от класса `Eremex.AvaloniaUI.Controls.Editors.BaseEditor`.

- Вам необходимо указать пользовательский конвертер значений в выражении привязки данных.

```
<DataTemplate DataType="pepa:Color">
    <mxe:PopupColorEditor x:Name="PART_Editor"
     EditorValue="{Binding Path=Value, Converter={ecadpg:EcadColorToAvaloniaColorConverter}}"/>
</DataTemplate>
```

Динамическое назначение редакторов на основе типа данных столбца TreeList

Контрол TreeList позволяет назначать редакторы, встроенные в `DataTemplate`s, ячейкам столбцов на основе типа данных привязанного поля столбца. Используйте свойство `TreeListControl.CellTemplate` для этой цели.

Пример - Как связать редактор с определенным типом данных

Следующий пример привязывает `TextEditor`, который рисует значения зеленым цветом, к столбцам, привязанным к целочисленным полям:

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl.CellTemplate>
    <DataTemplate DataType="sys:Int32">
        <mxe:TextEditor x:Name="PART_Editor1" EditorValue="{Binding Value}" Foreground="Green"/>
    </DataTemplate>
</mxtl:TreeListControl.CellTemplate>
```

Пример - Как связать редакторы с несколькими типами данных

Предположим, что у вас есть список объектов `DataTemplate`, связанных с различными типами данных, и вы хотите создать встроенные редакторы столбцов на основе этого списка.

Для выполнения этой задачи выполните следующие действия:

- Создайте пользовательский класс (*CellTemplateLocator*) в коде, который возвращает `DataTemplate` для определенного типа данных и создает контрол, связанный с этим типом данных.

```
using Avalonia.Collections;
using Avalonia.Controls.Templates;

namespace AvaloniaApplication1.Views;

public class CellTemplateLocator : AvaloniaList<IDataTemplate>, IDataTemplate
{
    public Control Build(object? param)
    {
        var cellData = param as Eremex.AvaloniaUI.Controls.DataControl.Visuals.CellData;
        if (cellData == null) return null;
        return this.First(x => x.Match(cellData.Value)).Build(cellData.Value);
    }

    public bool Match(object? data)
```

```
    {
        var cellData = data as Eremex.AvaloniaUI.Controls.DataControl.Visuals.CellData;
        if (cellData == null) return false;
        return this.Any(x => x.Match(cellData.Value));
    }
}
```

- Инициализируйте свойство `TreeListControl.CellTemplate` с помощью объекта *CellTemplateLocator*.
- Заполните объект *CellTemplateLocator* объектами `DataTemplate`, связанными с вашими типами данных.

В следующем примере определены два объекта `DataTemplate` с редакторами, связанными с типами данных *String* и *Integer* соответственно.

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:AvaloniaApplication1.Views"

<mxtl:TreeListControl.CellTemplate>
    <local:CellTemplateLocator>
        <DataTemplate DataType="sys:String">
            <mxe:TextEditor x:Name="PART_Editor" EditorValue="{Binding Value}" Foreground="Red"/>
        </DataTemplate>
        <DataTemplate DataType="sys:Int32">
            <mxe:TextEditor x:Name="PART_Editor1" EditorValue="{Binding Value}" Foreground="Green"/>
        </DataTemplate>
    </local:CellTemplateLocator>
</mxtl:TreeListControl.CellTemplate>
```

Сортировка данных

Функция сортировки данных позволяет сортировать одноуровневые узлы в TreeList и контролах TreeView в порядке возрастания или убывания.

TreeList поддерживает сортировку данных по одному или нескольким столбцам. Контрол также позволяет вашим пользователям сортировать данные щелчком мыши по заголовку столбца и с помощью контекстного меню заголовка столбца.

Действия конечного пользователя (TreeList)

В контроле TreeList пользователь может щелкнуть левой кнопкой мыши заголовок столбца, чтобы отсортировать этот столбец. Последующий щелчок по заголовку того же столбца изменяет направление сортировки на противоположное. Чтобы отменить сортировку, удерживайте нажатой клавишу CTRL и щелкните заголовок столбца.

Чтобы отсортировать данные по нескольким столбцам, пользователю необходимо удерживать нажатой клавишу SHIFT, а затем щелкнуть заголовки столбцов в требуемом порядке. Контрол будет сортировать данные по первому щелкнутому столбцу, затем по второму столбцу и так далее.

Пользователь также может изменить свойства сортировки столбца из контекстного меню заголовка столбца. Щелкните правой кнопкой мыши заголовок столбца, чтобы вызвать это меню.

Используйте следующие свойства, чтобы запретить пользователю сортировать данные:

- Установите для `TreeListControl.AllowSorting` значение `false`, чтобы запретить пользователю выполнять сортировку по любому столбцу.
- Установите для `ColumnBase.AllowSorting` значение `false`, чтобы запретить пользователю выполнять сортировку по определенному столбцу.

Эти свойства не мешают вам сортировать данные в коде.

Контрол TreeView не имеет функций для сортировки данных из пользовательского интерфейса. Вы можете сортировать данные в коде в этом контроле.

Сортировка в коде (TreeList)

Вы можете отсортировать данные TreeList по одному или нескольким столбцам. Когда вы сортируете данные по нескольким столбцам, контрол TreeList переупорядочивает узлы в соответствии со значениями первого столбца сортировки. Затем он переупорядочивает отсортированные узлы по второму столбцу сортировки, сохраняя порядок значений в первом столбце, и так далее.

Используйте следующие свойства для сортировки данных по столбцу:

- Установите для свойства `ColumnBase.SortDirection` значение `Ascending` или `Descending`. Это свойство определяет порядок сортировки данных в столбце.
- Установите для свойства `ColumnBase.SortIndex` неотрицательное значение, чтобы отсортировать данные по столбцу в порядке возрастания. Свойство `ColumnBase.SortIndex` определяет позицию столбца среди отсортированных столбцов.

Выполните одно из следующих действий, чтобы очистить сортировку:

- Установите для свойства `SortDirection` столбца значение `null`, чтобы очистить сортировку для этого столбца.
- Вызовите унаследованный метод `DataControlBase.ClearSorting` контрола, чтобы удалить сортировку, примененную ко всем столбцам.

Следующий код очищает сортировку, а затем сортирует данные по двум столбцам:

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel;

treeList.ClearSorting();
treeListColumn1.SortDirection = ListSortDirection.Ascending;
treeListColumn3.SortDirection = ListSortDirection.Descending;
```

Вы можете обернуть свой код методами `BeginDataUpdate` и `EndDataUpdate`, чтобы предотвратить ненужные обновления при изменении множественной сортировки контрола и других свойств настроек.

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel;

treeList.BeginDataUpdate();
treeList.ClearSorting();
treeListColumn1.SortIndex = 0;
treeListColumn3.SortIndex = 1;
treeListColumn2.SortDirection = ListSortDirection.Descending;
treeList.EndDataUpdate();
```

Сортировка в коде (TreeView)

Чтобы отсортировать данные в TreeView, используйте свойство `SortDirection` вашего объекта `TreeViewControl`. Установите для этого свойства значение `Ascending` или `Descending`, чтобы отсортировать данные в требуемом порядке.

```
treeViewControl1.SortDirection = ListSortDirection.Ascending;
```

Выполните одно из следующих действий, чтобы очистить сортировку:

- Установите для свойства `SortDirection` значение `null`.
- Вызовите унаследованный метод `DataControlBase.ClearSorting` контрола.

Режим сортировки (TreeList и TreeView)

Дефолтное поведение контролов заключается в сравнении значений редактирования ячеек при сортировке данных. Встроенные редакторы, встроенные в ячейки, могут форматировать редактируемые значения для представления их произвольным способом на экране. В некоторых случаях вы можете захотеть выполнить сортировку по тексту отображения ячейки вместо значений редактирования ячейки.

Свойство `ColumnBase.SortMode` (в TreeList) и свойство `TreeViewControl.SortMode` (в TreeView) определяют алгоритм сравнения, используемый для сортировки значений ячеек. Доступны следующие опции:

- `SortMode.Value` — Алгоритм сортировки по умолчанию, который сравнивает значения редактирования ячеек.
- `SortMode.DisplayText` — Алгоритм сортировки по умолчанию, который сравнивает текст, отображаемый в ячейках.
- `SortMode.Custom` — Установите для свойства `SortMode` значение `Custom`, а затем обработайте события `TreeListControl.CustomColumnSort` и `TreeViewControl.CustomSort` для реализации пользовательской процедуры сортировки.

Когда вы обрабатываете событие `CustomColumnSort` / `CustomSort`, вы должны сравнить два узла, указанных в аргументах события. Присвойте результат сравнения параметру события `Result` следующим образом:

- Установите для `Result` значение **-1**, если первый узел должен отображаться над вторым узлом при сортировке данных в порядке возрастания.
- Установите для `Result` значение `1`, если первый узел должен отображаться ниже второго узла при сортировке данных в порядке возрастания.

- Установите `Result` на `0`, если два узла равны.

Следующий пример обрабатывает событие `TreeListControl.CustomColumnSort` для сортировки данных в столбце "*FileName*" произвольным способом. В столбце "*FileName*" хранятся имена файлов в стандартном формате "*filename.ext*". Пользовательская процедура сортировки сортирует имена файлов по их расширениям.

```
<mxtl:TreeListControl Name="treeList" CustomColumnSort="TreeList_CustomColumnSort">
    <mxtl:TreeListControl.Columns>
        <mxtl:TreeListColumn  Width="*" FieldName="FileName"  SortMode="Custom"/>
    </mxtl:TreeListControl.Columns>
</mxtl:TreeListControl>

private void TreeList_CustomColumnSort(object? sender, TreeListCustomColumnSortEventArgs e)
{
    if (e.Column.FieldName == "FileName")
    {
        string fileName1 = Convert.ToString(e.Value1);
        string fileName2 = Convert.ToString(e.Value2);

        string newfileName1 = extractFileExtension(fileName1) + "." + fileName1;
        string newfileName2 = extractFileExtension(fileName2) + "." + fileName2;
        e.Result = String.Compare(newfileName1, newfileName2);
    }
}

string extractFileExtension(string fileName)
{
    string res = "";
    int dotIndex = fileName.LastIndexOf('.');
    if (dotIndex > 0)
        res = fileName.Substring(dotIndex + 1);
    return res;
}
```

Дополнительный API

Свойства

- `TreeListControlBase.AutoScrollOnSorting` — Указывает, будет ли контрол автоматически прокручивать порт просмотра, чтобы сделать сфокусированный узел видимым при сортировке данных.

События

- `TreeListControlBase.StartSorting` — срабатывает, когда данные собираются быть отсортированными.
- `TreeListControlBase.EndSorting` — срабатывает, когда сортировка данных завершена.

Поиск и фильтрация

Контролы TreeList и TreeView поддерживают функции, которые позволяют вам и пользователю фильтровать данные и выполнять поиск.

Панель поиска (TreeList и TreeView)

Панель поиска помогает пользователю быстро найти узлы по содержащимся в них данным. Установите для свойства `SearchPanelDisplayMode` контрола (унаследованного от класса `DataControlBase`) одно из следующих значений, чтобы включить панель поиска:

- `SearchPanelDisplayMode.Always` — Контрол постоянно отображает панель поиска.
- `SearchPanelDisplayMode.HotKey` — Контрол отображает панель поиска, когда пользователь нажимает горячую клавишу CTRL+F. Горячая клавиша ESC очищает панель поиска. Последующее нажатие клавиши ESC закрывает панель. Пользователь также может активировать панель поиска из контекстного меню заголовка столбца.

Контрол фильтрует данные (используя оператор сравнения Contains), когда пользователь вводит текст на панели поиска. Свойство `TreeListControlBase.FilterMode` указывает, какие узлы отображаются при обнаружении совпадений. Поддерживаются два режима фильтрации:

- `FilterMode.ShowMatches` — Отображает узлы, соответствующие тексту поиска.
- `FilterMode.ShowMatchesWithAncestors` — Отображает узлы, соответствующие тексту поиска, и их родительские узлы.

Функция поиска не чувствительна к регистру.

Связанный API

- `DataControlBase.SearchText` — Получает или задает текст для поиска. Вы можете присвоить значение этому свойству, чтобы отфильтровать контрол в коде. Эта функция фильтрации поддерживается, даже если панель поиска скрыта или отключена (свойству `SearchPanelDisplayMode` присвоено значение `SearchPanelDisplayMode.Never`).
- `DataControlBase.IsSearchPanelVisible` — Определяет, видна ли панель поиска в данный момент.
- `ShowSearchPanelCloseButton` — Позволяет скрыть встроенную кнопку закрытия панели поиска.

Пример

Следующий код делает панель поиска всегда видимой и позволяет отображать отфильтрованные узлы вместе с их родительскими элементами. Свойство `SearchText` определяет текст, отображаемый на панели поиска.

```
treeList.SearchPanelDisplayMode = SearchPanelDisplayMode.Always;
treeList.FilterMode = FilterMode.ShowMatchesWithAncestors;
treeList.SearchText = "department";
```

Строка автоматической фильтрации (TreeList)

Строка автоматического фильтра - это специальная строка, отображаемая над всеми узлами TreeList. Это позволяет пользователю вводить текст в свои ячейки, чтобы отфильтровать данные по соответствующим столбцам.

Установите для свойства `TreeList.ShowAutoFilterRow` значение `true`, чтобы включить автоматическую фильтрацию строки.

Вы можете использовать свойство `ColumnBase.AutoFilterCondition`, чтобы указать операцию сравнения данных для определенных ячеек (столбцов) строки автоматической фильтрации. Когда пользователь вводит текст, TreeList использует указанную операцию сравнения для сравнения значений столбцов при поиске целевых узлов. Поддерживаются следующие операции сравнения данных:

- `Equals` — Значения узлов в целевом столбце должны соответствовать введенному значению.
- `Contains` — Значения узлов в целевом столбце должны содержать введенное значение. Этот режим подходит для строковых значений.
- `StartsWith` — Значения узлов в целевом столбце должны начинаться с введенного значения. Этот режим подходит для строковых значений.
- `Default` — Опция `Default` эквивалентна опции `Contains` для типов данных String и Object. Эта опция эквивалентна опции `Equals` для других типов данных.

Свойство `ColumnBase.AutoFilterValue` позволяет вам установить значение для конкретной ячейки строки автоматической фильтрации в коде. Вы можете использовать `ColumnBase.AutoFilterValue` для фильтрации TreeList, даже если строка автоматической фильтрации скрыта.

Режим фильтрации контрола по умолчанию заключается в отображении только тех узлов, которые соответствуют указанным критериям. Установите для свойства `TreeListControlBase.FilterMode` значение `FilterMode.ShowMatchesWithAncestors`, чтобы отображать целевые узлы вместе с их родительскими.

Функциональность фильтра не зависит от регистра.

Пример

Следующий код активирует строку автоматической фильтрации и отображает узлы, значения которых в столбце "Имя" начинаются с "M".

```
treeList1.ShowAutoFilterRow = true;
TreeListColumn colName = treeList1.Columns["Name"];
colName.AutoFilterCondition = AutoFilterCondition.StartsWith;
colName.AutoFilterValue = "M";
```

Фильтрация в коде

Свойства `DataControlBase.SearchText` и `ColumnBase.AutoFilterValue` можно использовать для фильтрации контролов TreeList и TreeView в коде, даже если панель поиска и строка автоматической фильтрации скрыты.

Если вы используете свойство `DataControlBase.SearchText`, TreeList фильтрует узлы путем поиска введенного текста во всех столбцах. Используется операция сравнения содержимого.

Если вы используете свойство `ColumnBase.AutoFilterValue`, TreeList выполняет поиск введенного текста в указанном столбце. Свойство `ColumnBase.AutoFilterCondition` определяет операцию сравнения.

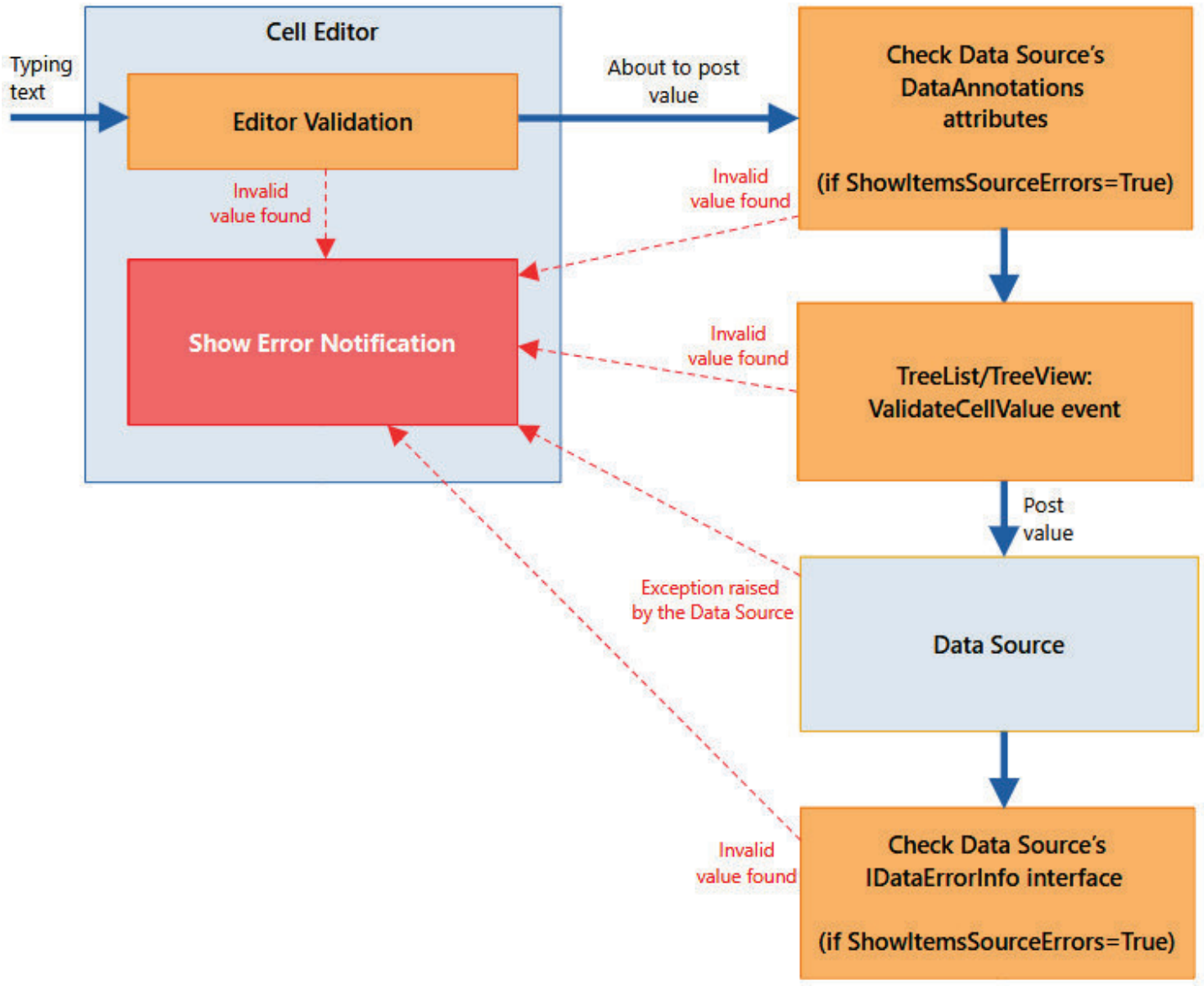
Валидация данных

Механизм валидации данных позволяет проверять значения ячеек и показывать ошибки в ячейках, содержащих недопустимые данные.

Контролы TreeList и TreeView выполняют валидацию данных, когда пользователь изменяет значение ячейки, а затем пытается сохранить (опубликовать) это значение. Контролы также запускают механизм валидации для вновь отображаемых ячеек и ячеек, обновленных в коде, даже если пользователь не изменял значения ячеек.

Проверка значений при изменении пользователем данных ячейки

Контролы TreeList и TreeView активируют механизм валидации значений ячеек, когда пользователь изменяет значение ячейки и пытается сохранить его в источнике данных (например, пользователь нажимает клавишу ENTER или перемещает фокус на другую ячейку). На следующей диаграмме показаны этапы механизма валидации, выполняемые, когда пользователь изменяет значение ячейки:



1. Встроенный редактор ячейки выполняет валидацию начального значения во время ввода данных. Например, `SpinEditor`, который принимает только числовые значения, показывает ошибки, если пользователь пытается ввести букву.

   Пользователь не может покинуть ячейку до тех пор, пока не будет введено допустимое значение или нажата клавиша ESC, которая возвращает предыдущее значение.

2. Контрол TreeList/TreeView проверяет, применяются ли Атрибуты DataAnnotations к бизнес-объекту источника данных, а затем проверяет данные в соответствии с этими правилами. Этот этап валидации включен, если свойством `DataControlBase.ShowItemsSourceErrors` является `true` (значение по умолчанию).

   Пользователь не может покинуть ячейку до тех пор, пока не будет введено допустимое значение или нажата клавиша ESC, которая возвращает предыдущее значение.

3. Контрол запускает Событие `DataControlBase.ValidateCellValue` , которое вы можете обработать для реализации валидации пользовательского значения.

   Пользователь не может покинуть ячейку до тех пор, пока не будет введено допустимое значение или нажата клавиша ESC, которая возвращает предыдущее значение.

4. Контрол публикует значение ячейки. На этом шаге источник данных может вызывать исключения, если опубликованные данные неверны. Контрол показывает ошибку в ячейке, если перехватывается какое-либо исключение.

   Пользователь не может покинуть ячейку до тех пор, пока не будет введено допустимое значение или нажата клавиша ESC, которая возвращает предыдущее значение.

5. Контрол TreeList/TreeView проверяет, реализует ли бизнес-объект Интерфейс `IDataErrorInfo` , и извлекает ошибки ячейки, если таковые имеются, используя свойство `IDataErrorInfo.Item[String]` . Этот этап валидации включен, если свойство `DataControlBase.ShowItemsSourceErrors` имеет значение `true` (значение по умолчанию).

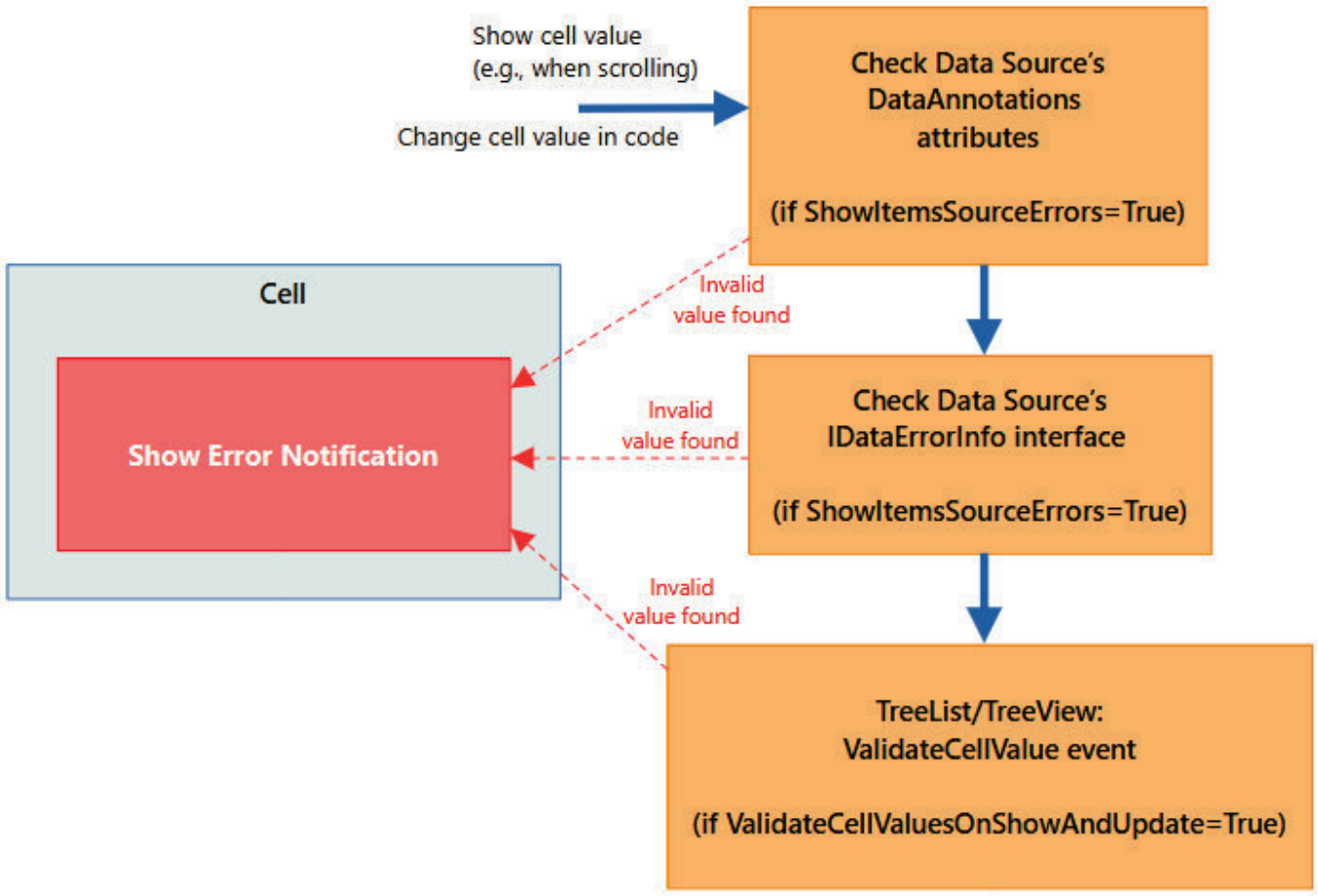
   Контрол позволяет пользователю покинуть ячейку, если значение ячейки является недопустимым, поскольку значение уже было сохранено.

Проверка значений вновь отображаемых ячеек и ячеек, обновляемых в коде

Контролы поддерживают механизм валидации для вновь отображаемых и обновленных ячеек, даже если пользователь не изменял значения ячеек. Контрол проверяет значения ячеек в следующих случаях:

- Первоначально отображается контрол, и поэтому ячейки рисуются в пределах области просмотра.
- Ячейка становится видимой при прокрутке контрола.
- Значение ячейки изменяется в коде, и поэтому контролу необходимо перерисовать ячейку.

На следующей диаграмме показаны этапы механизма валидации в этих сценариях:



1. Контрол TreeList/TreeView проверяет, применяются ли Атрибуты DataAnnotations к бизнес-объекту источника данных, а затем проверяет данные в соответствии с этими правилами. Этот этап валидации включен, если свойство `DataControlBase.ShowItemsSourceErrors` имеет значение `true` (значение по умолчанию).

2. Контрол TreeList/TreeView проверяет, реализует ли бизнес-объект Интерфейс `IDataErrorInfo` , и извлекает ошибки ячейки, если таковые имеются, используя свойство `IDataErrorInfo.Item[String]` . Этот этап валидации включен, если свойством `DataControlBase.ShowItemsSourceErrors` является `true` (значение по умолчанию).

3. Контрол запускает Событие `DataControlBase.ValidateCellValue` , которое вы можете обработать для реализации валидации пользовательского значения. Этот этап валидации включен, если свойством `DataControlBase.ValidateCellValuesOnShowAndUpdate` является `true` (значение по умолчанию - `false`).

Правила валидации источника данных

Если свойство `DataControlBase.ShowItemsSourceErrors` включено (дефолтное поведение), контрол TreeList/TreeView проверяет данные, используя атрибуты `DataAnnotations` и интерфейс

`IDataErrorInfo`, применяемые к бизнес-объекту источника данных.

Атрибуты DataAnnotations

Вы можете применить атрибуты валидации `DataAnnotations` (потомки `System.ComponentModel.DataAnnotations.ValidationAttribute`) к бизнес-объекту, чтобы указать правила валидации для свойств объекта. В приведенном ниже списке приведены наиболее распространенные атрибуты валидации:

- `CompareAttribute` — Предоставляет атрибут, который сравнивает два свойства.
- `CustomValidationAttribute` — Указывает пользовательский метод валидации, который используется для проверки свойства или экземпляра класса.
- `MaxLengthAttribute` — Указывает максимальную длину массива или строковых данных, допустимую в свойстве.
- `MinLengthAttribute` — Указывает минимальную длину массива или строковых данных, допустимую в свойстве.
- `RangeAttribute` — Задает ограничения числового диапазона для значения поля данных.
- `RegularExpressionAttribute` — Указывает, что значение поля данных должно соответствовать указанному регулярному выражению.
- `RequiredAttribute` — Указывает, что требуется значение поля данных.
- `StringLengthAttribute` — Указывает минимальную и максимальную длину символов, которые допустимы в поле данных.

Пример

Следующий код применяет атрибут `StringLength` к свойству *Code* бизнес-объекта. Атрибут заставляет пользователя вводить строку, содержащую не менее 4, но не более 8 символов.

```
public partial class Department : ObservableObject, IDataErrorInfo
{
    //...

    [ObservableProperty]
    [property: StringLength(8, MinimumLength = 4)]
    public string code = "0000";
}
```

Интерфейс IDataErrorInfo

Вы можете реализовать следующее `System.ComponentModel.IDataErrorInfo` интерфейс для бизнес-объекта для указания правил валидации свойств бизнес-объекта.

В настоящее время контролы TreeList и TreeView поддерживают ошибки только для отдельных ячеек, а не для целых строк. Таким образом, действует только свойство `IDataErrorInfo.Item[String]`, а свойство `IDataErrorInfo.Error` игнорируется.

Пример

В следующем примере реализован `IDataErrorInfo` для бизнес-объекта, который возвращает ошибку, если свойство *Phone* пусто.

```
public partial class Department : ObservableObject, IDataErrorInfo
{
    //...

    [ObservableProperty]
    public string phone = "0";

    string IDataErrorInfo.this[string columnName]
    {
        get
        {
            if (columnName != "Phone")
                return "";
            return string.IsNullOrEmpty(this.Phone) ? "Please specify the phone number" : "";
        }
    }
    string IDataErrorInfo.Error
    {
        get { return ""; }
    }
}
```

Событие ValidateCellValue для контрола

Событие `DataControlBase.ValidateCellValue` позволяет вам реализовать логику валидации пользовательских значений.

Инициирование события `DataControlBase.ValidateCellValue` является необязательным этапом механизма валидации, вызываемого после того, как пользователь изменил значение ячейки.

Механизм валидации также используется для проверки достоверности вновь отображаемых ячеек и ячеек, обновленных в коде. В этом случае событие `DataControlBase.ValidateCellValue` срабатывает только в том случае, если унаследованным свойством `DataControlBase.ValidateCellValuesOnShowAndUpdate` является `true` (значение по умолчанию - `false`).

Пример

Следующий пример обрабатывает событие `ValidateCellValue`, чтобы показывать ошибки, когда значение свойства *Date1* больше значения свойства *Date2*.

```
public partial class Department : ObservableObject, IDataErrorInfo
{
    [ObservableProperty]
    public DateTime date1 = new DateTime();

    [ObservableProperty]
    public DateTime date2 = new DateTime();
}

treeList1.ValidateCellValue += TreeList1_ValidateCellValue;

// Uncomment the following code line to use the ValidateCellValue event handler
// to check cells when they are displayed or modified in code:

// treeList1.ValidateCellValuesOnShowAndUpdate = true;

private void TreeList1_ValidateCellValue(object? sender, TreeListValidateCellValueEventArgs e)
{
    Department dep = e.Node.Content as Department;

    if(e.Column.FieldName == "Date1")
    {
        DateTime value1 = Convert.ToDateTime(e.Value);
        DateTime value2 = dep.Date2;
        if(value1 > value2)
        {
            e.ErrorContent = "Date1 must be less than Date2";
            return;
        }
    }
}
```

Контекстные меню

Контролы TreeList и TreeView поддерживают всплывающие меню, вызываемые при щелчке правой кнопкой мыши по контролам. Вы можете настроить эти меню таким образом, чтобы предоставлять пользователям пользовательские контекстно-зависимые команды.

Встроенное контекстное меню для заголовка столбца (TreeList)

Когда пользователь щелкает правой кнопкой мыши заголовок столбца TreeList, контрол отображает встроенное меню заголовка столбца. Он содержит команды, которые позволяют пользователю изменять свойства сортировки для выбранного столбца.

Используйте свойство `TreeListControl.ColumnMenu`, чтобы настроить это меню. Вы можете назначить объект `Eremex.AvaloniaUI.Controls.Bars.PopupMenu` свойству `TreeListControl.ColumnMenu`, чтобы заменить меню по умолчанию.

Чтобы настроить существующее меню заголовка столбца (добавить или удалить элементы по умолчанию), откройте меню после его инициализации (например, в обработчике событий `Initialized` вашего TreeList), а затем измените меню.

Пример - Как заменить дефолтное меню для заголовка столбца

Следующий код создает пользовательское меню заголовка столбца, содержащее элемент меню *Clear Column Data*. Код привязывает элемент меню к команде *ClearColumnDataCommand*, определенной в ViewModel.

Свойство `DataContext` заголовка элементов меню столбца (объекты `ToolbarButtonItem` в этом примере) содержит объект `TreeListColumn`, для которого было вызвано меню. `DataContext` объектов `TreeListColumn` соответствует `DataContext` в TreeList (объект *ViewModel* в этом примере).

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Name="treeList1"
                      >
    <mxtl:TreeListControl.ColumnMenu>
        <mxb:PopupMenu>
            <mxb:ToolbarButtonItem
                Header="Clear Column Data"
                Command="{Binding DataContext.ClearColumnDataCommand}"
                CommandParameter="{Binding FieldName}">
            </mxb:ToolbarButtonItem>
        </mxb:PopupMenu>
```

```
    </mxtl:TreeListControl.ColumnMenu>
</mxtl:TreeListControl>
```

```
public MainView()
{
    DataContext = new ViewModel();
}

public partial class ViewModel : ObservableObject
{
    [RelayCommand]
    void ClearColumnData(string fieldName)
    {
        //...
    }
}
```

Пример - Как изменить существующее меню для заголовка столбца

Следующий пример добавляет пользовательскую команду в дефолтное меню заголовка столбца.

В примере обрабатывается событие `TreeListControl.Initialized` для доступа к дефолтному меню заголовка столбца после его инициализации и добавляется команда *Refresh Data* в меню.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Name="treeList1"
                      Initialized="OnInitialized"
                    >
```

```
using Eremex.AvaloniaUI.Controls.Bars;
using Eremex.AvaloniaUI.Controls.TreeList;

private void OnInitialized(object? sender, System.EventArgs e)
{
    ToolbarButtonItem btn1 = new ToolbarButtonItem();
    btn1.Header = "Refresh Data";
    btn1.ShowSeparator = true;
    btn1.Command = new RelayCommand<TreeListControl>(UpdateTreeList);
    btn1.CommandParameter = treeList1;
    treeList1.ColumnMenu.Items.Add(btn1);
}

[RelayCommand]
void UpdateTreeList(TreeListControl treeList)
{
    //...
}
```

Меню для ячеек строк (TreeList и TreeView)

Контролы TreeList и TreeView имеют встроенное меню ячеек строк, заданное свойством `TreeListControlBase.RowCellMenu`. Изначально это меню пусто. Чтобы отобразить это меню, заполните его элементами в XAML или в code behind.

Пример - Как отобразить одинаковые команды контекстного меню для всех строк

Следующий пример добавляет команду "*Delete Row*" в меню ячейки строки (`TreeListControlBase.RowCellMenu`) для всех строк.

Приведенный ниже код XAML инициализирует свойство `TreeListControlBase.RowCellMenu` с помощью объекта `PopupMenu`. Всплывающее меню определяет один элемент (`ToolbarButtonItem`), свойства которого определяют заголовок элемента, команду и параметр команды.

Когда вызывается меню, его `DataContext` содержит специальный объект `CellData`, который позволяет вам получить доступ к информации, зависящей от контекста:

- Средство настройки свойств `Command` показывает, как получить доступ к контролу (объекту `TreeListControl`), `DataContext` контрола и команде *DeleteRow*, определенной в объекте `DataContext` контрола (`DataContext` контрола содержит экземпляр класса *ViewModel*).
- Установщик `CommandParameter` показывает, как получить доступ к нижележащему бизнес-объекту (строке данных), который соответствует выбранному узлу.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Name="treeList1"
                      AutoGenerateColumns="True"
                      ItemsSource="{Binding Departments}"
                      ChildrenFieldName="Children"
                      HasChildrenFieldName="HasChildren"
                      >
    <mxtl:TreeListControl.RowCellMenu>
        <mxb:PopupMenu>
            <mxb:PopupMenu.Items>
                <mxb:ToolbarButtonItem Header="Delete Row"
                 Command="{Binding DataControl.DataContext.DeleteRowCommand}"
                 CommandParameter="{Binding Row}">
                </mxb:ToolbarButtonItem>
            </mxb:PopupMenu.Items>
        </mxb:PopupMenu>
    </mxtl:TreeListControl.RowCellMenu>
</mxtl:TreeListControl>
```

```
public partial class MainView : UserControl
{
    ViewModel viewModel = new ViewModel();

    public MainView()
```

```
    {
        DataContext = viewModel;

        Department depOperations = new Department() {
            Name = "Operations", Phone = "1110", IsRoot = true };
        Department depManufacturing = new Department() {
            Name = "Manufacturing", Phone = "1111" };
        Department depQuality = new Department() { Name = "Quality", Phone = "1112" };
        depOperations.Children.Add(depManufacturing);
        depOperations.Children.Add(depQuality);

        Department depMarketing = new Department() {
            Name = "Marketing", Phone = "3120", IsRoot = true };
        Department depSales = new Department() { Name = "Sales", Phone = "3121" };
        Department depCRM = new Department() { Name = "CRM", Phone = "3122" };
        depMarketing.Children.Add(depSales);
        depMarketing.Children.Add(depCRM);

        Department depAccountsAndFinance = new Department() {
            Name = "Accounts & Finance", Phone = "5780", IsRoot = true };
        Department depAccounts = new Department() { Name = "Sales", Phone = "5781" };
        Department depFinance = new Department() { Name = "Finance", Phone = "5782" };
        depAccountsAndFinance.Children.Add(depAccounts);
        depAccountsAndFinance.Children.Add(depFinance);

        Department depHumanResources = new Department() { Name = "Human Resources",
         Phone = "7370", IsRoot = true };
        Department depHR = new Department() { Name = "HR", Phone = "7370" };
        depHumanResources.Children.Add(depHR);

        viewModel.Departments.Add(depOperations);
        viewModel.Departments.Add(depMarketing);
        viewModel.Departments.Add(depAccountsAndFinance);
        viewModel.Departments.Add(depHumanResources);

        InitializeComponent();
    }
}

public partial class ViewModel : ObservableObject
{
    public ViewModel() { }

    public ObservableCollection<Department> Departments { get; } = new();

    [RelayCommand]
    void DeleteRow(Department row)
    {
        DeleteRowRecursively(Departments, row);
    }

    bool DeleteRowRecursively(ObservableCollection<Department> departments, Department row)
    {
        foreach (Department dep in departments)
        {
            if (dep == row)
            {
```

```
                departments.Remove(row);
                return true;
            }
            if (DeleteRowRecursively(dep.Children, row))
                break;
        }
        return false;
    }
}

public partial class Department : ObservableObject
{
    [ObservableProperty]
    public string name = "";

    [ObservableProperty]
    public string phone = "0";

    [Browsable(false)]
    public bool IsRoot { get; set; } = false;

    public ObservableCollection<Department> Children { get; } = new();

    public bool HasChildren { get { return Children.Count > 0; } }

    public void AddDepartment(Department department)
    {
        Children.Add(department);
        if (Children.Count == 1)
            OnPropertyChanged(nameof(HasChildren));
    }
}
```

Пример - Как отобразить разные команды контекстного меню для разных строк

В следующем примере показаны различные команды в контекстном меню (`TreeListControlBase.RowCellMenu`) для корневой и вложенной строк. В корневом контекстном меню отображается команда "*Add Child Dep*", в то время как в контекстном меню для вложенных строк отображается команда "*Delete Row*".

Код XAML определяет контекстное меню (объект `PopupMenu`) с помощью команд "*Add Child Dep*" и "*Delete Row*". Команда "*Add Child Dep*" отображается для корневых узлов. Команда "*Delete Row*" отображается для вложенных узлов. Видимостью этих команд динамически управляет свойство *IsRoot* бизнес-объекта.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Name="treeList1"
                      AutoGenerateColumns="True"
                      ItemsSource="{Binding Departments}"
                      ChildrenFieldName="Children"
                      HasChildrenFieldName="HasChildren"
                    >
```

```
    <mxtl:TreeListControl.RowCellMenu>
        <mxb:PopupMenu>
            <mxb:PopupMenu.Items>
                <mxb:ToolbarButtonItem
                    Header="Add Child Dep"
                    Command="{Binding DataControl.DataContext.AddChildRowCommand}"
                    CommandParameter="{Binding Row}"
                    Glyph="/Assets/add.png"
                    IsVisible="{Binding Row.IsRoot}">
                </mxb:ToolbarButtonItem>
                <mxb:ToolbarButtonItem
                    Header="Delete Row"
                    Command="{Binding DataControl.DataContext.DeleteRowCommand}"
                    CommandParameter="{Binding Row}"
                    Glyph="/Assets/delete.png"
                    IsVisible="{Binding !Row.IsRoot}">
                </mxb:ToolbarButtonItem>
            </mxb:PopupMenu.Items>
        </mxb:PopupMenu>
    </mxtl:TreeListControl.RowCellMenu>
</mxtl:TreeListControl>
```

```
public partial class MainView : UserControl
{
    ViewModel viewModel = new ViewModel();

    public MainView()
    {
        DataContext = viewModel;

        Department depOperations = new Department() {
            Name = "Operations", Phone = "1110", IsRoot = true };
        Department depManufacturing = new Department() {
            Name = "Manufacturing", Phone = "1111" };
        Department depQuality = new Department() { Name = "Quality", Phone = "1112" };
        depOperations.Children.Add(depManufacturing);
        depOperations.Children.Add(depQuality);

        Department depMarketing = new Department() {
            Name = "Marketing", Phone = "3120", IsRoot = true };
        Department depSales = new Department() { Name = "Sales", Phone = "3121" };
        Department depCRM = new Department() { Name = "CRM", Phone = "3122" };
        depMarketing.Children.Add(depSales);
        depMarketing.Children.Add(depCRM);

        Department depAccountsAndFinance = new Department() {
            Name = "Accounts & Finance", Phone = "5780", IsRoot = true };
        Department depAccounts = new Department() { Name = "Sales", Phone = "5781" };
        Department depFinance = new Department() { Name = "Finance", Phone = "5782" };
        depAccountsAndFinance.Children.Add(depAccounts);
        depAccountsAndFinance.Children.Add(depFinance);

        Department depHumanResources = new Department() { Name = "Human Resources",
         Phone = "7370", IsRoot = true };
        Department depHR = new Department() { Name = "HR", Phone = "7370" };
        depHumanResources.Children.Add(depHR);
```

```
        viewModel.Departments.Add(depOperations);
        viewModel.Departments.Add(depMarketing);
        viewModel.Departments.Add(depAccountsAndFinance);
        viewModel.Departments.Add(depHumanResources);

        InitializeComponent();
    }
}

public partial class ViewModel : ObservableObject
{
    public ViewModel() { }

    public ObservableCollection<Department> Departments { get; } = new();

    [RelayCommand]
    void AddChildRow(Department parentRow)
    {
        parentRow.AddDepartment(new Department() { Name = "New dep", Phone = "0000" });
    }

    [RelayCommand]
    void DeleteRow(Department row)
    {
        DeleteRowRecursively(Departments, row);
    }

    bool DeleteRowRecursively(ObservableCollection<Department> departments, Department row)
    {
        foreach (Department dep in departments)
        {
            if (dep == row)
            {
                departments.Remove(row);
                return true;
            }
            if (DeleteRowRecursively(dep.Children, row))
                break;
        }
        return false;
    }
}

public partial class Department : ObservableObject
{
    [ObservableProperty]
    public string name = "";

    [ObservableProperty]
    public string phone = "0";

    [Browsable(false)]
    public bool IsRoot { get; set; } = false;

    public ObservableCollection<Department> Children { get; } = new();
```

```
    public bool HasChildren { get { return Children.Count > 0; } }

    public void AddDepartment(Department department)
    {
        Children.Add(department);
        if (Children.Count == 1)
            OnPropertyChanged(nameof(HasChildren));
    }
}
```

Пример - Как генерировать команды контекстного меню из ViewModel

В этом примере показано, как заполнить меню ячейки строки контрола TreeList элементами, определенными в ViewModel.

В примере свойству `TreeListControlBase.RowCellMenu` присваивается значение объекта `PopupMenu`. Меню заполнено элементами `ToolbarButtonItem`, созданными из объектов, указанных в коллекции `PopupMenu.ItemsSource`.

Свойство `PopupMenu.ItemsSource` определяет источник объектов, отображаемых в виде всплывающих элементов меню. Код XAML связывает свойство `PopupMenu.ItemsSource` со свойством *ViewModel.MenuItems*, используя следующее выражение привязки:

```
<mxb:PopupMenu ItemsSource="{Binding DataControl.DataContext.MenuItems}">
```

Когда для ячейки TreeList отображается `PopupMenu`, `DataContext` меню содержит объект `Eremex.AvaloniaUI.Controls.TreeList.CellData`. Объект `CellData` предоставляет свойство `DataControl`, которое позволяет вам получить доступ к контролу, для которого отображается меню. Объект `CellData` также содержит другие свойства для доступа к информации, связанной с ячейкой (объект столбца, строки и т.д.).

`DataContext` из элементов `PopupMenu` - это объекты, хранящиеся в коллекции `PopupMenu.ItemsSource` (объекты *MenuItemViewModel*). Следующий код привязывает свойства `ToolbarButtonItem`, `Header` и `Command` к данным, указанным объектами *MenuItemViewModel*.

```
<mxb:PopupMenu.Styles>
    <Style Selector="mxb|ToolbarButtonItem">
        <Setter Property="Header" Value="{Binding Header}"/>
        <Setter Property="Command" Value="{Binding Command}"/>
    </Style>
</mxb:PopupMenu.Styles>
```

Команде `ToolbarButtonItem` требуется информация о строке, по которой был выполнен щелчок правой кнопкой мыши. Чтобы передать строку данных команде, код XAML устанавливает свойство `ToolbarButtonItem.CommandParameter` следующим образом:

```
<mxb:PopupMenu.Styles>
    <Style Selector="mxb|ToolbarButtonItem">
```

```
        ...
        <Setter Property="CommandParameter"
         Value="{Binding $parent[mxtl:CellControl].DataContext.Row}"/>
    </Style>
</mxb:PopupMenu.Styles>
```

Здесь выражение `$parent[mxtl:CellControl]` обходит логическое дерево, чтобы найти объект `Eremex.AvaloniaUI.Controls.TreeList.CellControl` (он является родительским для `DataContext` `ToolbarButtonItem`). Объект `CellControl.DataContext` хранит объект `Eremex.AvaloniaUI.Controls.TreeList.CellData`, который позволяет вам получить доступ к строке данных из свойства `CellData.Row`.

Полный код показан ниже.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl Name="treeList1"
                      AutoGenerateColumns="True"
                      ItemsSource="{Binding Departments}"
                      ChildrenFieldName="Children"
                      HasChildrenFieldName="HasChildren"
                      ExpandStateFieldName="IsExpanded"
                      FocusedItem="{Binding SelectedDepartment, Mode=TwoWay}"
                    >
    <mxtl:TreeListControl.RowCellMenu>
        <mxb:PopupMenu ItemsSource="{Binding DataControl.DataContext.MenuItems}">
            <mxb:PopupMenu.Styles>
                <Style Selector="mxb|ToolbarButtonItem">
                    <Setter Property="Header" Value="{Binding Header}"/>
                    <Setter Property="Command" Value="{Binding Command }"/>
                    <Setter Property="CommandParameter"
                     Value="{Binding $parent[mxtl:CellControl].DataContext.Row}"/>
                </Style>
            </mxb:PopupMenu.Styles>
        </mxb:PopupMenu>
    </mxtl:TreeListControl.RowCellMenu>
</mxtl:TreeListControl>
```

```
public partial class MainView : UserControl
{
    ViewModel viewModel = new ViewModel();

    public MainView()
    {
        DataContext = viewModel;

        Department depOperations = new Department() {
            Name = "Operations", Phone = "1110", IsRoot = true };
        Department depManufacturing = new Department() { Name = "Manufacturing", Phone = "1111" };
        Department depQuality = new Department() { Name = "Quality", Phone = "1112" };
        depOperations.Children.Add(depManufacturing);
```

```
        depOperations.Children.Add(depQuality);

        Department depMarketing = new Department() {
            Name = "Marketing", Phone = "3120", IsRoot = true };
        Department depSales = new Department() { Name = "Sales", Phone = "3121" };
        Department depCRM = new Department() { Name = "CRM", Phone = "3122" };
        depMarketing.Children.Add(depSales);
        depMarketing.Children.Add(depCRM);

        Department depAccountsAndFinance = new Department() {
            Name = "Accounts & Finance", Phone = "5780", IsRoot = true };
        Department depAccounts = new Department() { Name = "Sales", Phone = "5781" };
        Department depFinance = new Department() { Name = "Finance", Phone = "5782" };
        depAccountsAndFinance.Children.Add(depAccounts);
        depAccountsAndFinance.Children.Add(depFinance);

        Department depHumanResources = new Department() {
            Name = "Human Resources", Phone = "7370", IsRoot = true };
        Department depHR = new Department() { Name = "HR", Phone = "7370" };
        depHumanResources.Children.Add(depHR);

        viewModel.Departments.Add(depOperations);
        viewModel.Departments.Add(depMarketing);
        viewModel.Departments.Add(depAccountsAndFinance);
        viewModel.Departments.Add(depHumanResources);

        InitializeComponent();
    }
}

public partial class ViewModel : ObservableObject
{
    [ObservableProperty]
    Department? selectedDepartment;

    public ViewModel()
    {
        MenuItemViewModel menuItem1 = new MenuItemViewModel();
        menuItem1.Header = "Add Dep";
        menuItem1.Command = new RelayCommand<Department>(AddChildRow);
        MenuItems.Add(menuItem1);
    }

    public ObservableCollection<Department> Departments { get; } = new();

    public ObservableCollection<MenuItemViewModel> MenuItems { get; } = new();

    [RelayCommand]
    void AddChildRow(Department? parentRow)
    {
        if (parentRow == null) return;

        Department newDepartment = new() { Name = "New dep", Phone = "0000" };
        parentRow.AddDepartment(newDepartment);
        parentRow.IsExpanded = true;
        SelectedDepartment = newDepartment;
    }
```

```
}

public partial class MenuItemViewModel : ObservableObject
{
    [ObservableProperty]
    string? header;

    [ObservableProperty]
    ICommand? command;
}

public partial class Department : ObservableObject
{
    [ObservableProperty]
    public string name = "";

    [ObservableProperty]
    public string phone = "0";

    [ObservableProperty]
    public bool isExpanded;

    [Browsable(false)]
    public bool IsRoot { get; set; } = false;

    public ObservableCollection<Department> Children { get; } = new();

    public bool HasChildren => Children.Count > 0;

    public void AddDepartment(Department department)
    {
        Children.Add(department);
        if (Children.Count == 1)
            OnPropertyChanged(nameof(HasChildren));
    }
}
```

Настройка меню при его показе

Вы можете обработать событие `PopupMenu.Opening` для динамической настройки меню TreeList. Событие происходит, когда вот-вот отобразится всплывающее меню.

Пример - Как отобразить контекстное меню для первого столбца

В следующем примере свойству `TreeListControlBase.RowCellMenu` присваивается пустое значение `PopupMenu`, а затем обрабатывается событие `PopupMenu.Opening` для заполнения меню элементами, когда пользователь щелкает правой кнопкой мыши ячейки в первом видимом столбце TreeList. Меню остается пустым (и, следовательно, не отображается), когда пользователь щелкает правой кнопкой мыши в других столбцах.

Созданное меню содержит кнопку-переключатель "*Show/Hide Root Indent*", которая переключает видимость отступа TreeList (свойство `TreeListControlBase.ShowRootIndent`).

```
<mxtl:TreeListControl Name="treeList1" ...>
    <mxtl:TreeListControl.RowCellMenu>
        <mxb:PopupMenu Opening="RowCellMenuOpening"/>
    </mxtl:TreeListControl.RowCellMenu>
</mxtl:TreeListControl>
```

```
void RowCellMenuOpening(object? sender, CancelEventArgs e)
{
    if (sender == null) return;
    PopupMenu menu = sender as PopupMenu;
    menu.Items.Clear();

    CellData cellData = menu.DataContext as CellData;
    TreeListControl control = cellData.DataControl as TreeListControl;
    TreeListColumn column = cellData.Column as TreeListColumn;
    //// Access the underlying data row, when required.
    //object dataRow = cellData.Row;

    if(column.VisibleIndex == 0)
    {
        ToolbarCheckItem btn1 = new ToolbarCheckItem();
        btn1.IsChecked = control.ShowRootIndent;
        btn1.Header = (control.ShowRootIndent) ? "Hide Root Indent" : "Show Root Indent";
        btn1.Command = new RelayCommand<TreeListControl>(ShowRootIndentCommand);
        btn1.CommandParameter = control;
        menu.Items.Add(btn1);
    }
}

[RelayCommand]
void ShowRootIndentCommand(TreeListControl treeList)
{
    treeList.ShowRootIndent = !treeList.ShowRootIndent;
}
```

Отображение контекстного меню для контролов, используя attached-свойство ToolbarManager.

Компонент `Eremex.AvaloniaUI.Controls.Bars.ToolbarManager` предоставляет attached-свойство `ContextPopup`, которое позволяет назначать контекстное меню любому контролу, включая TreeList и TreeView. Это контекстное меню отображается для регионов TreeList/TreeView, в которых нет контекстных меню по умолчанию, и для регионов с пустыми меню по умолчанию.

Пример - Как назначить контекстное меню, используя attached-свойство `ToolbarManager.ContextPopup`

Следующий код использует attached-свойство `ToolbarManager.ContextPopup` для указания контекстного меню для контрола TreeList. Меню содержит переключающий элемент меню *Show Column Header Panel/Hide Column Header Panel*, который переключает видимость опции `TreeListControl.ShowColumnHeaders`.

```
<mxtl:TreeListControl Name="treeList1"
                >
    <mxtl:TreeListControl.Styles>
        <Style Selector="mxtl|TreeListControl">
            <Setter Property="mxb:ToolbarManager.ContextPopup" >
                <Template>
                    <mxb:PopupMenu Tag="treeList1">
                        <mxb:ToolbarCheckItem Header="Show Column Header Panel"
                         IsChecked="{Binding $parent[mxtl:TreeListControl].ShowColumnHeaders, Mode=TwoWay}"
                         IsVisible="{Binding !$parent[mxtl:TreeListControl].ShowColumnHeaders}"/>
                        <mxb:ToolbarCheckItem Header="Hide Column Header Panel"
                         IsChecked="{Binding $parent[mxtl:TreeListControl].ShowColumnHeaders, Mode=TwoWay}"
                         IsVisible="{Binding $parent[mxtl:TreeListControl].ShowColumnHeaders}"/>
                    </mxb:PopupMenu>
                </Template>
            </Setter>
        </Style>
    </mxtl:TreeListControl.Styles>

</mxtl:TreeListControl>
```

Стили

Контролы TreeList и TreeView поддерживают систему стилизации Avalonia, которая позволяет настраивать свойства визуальных элементов контролов. Используйте свойство `TreeListControl.Styles`, чтобы применить стили к следующим визуальным элементам:

- Заголовок столбца
- Ряд
- Ячейка строки
- Отступ строки

Типичный стиль содержит селектор стилей и набор средств настройки свойств. Селектор определяет целевой объект контрола, который инкапсулирует визуальный элемент. Установщики свойств определяют свойства целевого контрола и их значения.

Следующий код создает простой стиль, который устанавливает фон для всех строк TreeList одинакового цвета.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl.Styles>
    <Style Selector="mxtl|TreeListRowControl">
        <Setter Property="Background" Value="Ivory"/>
    </Style>
</mxtl:TreeListControl.Styles>
```

Применение стилей к определенным состояниям контрола

Селекторы стилей поддерживают псевдоклассы в Avalonia. Псевдокласс - это ключевое слово, добавляемое к селектору. Он определяет состояние целевого контрола, к которому применяется стиль.

Контролы TreeList и TreeView предоставляют пользовательские псевдоклассы для обращения к определенным состояниям визуальных элементов контролов. В разделах ниже перечислены пользовательские псевдоклассы, поддерживаемые для визуальных элементов контролов.

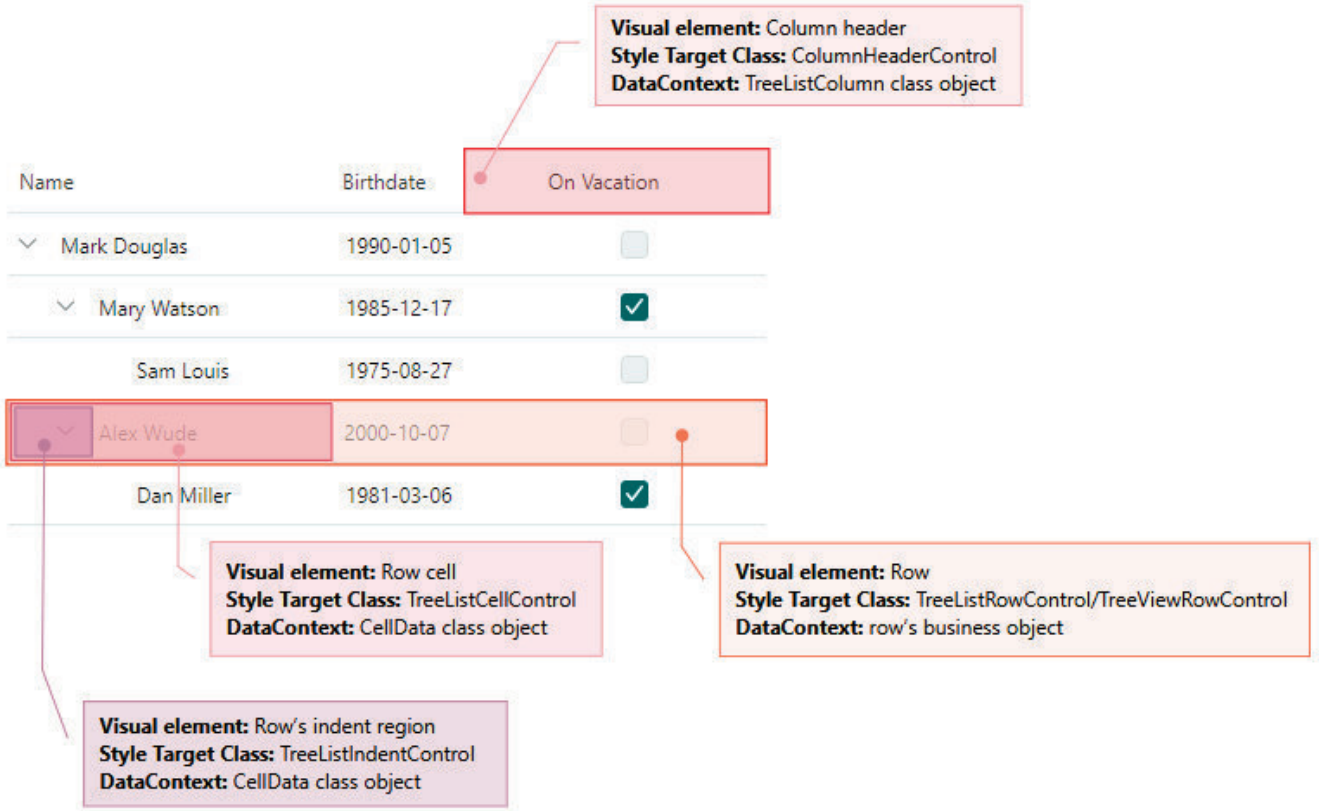
Приведенный ниже код создает селектор стилей, который нацелен на строки TreeList. Псевдокласс `:focusedAndSelectedState` используется только для применения стиля к строкам, которые находятся в сфокусированном состоянии.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<Style Selector="mxtl|TreeListRowControl:focusedAndSelectedState">
```

```
    ...
</Style>
```

Контролы, определяющие визуальные элементы

Синтаксис селектора стилей должен указывать целевой контрол (объект контрола, который инкапсулирует визуальный элемент), к которому применяется стиль. На следующей картинке показаны визуальные элементы TreeList/TreeView и соответствующие целевые контролы.



Стиль строки (узла)

Строка - это визуальный элемент, отображающий ячейки.

Используйте следующую информацию, чтобы настроить стиль строки:

Целевой класс контрола

`TreeListRowControl` (в контроле TreeList), `TreeViewRowControl` (в контроле TreeView)

DataContext

Объект данных (бизнес) строки

Пользовательские псевдоклассы

- ":selectedState" — строка выбрана, но не сфокусирована. Это состояние действует в режиме выбора нескольких строк.

- ":focusedState" — строка сфокусирована, но не выбрана. Это состояние действует в режиме выбора нескольких строк.
- ":focusedAndSelectedState" — строка сфокусирована и выбрана. Это состояние действует в режимах выбора нескольких и одной строки.
- ":editingState" — Узел имеет активный редактор встроенных файлов.

Пример - Как настроить стиль сфокусированного узла

Следующий код создает стиль, который настраивает фон сфокусированного узла TreeList.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl.Styles>
    <Style Selector="mxtl|TreeListRowControl:focusedAndSelectedState">
        <Setter Property="Background" Value="SkyBlue"/>
    </Style>
</mxtl:TreeListControl.Styles>
```

Пример - Как настроить стиль узла в соответствии со значением столбца

Следующий код настраивает фон строк (узлов), для свойства *OnVacation* которых установлено значение `true`.

Объект `Binding` получает логическое значение свойства *OnVacation* из DataContext, который содержит бизнес-объект строки. Затем он преобразует логическое значение в кисть с помощью конвертера `Eremex.AvaloniaUI.Controls.BoolToObjectConverter` и возвращает результат преобразования.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mx="clr-namespace:Eremex.AvaloniaUI.Controls;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl.Styles>
    <Style Selector="mxtl|TreeListRowControl">
        <Setter Property="Background">
            <Setter.Value>
                <Binding Path="OnVacation">
                    <Binding.Converter>
                        <mx:BoolToObjectConverter>
                            <mx:BoolToObjectConverter.TrueValue>
                                <SolidColorBrush Color="LightYellow"/>
                            </mx:BoolToObjectConverter.TrueValue>
                        </mx:BoolToObjectConverter>
                    </Binding.Converter>
                </Binding>
            </Setter.Value>
        </Setter>
    </Style>
</mxtl:TreeListControl.Styles>
```

Стиль ячейки строки

Ячейки строк (узлов) отображают значения столбцов. В контроле TreeView в каждой строке отображается только одно значение.

Чтобы настроить стиль ячейки строки, используйте следующую информацию:

Целевой класс контрола

`TreeListCellControl`

DataContext

Объект `Eremex.AvaloniaUI.Controls.DataControl.Visuals.CellData`. Основными свойствами, предоставляемыми объектом `CellData`, являются:

- `Column` — столбец (объект `ColumnBase`), который отображает ячейку. Когда вы применяете стиль к объекту TreeView, свойство `CellData.Column` содержит объект `ColumnBase`, который инкапсулирует столбец значений TreeView.
- `DataControl` — Контрол TreeList/TreeView (объект `DataControlBase`).
- `Row` — нижележащий объект данных (бизнес) строки.
- `ValidationInfo` — объект, содержащий информацию о валидации ячейки.
- `Value` — Значение ячейки.

Пользовательские псевдоклассы

Пользовательские псевдоклассы соответствуют тем, которые применяются к объектам row. Они показаны ниже:

- ":selectedState" — строка выбрана, но не сфокусирована. Это состояние действует в режиме выбора нескольких строк.
- ":focusedState" — строка сфокусирована, но не выбрана. Это состояние действует в режиме выбора нескольких строк.
- ":focusedAndSelectedState" — строка сфокусирована и выбрана. Это состояние действует в режимах выбора нескольких и одной строки.
- ":editingState" — Узел имеет активный редактор встроенных файлов.

Пример - Как настроить стиль ячейки в соответствии со значением ячейки

Объект Style в приведенном ниже коде задает фон ячеек, содержащих значение `true` в свойстве строки *OnVacation*.

Стиль задает свойству `Background` контролов ячеек (объектов `TreeListCellControl`) значение brush, возвращаемое пользовательским конвертером *OnVacationCellValueToBrushConverter*. Конвертер возвращает кисть в соответствии с столбцом ячейки и значением.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mx="clr-namespace:Eremex.AvaloniaUI.Controls;assembly=Eremex.Avalonia.Controls"
xmlns:local="clr-namespace:AvaloniaApp1"

<Grid.Resources>
    <local:OnVacationCellValueToBrushConverter x:Key="myOnVacationCellValueToBrushConverter"/>
</Grid.Resources>

<mxtl:TreeListControl.Styles>
    <Style Selector="mxtl|TreeListCellControl">
        <Setter Property="Background">
            <Setter.Value>
                <MultiBinding Converter="{StaticResource myOnVacationCellValueToBrushConverter}">
```

```
                    <Binding Path="Row.OnVacation"/>
                    <Binding Path="Column"/>
                </MultiBinding>
            </Setter.Value>
        </Setter>
    </Style>
</mxtl:TreeListControl.Styles>
```

```
public class OnVacationCellValueToBrushConverter : IMultiValueConverter
{
    public object? Convert(IList<object?> values, Type targetType, object? parameter, CultureInfo culture)
    {
        bool onVacation = (bool)values[0];
        TreeListColumn column = (TreeListColumn)values[1];
        if (column.FieldName == "OnVacation" && onVacation)
            return new SolidColorBrush(Colors.Aqua);
        return null;
    }
}
```

Стиль для области отступа строки

В области отступа отображаются кнопки развертывания и картинки узлов. Область отступа - это часть ячейки, которая отображает иерархию узлов.

Чтобы настроить стиль области отступа строки, используйте следующую информацию:

Целевой класс контрола

`TreeListIndentControl`

DataContext

Объект `Eremex.AvaloniaUI.Controls.DataControl.Visuals.CellData`. Смотрите раздел Стиль ячейки строки для получения дополнительной информации.

Пользовательские псевдоклассы

Пользовательские псевдоклассы соответствуют тем, которые применяются к объектам row. Они показаны ниже:

- ":selectedState" — строка выбрана, но не сфокусирована. Это состояние действует в режиме выбора нескольких строк.
- ":focusedState" — строка сфокусирована, но не выбрана. Это состояние действует в режиме выбора нескольких строк.
- ":focusedAndSelectedState" — строка сфокусирована и выбрана. Это состояние действует в режимах выбора нескольких и одной строки.
- ":editingState" — Узел имеет активный редактор встроенных файлов.

Пример - Как настроить область отступа определенной строки

Следующий код создает стиль, который настраивает фон отступов в строках, для которых значения строк *HasChildren* установлены на `true`.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:mx="clr-namespace:Eremex.AvaloniaUI.Controls;assembly=Eremex.Avalonia.Controls"

<mxtl:TreeListControl.Styles>
    <Style Selector="mxtl|TreeListIndentControl">
        <Setter Property="Background">
            <Setter.Value>
                <Binding Path="Row.HasChildren">
                    <Binding.Converter>
                        <mx:BoolToObjectConverter>
                            <mx:BoolToObjectConverter.TrueValue>
                                <SolidColorBrush Color="Coral"/>
                            </mx:BoolToObjectConverter.TrueValue>
                        </mx:BoolToObjectConverter>
                    </Binding.Converter>
                </Binding>
            </Setter.Value>
        </Setter>
    </Style>
</mxtl:TreeListControl.Styles>
```

Стиль заголовка столбца (TreeList)

В заголовке столбца отображается заголовок столбца и индикаторы сортировки.

Используйте следующую информацию, чтобы настроить стиль заголовка столбца:

Целевой класс контрола

`ColumnHeaderControl`

DataContext

`TreeListColumn`

Пользовательские псевдоклассы

- ":сортировка по возрастанию" — столбец сортируется в порядке возрастания.
- ":sortdescending" — столбец сортируется в порядке убывания.
- ":перетаскивание" — перетаскивается столбец.

Пример - Как настроить стиль определенного столбца

Следующий код применяет стиль полужирного шрифта к заголовку столбца TreeList *OnVacation*.

```
xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
xmlns:local="clr-namespace:AvaloniaApp1"

<Grid.Resources>
    <local:ColumnToFontWeightConverter x:Key="myColumnToFontWeightConverter"/>
</Grid.Resources>

<mxtl:TreeListControl.Styles>
    <Style Selector="mxtl|ColumnHeaderControl">
        <Setter Property="FontWeight" Value="{Binding Converter={StaticResource myColumnToFontWeightConverter}}">
```

```
        </Setter>
    </Style>
</mxtl:TreeListControl.Styles>
```

```
public class ColumnToFontWeightConverter : IValueConverter
{
    public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
    {
        TreeListColumn col = value as TreeListColumn;
        if(col!=null && col.FieldName=="OnVacation")
        {
            return FontWeight.Bold;
        }

        return null;
    }

    public object? ConvertBack(object? value, Type targetType, object? parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Примеры для контролов TreeList и Tree View

- Как создать контрол TreeList и привязать его к иерархическому источнику данных
- Как создать контрол TreeView и привязать его к самореферентному источнику данных

Как создать контрол TreeView и привязать его к самореферентному источнику данных

В этом примере создается контрол `TreeViewControl`, который отображает иерархическую коллекцию объектов *Employee*. Имя сотрудника, который в данный момент находится в TreeList, отображается в текстовом редакторе.

Класс *Employee* является [самореферентным источником данных](). Он хранит информацию о родительско-дочерних отношениях в двух служебных свойствах (*ID* и *ParentID*).

В приведенном примере для настройки `TreeViewControl` используются следующие основные свойства:

- `DataControlBase.ItemsSource` — Указывает источник данных контрола.
- `TreeListControlBase.KeyFieldName` — указывает имя *Key field* (свойства), в котором хранятся уникальные идентификаторы записей.
- `TreeListControlBase.ParentFieldName` — указывает имя *Key field* родительской записи (свойство).
- `TreeListControlBase.RootValue` — Идентифицирует корневые записи в связанной коллекции. Свойство `RootValue` определяет значение, которое имеют корневые записи в *Parent key field* (свойство).
- `TreeViewControl.DataFieldName` — указывает имя поля (свойства), данные которого отображаются в `TreeViewControl`.

Когда пользователь фокусирует узел, вы можете использовать унаследованное свойство `DataControlBase.FocusedItem` TreeView для извлечения нижележащего объекта данных сфокусированного узла. В примере свойство `DataControlBase.FocusedItem` возвращает объект *Employee*. Значение *Name* сфокусированного объекта *Employee* отображается в `TextEditor`.

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:vm="using:TreeControls"

        mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
        xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
        xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

```
        x:Class="TreeControls.MainWindow"
        Title="TreeControls">
    <Grid ColumnDefinitions="*, 200">
        <mxtl:TreeViewControl
            Grid.Column="0"
            Name="treeView1"
            ItemsSource="{Binding Employees}"
            DataFieldName="Name"
            KeyFieldName="ID"
            ParentFieldName="ParentID"
            FocusedItem="{Binding Current}"
            >
            <mxtl:TreeViewControl.RootValue>
                <sys:Int32>-1</sys:Int32>
            </mxtl:TreeViewControl.RootValue>
        </mxtl:TreeViewControl>
        <StackPanel Orientation="Vertical" Grid.Column="1" >
            <Label Content="Focused Item:"></Label>

        <mxe:TextEditor
            ReadOnly="True"
            EditorValue="{Binding Current.Name}"
            >
        </mxe:TextEditor>
        </StackPanel>
    </Grid>
</Window>
```

```
using Avalonia.Controls;
using CommunityToolkit.Mvvm.ComponentModel;
using System;
using System.Collections.ObjectModel;

namespace TreeControls
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            MyViewModel viewModel = new MyViewModel();
            this.DataContext = viewModel;

            InitializeComponent();

            treeView1.ExpandAllNodes();
        }
    }

    public partial class MyViewModel : ObservableObject
    {
        public MyViewModel()
        {
            Employees = new ObservableCollection<Employee>
            {
```

```
                new Employee() { ID = 0, ParentID = -1, Name = "Serge Smolin",
                 Birthdate = new DateTime(1990, 01, 5) },
                new Employee() { ID = 1, ParentID = 0, Name = "Alex Douglas",
                 Birthdate = new DateTime(1975, 8, 27) },
                new Employee() { ID = 2, ParentID = 0, Name = "Dennis Parker",
                 Birthdate = new DateTime(1985, 12, 17) },
                new Employee() { ID = 3, ParentID = 1, Name = "Pavel Morris",
                 Birthdate = new DateTime(1987, 10, 15) },
                new Employee() { ID = 4, ParentID = 2, Name = "Mary Thompson",
                 Birthdate = new DateTime(1991, 03, 16) },
                new Employee() { ID = 5, ParentID = 3, Name = "Vera Liskina",
                 Birthdate = new DateTime(1991, 04, 16) }
            };

            Current = null;
        }

        public ObservableCollection<Employee> Employees { get; set; }

        [ObservableProperty]
        public Employee? current;
    }

    public partial class Employee : ObservableObject
    {
        [ObservableProperty]
        public string name = "";

        [ObservableProperty]
        public DateTime? birthdate = null;

        public int ID { get; set; }

        public int ParentID { get; set; }
    }
}
```

Как создать контрол TreeList и привязать его к иерархическому источнику данных

В этом примере создаются следующие контролы:

- Контрол `TreeListControl`, который отображает иерархический список объектов *Employee*. Включен выбор нескольких узлов, что позволяет вам выбирать (выделять) несколько узлов одновременно.
- Текстовый редактор, отображающий имя сотрудника, который в данный момент находится в TreeList.
- Контрол со списком, в котором отображаются имена сотрудников, выбранных (выделенных) в TreeList.

TreeList привязан к бизнес-объекту *Employee*, который представляет собой Иерархический источник данных . Он содержит коллекцию *Subordinates*, содержимое которой должно отображаться в виде дочерних узлов.

В приведенном примере для настройки `TreeListControl` используются следующие основные свойства:

- `DataControlBase.ItemsSource` — Указывает источник данных контрола.
- `TreeListControl.Columns` — Определяет коллекцию столбцов TreeList, привязанных к свойствам источника данных.
- `TreeListControlBase.ChildrenFieldName` — Указывает имя поля (свойства), в котором хранятся дочерние данные в нижележащем бизнес-объекте.
- `TreeListControlBase.HasChildrenFieldName` — указывает имя поля (свойства), которое возвращает `true`, если бизнес-объект имеет дочерние данные, и `false`, в противном случае.
- `TreeListControlBase.SelectionMode` — Позволяет выбирать несколько узлов.

В режиме выбора нескольких узлов пользователь может выбрать (выделить) несколько узлов с помощью мыши и клавиатуры. Например, пользователь может удерживать нажатой клавишу CTRL и щелкать по отдельным узлам, чтобы выбрать их. TreeList позволяет вам получить доступ к выбранным в данный момент узлам из коллекции `DataControlBase.SelectedItems`.

Следующие два контрола используются для отображения информации о сфокусированных в данный момент и выбранных узлах TreeList:

- Текстовый редактор отображает имя объекта *Employee* сфокусированного узла. Контрол привязан к свойству `FocusedItem` TreeList.
- В окне списка отображаются имена объектов *Employee*, которые соответствуют выбранным узлам TreeList.

Пример находит два узла TreeList по именам и выбирает их при запуске приложения.

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

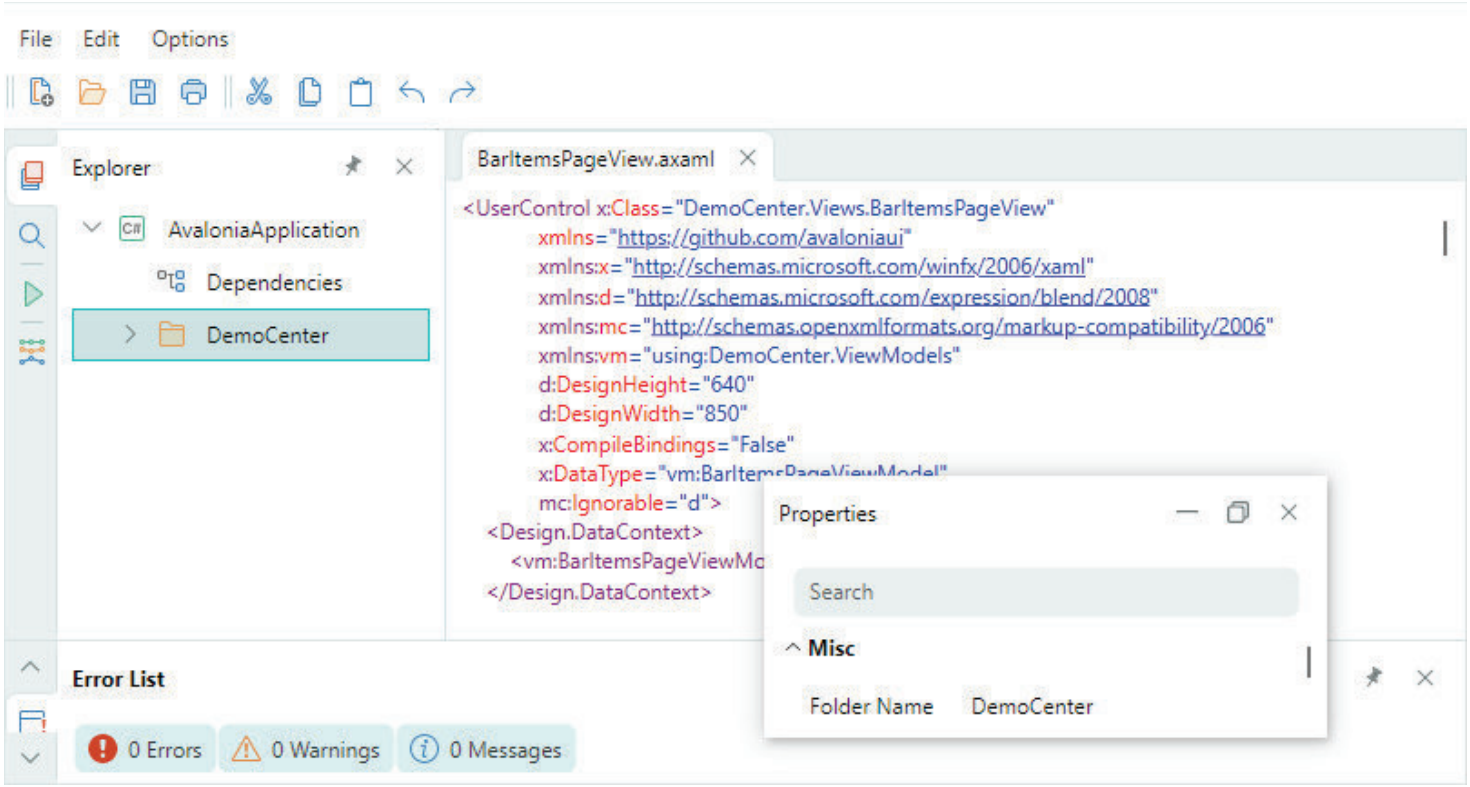
```
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:vm="using:TreeControls"
        mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
        xmlns:mxtl="clr-namespace:Eremex.AvaloniaUI.Controls.TreeList;assembly=Eremex.Avalonia.Controls"
        xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        x:Class="TreeControls.MainWindow"
        Title="TreeControls">
    <Grid ColumnDefinitions="*, 200">
        <mxtl:TreeListControl
            Grid.Column="0"
            Name="treeList1"
            ItemsSource="{Binding Employees}"
            ChildrenFieldName="Subordinates"
            HasChildrenFieldName="HasChildren"
            SelectionMode="Multiple"
            SelectedItems="{Binding SelectedEmployees}"
            >
            <mxtl:TreeListControl.Columns>
                <mxtl:TreeListColumn Name="colName" FieldName="Name" Header="Name"/>
                <mxtl:TreeListColumn Name="colBirthdate" FieldName="Birthdate" Header="Birthdate"/>
            </mxtl:TreeListControl.Columns>
        </mxtl:TreeListControl>

        <StackPanel Orientation="Vertical" Grid.Column="1" >
            <Label Content="Focused Item:"></Label>

            <mxe:TextEditor
                ReadOnly="True"
                EditorValue="{Binding #treeList1.FocusedItem.Name}"
            >
            </mxe:TextEditor>
            <Label Content="Selected Items:" Margin="0,20,0,0"></Label>
            <ListBox Name="listBox"
                     Margin="6"
                     ItemsSource="{Binding SelectedEmployees}"
                 >
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding Name}"/>
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
        </StackPanel>
    </Grid>
</Window>
```

```
using Avalonia.Controls;
using CommunityToolkit.Mvvm.ComponentModel;
using Eremex.AvaloniaUI.Controls.TreeList;
using System;
using System.Collections.ObjectModel;

namespace TreeControls
{
    public partial class MainWindow : Window
```

```
    {
        public MainWindow()
        {
            MyViewModel viewModel = new MyViewModel();
            this.DataContext = viewModel;

            InitializeComponent();

            treeList1.SelectionMode = Eremex.AvaloniaUI.Controls.DataControl.RowSelectionMode.Multiple;
            treeList1.ExpandAllNodes();
            TreeListNode node1 = treeList1.FindNode(node => (node.Content as Employee).Name.Contains("Sam"));
            TreeListNode node2 = treeList1.FindNode(node => (node.Content as Employee).Name.Contains("Dan"));
            treeList1.BeginSelection();
            treeList1.ClearSelection();
            treeList1.SelectNode(node1);
            treeList1.SelectNode(node2);
            treeList1.EndSelection();
        }
    }

    public partial class MyViewModel : ObservableObject
    {
        public MyViewModel()
        {
            Employee p1 = new Employee() { Name = "Mark Douglas", Birthdate = new DateTime(1990, 01, 5) };
            Employee p2 = new Employee() { Name = "Mary Watson", Birthdate = new DateTime(1985, 12, 17) };
            Employee p3 = new Employee() { Name = "Alex Wude", Birthdate = new DateTime(2000, 10, 7) };
            Employee p4 = new Employee() { Name = "Sam Louis", Birthdate = new DateTime(1975, 8, 27) };
            Employee p5 = new Employee() { Name = "Dan Miller", Birthdate = new DateTime(1981, 3, 6) };
            p1.Subordinates.Add(p2);
            p1.Subordinates.Add(p3);
            p2.Subordinates.Add(p4);
            p3.Subordinates.Add(p5);

            Employees = new ObservableCollection<Employee>() { p1 };

            SelectedEmployees = new();
        }

        public ObservableCollection<Employee> Employees { get; set; }

        public ObservableCollection<Employee>? SelectedEmployees { get; set; }
    }

    public partial class Employee : ObservableObject
    {
        [ObservableProperty]
        public string name = "";

        [ObservableProperty]
        public DateTime? birthdate = null;

        public ObservableCollection<Employee> Subordinates { get; } = new();

        public bool HasChildren { get { return Subordinates.Count > 0; } }
    }
}
```

Интерфейс Докинга

Компонент **Dock Manager**, поставляемый как часть библиотеки Контролов Eremex для Avalonia UI, позволяет реализовать классический Интерфейс Докинга, который вы можете найти в популярных IDE. Вы можете создавать панели инструментов, которые поддерживают операции докинга, автоматического скрытия и перемещения. Специальные контейнеры для Документов предназначены для отображения основного содержимого вашего окна. Вы можете создать несколько Документов и организовать их в пользовательский интерфейс с вкладками.



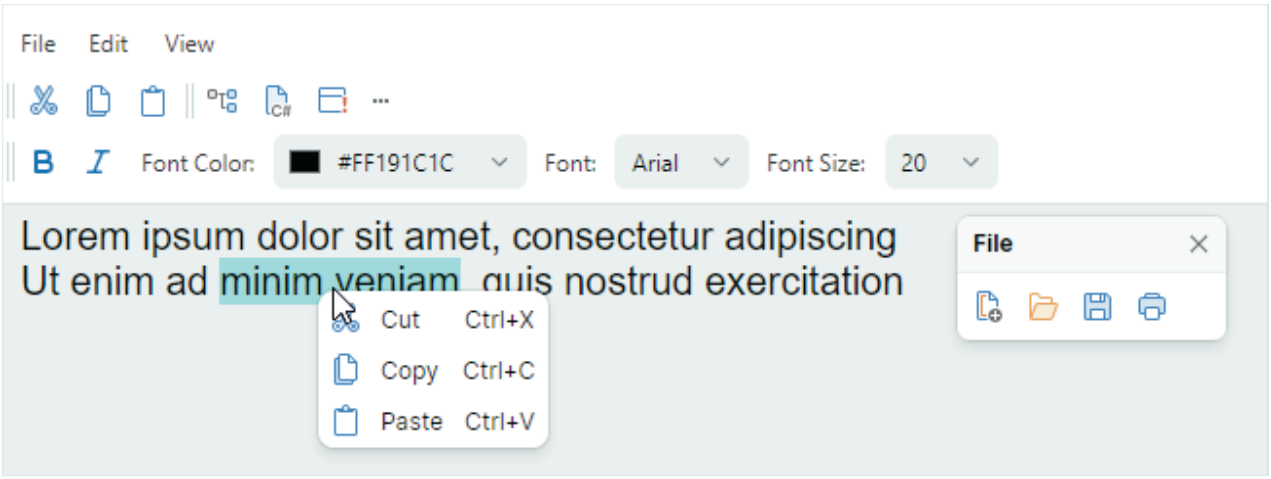
Основные функции библиотеки докинга включают в себя:

- Пользовательский интерфейс, вдохновленный Visual Studio IDE.
- Пристыкованные панели позволяют создавать закрепляемые панели инструментов.
- Плавающие панели.
- Функция автоматического скрытия панели.
- Пользователи могут переставлять панели и изменять их размер любым удобным им способом.
- Интегрированная поддержка Документов, которые представляют собой специальные контейнеры, используемые для отображения основного содержимого вашего окна.
- Объединение нескольких Документов в пользовательском интерфейсе с вкладками.

- Встроенные контекстные команды для выполнения операций с панелями и Документами.
- Поддержка MVVM.
- Механизм сериализации и десериализации.

Панели инструментов и меню

Библиотека контролов Eremex включает компонент `ToolbarManager`, который помогает создавать традиционные Панели Инструментов и Меню в вашем приложении. `ToolbarManager` предлагает широкие возможности для размещения панелей и управления ими, создания контекстных меню, настройки вида панели инструментов и свойств поведения, а также настройки панелей во время выполнения программы.



Вы можете использовать несколько типов элементов панели для создания пользовательского интерфейса меню: обычные кнопки, кнопки-переключатели, редакторы, метки, подменю и элементы групп.

Панели инструментов могут быть настроены пользователями. Во время выполнения программы они могут выбрать отображение нужных им команд.

Основные функции библиотеки Панелей Инструментов и Меню включают в себя:

- Настройка пользователем:
    - Перетаскивание панели — позволяет пользователю переставлять панели с помощью маркеров перетаскивания.
    - Режим настройки и окно настройки — В окне настройки пользователь может изменять видимость существующих панелей инструментов и создавать пользовательские панели инструментов. Пользователь может скрывать, восстанавливать и перемещать элементы панели инструментов с помощью операций перетаскивания.
    - Быстрая настройка — Пользователь может перемещать элементы внутри панелей и между ними с помощью перетаскивания, удерживая нажатой клавишу Alt. Нет

необходимости активировать режим настройки /окно настройки для выполнения быстрой настройки.

- Многочисленные опции размещения панелей:
  - Горизонтальное и вертикальное направления отдельных панелей.
  - Любая позиция в пределах окна.
  - Растягивание панели.
  - Адаптивное размещение панелей инструментов — панели инструментов автоматически скрывают и восстанавливают свои элементы при изменении занимаемого ими пространства.
  - Опция для отображения/скрытия маркеров перетаскивания панели.
  - Опция для отображения/скрытия кнопок настройки панели.
- Поддерживаемые элементы панели: `ToolbarButtonItem`, `ToolbarCheckItem`, `ToolbarMenuItem`, `ToolbarEditorItem`, `ToolbarTextItem`, `ToolbarItemGroup` и `ToolbarCheckItemGroup`.

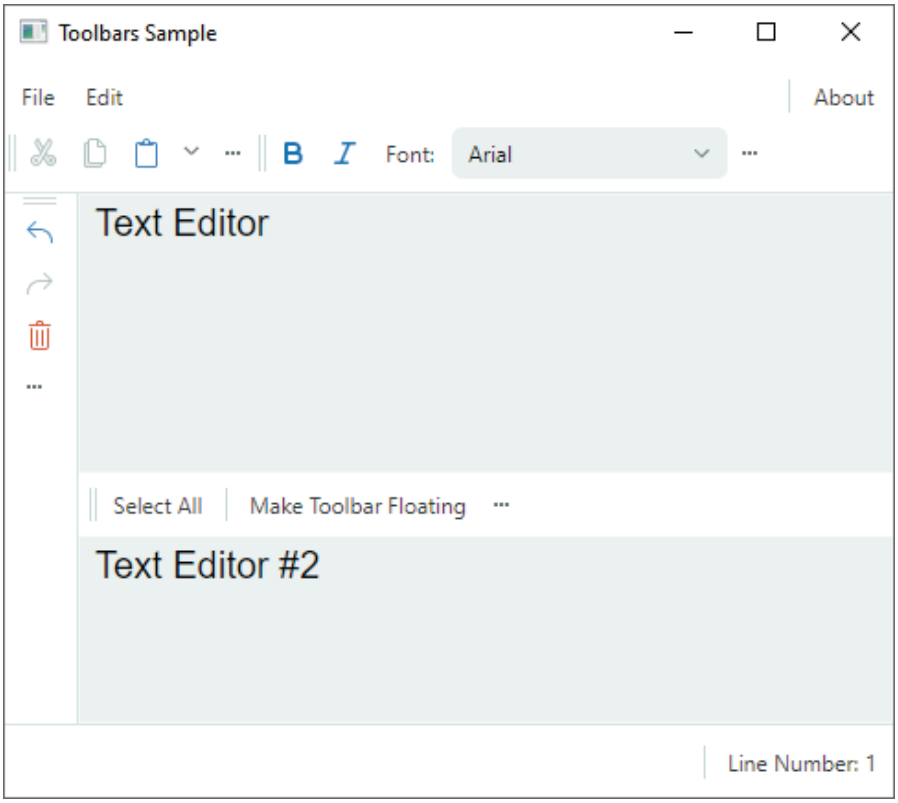
- Контекстные меню — Вы можете создавать контекстные меню для внешних контролов. Свойства стиля контекстных меню согласуются со всеми компонентами библиотеки Панели Инструментов и Меню.

Дополнительные сведения см. в следующих разделах:

- Начало работы с панелями инструментов
- Панели инструментов
- Контекстные меню
- Элементы панели инструментов

Начало работы с панелями инструментов

В этом руководстве показано, как использовать библиотеку панелей инструментов Eremex для создания пользовательского интерфейса панели инструментов с нуля. В нем представлены контролы для реализации пользовательского интерфейса панели инструментов и демонстрируются свойства основных панелей инструментов.



В руководстве создается пользовательский интерфейс панели инструментов для двух текстовых редакторов, размещенных в центре окна. Пользовательский интерфейс панели инструментов состоит из главного меню, панели состояния и обычных панелей инструментов, на которых отображаются различные элементы: кнопки, кнопки-переключатели, встроенные редакторы, подменю и текстовые элементы.

Все панели инструментов, кроме одной, пристыкованы к краям окна. На этих панелях инструментов есть команды, которые работают с первым текстовым редактором. Одна панель инструментов (автономная панель инструментов) размещена между текстовыми редакторами. Он предоставляет команды для второго текстового редактора.

В руководстве также показано, как привязать текстовый редактор к контекстному меню из Библиотеки Панелей Инструментов и Меню.

1. Создание компонента ToolbarManager

Начните с определения компонента `ToolbarManager` в XAML.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>

<mxb:ToolbarManager IsWindowManager="True">
    <Grid RowDefinitions="Auto, *, Auto, *, Auto" ColumnDefinitions="Auto, *, Auto">
        <TextBox Grid.Row="1" Grid.Column="1" x:Name="textBox1"  Text="Text Editor"
         AcceptsReturn="True"/>
        <TextBox x:Name="textBox2"  Grid.Row="3" Grid.Column="1" Text="Text Editor #2"
         AcceptsReturn="True"/>
    </Grid>
</mxb:ToolbarManager>
```

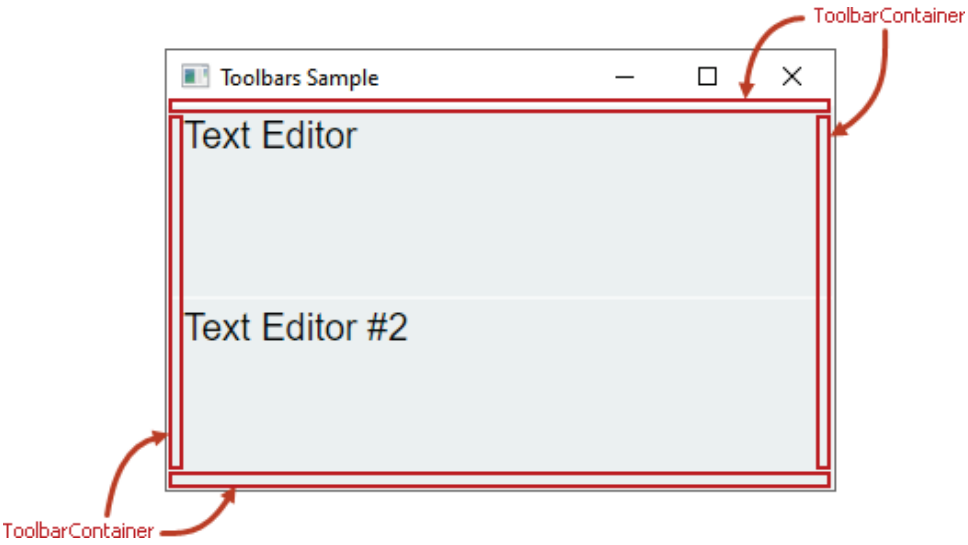
`ToolbarManager` - это основной компонент, который управляет панелями инструментов, контекстными меню и элементами меню. Компонент обрабатывает горячие клавиши, вызывает команды, связанные с соответствующими элементами, поддерживает настройку среды выполнения панели инструментов и выполняет сериализацию и десериализацию пользовательского интерфейса панели.

Компонент `ToolbarManager` должен обернуть клиентский контрол (контролы), для которого создан пользовательский интерфейс панели инструментов.

2. Создание контейнеров панелей инструментов

Чтобы разрешить пристыковывать панель инструментов к определенной позиции в окне/UserControl, сначала создайте контейнер панели инструментов (`ToolbarContainerControl`). Контейнер панели инструментов - это контрол, который отображает панели инструментов в пристыкованном состоянии и поддерживает операции перетаскивания панели инструментов.

В XAML создайте четыре контейнера панели инструментов (объекты `ToolbarContainerControl`) вдоль верхнего, нижнего, левого и правого краев окна. После этого вы сможете, чтобы пристыковать панели инструментов в этих позициях.

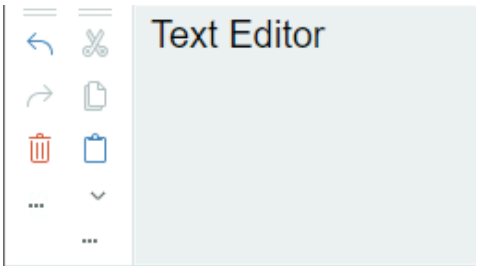



```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarManager IsWindowManager="True">
    <Grid RowDefinitions="Auto, *, Auto, *, Auto" ColumnDefinitions="Auto, *, Auto">
        <mxb:ToolbarContainerControl DockType="Top" Grid.ColumnSpan="3"/>

        <mxb:ToolbarContainerControl DockType="Left" Grid.Row="1" Grid.Column="0"
         Grid.RowSpan="3"/>

        <TextBox Grid.Row="1" Grid.Column="1" x:Name="textBox1"  Text="Text Editor"
         AcceptsReturn="True"/>
        <TextBox x:Name="textBox2"  Grid.Row="3" Grid.Column="1" Text="Text Editor #2"
         AcceptsReturn="True"/>

        <mxb:ToolbarContainerControl DockType="Right" Grid.Row="1" Grid.Column="2"
         Grid.RowSpan="3"/>

        <mxb:ToolbarContainerControl DockType="Bottom" Grid.Row="4" Grid.ColumnSpan="3"/>
    </Grid>
</mxb:ToolbarManager>
```

Опции контейнера панелей инструментов

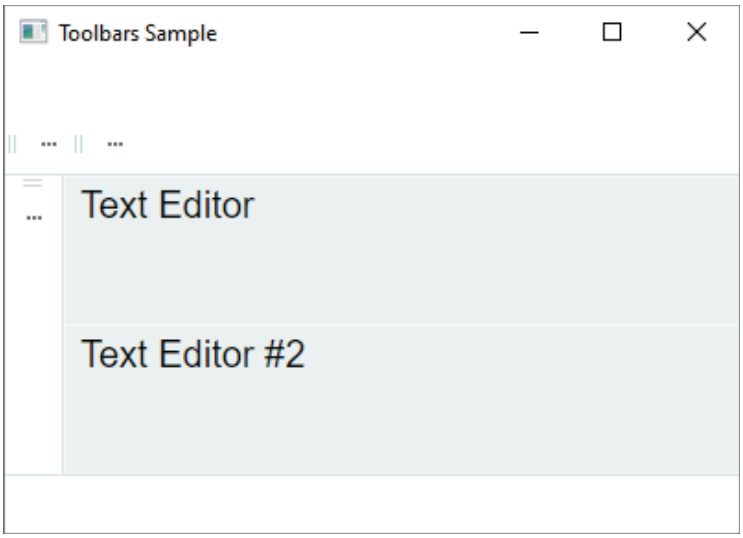
Основным свойством `ToolbarContainerControl` является `ToolbarContainerControl.DockType`, которое определяет, как контейнер пристыкован к своему родительскому элементу. Вы можете установить свойство `DockType` равным `Left`, `Right`, `Top`, `Bottom` и `Standalone`.

Свойство `DockType` определяет видимость границ контейнера и выравнивание вложенных панелей инструментов по умолчанию. Например, если опция контейнера `DockType` равна `Left`, контейнер рисует границу по своему правому краю и размещает вложенные панели инструментов вертикально. На картинке ниже показан контейнер панели инструментов, для свойства которого для опции `DockType` установлено значение `Left`. Дочерние панели инструментов ориентированы вертикально в соответствии с свойством `DockType`.



3. Создание панелей инструментов

Добавьте панели инструментов (объекты `Toolbar`) в необходимые контейнеры панелей инструментов.




```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarManager IsWindowManager="True">
    <Grid RowDefinitions="Auto, *, Auto, *, Auto"
     ColumnDefinitions="Auto, *, Auto">
        <mxb:ToolbarContainerControl DockType="Top" Grid.ColumnSpan="3">
            <mxb:Toolbar x:Name="MainMenu" ToolbarName="Main Menu"
             DisplayMode="MainMenu">
            </mxb:Toolbar>

            <mxb:Toolbar x:Name="EditToolbar" ToolbarName="Edit"
             ShowCustomizationButton="True">
            </mxb:Toolbar>

            <mxb:Toolbar x:Name="FontToolbar" ToolbarName="Font"
             ShowCustomizationButton="True">
            </mxb:Toolbar>
        </mxb:ToolbarContainerControl>

        <mxb:ToolbarContainerControl DockType="Left" Grid.Row="1" Grid.Column="0"
        Grid.RowSpan="3">
            <mxb:Toolbar x:Name="TextEditingToolbar" ToolbarName="Text Editing"
             ShowCustomizationButton="True" >
            </mxb:Toolbar>
```

```
        </mxb:ToolbarContainerControl>

        <TextBox Grid.Row="1" Grid.Column="1" x:Name="textBox1"  Text="Text Editor"
         AcceptsReturn="True" CornerRadius="0" FontFamily="Arial" FontSize="20"/>
        <TextBox x:Name="textBox2"  Grid.Row="3" Grid.Column="1" Text="Text Editor #2"
         AcceptsReturn="True" CornerRadius="0" FontFamily="Arial" FontSize="20"/>

        <mxb:ToolbarContainerControl DockType="Right" Grid.Row="1"
         Grid.Column="2" Grid.RowSpan="3"/>

        <mxb:ToolbarContainerControl DockType="Bottom"
         Grid.Row="4" Grid.ColumnSpan="3">
            <mxb:Toolbar DisplayMode="StatusBar" ToolbarName="Status Bar"
             x:Name="StatusBar">
            </mxb:Toolbar>
        </mxb:ToolbarContainerControl>
    </Grid>
</mxb:ToolbarManager>
```

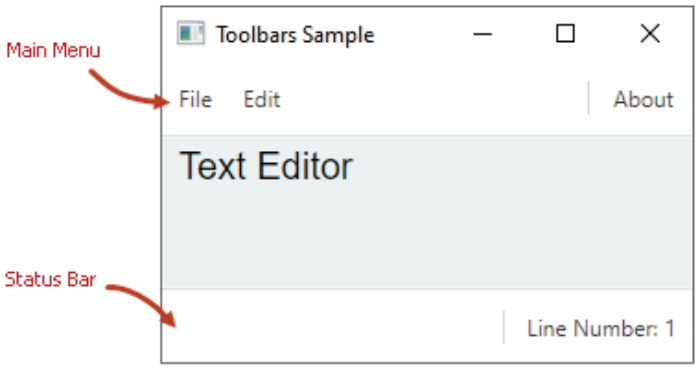
Приведенный выше фрагмент заполняет три контейнера панели инструментов панелями инструментов и оставляет один контейнер панели инструментов пустым. Пользователи смогут перетаскивать панели инструментов в любой из четырех контейнеров панелей инструментов во время выполнения программы.

Главное меню и панель состояния

Чтобы указать, что панель инструментов является главным меню или панелью состояния, установите для ее свойства `Toolbar.DisplayMode` значения `MainMenu` и `StatusBar` соответственно.

```
<mxb:Toolbar x:Name="MainMenu" ToolbarName="Main Menu" DisplayMode="MainMenu">
</mxb:Toolbar>
```

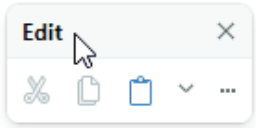
Главное меню и панель состояния имеют отличительные свойства внешнего вида и поведения. Например, они не содержат маркера перетаскивания, поэтому пользователи не могут их перетаскивать. Пользователь не может скрыть главное меню и панель состояния во время выполнения программы.



Опции панели инструментов

Объекты `Toolbar` предоставляют множество опций для настройки их вида, размещения и свойств поведения. Некоторые из этих опций включают:

- `ToolbarName` — Отображаемое имя панели инструментов. Названия панелей инструментов отображаются в окне настройки, а также когда панель инструментов находится в плавающем состоянии.



- `ShowCustomizationButton` — Определяет видимость кнопки настройки, используемой для активации режима настройки и открытия окна настройки.

- `AllowDragToolbar` — Определяет видимость маркера перетаскивания, который позволяет пользователям перетаскивать панель инструментов.

- `DockType` — Это свойство позволяет перемещать панель инструментов в определенный контейнер панели инструментов в коде или делать панель инструментов плавающей.
- `StretchToolbar` — Включает растягивание панели инструментов. В этом режиме никакая другая панель инструментов не может отображаться в той же строке.
- `WrapItems` — Включает многорядное размещение панели инструментов.

4. Создание элементов панели инструментов

Следующим шагом является заполнение панелей инструментов элементами панели инструментов: обычными кнопками, кнопками-переключателями, встроенными редакторами, подменю и текстовыми элементами. Элементы панели инструментов инкапсулируются классами, производными от класса `ToolbarItem`, который предоставляет общие опции элемента панели инструментов.

В этом руководстве создаются следующие элементы панели инструментов:

ToolbarButtonItem

Обычная кнопка, которая запускает команду, указанную свойством `Command`.

```
<mxb:Toolbar x:Name="EditToolbar" ToolbarName="Edit" ShowCustomizationButton="True"  >
    <mxb:ToolbarButtonItem Header="Cut" Command="{Binding #textBox1.Cut}"
     IsEnabled="{Binding #textBox1.CanCut}"
```

```
     Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCut.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Copy" Command="{Binding #textBox1.Copy}"
     IsEnabled="{Binding #textBox1.CanCopy}"
     Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCopy.svg'}"
     Category="Edit"/>
</mxb:Toolbar>
```

Общие опции элементов панели инструментов

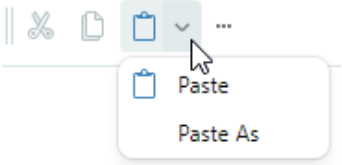
Базовый класс `ToolbarItem` предоставляет общие опции, наследуемые всеми элементами панели инструментов. Некоторые из этих опций включают:

- `Alignment` — Выравнивание элемента на панели инструментов.
- `Category` — категория, к которой принадлежит элемент. Категории используются для организации элементов в логические группы в окне настройки.
- `Command` — Команда, выполняемая при нажатии на кнопку.
- `CommandParameter` — Командный параметр, передаваемый указанной команде.
- `DisplayMode` — Определяет, отображать ли только картинку, заголовок или и то, и другое.
- `Header` — Текст для отображения элемента.
- `Glyph` — Картинка элемента.
- `GlyphAlignment` — Выравнивание картинки относительно заголовка элемента.
- `GlyphSize` — Размер отображаемой картинки.
- `ShowSeparator` — Позволяет отображать разделитель перед элементом.

Присвоение всплывающего контрола/меню элементу ToolbarButtonItem

Вы можете привязать контрол/меню всплывающего окна к объекту `ToolbarButtonItem`. Всплывающее окно активируется щелчком по стрелке встроенного всплывающего окна или по самому элементу (смотрите опцию `DropDownArrowVisibility` ниже для получения дополнительной информации).

Давайте привяжем кнопку *Paste* (`ToolbarButtonItem`) к всплывающему окну меню. В меню всплывающего окна отобразятся команды *Paste* и *Paste As*.




```
<mxb:ToolbarButtonItem Header="Paste" Command="{Binding #textBox1.Paste}"
 IsEnabled="{Binding #textBox1.CanPaste}"
 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditPaste.svg'}" Category="Edit">
    <mxb:ToolbarButtonItem.DropDownControl>
        <mxb:PopupMenu>
            <mxb:ToolbarButtonItem Header="Paste" Command="{Binding #textBox1.Paste}"
             IsEnabled="{Binding #textBox1.CanPaste}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditPaste.svg'}"/>
            <mxb:ToolbarButtonItem Header="Paste As" Command="{Binding PasteAsCommand}"
             IsEnabled="{Binding #textBox1.CanPaste}"/>
```

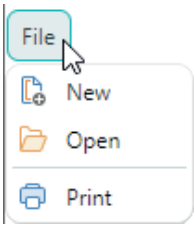
```
        </mxb:PopupMenu>
    </mxb:ToolbarButtonItem.DropDownControl>
</mxb:ToolbarButtonItem>
```

Следующие свойства используются для указания контрола всплывающего окна и способа его отображения:

- `DropDownControl` — Получает или устанавливает контрол/меню всплывающего окна, связанные с элементом. Это свойство принимает объекты `PopupContainer` и `PopupMenu`.
- `DropDownArrowVisibility` — Указывает, отображает ли элемент стрелку всплывающего окна, используемую для вызова связанного контрола всплывающего окна. Поддерживаемые опции включают:
    - `ShowArrow` — Видна стрелка всплывающего окна. Элемент и стрелка действуют как единая кнопка. Щелчок по ним отображает связанный контрол всплывающего окна.
    - `ShowSplitArrow` или `Default` — Видна стрелка всплывающего окна. Она действует как отдельная кнопка, встроенная в элемент. Щелчок по стрелке всплывающего окна вызывает связанный с ним контрол всплывающего окна. Щелчок по элементу вызывает его команду.
    - `Hide` — Стрелка всплывающего окна скрыта. Щелчок по элементу вызывает контрол всплывающего окна.

ToolbarMenuItem

Кнопка, вызывающая подменю. Чтобы добавить элементы в подменю, определите их между начальным и конечным тегами `<ToolbarMenuItem>` в разметке XAML или добавьте их в коллекцию `Items`.




```
<mxb:Toolbar x:Name="MainMenu" ToolbarName="Main Menu" DisplayMode="MainMenu">
    <mxb:ToolbarMenuItem Header="File" Category="File">
        <mxb:ToolbarButtonItem Header="New"
         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FileNew.svg'}"
         Category="File" Command="{Binding NewFileCommand}"/>
        <mxb:ToolbarButtonItem Header="Open"
         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FileOpen.svg'}"
         Category="File" Command="{Binding OpenFileCommand}"/>
        <mxb:ToolbarButtonItem Header="Print"
         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FilePrint.svg'}"
         Category="File" Command="{Binding PrintCommand}" ShowSeparator="True"/>
    </mxb:ToolbarMenuItem>
</mxb:Toolbar>
```

Вы можете добавить в подменю все поддерживаемые типы элементов.

ToolbarCheckItem

Кнопка-переключатель, которая может находиться как в обычном, так и в нажатом состоянии.

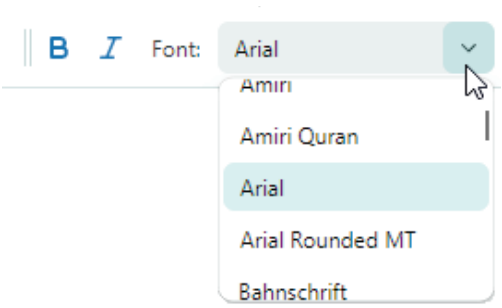
```
<mxb:Toolbar x:Name="FontToolbar" ToolbarName="Font" ShowCustomizationButton="False">
    <mxb:ToolbarCheckItem Header="Bold"
     IsChecked="{Binding #textBox1.FontWeight,
      Converter={local:BoolToFontWeightConverter}, Mode=TwoWay}"
     Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FontBold.svg'}"
     Category="Font"/>
    ...
</mxb:Toolbar>
```

Опции ToolbarCheckItem

- `IsChecked` — Возвращает или устанавливает состояние нажатия кнопки.
- `CheckedChanged` — Событие, которое срабатывает при изменении нажатого состояния.

ToolbarEditorItem

Элемент, позволяющий встроить редактор Eremex в панель инструментов или меню.




```
<mxb:Toolbar x:Name="FontToolbar" ToolbarName="Font" ShowCustomizationButton="False">
    <mxb:ToolbarEditorItem Header="Font:" EditorWidth="150" Category="Font"
     EditorValue="{Binding #textBox1.FontFamily,
      Converter={local:FontNameToFontFamilyConverter}}">
        <mxb:ToolbarEditorItem.EditorProperties>
            <mxe:ComboBoxEditorProperties
             ItemsSource="{Binding $parent[local:MainWindow].Fonts}"
             IsTextEditable="False"/>
        </mxb:ToolbarEditorItem.EditorProperties>
    </mxb:ToolbarEditorItem>
    ...
</mxb:Toolbar>
```

Опции ToolbarEditorItem

- `EditorValue` — Позволяет вам устанавливать и считывать значение встроенного редактора.
- `EditorProperties` — Указывает тип редактора, который должен быть встроен в панель инструментов/меню. В приведенном выше фрагменте кода свойству `EditorProperties` присвоено

свойство объекта `ComboBoxEditorProperties`. Этот объект содержит свойства, специфичные для контрола `ComboBoxEditor`. Панель инструментов автоматически создаст контрол `ComboBoxEditor` во время выполнения программы из указанного объекта `ComboBoxEditorProperties`.

ToolbarTextItem

Текстовая метка. Щелчок по текстовой метке не вызывает никакого действия (команды).


```
<mxb:Toolbar DisplayMode="StatusBar" ToolbarName="Status Bar" x:Name="StatusBar"
 ShowCustomizationButton="False">
    <mxb:ToolbarTextItem  Name="tbTextItem1" Alignment="Far" ShowSeparator="True"
     ShowBorder="False" Category="Info" CustomizationName="Position Info"
     Header="{Binding $parent[local:MainWindow].LineNumber}"/>
</mxb:Toolbar>
```

Опции ToolbarTextItem

- `SizeMode` — Получает или устанавливает, будет ли размер элемента автоматически подогнан под его содержимое или растянут, чтобы занять доступное пространство на панели инструментов.
- `ShowBorder` — Получает или устанавливает, следует ли отображать рамку вокруг элемента.

Другие типы элементов панели инструментов

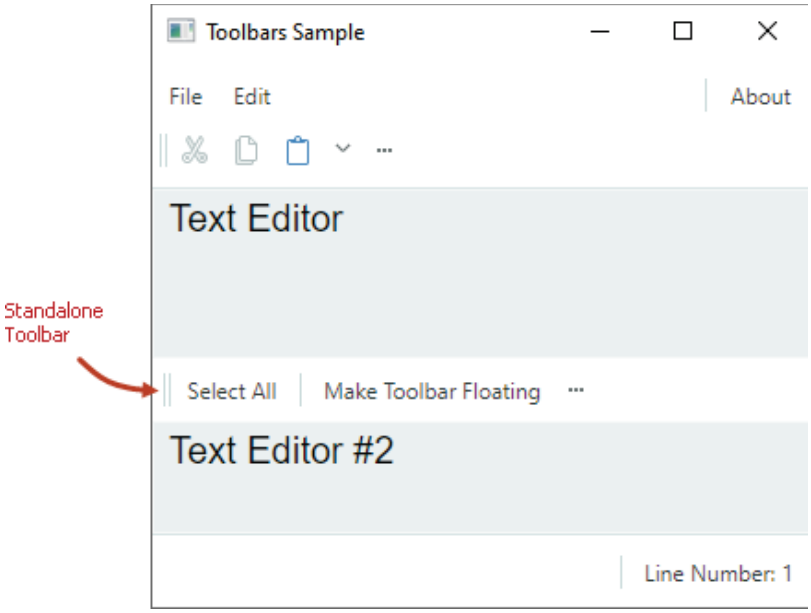
Библиотека панелей инструментов также поддерживает другие типы элементов панели инструментов, которые не продемонстрированы в этом руководстве:

- `ToolbarItemGroup` — Группа элементов панели инструментов.
- `ToolbarCheckItemGroup` — Группа переключающих кнопок. Используйте его для создания группы взаимоисключающих переключающих элементов или группы, которая поддерживает одновременный выбор нескольких элементов.

Смотрите следующий раздел для получения дополнительной информации: Элементы панели инструментов .

5. Создание автономной панели инструментов

Вы можете размещать панели инструментов в любой позиции внутри окна, а не только по его краям. Например, вы можете разместить панели инструментов с командами рядом с целевым контролом. Эти панели инструментов называются "автономными", поскольку они находятся в "автономных" контейнерах панелей инструментов.



Чтобы создать автономную панель инструментов, выполните следующие действия:

1. Создайте контейнер панели инструментов (`ToolbarContainerControl`) в требуемой позиции. Установите для его свойства `DockType` значение `Standalone` .

   Автономные контейнеры панели инструментов не имеют границ.

2. Добавьте панель инструментов с командами в этот контейнер панели инструментов.

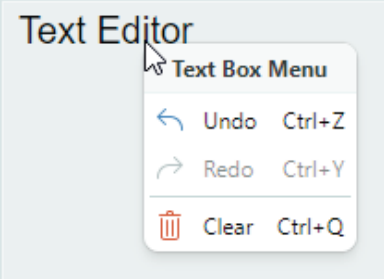
Приведенный ниже код отображает автономную панель инструментов между двумя текстовыми редакторами. Команда *Select All* на панели инструментов выделяет текст во втором текстовом редакторе.

```
<mxb:ToolbarManager Name="toolbarManager1" IsWindowManager="True" >
    <Grid RowDefinitions="Auto, *, Auto, *, Auto" ColumnDefinitions="Auto, *, Auto">
        ...
        <TextBox Grid.Row="1" Grid.Column="1" x:Name="textBox1"  Text="Text Editor"
         AcceptsReturn="True"/>

        <mxb:ToolbarContainerControl DockType="Standalone" Grid.Row="2" Grid.Column="1">
            <mxb:Toolbar x:Name="TextEditor2Toolbar" ToolbarName="Standalone Toolbar"
             ShowCustomizationButton="True" AllowDragToolbar="true"  >
                <mxb:ToolbarButtonItem Header="Select All"
                 Command="{Binding #textBox2.SelectAll}"
                 Category="TextEditor2 Toolbar"/>
                <mxb:ToolbarButtonItem Header="Make Toolbar Floating"
                 Command="{Binding $parent[local:MainWindow].MakeToolbar2Floating}"
                 ShowSeparator="True"
                 Category="TextEditor2 Toolbar"/>
            </mxb:Toolbar>
        </mxb:ToolbarContainerControl>

        <TextBox x:Name="textBox2"  Grid.Row="3" Grid.Column="1" Text="Text Editor #2"
         AcceptsReturn="True"/>
```

```
    </Grid>
</mxb:ToolbarManager>
```

6. Назначение контекстного меню текстовому редактору

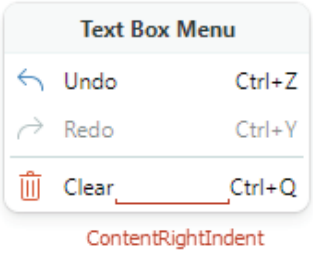
Чтобы указать контекстное меню для контрола, создайте объект `PopupMenu` и назначьте его целевому контролу, используя attached-свойство `ToolbarManager.ContextPopup`. Во всплывающие меню можно добавлять элементы панели инструментов любого типа.




```
<TextBox Grid.Row="1" Grid.Column="1" x:Name="textBox1"  Text="Text Editor"
 AcceptsReturn="True"
 CornerRadius="0" FontFamily="Arial" FontSize="20" >
    <mxb:ToolbarManager.ContextPopup>
        <mxb:PopupMenu ShowIconStrip="True" Header="Text Box Menu" ShowHeader="True">
            <mxb:ToolbarButtonItem Header="Undo" HotKeyDisplayString="Ctrl+Z"
             Command="{Binding #textBox1.Undo}" IsEnabled="{Binding #textBox1.CanUndo}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditUndo.svg'}"
             Category="Edit"/>
            <mxb:ToolbarButtonItem Header="Redo" HotKeyDisplayString="Ctrl+Y"
             Command="{Binding #textBox1.Redo}" IsEnabled="{Binding #textBox1.CanRedo}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditRedo.svg'}"  C
             ategory="Edit"/>
            <mxb:ToolbarSeparatorItem/>
            <mxb:ToolbarButtonItem Header="Clear" Command="{Binding #textBox1.Clear}"
             HotKeyDisplayString="Ctrl+Q"  Category="Edit"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditDelete.svg'}"/>
        </mxb:PopupMenu>
    </mxb:ToolbarManager.ContextPopup>
</TextBox>
```

Опции всплывающего меню

- `ContentRightIndent` — Задает ширину пустого пространства справа от текста элементов меню.



- `Header` — Позволяет указать заголовок меню.
- `ShowHeader` — Возвращает или устанавливает, виден ли заголовок меню.
- `ShowIconStrip` — Получает или устанавливает, следует ли отображать вертикальную полосу значков для элементов меню. Значок элемента меню определяется свойством `Glyph` элемента.

7. Назначение горячих клавиш для элементов панели инструментов

Используйте свойство `ToolbarItem.HotKey`, чтобы назначить элементам горячие клавиши.

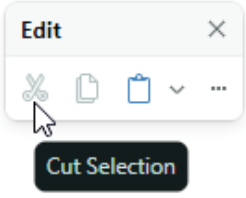
```
<mxb:ToolbarButtonItem
    Header="Clear" Command="{Binding #textBox1.Clear}" HotKey="Ctrl+Q"
    Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditDelete.svg'}"  Category="Edit"/>
```

Границы компонента `ToolbarManager` определяют дефолтная область действия горячих клавиш. Если фокус находится в пределах области действия горячих клавиш, `ToolbarManager` может перехватывать и обрабатывать горячие клавиши. Вы можете установить для свойства `ToolbarManager.IsWindowManager` значение `true`, чтобы расширить область действия горячих клавиш на все окно. В этом случае `ToolbarManager` регистрирует горячие клавиши в окне и может обрабатывать горячие клавиши, даже если фокус находится за пределами `ToolbarManager`.

Смотрите следующий раздел для получения дополнительной информации: Горячие клавиши элемента панели инструментов .

8. Назначение всплывающих подсказок элементам панели инструментов

Свойство `ToolTip` позволяет вам указывать всплывающие подсказки для элементов панели инструментов.



```
<mxb:ToolbarButtonItem Header="Cut" Command="{Binding #textBox1.Cut}"
 IsEnabled="{Binding #textBox1.CanCut}"
 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCut.svg'}"
 Category="Edit"
 ToolTip.Tip="Cut Selection"/>
```

9. Плавающая панель инструментов

Давайте создадим панель инструментов, плавающую в коде. Убедитесь, что у целевой панели инструментов есть имя, чтобы вы могли получить к ней доступ. После того как вы получите объект панели инструментов, установите для его свойства `Toolbar.DockType` значение `Floating`. Используйте свойство `Toolbar.FloatingPosition`, чтобы задать местоположение плавающей панели инструментов.

```
EditToolbar.DockType = Eremex.AvaloniaUI.Controls.Bars.MxToolbarDockType.Floating;
EditToolbar.FloatingPosition = new PixelPoint(200, 200);
```

Чтобы создать плавающую панель инструментов в XAML, определите объект `Toolbar` в коллекции `ToolbarManager.Toolbars` и задайте свойству `Toolbar.DockType` значение `Floating`.

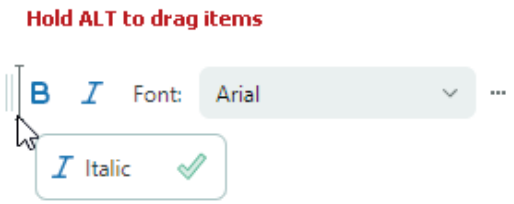
Смотрите следующий раздел для получения дополнительной информации: Плавающие панели инструментов .

10. Время выполнения

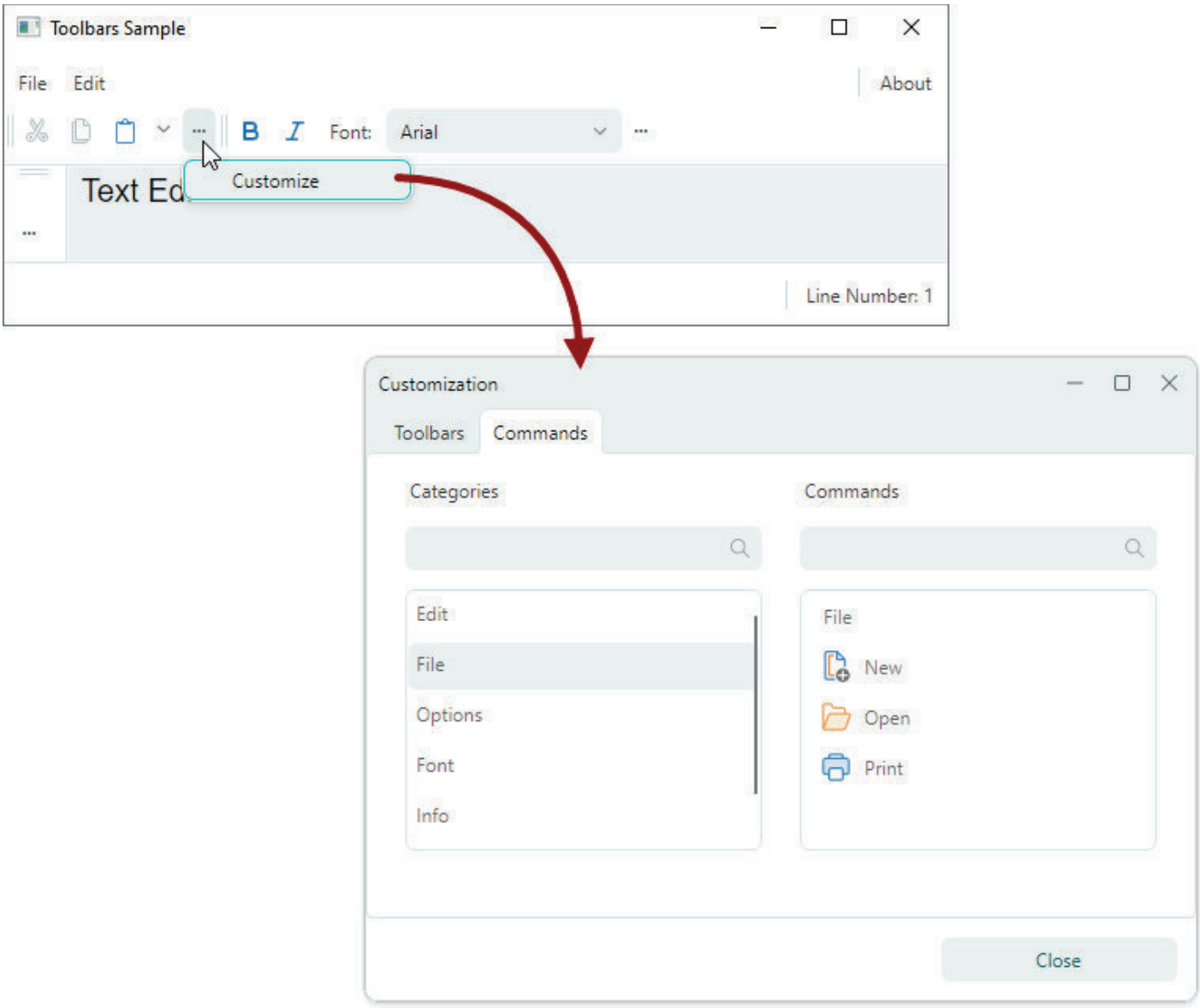
Библиотека панелей инструментов поддерживает настройку панели инструментов пользователями во время выполнения программы. Запустите приложение, чтобы увидеть эти функции в действии:

- Перетаскивание панелей - Панели инструментов отображают маркеры перетаскивания, которые позволяют изменять порядок расположения панелей.

- Быстрая настройка панели инструментов — вы можете быстро перемещать элементы внутри панелей и между ними с помощью перетаскивания, удерживая нажатой клавишу Alt.



- Режим настройки и окно настройки — нажмите кнопку настройки на панели инструментов ("..."), а затем выберите команду "Настроить". При активации режима настройки отображается окно настройки:



В режиме настройки вы можете выполнить следующее:

- Скрывайте и восстанавливайте панели инструментов.
- Создавайте пользовательские панели инструментов и управляйте ими.
- Скрывайте, восстанавливайте и переставляйте элементы панели инструментов между панелями и подменю.

11. Полный код

Ниже вы можете найти полный код этого руководства.

Картинки в формате SVG, используемые в этом примере, помещены в папку `bars_sample/Images/Toolbars`. У них для свойства `Build Action` установлено значение `AvaloniaResource`.

MainWindow.axaml:

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
                xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
        xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

        xmlns:mxcom="clr-namespace:Eremex.AvaloniaUI.Controls.Common;assembly=Eremex.Avalonia.Controls"
        xmlns:col="using:System.Collections"
                xmlns:sys="clr-namespace:System;assembly=mscorlib"
                xmlns:local="clr-namespace:BarsSample"
        mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
        x:Class="BarsSample.MainWindow"
        Title="Toolbars Sample"
        Width="600" Height="400"
        >
    <Window.DataContext>
        <local:MainViewModel/>
    </Window.DataContext>

    <mxb:ToolbarManager Name="toolbarManager1" IsWindowManager="True" >
        <Grid RowDefinitions="Auto, *, Auto, *, Auto" ColumnDefinitions="Auto, *, Auto">
            <mxb:ToolbarContainerControl DockType="Top" Grid.ColumnSpan="3">
                <mxb:Toolbar x:Name="MainMenu" ToolbarName="Main Menu" DisplayMode="MainMenu" >
                    <mxb:ToolbarMenuItem Header="File" Category="File">
                        <mxb:ToolbarButtonItem Header="New"
                         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FileNew.svg'}"
                         Category="File"
                         Command="{Binding NewFileCommand}"/>
                        <mxb:ToolbarButtonItem Header="Open"
                         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FileOpen.svg'}"
                         Category="File"
                         Command="{Binding OpenFileCommand}"/>
                        <mxb:ToolbarButtonItem Header="Print"
                         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FilePrint.svg'}"
                         Category="File"
                         Command="{Binding PrintCommand}" ShowSeparator="True"/>
                    </mxb:ToolbarMenuItem>

                    <mxb:ToolbarMenuItem Header="Edit" Category="Edit" >
                        <mxb:ToolbarButtonItem Header="Cut"
                         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCut.svg'}"
                         Category="Edit"
                         Command="{Binding #textBox1.Cut}" IsEnabled="{Binding #textBox1.CanCut}"/>
                        <mxb:ToolbarButtonItem Header="Copy"
                         Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCopy.svg'}"
                         Category="Edit"
                         Command="{Binding #textBox1.Copy}" IsEnabled="{Binding #textBox1.CanCopy}"/>
```

```
                <mxb:ToolbarButtonItem Header="Paste"
                 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditPaste.svg'}"
                 Category="Edit"
                 Command="{Binding #textBox1.Paste}" IsEnabled="{Binding #textBox1.CanPaste}"/>
            </mxb:ToolbarMenuItem>
            <mxb:ToolbarButtonItem Header="About" Category="Options" ShowSeparator="True"
             Alignment="Far" Command="{Binding AboutCommand}"/>
        </mxb:Toolbar>

        <mxb:Toolbar x:Name="EditToolbar" ToolbarName="Edit" ShowCustomizationButton="True"  >
            <mxb:ToolbarButtonItem Header="Cut" Command="{Binding #textBox1.Cut}"
             IsEnabled="{Binding #textBox1.CanCut}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCut.svg'}"
             Category="Edit"/>
            <mxb:ToolbarButtonItem Header="Copy" Command="{Binding #textBox1.Copy}"
             IsEnabled="{Binding #textBox1.CanCopy}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCopy.svg'}"
             Category="Edit"/>
            <mxb:ToolbarButtonItem Header="Paste"
             Command="{Binding #textBox1.Paste}"
             IsEnabled="{Binding #textBox1.CanPaste}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditPaste.svg'}"
             Category="Edit"
             DropDownArrowVisibility="ShowArrow" DropDownArrowAlignment="Default">
                <mxb:ToolbarButtonItem.DropDownControl>
                    <mxb:PopupMenu>
                        <mxb:ToolbarButtonItem Header="Paste" Command="{Binding #textBox1.Paste}"
                         IsEnabled="{Binding #textBox1.CanPaste}"/>
                        <mxb:ToolbarButtonItem Header="Paste As"
                         Command="{Binding PasteAsCommand}"
                         IsEnabled="{Binding #textBox1.CanPaste}"/>
                    </mxb:PopupMenu>
                </mxb:ToolbarButtonItem.DropDownControl>
            </mxb:ToolbarButtonItem>
        </mxb:Toolbar>

        <mxb:Toolbar x:Name="FontToolbar" ToolbarName="Font" ShowCustomizationButton="True" >
            <mxb:ToolbarCheckItem Header="Bold"
             IsChecked="{Binding #textBox1.FontWeight,
              Converter={local:BoolToFontWeightConverter}, Mode=TwoWay}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FontBold.svg'}"
             Category="Font"/>
            <mxb:ToolbarCheckItem Header="Italic"
             IsChecked="{Binding #textBox1.FontStyle,
              Converter={local:BoolToFontStyleConverter}, Mode=TwoWay}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FontItalic.svg'}"
             Category="Font"/>
            <mxb:ToolbarEditorItem Header="Font:" IsVisible="" EditorWidth="150"
            Category="Font"
             EditorValue="{Binding #textBox1.FontFamily,
              Converter={local:FontNameToFontFamilyConverter}}">
                <mxb:ToolbarEditorItem.EditorProperties>
                    <mxe:ComboBoxEditorProperties
                     ItemsSource="{Binding $parent[local:MainWindow].Fonts}"
                     IsTextEditable="False" PopupMaxHeight="145"/>
                </mxb:ToolbarEditorItem.EditorProperties>
```

```
            </mxb:ToolbarEditorItem>
        </mxb:Toolbar>
    </mxb:ToolbarContainerControl>

    <mxb:ToolbarContainerControl DockType="Left" Grid.Row="1" Grid.Column="0"
     Grid.RowSpan="3">
        <mxb:Toolbar x:Name="TextEditingToolbar" ToolbarName="Text Editing"
         ShowCustomizationButton="True" >
            <mxb:ToolbarButtonItem Header="Undo" HotKeyDisplayString="Ctrl+Z"
             Command="{Binding #textBox1.Undo}" IsEnabled="{Binding #textBox1.CanUndo}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditUndo.svg'}"
             Category="Edit"/>
            <mxb:ToolbarButtonItem Header="Redo" HotKeyDisplayString="Ctrl+Y"
             Command="{Binding #textBox1.Redo}" IsEnabled="{Binding #textBox1.CanRedo}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditRedo.svg'}"
             Category="Edit"/>
            <mxb:ToolbarButtonItem Header="Clear" Command="{Binding #textBox1.Clear}"
            HotKey="Ctrl+Q"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditDelete.svg'}"
             Category="Edit"/>
        </mxb:Toolbar>
    </mxb:ToolbarContainerControl>

    <TextBox Grid.Row="1" Grid.Column="1" x:Name="textBox1"  Text="Text Editor"
     AcceptsReturn="True" CornerRadius="0" FontFamily="Arial" FontSize="20" >
        <mxb:ToolbarManager.ContextPopup>
            <mxb:PopupMenu ShowIconStrip="True" Header="Text Box Menu" ShowHeader="True">
                <mxb:ToolbarButtonItem Header="Undo" HotKeyDisplayString="Ctrl+Z"
                 Command="{Binding #textBox1.Undo}" IsEnabled="{Binding #textBox1.CanUndo}"
                 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditUndo.svg'}"
                 Category="Edit"/>
                <mxb:ToolbarButtonItem Header="Redo" HotKeyDisplayString="Ctrl+Y"
                 Command="{Binding #textBox1.Redo}" IsEnabled="{Binding #textBox1.CanRedo}"
                 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditRedo.svg'}"
                 Category="Edit"/>
                <mxb:ToolbarSeparatorItem/>
                <mxb:ToolbarButtonItem Header="Clear"
                 Command="{Binding #textBox1.Clear}" HotKeyDisplayString="Ctrl+Q"
                 Category="Edit"
                 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditDelete.svg'}"/>
            </mxb:PopupMenu>
        </mxb:ToolbarManager.ContextPopup>
    </TextBox>

    <mxb:ToolbarContainerControl DockType="Standalone" Grid.Row="2" Grid.Column="1">
        <mxb:Toolbar x:Name="TextEditor2Toolbar" ToolbarName="Standalone Toolbar"
         ShowCustomizationButton="True" AllowDragToolbar="true"  >
            <mxb:ToolbarButtonItem Header="Select All"
             Command="{Binding #textBox2.SelectAll}"
             Category="TextEditor2 Toolbar"/>
            <mxb:ToolbarButtonItem Header="Make Toolbar Floating"
             Command="{Binding $parent[local:MainWindow].MakeToolbar2Floating}"
             ShowSeparator="True"
                                    Category="TextEditor2 Toolbar"/>
        </mxb:Toolbar>
    </mxb:ToolbarContainerControl>
```

```
            <TextBox x:Name="textBox2"  Grid.Row="3" Grid.Column="1" Text="Text Editor #2"
             AcceptsReturn="True" CornerRadius="0" FontFamily="Arial" FontSize="20"/>

            <mxb:ToolbarContainerControl DockType="Right" Grid.Row="1" Grid.Column="2"
             Grid.RowSpan="3"/>

            <mxb:ToolbarContainerControl DockType="Bottom" Grid.Row="4" Grid.ColumnSpan="3">
                <mxb:Toolbar DisplayMode="StatusBar" ToolbarName="Status Bar" x:Name="StatusBar">
                    <mxb:ToolbarTextItem  Name="tbTextItem1" Alignment="Far" ShowSeparator="True"
                     ShowBorder="False" Category="Info" CustomizationName="Position Info"
                     Header="{Binding $parent[local:MainWindow].LineNumber}"/>

                </mxb:Toolbar>
            </mxb:ToolbarContainerControl>
        </Grid>
    </mxb:ToolbarManager>
</Window>
```

App.axaml:

```
<Application xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                         RequestedThemeVariant="Light"
             x:Class="BarsSample.App">
        <Application.Resources>
                <ResourceDictionary>
                </ResourceDictionary>
        </Application.Resources>
    <Application.Styles>
        <FluentTheme/>
                <StyleInclude Source="avares://Eremex.Avalonia.Controls/Themes/Light/Theme.axaml"/>
                <StyleInclude Source="avares://Eremex.Avalonia.Controls/Themes/Generic.axaml"/>
    </Application.Styles>
</Application>
```

MainWindow.axaml.cs:

```
using Avalonia;
using Avalonia.Controls;
using Avalonia.Media;
using CommunityToolkit.Mvvm.Input;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Runtime.CompilerServices;

namespace BarsSample
{
    public partial class MainWindow : Window, INotifyPropertyChanged
    {
```

```
        public MainWindow()
        {
            InitializeComponent();
            textBox1.PropertyChanged += TextBox_PropertyChanged;
        }

        private void TextBox_PropertyChanged(object? sender, AvaloniaPropertyChangedEventArgs e)
        {
            if (e.Property == TextBox.CaretIndexProperty)
            {
                NotifyPropertyChanged("LineNumber");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged([CallerMemberName] String propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }

        IReadOnlyList<string> fonts;
        public IReadOnlyList<string> Fonts => fonts ??
            (fonts = FontManager.Current.SystemFonts.Select(x => x.Name).OrderBy(x => x).ToList());

        [RelayCommand]
        public void MakeToolbar2Floating()
        {
            TextEditor2Toolbar.DockType = Eremex.AvaloniaUI.Controls.Bars.MxToolbarDockType.Floating;
            TextEditor2Toolbar.FloatingPosition = new PixelPoint(200, 200);
        }

        public string LineNumber
        {
            get
            {
                TextBox textBox = this.textBox1;
                string text = textBox.Text;
                string newLine = textBox.NewLine;

                int currentIndex = 0;
                int lineNumber = 0;
                while (currentIndex <= textBox.CaretIndex)
                {
                    lineNumber++;
                    int newLineIndex = text.IndexOf(newLine, currentIndex);
                    if (newLineIndex >= 0)
                        currentIndex = newLineIndex + newLine.Length;
                    else
                        break;
                }
                return "Line Number: " + lineNumber;
            }
        }
    }
```

```
}
```

MainViewModel.cs:

```
using Avalonia.Data.Converters;
using Avalonia.Markup.Xaml;
using Avalonia.Media;
using CommunityToolkit.Mvvm.Input;
using Eremex.AvaloniaUI.Controls.Common;
using System;
using System.Globalization;

namespace BarsSample
{
    public partial class MainViewModel : ViewModelBase
    {
        public MainViewModel()
        {

        }

        [RelayCommand]
        void About()
        {

        }

        [RelayCommand]
        void NewFile()
        {

        }

        [RelayCommand]
        void OpenFile()
        {

        }

        [RelayCommand]
        void Print()
        {

        }
    }

    public class BoolToFontWeightConverter : MarkupExtension, IValueConverter
    {
        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            return this;
        }

        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
```

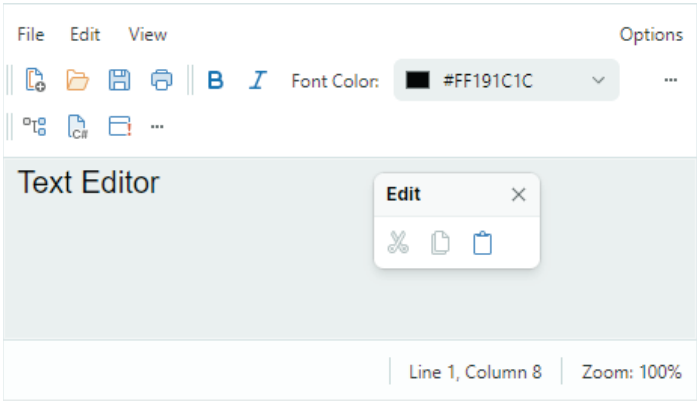
```
        {
            return ((FontWeight)value) == FontWeight.Bold;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return (bool)value ? FontWeight.Bold : FontWeight.Normal;
        }
    }

    public class BoolToFontStyleConverter : MarkupExtension, IValueConverter
    {
        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            return this;
        }

        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return (FontStyle)value == FontStyle.Italic;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return (bool)value ? FontStyle.Italic : FontStyle.Normal;
        }
    }

    public class FontNameToFontFamilyConverter : MarkupExtension, IValueConverter
    {
        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            return this;
        }

        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return ((FontFamily)value).Name;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return new FontFamily((string)value);
        }
    }
}
```

Смотрите также

- Панели инструментов
- Контекстные меню
- Элементы панели инструментов

Панели инструментов

Компонент `ToolbarManager` позволяет вам создать классический пользовательский интерфейс панели инструментов и Контекстные меню в вашем приложении. Вы можете пристыковывать панели инструментов к краям окна/UserControl и создавать плавающие панели инструментов. Библиотека панелей инструментов Eremex также позволяет размещать панели инструментов в любом месте вашего окна/UserControl.



Когда вы используете компонент `ToolbarManager`, вам необходимо создать контейнеры панели инструментов чтобы пристыковать панели инструментов. Контейнер панели инструментов - это визуальный контейнер, используемый для отображения пристыкованных панелей инструментов. Контейнеры панели инструментов являются дочерними элементами компонента `ToolbarManager`.

Создание панелей инструментов

Контрол `Toolbar` реализует панель инструментов. Используйте его, чтобы добавить обычную панель, главное меню и панель состояния в ваше приложение.

Выполните следующие действия, чтобы создать панели инструментов:

- Создайте компонент `ToolbarManager`.
- Создайте контейнеры панели инструментов.
- Создавайте панели инструментов.
- Заполните панели инструментов элементами.

Создание ToolbarManager

Чтобы создать панели инструментов, сначала определите компонент `ToolbarManager` (потомок класса `Avalonia.Controls.Border`). Клиентский контрол, для которого создается пользовательский интерфейс панели инструментов, должен быть размещен внутри компонента `ToolbarManager`. Объект `ToolbarManager` определяет область, в которой функционируют панели инструментов и всплывающие меню.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarManager IsWindowManager="True">
    <Grid RowDefinitions="Auto, *, Auto" ColumnDefinitions="Auto, *, Auto">
        <TextBox x:Name="textBox" Text="Text Editor" AcceptsReturn="True"
         CornerRadius="0" FontFamily="Arial" FontSize="20"
         helpers:TextBoxHelper.IsEnabled="True"/>
```

```
    </Grid>
</mxb:ToolbarManager>
```

> **ПОДСКАЗКА**
>
> Вы можете назначить горячие клавиши элементам панели инструментов. Чтобы разрешить объекту `ToolbarManager` обрабатывать эти горячие клавиши, когда фокус клавиатуры находится за пределами клиентской области `ToolbarManager`, включите опцию `ToolbarManager.IsWindowManager`. Смотрите следующий раздел для получения информации о области действия горячих клавиш элементов панели инструментов: Горячие клавиши .

Создание контейнеров панелей инструментов

Добавьте контейнеры панели инструментов (`ToolbarContainerControl`) к компоненту `ToolbarManager`, чтобы панели инструментов можно было пристыковывать к определенной позиции в окне/UserControl. Контейнер панели инструментов определяет область, в которой панели инструментов можно пристыковывать и перетаскивать.

В следующем примере создаются четыре контейнера панели инструментов по краям контрола Grid, что позволяет позже пристыковать панели инструментов в этих позициях.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarManager IsWindowManager="True">
    <Grid RowDefinitions="Auto, *, Auto" ColumnDefinitions="Auto, *, Auto">
        <mxb:ToolbarContainerControl Grid.ColumnSpan="3" DockType="Top">
        </mxb:ToolbarContainerControl>

        <mxb:ToolbarContainerControl DockType="Left" Grid.Row="1"/>

        <TextBox x:Name="textBox" Text="Text Editor" AcceptsReturn="True"
         CornerRadius="0" FontFamily="Arial" FontSize="20"
         helpers:TextBoxHelper.IsEnabled="True"/>

        <mxb:ToolbarContainerControl DockType="Right" Grid.Row="1"
         Grid.Column="2"/>

        <mxb:ToolbarContainerControl Grid.Row="2" Grid.ColumnSpan="3"
         DockType="Bottom">
        </mxb:ToolbarContainerControl>
    </Grid>
</mxb:ToolbarManager>
```

Инициализируйте свойство `ToolbarContainerControl.DockType` для созданных контейнеров, чтобы указать, как они будут пристыкованы. Тип стыковки контейнера определяет выравнивание вложенных панелей инструментов по умолчанию и свойства внешнего вида, применяемые к контейнеру панели инструментов. Доступные типы стыковок включают:

- `Left`, `Right`, `Top`, `Bottom` — Контейнер панели инструментов пристыкован к соответствующей стороне родительского контрола. Контейнеры, имеющие эти типы стыковок, рисуют пограничные линии, отделяющие их от остальной клиентской области. Например, контейнер панели инструментов, пристыкованный к верхней части, рисует линию границы внизу.
- `Standalone` — Этот режим следует применять к автономным контейнерам панели инструментов . Автономный контейнер панели инструментов предназначен для отображения панелей инструментов в пользовательской позиции в окне/UserControl. Автономные контейнеры панели инструментов не имеют границ.

Вы можете определить пустой `ToolbarContainerControl`, чтобы разрешить добавление панели инструментов в этот контейнер позже (пользователем или в коде).

Добавление панелей инструментов

Добавьте контролы `Toolbar` в контейнеры панелей инструментов, чтобы пристыковать панели инструментов в соответствующих позициях.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarContainerControl Grid.ColumnSpan="3" DockType="Top">
    <mxb:Toolbar x:Name="MainMenu" ToolbarName="Main Menu"
     DisplayMode="MainMenu">
    </mxb:Toolbar>

    <mxb:Toolbar x:Name="EditToolbar" ToolbarName="Edit"
     ShowCustomizationButton="False">
    </mxb:Toolbar>
</mxb:ToolbarContainerControl>

<mxb:ToolbarContainerControl Grid.Row="2" Grid.ColumnSpan="3"
 DockType="Bottom">
    <mxb:Toolbar DisplayMode="StatusBar" ToolbarName="Status Bar"
     x:Name="StatusBar">
    </mxb:Toolbar>
</mxb:ToolbarContainerControl>
```

Дополнительную информацию смотрите в следующих разделах:

- Плавающие панели инструментов
- Автономные панели инструментов

Добавление элементов панелей инструментов

Панели инструментов могут отображать различные типы элементов панели инструментов, продемонстрированные ниже:

- `ToolbarButtonItem` — элемент, который может действовать как обычная кнопка или кнопка всплывающего окна (если вы привязали к нему контрол/меню всплывающего окна).
- `ToolbarCheckItem` — Переключающая кнопка.
- `ToolbarMenuItem` — элемент, отображающий подменю.
- `ToolbarEditorItem` — элемент, отображающий встроенный редактор.
- `ToolbarTextItem` — Текстовая метка.
- `ToolbarItemGroup` — Группа элементов панели инструментов.
- `ToolbarCheckItemGroup` — Группа переключающих кнопок. Используйте его для создания группы взаимоисключающих переключающих элементов или группы, которая поддерживает одновременный выбор нескольких элементов.
- `ToolbarSeparatorItem` — Рисует разделитель между соседними элементами.

Чтобы добавить элементы панели инструментов на панель инструментов, определите их как дочерние элементы контрола `Toolbar` в XAML или добавьте их в коллекцию `Toolbar.Items` в коде.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:Toolbar x:Name="EditToolbar" ToolbarName="Edit" ShowCustomizationButton="False">
    <mxb:ToolbarButtonItem Header="Cut" Command="{Binding #textBox.Cut}"
     IsEnabled="{Binding #textBox.CanCut}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Cut.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Copy" Command="{Binding #textBox.Copy}"
     IsEnabled="{Binding #textBox.CanCopy}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Copy.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Paste" Command="{Binding #textBox.Paste}"
     IsEnabled="{Binding #textBox.CanPaste}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Paste.svg'}"
     Category="Edit"/>
</mxb:Toolbar>
```

Смотрите раздел Элементы панели инструментов для получения дополнительной информации.

Выбор типа панели инструментов - Обычная панель, Главное меню или Панель состояния

Чтобы указать тип панели инструментов (обычная панель, главное меню или панель состояния), используйте свойство `Toolbar.DisplayMode`. Этому свойству можно присвоить одно из следующих значений:

- `ToolbarDisplayMode.Default` — Объект `Toolbar` отображается как обычная панель инструментов.
- `ToolbarDisplayMode.MainMenu` — Объект `Toolbar` отображается в качестве главного меню. Функции главного меню включают:
    - Пользователь не может скрыть его во время выполнения программы.
    - Главное меню получает фокусировку, когда пользователь нажимает клавишу ALT.
    - Главное меню растягивается по горизонтали, подгоняя его под ширину контейнера.
    - Главное меню не поддерживает расположение элементов в несколько линий, плавающий режим и операции перетаскивания.
- `ToolbarDisplayMode.StatusBar` — Объект `Toolbar` отображается в панели состояния приложения. Функции панели состояния включают в себя:
    - Пользователь не может скрыть его во время выполнения программы.
    - Панель состояния растягивается по горизонтали подогнано под ширину контейнера.
    - Панель состояния не поддерживает расположение элементов в несколько линий, плавающий режим и операции перетаскивания.

В следующем примере создается панель состояния с двумя элементами. Панель состояния размещена в контейнере панели инструментов, отображаемом в нижней части родительского контрола Grid для расширенных таблиц.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarContainerControl Grid.Row="2" Grid.ColumnSpan="3" DockType="Bottom">
    <mxb:Toolbar DisplayMode="StatusBar" ToolbarName="Status Bar" x:Name="StatusBar"
     ShowCustomizationButton="False">
        <mxb:ToolbarTextItem Alignment="Far" ShowSeparator="True" ShowBorder="False"
         Category="Info" CustomizationName="Position Info">
            <mxb:ToolbarTextItem.Header>
                <MultiBinding Converter="{helpers:LineAndColumnToTextConverter}">
                    <Binding ElementName="textBox" Path="(helpers:TextBoxHelper.Line)"/>
                    <Binding ElementName="textBox" Path="(helpers:TextBoxHelper.Column)"/>
                </MultiBinding>
            </mxb:ToolbarTextItem.Header>
        </mxb:ToolbarTextItem>
        <mxb:ToolbarTextItem Header="{Binding #scaleDecorator.Scale, StringFormat={}Zoom: {0:P0}}"
         ShowBorder="False" Alignment="Far" ShowSeparator="True" Category="Info"
         CustomizationName="Zoom Info"/>
    </mxb:Toolbar>
</mxb:ToolbarContainerControl>
```

Размещение и позиция панелей инструментов

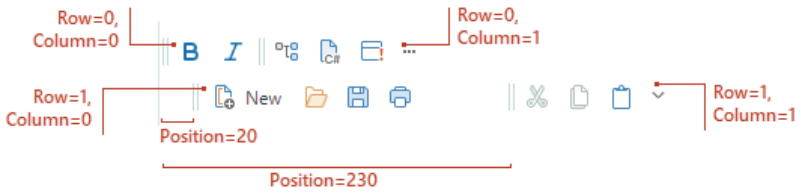
Когда панели инструментов находятся в контейнерах панелей инструментов, они выравниваются в соответствии с типом стыковки контейнера (свойство `ToolbarContainerControl.DockType`).:

- Панели инструментов ориентированы горизонтально, если свойству `ToolbarContainerControl.DockType` присвоено значение `Top`, `Bottom` или `Standalone`.
- Панели инструментов ориентированы вертикально, если свойству `ToolbarContainerControl.DockType` присвоено значение `Left` или `Right`.

Объект `Toolbar` имеет свой собственный параметр типа стыковки, указанный свойством `Toolbar.DockType`. Вы можете использовать свойство `Toolbar.DockType` для перемещения панели инструментов в определенный контейнер панели инструментов в коде. Когда вы устанавливаете свойство `Toolbar.DockType` в `Left`, `Right`, `Top` или `Bottom`, панель инструментов перемещается в контейнер панели инструментов, для опции `ToolbarContainerControl.DockType` свойство которого имеет соответствующее значение.

Панели инструментов могут быть размещены в несколько рядов внутри родительского контейнера панели инструментов. Вы можете использовать следующие свойства для отображения панели инструментов в определенной строке и перемещения панели инструментов внутри строки:

- `Toolbar.Row` — Получает или устанавливает индекс строки, в которой отображается панель инструментов, начинающийся с нуля. Индексы строк начинаются с `0`. Это свойство действует только для обычных панелей инструментов. Невозможно изменить индекс строки для главного меню, равно как и разместить обычную панель инструментов перед главным меню.
- `Toolbar.Column` — Возвращает или устанавливает начинающийся с нуля порядок расположения панели инструментов в строке.
- `Toolbar.Position` — Возвращает или устанавливает минимальное смещение панели инструментов внутри строки.



Размещение элементов на панелях инструментов

Адаптивное размещение

Панели инструментов автоматически скрывают и восстанавливают свои элементы при изменении размера родительского контейнера. Отключите опцию `Toolbar.AllowShrinkToolbar`, чтобы предотвратить сворачивание определенной панели при уменьшении размера контейнера.

Растягивание панели инструментов

Панель инструментов может быть растянута и подогнана под ширину контейнера. Установите для опции `Toolbar.StretchToolbar` значение `true`, чтобы включить растягивание панели инструментов. В этом режиме никакая другая панель инструментов не может отображаться в той же строке.

Главное меню и панель состояния всегда занимают целые строки. Опция `Toolbar.StretchToolbar` не действует для этих панелей инструментов.

Расположение элементов в несколько рядов

Панель инструментов может отображать свои элементы в несколько строк, когда ширины родительского контейнера недостаточно для отображения элементов в одной строке. Установите для свойства `Toolbar.WrapItems` значение `true`, чтобы включить размещение нескольких строк.

Плавающие панели инструментов

Чтобы создать плавающую панель инструментов в XAML, добавьте объект `Toolbar` в коллекцию `ToolbarManager.Toolbars` и задайте свойству `Toolbar.DockType` значение `Floating`. Используйте свойство `Toolbar.FloatingPosition`, чтобы задать местоположение плавающей панели инструментов.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarManager.Toolbars>
```

```
    <mxb:Toolbar x:Name="bar1" DockType="Floating" FloatingPosition="200,200"
                         ToolbarName="Toolbar 1"  >
        <mxb:ToolbarButtonItem Header="Script"
         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Script, Icon=CSharpFile.svg'}"
         Category="UserCommands"/>
        <mxb:ToolbarButtonItem Header="Settings"
         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=List of Bugs.svg'}"
         Category="UserCommands"/>
    </mxb:Toolbar>
</mxb:ToolbarManager.Toolbars>
```

В следующем фрагменте показано, как сделать панель инструментов плавающей в коде.

```
toolBar1.DockType = Eremex.AvaloniaUI.Controls.Bars.MxToolbarDockType.Floating;
toolBar1.FloatingPosition = new PixelPoint(200, 200);
```

Настройка панели инструментов во время выполнения

Изменение порядка панелей инструментов

Если опция `Toolbar.AllowDragToolbar` включена, на панели инструментов отображается маркер перетаскивания. Пользователь может нажать эту кнопку, а затем перетащить панель инструментов в другую позицию или сделать панель инструментов плавающей.

Перестановка элементов с помощью быстрой настройки

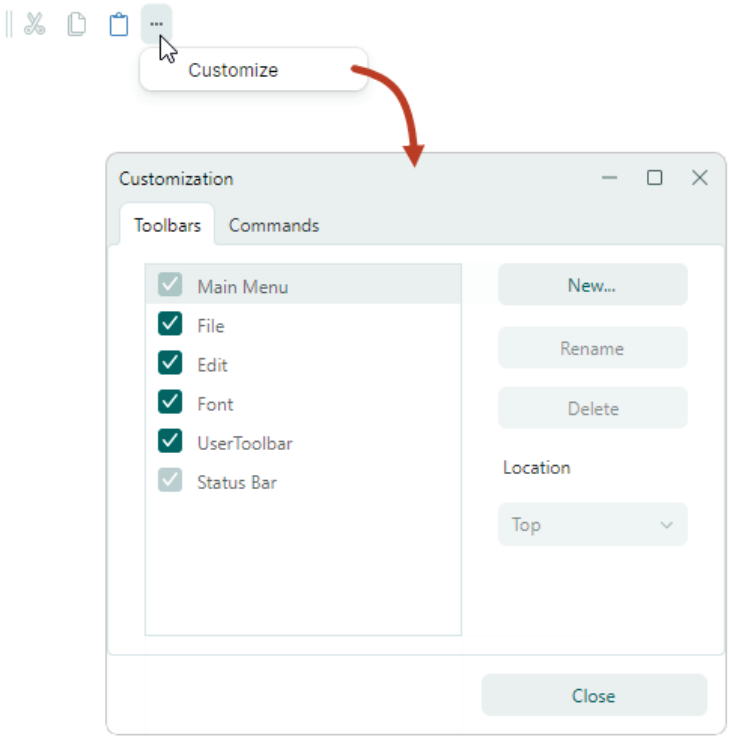
Пользователи могут изменять порядок размещения элементов панели инструментов, не активируя режим настройки. Им нужно нажать клавишу ALT, а затем перетащить элемент в нужную позицию. Вместо клавиши ALT нажмите комбинацию клавиш CTRL+ALT, чтобы скопировать перетаскиваемый элемент.

Режим настройки и окно настройки

Панели инструментов поддерживают режим настройки, в котором пользователи могут выполнять следующие операции:

- Получите доступ к видимым и скрытым панелям инструментов, а также показывайте и скрывайте их.
- Получите доступ ко всем элементам и управляйте их видимостью и позицией на панелях инструментов с помощью операций перетаскивания.
- Создавайте и удаляйте пользовательские панели инструментов.

Чтобы активировать режим настройки панели инструментов, пользователь может нажать кнопку настройки панели инструментов ('...'), а затем выбрать команду "Настроить". При активации режима настройки отображается окно настройки:



СОВЕТ

Включите опцию `Toolbar.ShowCustomizationButton`, чтобы отобразить кнопку настройки ('...') на панели инструментов.

Вызовите метод `ToolbarManager.ShowCustomizationWindow`, чтобы активировать режим настройки и отобразить окно настройки в коде.

В режиме настройки пользователь может использовать операции перетаскивания для скрытия, отображения и перемещения элементов панели инструментов:

- Перетащите элементы из панелей инструментов/подменю на вкладку "Команды" окна настройки, чтобы скрыть эти элементы.
- Перетащите элементы с вкладки "Команды" окна настройки на панели инструментов/подменю, чтобы отобразить эти элементы.
- Перетаскивайте элементы между панелями инструментов и подменю, чтобы изменить их размещение.

Окно настройки также позволяет пользователю скрывать/показывать панели инструментов и создавать пользовательские. Смотрите следующий раздел для получения дополнительной информации: Пользовательские панели инструментов .

Опции настройки панелей инструментов

Панели инструментов содержат следующие опции, которые позволяют вам настраивать их свойства просмотра и поведения:

- `Toolbar.ToolbarName` — Получает или устанавливает заголовок панели инструментов. Заголовок панели инструментов отображается в окне настройки и когда панель инструментов находится в плавающем состоянии.
- `Toolbar.AllowDragToolbar` — Возвращает или устанавливает, будет ли на панели инструментов отображаться маркер перетаскивания. Эта кнопка позволяет пользователю перетащить панель инструментов в другую пристыкованную позицию или сделать ее плавающей.
- `Toolbar.ShowCustomizationButton` — Получает или устанавливает, будет ли на панели инструментов отображаться кнопка настройки ('...'). Нажатие на эту кнопку вызывает меню, которое позволяет пользователю вызвать окно настройки.

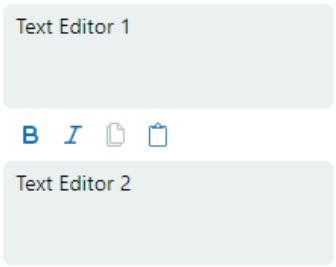
Автономные панели инструментов

Вы можете отображать панели инструментов в любой позиции в вашем окне/UserControl. Чтобы сделать это, создайте `ToolbarContainerControl` в этой позиции и добавьте панели инструментов в этот контейнер.

Свойство `ToolbarContainerControl.DockType` имеет значение `Standalone`. Этот тип стыковки применяет свойства внешнего вида, характерные для автономных контейнеров панели инструментов. Например, автономные контейнеры не имеют границ в светлой и темной темах Eremex.

Ориентация панелей инструментов в автономных контейнерах панелей инструментов по умолчанию - горизонтальная. Вы можете использовать свойство `ToolbarContainerControl.Orientation` для выравнивания автономных панелей по вертикали.

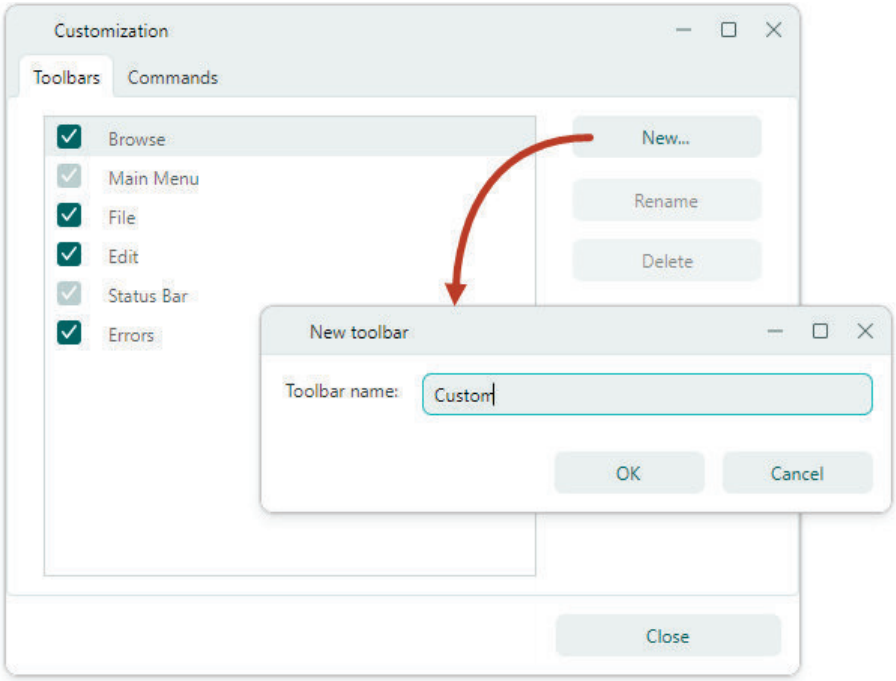
В следующем примере отображается автономная панель инструментов между двумя текстовыми полями. Панель содержит команды, которые выполняют действия со вторым текстовым полем.




```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<StackPanel Grid.Row="1" Grid.Column="1">
    <TextBox x:Name="textBox" Text="Text Editor 1" AcceptsReturn="True" FontSize="14" Height="70"/>

    <mxb:ToolbarContainerControl DockType="Standalone" Orientation="Horizontal"  >
        <mxb:Toolbar x:Name="EditToolbar2" ToolbarName="Edit (textBox2)" AllowDragToolbar="False" ShowCustomizationButton="False" >
            <mxb:ToolbarCheckItemGroup>
                <mxb:ToolbarCheckItem Header="Bold"
                 IsChecked="{Binding #textBox2.FontWeight, Converter={local:BoolToFontWeightConverter}, Mode=TwoWay}"
                 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FontBold.svg'}" Category="Font"/>
                <mxb:ToolbarCheckItem Header="Italic"
                 IsChecked="{Binding #textBox2.FontStyle, Converter={local:BoolToFontStyleConverter}, Mode=TwoWay}"
                 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FontItalic.svg'}" Category="Font"/>
            </mxb:ToolbarCheckItemGroup>
            <mxb:ToolbarButtonItem Header="Copy" Command="{Binding #textBox2.Copy}"
             IsEnabled="{Binding #textBox2.CanCopy}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditCopy.svg'}" Category="Edit"/>
            <mxb:ToolbarButtonItem Header="Paste"
                                   Command="{Binding #textBox2.Paste}"
                                   IsEnabled="{Binding #textBox2.CanPaste}"
                                   Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditPaste.svg'}"
                                   Category="Edit"/>
        </mxb:Toolbar>
    </mxb:ToolbarContainerControl>
    <TextBox x:Name="textBox2" Text="Text Editor 2" AcceptsReturn="True" FontSize="14" Height="70"/>
</StackPanel>
```

Пользовательские панели инструментов

Пользователь может создавать панели инструментов в окне настройки во время выполнения программы. Эти панели называются "пользовательскими панелями инструментов", и для них установлена опция `Toolbar.UserToolbar`, равная `true`.



В отличие от других панелей инструментов, пользовательские панели инструментов можно переименовывать и удалять в окне настройки.

При необходимости вы можете вручную установить для свойства `Toolbar.UserToolbar` значение `true`, чтобы создать пользовательскую панель инструментов в коде. Пользователи также могут переименовывать и удалять эту панель инструментов в окне настройки.

После создания пользовательской панели инструментов пользователь может заполнить ее командами, используя операции перетаскивания.

Область действия горячих клавиш

Свойство `ToolbarItem.HotKey` позволяет назначать горячие клавиши элементам. Компонент `ToolbarManager` может перехватывать горячие клавиши, если фокус находится в пределах области действия горячих клавиш.

Дефолтная область действия горячих клавиш - это клиентская область `ToolbarManager`. Установите для свойства `ToolbarManager.IsWindowManager` значение `true`, чтобы расширить область действия горячих клавиш на все окно. В этом случае компонент `ToolbarManager` регистрирует горячие клавиши элемента в окне. Он сможет перехватывать и обрабатывать горячие клавиши, даже если фокус находится за пределами его клиентской области.

Смотрите следующий раздел для получения дополнительной информации: Горячие клавиши элемента панели инструментов .

Элементы панели инструментов

Элементы панели инструментов используются для отображения кнопок, переключающих кнопок, текстовых меток, подменю и встроенных редакторов на панелях инструментов и меню. Вы можете добавить любое количество элементов в каждую панель/меню и создать несколько уровней иерархии с помощью подменю.

Добавление элементов в панель и контекстное меню

Используйте коллекции `Toolbar.Items` и `PopupMenu.Items` для заполнения элементами панелей и меню. В XAML вы можете определять элементы непосредственно между начальным и конечным тегами `<Toolbar>` / `<PopupMenu>`.

В следующем примере на панели инструментов отображаются три элемента.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:Toolbar x:Name="EditToolbar" ToolbarName="Edit" ShowCustomizationButton="False">
    <mxb:ToolbarButtonItem Header="Cut" Command="{Binding #textBox.Cut}"
     IsEnabled="{Binding #textBox.CanCut}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Cut.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Copy" Command="{Binding #textBox.Copy}"
     IsEnabled="{Binding #textBox.CanCopy}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Copy.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Paste" Command="{Binding #textBox.Paste}"
     IsEnabled="{Binding #textBox.CanPaste}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Paste.svg'}"
     Category="Edit"/>
</mxb:Toolbar>
```

Типы элементов панели инструментов

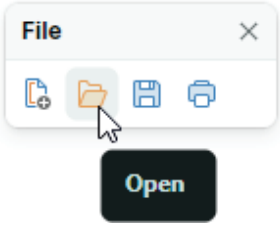
Библиотека панелей инструментов поддерживает несколько типов элементов панели инструментов. Все они являются потомками класса `ToolbarItem`, который содержит свойства, общие для всех элементов панели инструментов.

Общие свойства элементов панели инструментов

- `Alignment` — Выравнивание элемента на панели инструментов.
- `Category` — категория, к которой принадлежит элемент. Категории используются для организации элементов в логические группы в окне настройки. Смотрите следующий раздел для получения дополнительной информации: Категории элементов панели инструментов .
- `Command` — Команда, выполняемая при нажатии на кнопку.
- `CommandParameter` — Командный параметр, передаваемый указанной команде.
- `DisplayMode` — Получает или устанавливает, следует ли отображать только картинку, заголовок или и то, и другое.
- `Glyph` — Картинка элемента.
- `GlyphAlignment` — Выравнивание картинки относительно заголовка элемента.
- `GlyphSize` — Размер отображаемой картинки.
- `Header` — Текст для отображения элемента.
- `ShowSeparator` — Получает или устанавливает, следует ли отображать разделитель перед элементом. Вы также можете использовать элемент панели `ToolbarSeparatorItem` для вставки разделителя.

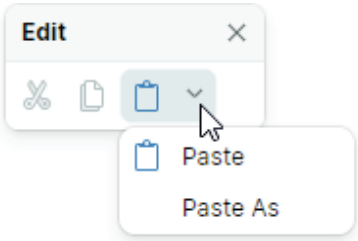
ToolbarButtonItem

`ToolbarButtonItem` - это наиболее распространенный элемент, который инкапсулирует обычную кнопку. Пользователь может нажать на кнопку, чтобы вызвать связанную команду.




```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarButtonItem Header="Open"
 Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/FileOpen.svg'}"
 Category="File" Command="{Binding OpenFileCommand}"/>
```

Вы можете привязать контрол всплывающего окна к объекту `ToolbarButtonItem`. Этот контрол будет отображаться, когда пользователь щелкнет по элементу или встроенной стрелке вниз.




```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarButtonItem Header="Paste"
                       Command="{Binding #textBox.Paste}"
                       IsEnabled="{Binding #textBox.CanPaste}"
                       Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditPaste.svg'}"
                       Category="Edit"
                       DropDownArrowVisibility="ShowArrow" DropDownArrowAlignment="Default"
                       >
    <mxb:ToolbarButtonItem.DropDownControl>
        <mxb:PopupMenu>
            <mxb:ToolbarButtonItem Header="Paste" Command="{Binding #textBox.Paste}"
             IsEnabled="{Binding #textBox.CanPaste}"/>
            <mxb:ToolbarButtonItem Header="Paste As" Command="{Binding PasteAsCommand}"
             IsEnabled="{Binding #textBox.CanPaste}"/>
        </mxb:PopupMenu>
    </mxb:ToolbarButtonItem.DropDownControl>
</mxb:ToolbarButtonItem>
```

Основные свойства ToolbarButtonItem

- `DropDownControl` — Получает или устанавливает контрол всплывающего окна (объект `Eremex.AvaloniaUI.Controls.Bars.IPopup`), связанный с элементом. Контрол появляется, когда пользователь щелкает по элементу или стрелке встроенного всплывающего окна (см. `DropDownArrowVisibility`). Следующие объекты реализуют интерфейс `Eremex.AvaloniaUI.Controls.Bars.IPopup`, и поэтому они могут отображаться в виде контролов всплывающего окна:
  - `Eremex.AvaloniaUI.Controls.Bars.PopupMenu` — Всплывающее меню. Вы можете добавить в меню все типы элементов панели инструментов, чтобы заполнить его содержимым.
  - `Eremex.AvaloniaUI.Controls.Bars.PopupContainer` — Контейнер с контролами. Используйте `PopupContainer` для отображения пользовательских контролов в всплывающем окне.
- `DropDownArrowVisibility` — Получает или устанавливает, отображает ли элемент стрелку всплывающего окна, используемую для вызова связанного контрола всплывающего окна. Поддерживаемые опции включают:

    - `ShowArrow` — Видна стрелка всплывающего окна. Элемент и стрелка действуют как единая кнопка. Щелчок по ним отображает связанный контрол всплывающего окна.
    - `ShowSplitArrow` или `Default` — Видна стрелка всплывающего окна. Она действует как отдельная кнопка, встроенная в элемент. Щелчок по стрелке всплывающего окна вызывает связанный с ним контрол всплывающего окна. Щелчок по элементу вызывает его команду.
    - `Hide` — Стрелка всплывающего окна скрыта. Щелчок по элементу вызывает контрол всплывающего окна.
- `DropDownArrowAlignment` — Возвращает или устанавливает позицию стрелки всплывающего окна.
- `DropDownOpenMode` — Получает или устанавливает, вызывается ли контрол всплывающего окна и когда, когда пользователь касается стрелки элемента/всплывающего окна. Поддерживаемые опции включают:
    - `Press` или `Default` — Всплывающее окно отображается при нажатии кнопки мыши.
    - `Click` — После нажатия и последующего отпускания курсора мыши над элементом отображается всплывающее окно.
    - `Never` — Всплывающее окно не отображается.
- `DropDownPress` — Событие, которое срабатывает при нажатии стрелки на всплывающее окно.

Смотрите также: Общие свойства элементов панели инструментов .

ToolbarCheckItem

`ToolbarCheckItem` инкапсулирует кнопку-переключателя, которая поддерживает два состояния - нормальное и нажатое.


```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarCheckItem Header="Bold"
 IsChecked="{Binding #textBox.FontWeight,
           Converter={helpers:BoolToFontWeightConverter}, Mode=TwoWay}"
 Glyph="{SvgImage 'avares://DemoCenter/Images/FontBold.svg'}" Category="Font"/>
```

Основные свойства и события ToolbarCheckItem

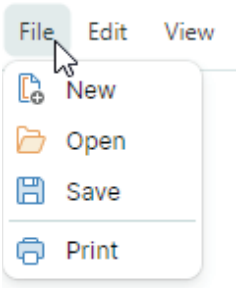
- `IsChecked` — Возвращает или устанавливает состояние нажатия кнопки.

- `CheckedChanged` — Событие, которое срабатывает при изменении состояния нажатия кнопки "Проверить".

Смотрите также: Общие свойства элементов панели инструментов .

ToolbarMenuItem

`ToolbarMenuItem` - это элемент, который отображает подменю при щелчке мыши.



Чтобы указать содержимое подменю, определите элементы между начальным и конечным тегами `<ToolbarMenuItem>` в XAML или добавьте элементы в коллекцию `ToolbarMenuItem.Items` в коде.

```
xmlns:mxb="clr-namespace:Eremex.AvaloniaUI.Controls.Bars;assembly=Eremex.Avalonia.Controls"

<mxb:ToolbarMenuItem Header="File" Category="File">
    <mxb:ToolbarButtonItem Header="New"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu,
      Icon=NewDraftAction.svg'}"
     Category="File" Command="{Binding NewFileCommand}"/>
    <mxb:ToolbarButtonItem Header="Open"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Folder Open.svg'}"
     Category="File" Command="{Binding OpenFileCommand}"/>
    <mxb:ToolbarButtonItem Header="Save"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Save.svg'}"
     Category="File" Command="{Binding SaveFileCommand}"/>
    <mxb:ToolbarButtonItem Header="Print"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Print.svg'}"
     ShowSeparator="True"
     Category="File" Command="{Binding PrintFileCommand}"/>
</mxb:ToolbarMenuItem>
```

Основные свойства и события ToolbarMenuItem

- `DropDownOpenMode` — Получает или устанавливает, вызывается ли подменю и каким образом, когда пользователь касается элемента. Поддерживаемые опции включают:
  - `Press` или `Default` — Меню отображается при нажатии кнопки мыши.
  - `Click` — Меню отображается после нажатия и последующего отпускания курсора мыши над элементом.
  - `Never` — Меню не отображается.

- `Items` — Набор элементов, отображаемых в подменю.

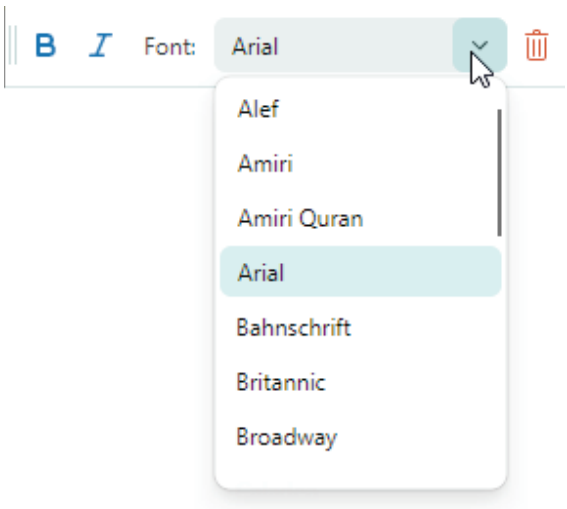
Events

- `Opening` — срабатывает, когда меню вот-вот будет отображено. Это событие позволяет отменить отображение меню.
- `Opened` — срабатывает после отображения меню.
- `Closing` — срабатывает, когда меню собирается закрываться. Это событие позволяет отменить закрытие меню.
- `Closed` — срабатывает после закрытия меню.

Смотрите также: Общие свойства элементов панели инструментов .

ToolbarEditorItem

`ToolbarEditorItem` позволяет отображать встроенный редактор.



Чтобы указать тип встроенного редактора и свойства настроек, используйте свойство `ToolbarEditorItem.EditorProperties`. Этому элементу можно присвоить свойства следующих типов редакторов:

- `ButtonEditorProperties` — Содержит свойства, специфичные для контрола `ButtonEditor`.
- `CheckEditorProperties` — Содержит свойства, специфичные для контрола `CheckEditor`.
- `ColorEditorProperties` — Содержит свойства, специфичные для контрола `ColorEditor`.
- `ComboBoxEditorProperties` — Содержит свойства, специфичные для контрола `ComboBoxEditor`.
- `HyperlinkEditorProperties` — Содержит свойства, специфичные для контрола `HyperlinkEditor`.
- `PopupColorEditorProperties` — Содержит свойства, специфичные для контрола `PopupColorEditor`.
- `PopupEditorProperties` — Содержит свойства, специфичные для контрола `PopupEditor`.

- `SegmentedEditorProperties` — Содержит свойства, специфичные для контрола `SegmentedEditor`.
- `SpinEditorProperties` — Содержит свойства, специфичные для контрола `SpinEditor`.
- `TextEditorProperties` — Содержит свойства, специфичные для контрола `TextEditor`.

В следующем примере объект *Font* `ToolbarEditorItem` отображает список шрифтов с помощью встроенного редактора combobox. Свойству `ToolbarEditorItem.EditorProperties` присвоено значение `ComboBoxEditorProperties`, которое соответствует контролу `ComboBoxEditor`.

```
<mxb:ToolbarEditorItem Header="Font:" EditorWidth="150" Category="Font"
 EditorValue="{Binding #textBox.FontFamily,
  Converter={helpers:FontNameToFontFamilyConverter}}">
    <mxb:ToolbarEditorItem.EditorProperties>
        <mxe:ComboBoxEditorProperties
         ItemsSource="{Binding $parent[view:ToolbarAndMenuPageView].Fonts}"
         IsTextEditable="False"/>
    </mxb:ToolbarEditorItem.EditorProperties>
</mxb:ToolbarEditorItem>
```

Основные свойства и события ToolbarEditorItem

- `ToolbarEditorItem.EditorProperties` — Получает или свойства объекта, который определяет тип и свойства встроенного редактора.
- `ToolbarEditorItem.EditorValue` — Получает или устанавливает значение встроенного редактора. Используйте это свойство для привязки данных.
- `ToolbarEditorItem.SizeMode` — Получает или устанавливает режим размера элемента. Это свойство позволяет вам растянуть элемент панели таким образом, чтобы он занимал все доступное пустое пространство внутри панели.
- `ToolbarEditorItem.EditorWidth` — Возвращает или устанавливает ширину встроенного редактора.
- `ToolbarEditorItem.EditorHeight` — Возвращает или устанавливает высоту встроенного редактора.
- `ToolbarEditorItem.EditorAlignment` — Получает или устанавливает выравнивание редактора относительно заголовка элемента.

Смотрите также: Общие свойства элементов панели инструментов .

ToolbarTextItem

`ToolbarTextItem` создает текстовую метку. Используйте этот элемент для отображения текста, который не может редактироваться пользователями.

```
<mxb:ToolbarTextItem Header="{Binding #scaleDecorator.Scale,
 StringFormat={}Zoom: {0:P0}}"
 ShowBorder="True" Alignment="Far" ShowSeparator="True" Category="Info"
 CustomizationName="Zoom Info"/>
```

Основные свойства и события ToolbarTextItem

- `ToolbarTextItem.SizeMode` — Получает или устанавливает режим размера элемента. Это свойство позволяет вам растянуть элемент таким образом, чтобы он занимал все доступное пустое пространство внутри панели.
- `ToolbarTextItem.ShowBorder` — Возвращает или устанавливает, видна ли граница элемента. Вы можете использовать свойство `ToolbarTextItem.BorderTemplate`, чтобы указать пользовательский шаблон для рисовать границу.
- `ToolbarTextItem.BorderTemplate` — Получает или устанавливает пользовательский шаблон для рисовать границу элемента. Этот шаблон действует, если включена опция `ToolbarTextItem.ShowBorder`.

Смотрите также: Общие свойства элементов панели инструментов .

ToolbarItemGroup

`ToolbarItemGroup` представляет собой группу (контейнер) элементов панели инструментов.

Когда вы объединяете элементы в группу, они функционируют как единое целое. Чтобы указать содержимое группы, определите элементы между начальным и конечным тегами `<ToolbarItemGroup>` или добавьте элементы в коллекцию `ToolbarItemGroup.Items` в коде.

```
<mxb:ToolbarItemGroup CustomizationName="Clipboard" >
    <mxb:ToolbarButtonItem Header="Cut" Command="{Binding #textBox.Cut}"
     IsEnabled="{Binding #textBox.CanCut}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Cut.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Copy" Command="{Binding #textBox.Copy}"
     IsEnabled="{Binding #textBox.CanCopy}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Copy.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Paste" Command="{Binding #textBox.Paste}"
     IsEnabled="{Binding #textBox.CanPaste}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Paste.svg'}"
     Category="Edit"/>
</mxb:ToolbarItemGroup>
```

Основные свойства и события ToolbarItemGroup

- `ToolbarItemGroup.Items` — Позволяет вам получить доступ к дочерним элементам группы.

ToolbarCheckItemGroup

`ToolbarCheckItemGroup` представляет собой группу переключающих элементов (контейнер), которая контролирует состояние нажатия своих дочерних элементов (объектов `ToolbarCheckItem`). Класс `ToolbarCheckItemGroup` позволяет вам создавать группу взаимоисключающих элементов (radio group) и группу, которая позволяет нажимать на несколько элементов одновременно.

Чтобы указать содержимое группы, добавьте элементы `ToolbarCheckItem` между начальным и конечным тегами `<ToolbarCheckItemGroup>` в XAML или добавьте элементы в коллекцию `ToolbarCheckItemGroup.Items` в коде.

```
<mxb:ToolbarCheckItemGroup CustomizationName="Check group" CheckType="Radio" >
    <mxb:ToolbarCheckItem Header="1" IsChecked="{Binding Option1}"   Category="Settings"/>
    <mxb:ToolbarCheckItem Header="2" IsChecked="{Binding Option2}"  Category="Settings"/>
    <mxb:ToolbarCheckItem Header="3" IsChecked="{Binding Option3}"  Category="Settings"/>
</mxb:ToolbarCheckItemGroup>
```

Объект `ToolbarCheckItemGroup` принимает дочерние элементы всех поддерживаемых типов элементов панели. Группа, однако, манипулирует только состояниями нажатия вложенных объектов `ToolbarCheckItem`.

Основные свойства и события ToolbarCheckItemGroup

- `ToolbarCheckItemGroup.CheckType` — Получает или устанавливает, может ли один или несколько элементов быть нажаты в группе одновременно. Поддерживаются следующие опции:
    - `Default` или `Multiple` — несколько элементов могут быть нажаты одновременно.
    - `Radio` — Группа взаимоисключающих элементов. Пользователь не может снять флажок с какого-либо элемента иначе, как установив другой.
    - `Single` — Группа взаимоисключающих элементов. Пользователь может снять флажки со всех элементов внутри группы.
- `ToolbarCheckItemGroup.Items` — Позволяет вам получить доступ к дочерним элементам группы.

Смотрите также: Общие свойства элементов панели инструментов .

ToolbarSeparatorItem

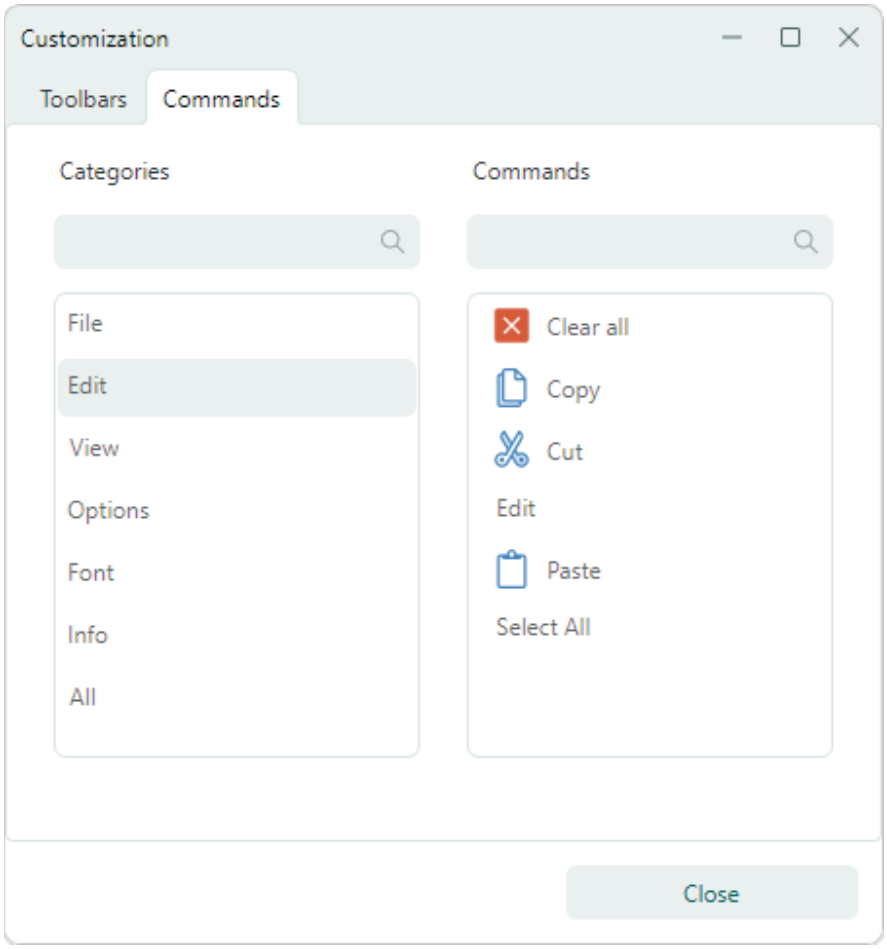
- `ToolbarSeparatorItem` — Рисует разделитель.

```
<mxb:ToolbarMenuItem Header="File" Category="File">
    <mxb:ToolbarButtonItem Header="New"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu,
      Icon=NewDraftAction.svg'}"
```

```
      Category="File"/>
     <mxb:ToolbarButtonItem Header="Open"
      Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Folder Open.svg'}"
      Category="File"/>
     <mxb:ToolbarButtonItem Header="Save"
      Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Save.svg'}"
      Category="File"/>
     <mxb:ToolbarSeparatorItem/>
     <mxb:ToolbarButtonItem Header="Print"
      Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Print.svg'}"
      Category="File"/>
</mxb:ToolbarMenuItem>
```

Категории элементов панели инструментов

Вы можете классифицировать элементы панели инструментов по категориям, чтобы включить группировку элементов в окне настройки. Пользователь может выбрать категорию в окне настройки, чтобы получить доступ к связанным элементам.



Используйте свойство `ToolbarItem.Category`, чтобы присвоить элементу категорию. Это свойство определяет название категории. Чтобы отнести группу элементов к одной и той же категории, установите для их свойства `ToolbarItem.Category` одинаковое название категории.

```
<mxb:ToolbarMenuItem Header="Edit" Category="Edit">
    <mxb:ToolbarButtonItem Header="Cut"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Cut.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Copy"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Copy.svg'}"
     Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Paste"
    Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Paste.svg'}"
    Category="Edit"/>
    <mxb:ToolbarButtonItem Header="Select All" Command="{Binding #textBox.SelectAll}"
    Category="Edit"
     ShowSeparator="True"/>
    <mxb:ToolbarButtonItem Header="Clear all" Command="{Binding #textBox.Clear}"
     Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Clear.svg'}"
     Category="Edit"/>
</mxb:ToolbarMenuItem>
```

Горячие клавиши

Вы можете использовать свойство `ToolbarItem.HotKey` для назначения горячих клавиш элементам.

```
<mxb:ToolbarButtonItem
    Header="Clear" Command="{Binding #textBox.Clear}" HotKey="Ctrl+Q"
    Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditDelete.svg'}"/>
```

Нажатие горячей клавиши активирует команду элемента при условии, что фокус находится в пределах области действия горячей клавиши. Дефолтная область действия горячих клавиш - это область пользовательского интерфейса в пределах компонента `ToolbarManager`. Когда фокус ввода находится за пределами области действия горячих клавиш, `ToolbarManager` не может перехватывать горячие клавиши.

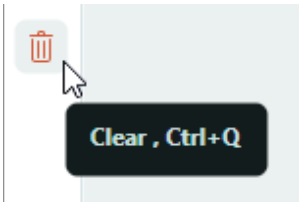
Свойство `ToolbarManager.IsWindowManager` позволяет вам расширить область действия горячих клавиш на все окно. Когда вы устанавливаете для этого свойства значение `true`, компонент `ToolbarManager` регистрирует горячие клавиши элемента в окне. Он сможет перехватывать и обрабатывать горячие клавиши, даже если фокус находится за пределами его клиентской области.

Отображение горячих клавиш

Горячие клавиши, назначенные элементам панели инструментов, отображаются в следующих случаях:

- В элементах, когда они находятся в подменю или всплывающих меню.

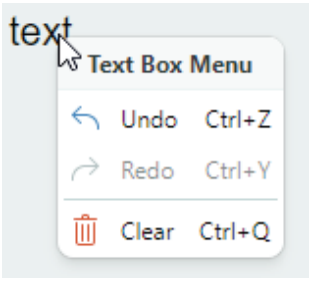
- Во всплывающих подсказках элементов.



Вы можете использовать свойство `ToolbarItem.HotKeyDisplayString` для указания текста для отображения горячей клавиши. Этот текст отображается, даже если элементу не назначена горячая клавиша. Это полезно, если целевая горячая клавиша уже зарегистрирована другим объектом для выполнения определенной операции, и вы хотите указать, что та же самая горячая клавиша связана с элементом панели инструментов.

Например, текстовое поле регистрирует горячие клавиши *Ctrl+Z* для выполнения операции отмены. Если элемент панели инструментов выполняет ту же операцию отмены в текстовом поле, не назначайте горячую клавишу *Ctrl+Z* этому элементу. Вместо этого установите для `ToolbarItem.HotKeyDisplayString` элемента значение "*Ctrl+Z*", чтобы отображать это горячие клавиши для пользователей во всплывающих подсказках и подменю/всплывающих меню.

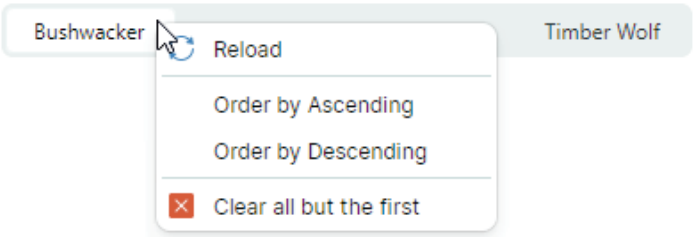
```
<mxb:ToolbarButtonItem Header="Undo" HotKeyDisplayString="Ctrl+Z"
    Command="{Binding $parent[TextBox].Undo}"
    IsEnabled="{Binding $parent[TextBox].CanUndo}"
    Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditUndo.svg'}"/>
```



Контекстные меню

Библиотека Панелей Инструментов и Меню содержит компонент `PopupMenu`, который позволяет создавать контекстные меню для контролов.

Чтобы указать контекстное меню, установите для attached-свойства `ToolbarManager.ContextPopup` для целевого контрола значение объекта `PopupMenu`.



Вы можете добавить все типы Элементы панели инструментов в контекстное меню. Определите элементы между начальным и конечным тегами `<PopupMenu>` в XAML или добавьте элементы в коллекцию `PopupMenu.Items` в коде.

```
<TextBox x:Name="textBox"  Text="Text Editor" AcceptsReturn="True" CornerRadius="0"
 FontFamily="Arial" FontSize="20" Height="200">
    <mxb:ToolbarManager.ContextPopup>
        <mxb:PopupMenu ShowIconStrip="True">
            <mxb:ToolbarButtonItem Header="Undo" HotKeyDisplayString="Ctrl+Z"
             Command="{Binding $parent[TextBox].Undo}"
             IsEnabled="{Binding $parent[TextBox].CanUndo}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditUndo.svg'}"/>
            <mxb:ToolbarButtonItem Header="Redo" HotKeyDisplayString="Ctrl+Y"
             Command="{Binding $parent[TextBox].Redo}" IsEnabled="{Binding $parent[TextBox].CanRedo}"
             Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditRedo.svg'}"/>
            <mxb:ToolbarSeparatorItem/>
            <mxb:ToolbarButtonItem Header="Clear" Command="{Binding $parent[TextBox].Clear}"
             HotKey="Ctrl+Q" Glyph="{SvgImage 'avares://bars_sample/Images/Toolbars/EditDelete.svg'}"/>
        </mxb:PopupMenu>
    </mxb:ToolbarManager.ContextPopup>
</TextBox>
```

Основные свойства и события всплывающего меню

- `ShowIconStrip` — Получает или устанавливает, следует ли отображать вертикальную полосу значков для элементов меню.
- `Header` — Позволяет указать заголовок для меню.
- `ShowHeader` — Возвращает или устанавливает, виден ли заголовок меню.

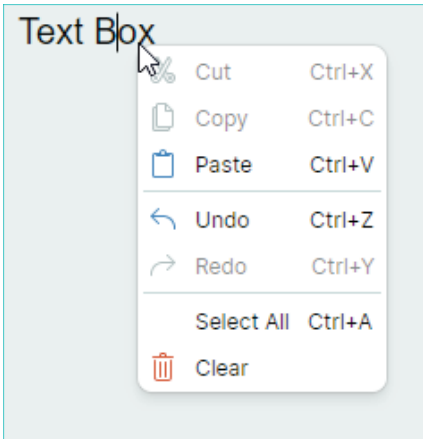
- `ContentRightIndent` — Задает ширину пустого пространства справа от текста элементов меню.

Events

- `Opening` — срабатывает, когда меню вот-вот будет отображено. Это событие позволяет отменить отображение меню.
- `Opened` — срабатывает после отображения меню.
- `Closing` — срабатывает, когда меню собирается закрываться. Это событие позволяет отменить закрытие меню.
- `Closed` — срабатывает после закрытия меню.

Пример - Как назначить контекстные меню контролам определенного типа

Вы можете определить меню в коллекции `Styles`, если вам нужно указать одно контекстное меню для нескольких контролов одного типа. В следующем примере задается attached-свойство `ToolbarManager.ContextPopup` для контролов `TextBox`, которые находятся в UserControl.

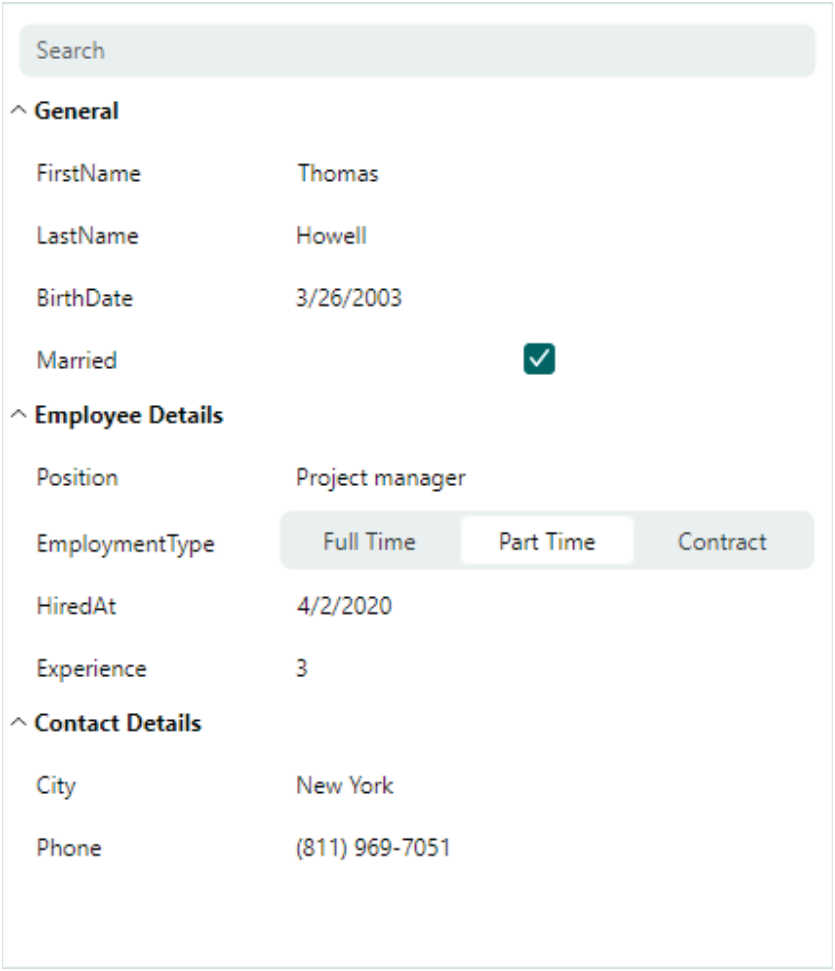



```
<UserControl.Styles>
    <Style Selector="TextBox">
        <Setter Property="mxb:ToolbarManager.ContextPopup">
            <Setter.Value>
                <Template>
                    <mxb:PopupMenu Focusable="False">
                        <mxb:ToolbarButtonItem Header="Cut" HotKeyDisplayString="Ctrl+X"
                         Command="{Binding $parent[TextBox].Cut}"
                         IsEnabled="{Binding $parent[TextBox].CanCut}"
                         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Cut.svg'}"/>
                        <mxb:ToolbarButtonItem Header="Copy" HotKeyDisplayString="Ctrl+C"
                         Command="{Binding $parent[TextBox].Copy}"
                         IsEnabled="{Binding $parent[TextBox].CanCopy}"
                         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Copy.svg'}"/>
                        <mxb:ToolbarButtonItem Header="Paste" HotKeyDisplayString="Ctrl+V"
                         Command="{Binding $parent[TextBox].Paste}"
                         IsEnabled="{Binding $parent[TextBox].CanPaste}"
                         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Paste.svg'}"/>
                        <mxb:ToolbarSeparatorItem/>
```

```
                        <mxb:ToolbarButtonItem Header="Undo" HotKeyDisplayString="Ctrl+Z"
                         Command="{Binding $parent[TextBox].Undo}"
                         IsEnabled="{Binding $parent[TextBox].CanUndo}"
                         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Undo.svg'}"/>
                        <mxb:ToolbarButtonItem Header="Redo" HotKeyDisplayString="Ctrl+Y"
                         Command="{Binding $parent[TextBox].Redo}"
                         IsEnabled="{Binding $parent[TextBox].CanRedo}"
                         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Basic, Icon=Redo.svg'}"/>
                        <mxb:ToolbarSeparatorItem/>
                        <mxb:ToolbarButtonItem Header="Select All" HotKeyDisplayString="Ctrl+A"
                         Command="{Binding $parent[TextBox].SelectAll}"/>
                        <mxb:ToolbarButtonItem Header="Clear"
                         Command="{Binding $parent[TextBox].Clear}"
                         Glyph="{SvgImage 'avares://DemoCenter/Images/Group=Context Menu, Icon=Delete.svg'}"/>
                    </mxb:PopupMenu>
                </Template>
            </Setter.Value>
        </Setter>
    </Style>
</UserControl.Styles>
```

PropertyGrid

Контрол `PropertyGridControl` позволяет пользователю просматривать свойства одного или нескольких объектов. Контрол отображает публичные свойства и их значения привязанных объектов в виде вертикального списка и позволяет пользователю редактировать значения свойств.



Основные функции контрола включают в себя:

- Автоматическое создание строк — контрол может использовать информацию, предоставляемую привязанным объектом (object), для автоматического создания строк.
- Ручное создание строк — вы можете вручную создавать строки для определенного набора свойств объекта.
- Операции редактирования данных — пользователь может редактировать значения ячеек, если включено редактирование данных. Вы можете встроить Eremex и пользовательские редакторы в ячейки, чтобы редактировать и представлять значения ячеек определенным образом.

- Поддержка атрибутов Data Annotation — PropertyGrid учитывает определенные атрибуты Data Annotation, применяемые к свойствам привязанного объекта. Вы можете использовать атрибуты Data Annotation, чтобы указать пользовательскую видимость, состояние "только для чтения", отображаемое имя, конвертер категорий и типов для автоматически сгенерированных строк.
- Категориальные строки — позволяют объединять строки в распахиваемые группы.
- Табуляционные строки — позволяют объединять строки в пользовательский интерфейс с вкладками.
- Панель поиска — помогает пользователю быстро находить строки по названиям

Смотрите следующий раздел для получения дополнительной информации: Обзор контрола PropertyGrid.

Обзор контрола PropertyGrid

`PropertyGridControl` отображает публичные свойства и их значения привязанных объектов в виде вертикального списка. Каждый элемент списка отображается в виде строки, содержащей две ячейки: "имя свойства" и "значение свойства".

PropertyGrid может автоматически заполнять строки из привязанных объектов. Вы также можете вручную создавать строки в коде. Определенные атрибуты, применяемые к свойствам привязанного объекта, позволяют вам указать пользовательскую видимость, состояние "только для чтения", отображаемое имя, категорию и конвертер типов.

Контрол поддерживает категориальные строки, которые позволяют объединять строки в распахиваемые группы.

Привязка данных и настройка строк

Когда вы привязываете контрол к объекту (объектам), контрол автоматически генерирует строки для всех публичных свойств. Определенные атрибуты позволяют настраивать опции генерации строк.

Контрол также поддерживает ручную генерацию обычных строк и категориальных строк. Смотрите следующие разделы для получения дополнительной информации:

- Привязка к одному объекту
- Привязка к нескольким объектам
- Создание строк
- Строки

Редактирование данных

Пользователи могут редактировать часть со "значением" строк, если включено редактирование данных. Контрол использует редакторы по умолчанию для отображения и редактирования значений строк распространенных типов данных (Boolean, Double, Integer, Enumerations и т.д.). При необходимости вы также можете указать пользовательские редакторы для определенных строк.

Смотрите следующие разделы, чтобы узнать, как указывать редакторы и получать к ним доступ, получать и устанавливать значения ячеек:

- Редактирование данных
- Пользовательские редакторы

Встроенный поиск

Панель поиска PropertyGrid позволяет пользователям быстро находить строки по названиям.

Обратитесь к следующему разделу, чтобы узнать больше:

- Поиск данных

Часто используемый API

Смотрите следующий раздел, чтобы ознакомиться с элементами API, которые помогают выполнять наиболее популярные задачи настройки:

- Часто используемый API

Привязка данных и создание строк

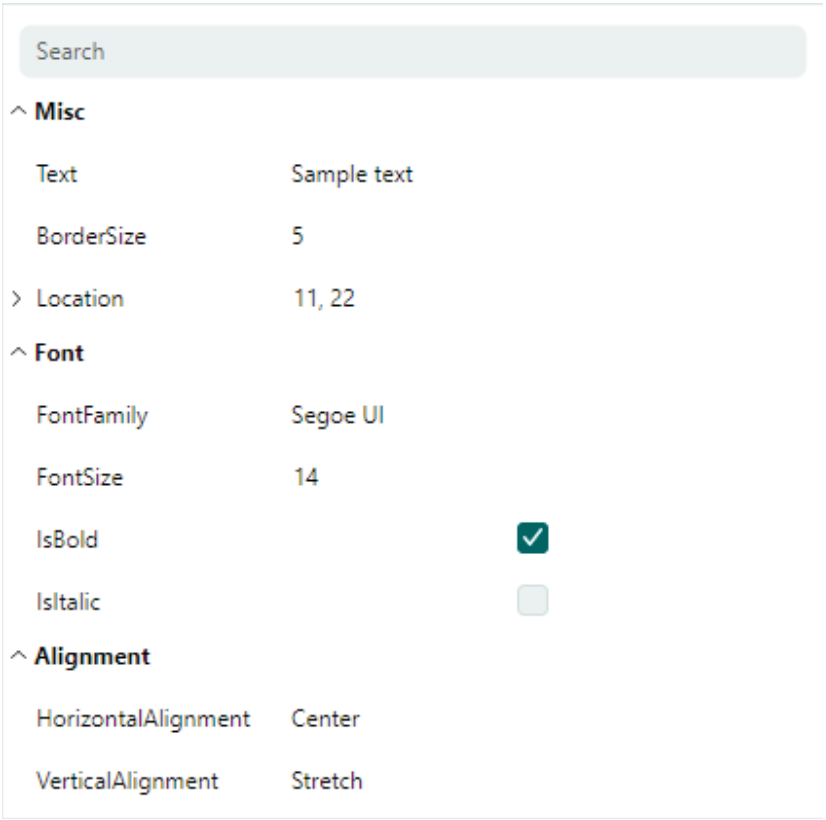
В этом разделе показано, как привязать PropertyGrid к одному или нескольким объектам. Контрол автоматически генерирует строки для публичных свойств привязанных объектов по умолчанию. Вы можете отключить автоматическую генерацию строк, создавать строки вручную, настраивать опции строк и указывать шаблоны строк.

Привязка к одному объекту

Используйте свойство `PropertyGridControl.SelectedObject`, чтобы привязать контрол к одному объекту. При привязке контрол автоматически отображает свойства объекта.

Пример

Следующий пример привязывает PropertyGrid к объекту *MyBusinessObject*, определенному в ViewModel.




```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
xmlns:local="using:PropertyGridTest"
```

```
<Window.DataContext>
    <local:SampleViewModel/>
</Window.DataContext>

<mxpg:PropertyGridControl
    Name="propertyGridControl1"
    SelectedObject="{Binding MyBusinessObject}"
    BorderThickness="1"
    UseModernAppearance="True"
    Margin="5"/>
```

```
using CommunityToolkit.Mvvm.ComponentModel;

public partial class SampleViewModel : ViewModelBase
{
    [ObservableProperty]
    MyBusinessObject myBusinessObject;
}

public partial class MyBusinessObject : ViewModelBase
{
    [ObservableProperty]
    string text = "Sample text";

    [ObservableProperty]
    double borderSize = 5;

    [property: Category("Font")]
    [ObservableProperty]
    string fontFamily = FontManager.Current.DefaultFontFamily.Name;

    [property: Category("Font")]
    [ObservableProperty]
    double fontSize = 14;

    [property: Category("Font")]
    [ObservableProperty]
    bool isBold = true;

    [property: Category("Font")]
    [ObservableProperty]
    bool isItalic;

    [property: Category("Alignment")]
    [ObservableProperty]
    HorizontalAlignment horizontalAlignment = HorizontalAlignment.Center;

    [property: Category("Alignment")]
    [ObservableProperty]
    VerticalAlignment verticalAlignment;
```

```
    [ObservableProperty]
    Point location = new Point(11, 22);
}

public class ViewModelBase : ObservableObject
{
}
```

Привязка к нескольким объектам

`PropertyGridControl` может отображать и редактировать свойства, которые являются общими для двух или более объектов. Присвойте список этих объектов свойству `PropertyGridControl.SelectedObjects`. Это приводит к тому, что контрол отображает только соответствующие свойства (те, которые имеют одинаковое имя и тип данных).

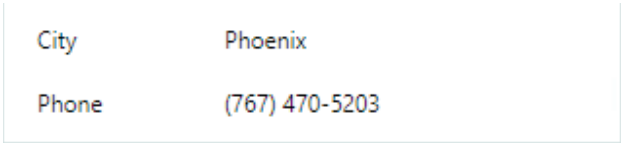
```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"

<mxpg:PropertyGridControl>
    <mxpg:PropertyGridControl.SelectedObjects>
        <local:MyList>
            <local:MyBusinessObject1/>
            <local:MyBusinessObject2/>
        </local:MyList>
    </mxpg:PropertyGridControl.SelectedObjects>
</mxpg:PropertyGridControl>
```

```
public class MyList : List<object>
{
}
```

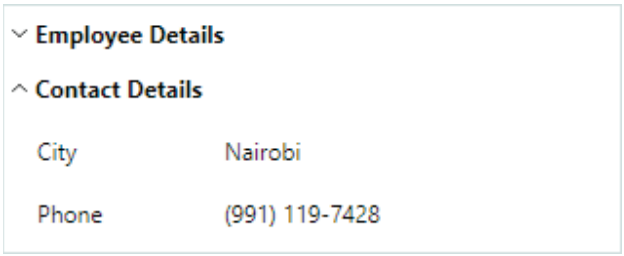
Создание строк

После привязки PropertyGrid к объекту (объектам) дефолтное поведение контрола заключается в автоматическом генерировании строк для отображения и редактирования свойств привязанного объекта(ов). Контрол автоматически генерирует следующие типы строк во время инициализации контрола:

- Строки данных (объекты `PropertyGridRow`) генерируются для всех публичных свойств. В этих строках отображаются имена и значения привязанных свойств.



- Категориальные строки (объекты `PropertyGridCategoryRow`) генерируются на основе атрибутов `System.ComponentModel.CategoryAttribute`, применяемых к нижележащим публичным свойствам. Соответствующие строки данных сгруппированы внутри этих категориальных строк.



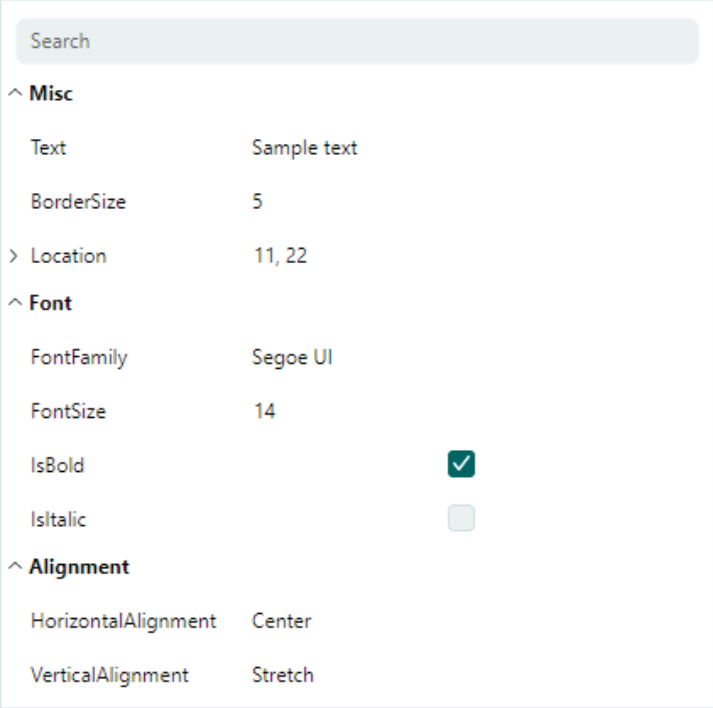
PropertyGrid предоставляет следующие функции настройки строк:

- Отключение автоматической генерации строк с помощью свойства `PropertyGridControl.AutoGenerateRows`.
- Создание строк вручную.
- Использование атрибутов аннотаций для настройки опций строк.
- Указание шаблонов данных для отображения отдельных строк.
- Генерация строк из коллекции моделей представления строк (`RowsSource`)

Смотрите следующий раздел для получения дополнительной информации: Строки .

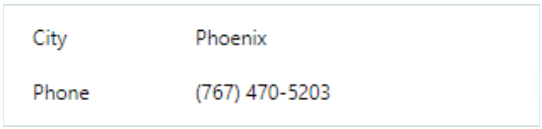
Строки в контроле PropertyGrid

PropertyGrid может автоматически создавать строки для свойств, предоставляемых привязанным объектом(ами) (см. `PropertyGridControl.SelectedObject` и `PropertyGridControl.SelectedObjects`). Автоматическая генерация строк включена по умолчанию. Вы можете отключить автоматическую генерацию строк с помощью опции `PropertyGridControl.AutoGenerateRows`, а затем создавать строки вручную.

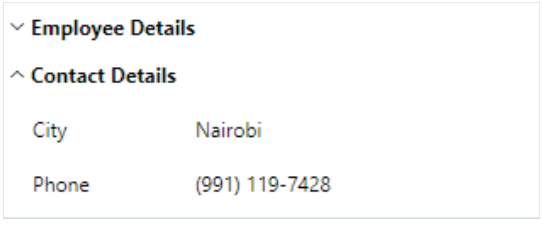


PropertyGrid поддерживает три типа строк:

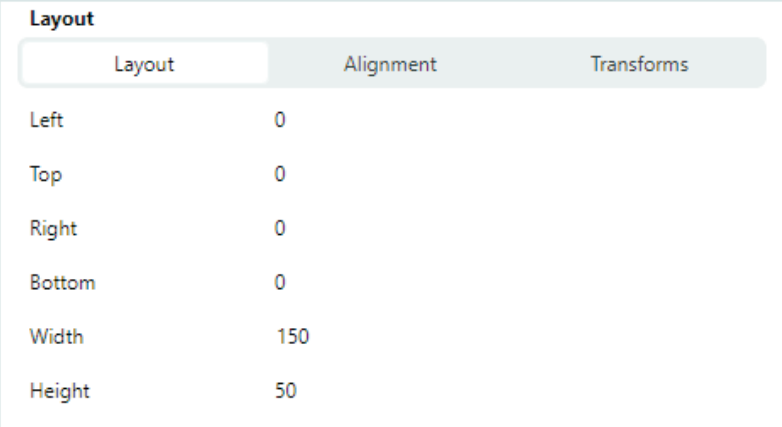
- Обычные строки (данных) (`PropertyGridRow`) — Отображают имена и значения привязанных свойств.

- Категориальные строки (`PropertyGridCategoryRow`) — используется для группировки других строк по категориям. Пользователи могут сворачивать и разворачивать категориальные строки, чтобы скрыть/показать их дочерние элементы.

- Табуляционная строка (`PropertyGridTabRow`) — используется для организации строк в пользовательский интерфейс с вкладками. Табуляционные строки не поддерживают функцию сворачивания/развертывания.



Создание строк

При включенной функции автоматической генерации строк пустой контрол PropertyGrid создает данные и категориальные строки на основе информации, полученной из привязанного объекта:

- Обычные строки создаются для всех публичных свойств.
- Категориальные строки генерируются на основе атрибутов `System.ComponentModel.CategoryAttribute` , применяемых к нижележащим публичным свойствам. Соответствующие строки данных сгруппированы в пределах этих категорий.

Если какая-либо строка была добавлена в контрол вручную (например, в XAML), автоматическая генерация строк не выполняется. Установите для свойства `PropertyGridControl.AutoGenerateRows` значение `false` , чтобы принудительно отключить автоматическую генерацию строк.

Используйте метод `PopulateRows` в коде для генерации данных и категориальных строк из привязанного объекта(ов). Этот метод очищает существующую коллекцию строк перед добавлением новых строк.

Вы можете применить определенные атрибуты Data Annotation к свойствам привязанного объекта, чтобы контролировать наличие, отображаемое имя и статус только для чтения сгенерированных строк PropertyGrid. Смотрите следующий раздел для получения более подробной информации: Использование атрибутов для настройки свойств строк .

Коллекция `PropertyGridControl.Rows` позволяет вам получать доступ к строкам контрола, добавлять и удалять отдельные элементы.

Создание строк данных

Используйте объекты `PropertyGridRow` для создания строк данных. Чтобы добавить строки данных на корневом уровне, добавьте объекты `PropertyGridRow` в коллекцию `PropertyGridControl.Rows` .

Основные свойства класса `PropertyGridRow` включают:

- `PropertyGridRow.FieldName` — Получает или задает имя публичного свойства, к которому привязана строка.
- `PropertyGridRow.Caption` — Получает или задает заголовок строки. Для автоматически сгенерированных строк свойство `Caption` содержит отображаемое имя свойства.

- `PropertyGridRow.AllowEditing` — Возвращает или устанавливает, включены ли операции редактирования значений.
- `PropertyGridRow.EditorProperties` — Позволяет вам назначить пользовательский встроенный редактор значению строки. Смотрите Редактирование данных .

Пример

Следующий код XAML создает три строки данных (объекты `PropertyGridRow`) и привязывает их к полям объекта, назначенного контексту данных контрола.

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"

<mxpg:PropertyGridControl x:Name="pGrid1" SelectedObject="{Binding}" Grid.Column="1">
    <mxpg:PropertyGridRow FieldName="Caption">
    </mxpg:PropertyGridRow>
    <mxpg:PropertyGridRow FieldName="OrderNo">
    </mxpg:PropertyGridRow>
    <mxpg:PropertyGridRow FieldName="InvoiceNo">
    </mxpg:PropertyGridRow>
</mxpg:PropertyGridControl>
```

Пример

Следующий пример создает строки данных в коде:

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"

<mxpg:PropertyGridControl x:Name="pGrid1" AutoGenerateRows="False"
 SelectedObject="{Binding}" Grid.Column="1">
</mxpg:PropertyGridControl>
```

```
using Eremex.AvaloniaUI.Controls.PropertyGrid;

pGrid1.Rows.Add(new PropertyGridRow() { FieldName = "Caption" });
pGrid1.Rows.Add(new PropertyGridRow() { FieldName = "OrderNo" });
pGrid1.Rows.Add(new PropertyGridRow() { FieldName = "InvoiceNo" });
```

Создание категориальных строк

PropertyGrid может генерировать категориальные строки из атрибутов `System.ComponentModel.CategoryAttribute`, применяемых к нижележащим публичным свойствам. Соответствующие строки данных сгруппированы в этих категориальных строках.

Вы можете отключить автоматическую генерацию строк и создать пользовательский набор строк данных и категориальных строк. Используйте объекты `PropertyGridCategoryRow` для определения категориальных строк. Добавьте объекты `PropertyGridCategoryRow` в коллекцию `PropertyGridControl.Rows`, чтобы отобразить их на корневом уровне.

Основные свойства класса `PropertyGridCategoryRow` включают:

- `PropertyGridCategoryRow.Rows` — Коллекция строк, отображаемых как дочерние элементы категориальной строки. Как правило, вы добавляете строки данных (объекты `PropertyGridRow`) в эту коллекцию.

- `PropertyGridCategoryRow.Caption` — Получает или задает отображаемое имя категориальной строки. Для автоматически сгенерированных категориальных строк это свойство возвращает значение `CategoryAttribute`.

Пример

Следующий XAML-код создает две категориальные строки (*Name* и *Details*). Они имеют одну и две строки данных в качестве дочерних, соответственно.

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
<mxpg:PropertyGridControl x:Name="pGrid1" SelectedObject="{Binding}" Grid.Column="1">
    <mxpg:PropertyGridCategoryRow Caption="Name">
        <mxpg:PropertyGridRow FieldName="Caption">
        </mxpg:PropertyGridRow>
    </mxpg:PropertyGridCategoryRow>
    <mxpg:PropertyGridCategoryRow Caption="Details">
        <mxpg:PropertyGridRow FieldName="OrderNo">
        </mxpg:PropertyGridRow>
        <mxpg:PropertyGridRow FieldName="InvoiceNo">
        </mxpg:PropertyGridRow>
    </mxpg:PropertyGridCategoryRow>
</mxpg:PropertyGridControl>
```

Пример

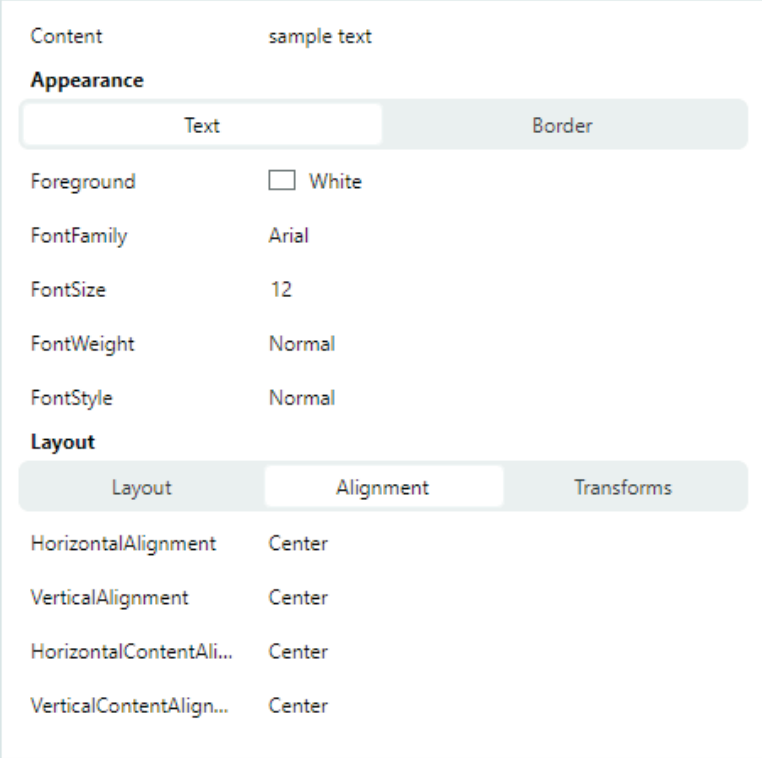
Следующий пример создает категориальные строки и помещает строки данных в созданные категории в коде.

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"

<mxpg:PropertyGridControl x:Name="pGrid1" AutoGenerateRows="False"
 SelectedObject="{Binding}" Grid.Column="1">
</mxpg:PropertyGridControl>
```

```
using Eremex.AvaloniaUI.Controls.PropertyGrid;

PropertyGridCategoryRow categoryRowName = new PropertyGridCategoryRow()
 { Caption = "Name" };
pGrid1.Rows.Add(categoryRowName);
categoryRowName.Rows.Add(new PropertyGridRow() { FieldName = "Caption" });

PropertyGridCategoryRow categoryRowDetails = new PropertyGridCategoryRow()
 { Caption = "Details" };
pGrid1.Rows.Add(categoryRowDetails);
categoryRowDetails.Rows.Add(new PropertyGridRow() { FieldName = "OrderNo" });
categoryRowDetails.Rows.Add(new PropertyGridRow() { FieldName = "InvoiceNo" });
```

Создать табуляционных строк

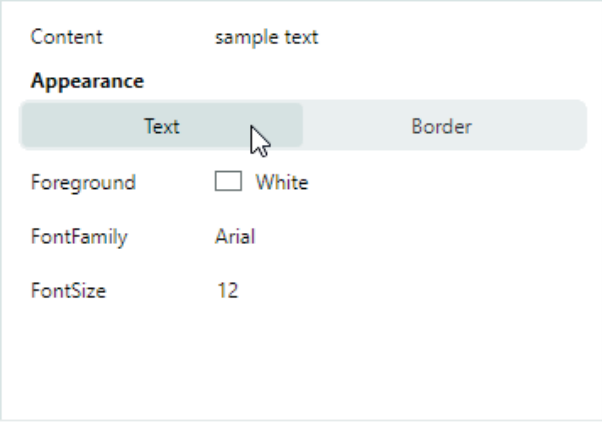
Табуляционная строка (`PropertyGridTabRow`) позволяет группировать строки в пользовательский интерфейс с вкладками. Он состоит из заголовка, переключателя вкладок и клиентской области. Когда пользователь выбирает вкладку, в клиентской области отображается набор свойств, соответствующий выбранной вкладке. На следующей картинке показан контрол PropertyGrid с двумя табуляционными строками (*Appearance* и *Layout*).:



Контрол заполняет коллекцию вкладок из дочерних элементов табуляционной строки (объекты `PropertyGridTabRowItem`). Каждый объект `PropertyGridTabRowItem` определяет коллекцию строк, связанных с этой вкладкой.

Пример

В следующем примере создается табуляционная строка *Appearance*, которая состоит из двух вкладок (*Text* и _Border_1). При выборе каждой вкладки отображается свой собственный набор свойств.




```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<UserControl.Resources>
    <DataTemplate x:Key="spinEditorTemplate1">
        <mxe:SpinEditor x:Name="PART_Editor" HorizontalContentAlignment="Stretch"/>
    </DataTemplate>
    <DataTemplate x:Key="spinEditorTemplate2">
        <mxe:SpinEditor x:Name="PART_Editor" HorizontalContentAlignment="Stretch"
         Increment="0.1"/>
    </DataTemplate>
```

```
</UserControl.Resources>

<mxpg:PropertyGridControl x:Name="propertyGrid"
                          SelectedObject="{Binding}"
                          UseModernAppearance="True"
                          ImmediatePostEditor="True"
                          BorderThickness="1,0" Grid.Column="1"
                          ShowSearchPanel="False"
                          >
    <mxpg:PropertyGridRow FieldName="Content"/>

    <mxpg:PropertyGridTabRow Caption="Appearance">
        <mxpg:PropertyGridTabRowItem Header="Text">
            <mxpg:PropertyGridRow FieldName="Foreground"/>
            <mxpg:PropertyGridRow FieldName="FontFamily">
                <mxpg:PropertyGridRow.EditorProperties>
                    <mxe:ComboBoxEditorProperties
                        IsTextEditable="False"
                        ItemsSource="{Binding Source={x:Static FontManager.Current},
                         Path=SystemFonts}"
                        ValueMember="Name" DisplayMember="Name"/>
                </mxpg:PropertyGridRow.EditorProperties>
            </mxpg:PropertyGridRow>
            <mxpg:PropertyGridRow FieldName="FontSize"
             CellTemplate="{StaticResource spinEditorTemplate1}"/>

        </mxpg:PropertyGridTabRowItem>

        <mxpg:PropertyGridTabRowItem Header="Border">
            <mxpg:PropertyGridRow FieldName="Background"/>
            <mxpg:PropertyGridRow FieldName="Opacity"
             CellTemplate="{StaticResource spinEditorTemplate2}"/>
            <mxpg:PropertyGridRow FieldName="BorderThickness"
             CellTemplate="{StaticResource spinEditorTemplate1}"/>
            <mxpg:PropertyGridRow FieldName="BorderBrush"/>
        </mxpg:PropertyGridTabRowItem>
    </mxpg:PropertyGridTabRow>
</mxpg:PropertyGridControl>
```

Использование атрибутов для настройки свойств строк

Вы можете применить атрибуты к свойствам привязанного объекта, чтобы настроить статус видимости, свойства просмотра и поведения для соответствующих строк в PropertyGridControl. Поддерживаются следующие атрибуты:

Атрибут `Browsable`

Атрибут `System.ComponentModel.BrowsableAttribute` контролирует наличие строк PropertyGrid, которые соответствуют определенным свойствам в привязанном объекте. Чтобы предотвратить создание отдельных строк, примените атрибут **Browsable(false)** к связанным свойствам.

```
public partial class MyBusinessObject : ObservableObject
{
    [Browsable(false)]
    public string Caption {
        get;set;
```

```
    }
}
```

Атрибут `Category`

Атрибут `System.ComponentModel.CategoryAttribute` задает название категории для свойства. Когда PropertyGrid обнаруживает этот атрибут, примененный к свойству, контрол создает категориальную строку с указанным именем категории и помещает соответствующую строку данных в эту категориальную строку.

```
[Category("Title")]
public string Caption {
    get;set;
}
```

Атрибут `DisplayName`

Атрибут `System.ComponentModel.DisplayNameAttribute` присваивает свойству пользовательское отображаемое имя. Когда этот атрибут указан, контрол PropertyGrid использует это отображаемое имя для подписей соответствующих строк.

```
[DisplayName("Name")]
public string Caption {
    get;set;
}
```

Атрибут `Readonly`

Атрибут `System.ComponentModel.ReadOnlyAttribute` помечает свойство как доступное только для чтения и предотвращает операции редактирования соответствующей строки в контроле PropertyGrid.

```
[ReadOnly(true)]
public string OrderId {
    get; set;
}
```

Атрибут `TypeConverter`

Атрибут `System.ComponentModel.TypeConverterAttribute` позволяет привязать объект `TypeConverter` (потомок `System.ComponentModel.TypeConverter`) к свойству.PropertyGrid использует этот конвертер для преобразования отображаемых значений в редактируемые. Функциональность конвертера типов вызывается в следующих случаях:

- Когда контрол собирается отобразить значение в ячейке или когда изменяется значение редактирования ячейки. `TypeConverter` преобразует значение редактирования ячейки в отображаемое значение.
- Когда пользователь редактирует ячейку, а затем перемещает фокус на другую ячейку. `TypeConverter` выполняет обратное преобразование.

Пользовательские шаблоны строк

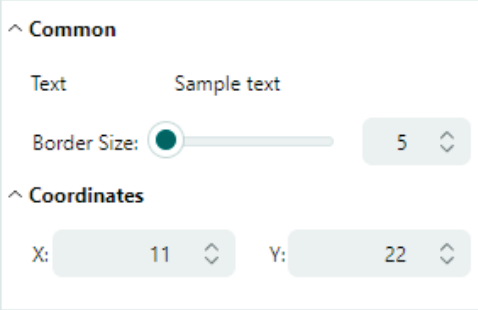
Дефолтная отрисовка строки данных состоит из областей заголовка и значений.

Вы можете использовать свойство `CellTemplate`, чтобы указать шаблон, используемый для отображения областей значений строк. Смотрите следующие разделы для получения дополнительной информации: Редактирование данных и Пользовательские редакторы

Используйте свойство `PropertyGridRow.RowTemplate` для отображения целых строк (областей заголовка и значений) произвольным способом. Это свойство определяет пользовательский шаблон строки.

Пример - Пользовательский шаблон строк

Следующий код привязывает PropertyGrid к объекту *MyBusinessObject*, который имеет свойства *BorderSize* и *Location* типов `Integer` и `Point` соответственно. Код определяет три объекта `PropertyGridRow`, два из которых используют пользовательские шаблоны. Шаблоны содержат пользовательские контролы для представления и редактирования свойств *BorderSize* и *Location*.



Шаблон строки для свойства *BorderSize* отображает метку, ползунок и `SpinEditor`. Ползунок и редактор привязаны к целевому свойству *BorderSize*.

Шаблон строки для редактирования свойства *Location* содержит две метки и два cde10074. Соответствующий объект `PropertyGridRow` привязан к свойству *Location*, в то время как объекты `SpinEditor` привязаны к вложенным полям *X* и *Y*. Альтернативной опцией является использование путей привязки *Location.X* и *Location.Y* для редакторов.

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<Window.DataContext>
    <local:SampleViewModel/>
</Window.DataContext>

<mxpg:PropertyGridControl
            x:Name="propertyGrid1"
            SelectedObject="{Binding Path=MyBusinessObject}"
            UseModernAppearance="True"
            Margin="10"
            BorderThickness="1"
            ShowSearchPanel="False"
            >

    <mxpg:PropertyGridCategoryRow Caption="Common">

        <mxpg:PropertyGridRow FieldName="Text"/>

        <mxpg:PropertyGridRow>
            <mxpg:PropertyGridRow.RowTemplate>
                <DataTemplate>
                    <Grid ColumnDefinitions="Auto, 2*, *" Margin="18,5,5,5">
                        <TextBlock Text="Border Size: "
```

```
                        VerticalAlignment="Center" Classes="PropertyGridRow_Modern"/>
                    <Slider Value="{Binding BorderSize,
                        Converter={local:MyDoubleToIntConverter}}"
                        Maximum="500" Focusable="False" Margin="0,0,5,0" Grid.Column="1"/>
                    <mxe:SpinEditor EditorValue="{Binding BorderSize}" Maximum="500"
                        Minimum="0" Grid.Column="2"/>
                </Grid>
            </DataTemplate>
        </mxpg:PropertyGridRow.RowTemplate>
    </mxpg:PropertyGridRow>
</mxpg:PropertyGridCategoryRow>

<mxpg:PropertyGridCategoryRow Caption="Coordinates">
    <mxpg:PropertyGridRow FieldName="Location">
        <mxpg:PropertyGridRow.RowTemplate>
            <DataTemplate>
                <Grid ColumnDefinitions="*, 20, *"  Margin="18,5,5,5" Grid.Column="0">
                    <Grid ColumnDefinitions="Auto, *" >
                        <TextBlock Text="X:" HorizontalAlignment="Right"
                            Classes="PropertyGridRow_Modern" VerticalAlignment="Center"/>
                        <mxe:SpinEditor EditorValue="{Binding X}" Minimum="0" Maximum="10000"
                            Grid.Column="1" VerticalAlignment="Center"/>
                    </Grid>
                    <Grid ColumnDefinitions="Auto, *" Grid.Column="2">
                        <TextBlock Text="Y:" HorizontalAlignment="Right"
                            Classes="PropertyGridRow_Modern" VerticalAlignment="Center"/>
                        <mxe:SpinEditor EditorValue="{Binding Y}" Minimum="0" Maximum="10000"
                            Grid.Column="1" VerticalAlignment="Center"/>
                    </Grid>
                </Grid>
            </DataTemplate>
        </mxpg:PropertyGridRow.RowTemplate>
    </mxpg:PropertyGridRow>
</mxpg:PropertyGridCategoryRow>
</mxpg:PropertyGridControl>
```

```
using System.Drawing;
using CommunityToolkit.Mvvm.ComponentModel;

public partial class SampleViewModel : ViewModelBase
{
    [ObservableProperty]
    MyBusinessObject myBusinessObject = new MyBusinessObject();
}
public partial class MyBusinessObject : ViewModelBase
{
    [ObservableProperty]
    string text = "Sample text";

    [ObservableProperty]
    double borderSize = 5;

    [ObservableProperty]
    string fontFamily = FontManager.Current.DefaultFontFamily.Name;

    [ObservableProperty]
    double fontSize = 14;

    [ObservableProperty]
    Point location = new Point(11, 22);
```

```
}

public class ViewModelBase : ObservableObject
{
}
```

Генерация строк из коллекции моделей представления строк (MVVM)

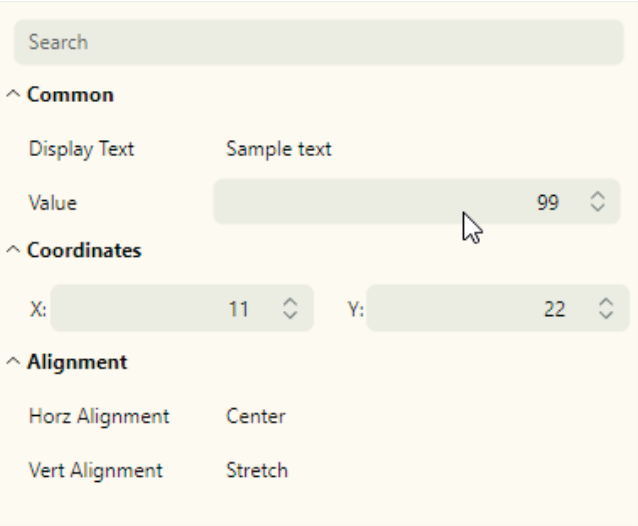
PropertyGrid позволяет вам использовать шаблон проектирования MVVM для заполнения контрола строками и инициализации строк из моделей представления.

Следующие свойства PropertyGrid поддерживают шаблон проектирования MVVM:

- `PropertyGridControl.RowsSource` — Источник моделей представления строк, которые будут отображаться как корневые строки.
- `PropertyGridCategoryRow.RowsSource` — Источник моделей представления строк, которые будут отображаться как дочерние элементы категориальной строки.
- `PropertyGridControl.RowsDataTemplates` — Коллекция шаблонов данных, которые определяют объекты `PropertyGridRow` и `PropertyGridCategoryRow`, используемые для отображения соответствующих моделей представления строк из коллекций `RowsSource`.

Пример

В следующем руководстве демонстрируется подход MVVM к заполнению строк. В этом примере инициализируются строки PropertyGrid из View Models и создается пользовательский интерфейс, показанный ниже:



В примере создаются категориальные строки *Common*, *Coordinates* и *Alignment* (объекты `PropertyGridCategoryRow`) из объектов *CategoryRowViewModel*.

Приведенные ниже модели представления используются для создания обычных строк в категориальных строках:

- *DefaultRowViewModel* — ViewModel, соответствующая строке по умолчанию (объект `PropertyGridRow` с дефолтными настройками). Тип данных значения строки по умолчанию определяет тип встроенного редактора. Объект *DefaultRowViewModel* используется для создания строк *Display Text*, *Horz Alignment* и *Vert Alignment*.

- *NumericSpinEditorRowViewModel* — ViewModel, соответствующая строке PropertyGrid со встроенным SpinEditor. Эта модель представления используется для создания строки *Value*.
- *PointEditorViewModel* — ViewModel, который соответствует строке PropertyGrid с пользовательским шаблоном строки. В этом шаблоне строк отображаются два SpinEditor и две метки, размещенные в линии для представления и редактирования значений типа данных `Point`.

Шаги:

1. Определите целевой бизнес-объект, данные которого необходимо отображать/редактировать в PropertyGrid. Привяжите этот объект к контролу с помощью элемента `PropertyGridControl.SelectedObject`.

```
public partial class MyBusinessObject : ViewModelBase
{
    [ObservableProperty]
    string text = "Sample text";

    [ObservableProperty]
    int value = 99;

    [ObservableProperty]
    double borderSize = 5;

    [property: Category("Font")]
    [ObservableProperty]
    string fontFamily = FontManager.Current.DefaultFontFamily.Name;

    [property: Category("Font")]
    [ObservableProperty]
    double fontSize = 14;

    [property: Category("Font")]
    [ObservableProperty]
    bool isBold = true;

    [property: Category("Font")]
    [ObservableProperty]
    bool isItalic;

    [property: Category("Alignment")]
    [ObservableProperty]
    HorizontalAlignment horizontalAlignment = HorizontalAlignment.Center;

    [property: Category("Alignment")]
    [ObservableProperty]
    VerticalAlignment verticalAlignment;

    [ObservableProperty]
    Point location = new Point(11, 22);
}
```

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"

<mxpg:PropertyGridControl
    SelectedObject="{Binding MyBusinessObject}"
    ...
```

```
    >
    ...
</mxpg:PropertyGridControl>
```

2. Создайте классы модели представления строк, содержащие свойства, используемые для инициализации строк PropertyGrid (например, свойства, используемые для инициализации свойств строки `PropertyGridRow.FieldName` и `PropertyGridRow.Caption`). Каждая модель представления строк обычно предоставляет уникальный набор свойств и функций. Модель представления для категориальных строк должна предоставлять объект `IEnumerable`, содержащий дочерние модели представления строк.

```
// A View Model that corresponds to regular data rows.
public class DefaultRowViewModel
{
    // The path to a target object's property.
    public string? FieldName { get; set; }

    // The property's display name.
    public string? Caption { get; set; }
}

// A View Model that corresponds to category rows.
public class CategoryRowViewModel
{
    // The category's display name.
    public string? Caption { get; set; }

    // Child row View Models that will be rendered as child rows.
    public IEnumerable? Items { get; set; }
}

// A View Model that corresponds to a data row with an embedded SpinEditor used to edit numeric values.
public class NumericSpinEditorRowViewModel
{
    public string? FieldName { get; set; }

    public string? Caption { get; set; }

    // The minimum allowed value for the SpinEditor.
    public int? MinValue { get; set; }
    // The maximum allowed value for the SpinEditor.
    public int? MaxValue { get; set; }
}

// A View Model for a data row that uses two standalone SpinEditors to edit values of the Point data type.
// This data row will be created from a custom row template.
public class PointEditorViewModel
{
    public string? FieldName { get; set; }
}
```

3. Создайте объект `IEnumerable`, который хранит экземпляры модели представления строк в порядке, который PropertyGrid должен использовать для отображения соответствующих строк.

```
public partial class SampleViewModel : ViewModelBase
{
    [ObservableProperty]
    IEnumerable myRowSource = GetMyRowSource();
```

```
    public static IEnumerable GetMyRowSource()
    {
        return new List<object>
            {
                new CategoryRowViewModel()
                {
                    Caption = "Common",
                    Items = new List<object>
                    {
                        new DefaultRowViewModel() { FieldName = "Text", Caption = "Display Text" },
                        new NumericSpinEditorRowViewModel() { FieldName = "Value", Caption = "Value",
                         MinValue=1, MaxValue=100 },
                    }
                },

                new CategoryRowViewModel()
                {
                    Caption = "Coordinates",
                    Items = new List<object>
                    {
                        new PointEditorViewModel() { FieldName = "Location" }
                    }
                },

                new CategoryRowViewModel()
                {
                    Caption = "Alignment",
                    Items = new List<object>
                    {
                        new DefaultRowViewModel() { FieldName = "HorizontalAlignment", Caption = "Horz Alignment" },
                        new DefaultRowViewModel() { FieldName = "VerticalAlignment", Caption = "Vert Alignment" },
                    }
                }
            };
    }
}
```

4. Установите свойство `PropertyGridControl.RowsSource` для созданного объекта `IEnumerable`.

```
xmlns:local="using:PropertyGridSample"
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"

<Window.DataContext>
    <local:SampleViewModel/>
</Window.DataContext>

<mxpg:PropertyGridControl
    SelectedObject="{Binding MyBusinessObject}"
    RowsSource="{Binding MyRowSource}"
    ...
    >
...
</mxpg:PropertyGridControl>
```

5. Используйте коллекцию `PropertyGridControl.RowsDataTemplates` для создания шаблонов данных, которые сопоставляют модели представления строк со строками PropertyGrid. Каждый шаблон данных должен определять объект `PropertyGridRow` или `PropertyGridCategoryRow` и инициализировать свойства строки, используя информацию, содержащуюся в соответствующей модели представления строк.

При определении объекта `PropertyGridCategoryRow` задайте свойству `PropertyGridCategoryRow.RowsSource` значение источника дочерних моделей представления строк.

```
<mxpg:PropertyGridControl
    ...>
    <mxpg:PropertyGridControl.RowsDataTemplates>
        <DataTemplates>
            <DataTemplate DataType="local:CategoryRowViewModel">
                <mxpg:PropertyGridCategoryRow
                    Caption="{Binding Path=Caption}"
                    RowsSource="{Binding Path=Items}"/>
            </DataTemplate>
            <DataTemplate DataType="local:DefaultRowViewModel">
                <mxpg:PropertyGridRow
                    FieldName="{Binding Path=FieldName}"
                    Caption="{Binding Path=Caption}"/>
            </DataTemplate>

            <DataTemplate DataType="local:NumericSpinEditorRowViewModel">
                <mxpg:PropertyGridRow FieldName="{Binding Path=FieldName}">
                    <mxpg:PropertyGridRow.EditorProperties >
                        <mxe:SpinEditorProperties
                            Minimum="{Binding MinValue}"
                            Maximum="{Binding MaxValue}"/>
                    </mxpg:PropertyGridRow.EditorProperties>
                </mxpg:PropertyGridRow>
            </DataTemplate>
            <DataTemplate DataType="local:PointEditorViewModel">
                <mxpg:PropertyGridRow
                    FieldName="{Binding Path=FieldName}"
                    RowTemplate="{DynamicResource ResourceKey=pointEditorTemplate}"/>
            </DataTemplate>
        </DataTemplates>
    </mxpg:PropertyGridControl.RowsDataTemplates>
</mxpg:PropertyGridControl>
```

> **СОВЕТ**
>
> Avalonia UI поддерживает иерархический поиск целевых `DataTemplate`s по логическому дереву. Помимо использования свойства `RowsDataTemplates`, вы можете определить шаблоны в коллекции `DataTemplates` родительского объекта контрола (parents), объекта `Window` или объекта `Application`.

5. Определите пользовательский шаблон *pointEditorTemplate* для отображения `PropertyGridRow`, который соответствует объекту *PointEditorViewModel*.

```
<Window.Resources>
    <DataTemplate x:Key="pointEditorTemplate">
        <Grid ColumnDefinitions="*, 20, *"  Margin="18,5,5,5" Grid.Column="0">
            <Grid ColumnDefinitions="Auto, *" >
                <TextBlock Text="X:" HorizontalAlignment="Right"
                 Classes="PropertyGridRow_Modern" VerticalAlignment="Center"/>
                <mxe:SpinEditor EditorValue="{Binding X}" Minimum="0" Maximum="10000"
                 Grid.Column="1" VerticalAlignment="Center"/>
            </Grid>
            <Grid ColumnDefinitions="Auto, *" Grid.Column="2">
                <TextBlock Text="Y:" HorizontalAlignment="Right"
```

```
                Classes="PropertyGridRow_Modern" VerticalAlignment="Center"/>
            <mxe:SpinEditor EditorValue="{Binding Y}" Minimum="0" Maximum="10000"
                Grid.Column="1" VerticalAlignment="Center"/>
        </Grid>
    </Grid>
</DataTemplate>
</Window.Resources>
```

Полный код

Ниже вы можете найти полный код руководства.

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
xmlns:local="using:PropertyGridSample"

<Window.DataContext>
    <local:SampleViewModel/>
</Window.DataContext>

<Window.Resources>
    <DataTemplate x:Key="pointEditorTemplate">
        <Grid ColumnDefinitions="*, 20, *"  Margin="18,5,5,5" Grid.Column="0">
            <Grid ColumnDefinitions="Auto, *" >
                <TextBlock Text="X:" HorizontalAlignment="Right"
                    Classes="PropertyGridRow_Modern" VerticalAlignment="Center"/>
                <mxe:SpinEditor EditorValue="{Binding X}" Minimum="0" Maximum="10000"
                    Grid.Column="1" VerticalAlignment="Center"/>
            </Grid>
            <Grid ColumnDefinitions="Auto, *" Grid.Column="2">
                <TextBlock Text="Y:" HorizontalAlignment="Right"
                    Classes="PropertyGridRow_Modern" VerticalAlignment="Center"/>
                <mxe:SpinEditor EditorValue="{Binding Y}" Minimum="0" Maximum="10000"
                    Grid.Column="1" VerticalAlignment="Center"/>
            </Grid>
        </Grid>
    </DataTemplate>
</Window.Resources>

<mxpg:PropertyGridControl
    Grid.Row="2" Grid.Column="5"
    Background="FloralWhite" BorderThickness="1"

    SelectedObject="{Binding MyBusinessObject}"
    RowsSource="{Binding MyRowSource}"
    UseModernAppearance="true"
    Margin="10">

<mxpg:PropertyGridControl
    Background="FloralWhite" BorderThickness="1"
    SelectedObject="{Binding MyBusinessObject}"
    RowsSource="{Binding MyRowSource}"
    UseModernAppearance="true"
    Margin="10">

    <mxpg:PropertyGridControl.RowsDataTemplates>
        <DataTemplates>
            <DataTemplate DataType="local:CategoryRowViewModel">
                <mxpg:PropertyGridCategoryRow
                    Caption="{Binding Path=Caption}"
                    RowsSource="{Binding Path=Items}"/>
```

```
            </DataTemplate>
            <DataTemplate DataType="local:DefaultRowViewModel">
                <mxpg:PropertyGridRow
                    FieldName="{Binding Path=FieldName}"
                    Caption="{Binding Path=Caption}"/>
            </DataTemplate>

            <DataTemplate DataType="local:NumericSpinEditorRowViewModel">
                <mxpg:PropertyGridRow FieldName="{Binding Path=FieldName}">
                    <mxpg:PropertyGridRow.EditorProperties >
                        <mxe:SpinEditorProperties
                            Minimum="{Binding MinValue}"
                            Maximum="{Binding MaxValue}"/>
                    </mxpg:PropertyGridRow.EditorProperties>
                </mxpg:PropertyGridRow>
            </DataTemplate>
            <DataTemplate DataType="local:PointEditorViewModel">
                <mxpg:PropertyGridRow
                    FieldName="{Binding Path=FieldName}"
                    RowTemplate="{DynamicResource ResourceKey=pointEditorTemplate}"/>
            </DataTemplate>
        </DataTemplates>
    </mxpg:PropertyGridControl.RowsDataTemplates>
</mxpg:PropertyGridControl>
```

```
using Eremex.AvaloniaUI.Controls.Common;
using Avalonia.Layout;
using System.Drawing;
using CommunityToolkit.Mvvm.ComponentModel;

namespace PropertyGridSample;

public partial class SampleViewModel : ViewModelBase
{
    [ObservableProperty]
    IEnumerable myRowSource = GetMyRowSource();

    [ObservableProperty]
    MyBusinessObject myBusinessObject = new MyBusinessObject();

    public static IEnumerable GetMyRowSource()
    {
        return new List<object>
            {
                new CategoryRowViewModel()
                {
                    Caption = "Common",
                    Items = new List<object>
                    {
                        new DefaultRowViewModel() { FieldName = "Text", Caption = "Display Text" },
                        new NumericSpinEditorRowViewModel() { FieldName = "Value", Caption = "Value",
                         MinValue=1, MaxValue=100 },
                    }
                },

                new CategoryRowViewModel()
                {
                    Caption = "Coordinates",
                    Items = new List<object>
                    {
```

```
                        new PointEditorViewModel() { FieldName = "Location" }
                    }
                },

                new CategoryRowViewModel()
                {
                    Caption = "Alignment",
                    Items = new List<object>
                    {
                        new DefaultRowViewModel() { FieldName = "HorizontalAlignment", Caption = "Horz Alignment" },
                        new DefaultRowViewModel() { FieldName = "VerticalAlignment", Caption = "Vert Alignment" },
                    }
                }
            };
    }
}

// A View Model that corresponds to regular data rows.
public class DefaultRowViewModel
{
    // The path to a target object's property.
    public string? FieldName { get; set; }

    // The property's display name.
    public string? Caption { get; set; }
}

// A View Model that corresponds to category rows.
public class CategoryRowViewModel
{
    // The category's display name.
    public string? Caption { get; set; }

    // Child row View Models that will be rendered as child rows.
    public IEnumerable? Items { get; set; }
}

// A View Model that corresponds to a data row with an embedded SpinEditor used to edit numeric values.
public class NumericSpinEditorRowViewModel
{
    public string? FieldName { get; set; }

    public string? Caption { get; set; }

    // The minimum allowed value for the SpinEditor.
    public int? MinValue { get; set; }
    // The maximum allowed value for the SpinEditor.
    public int? MaxValue { get; set; }
}

// A View Model for a data row that uses two standalone SpinEditors to edit values of the Point data type.
// This data row will be created from a custom row template.
public class PointEditorViewModel
{
    public string? FieldName { get; set; }
}

public partial class MyBusinessObject : ViewModelBase
{
    [ObservableProperty]
    string text = "Sample text";
```

```
    [ObservableProperty]
    int value = 99;

    [ObservableProperty]
    double borderSize = 5;

    [property: Category("Font")]
    [ObservableProperty]
    string fontFamily = FontManager.Current.DefaultFontFamily.Name;

    [property: Category("Font")]
    [ObservableProperty]
    double fontSize = 14;

    [property: Category("Font")]
    [ObservableProperty]
    bool isBold = true;

    [property: Category("Font")]
    [ObservableProperty]
    bool isItalic;

    [property: Category("Alignment")]
    [ObservableProperty]
    HorizontalAlignment horizontalAlignment = HorizontalAlignment.Center;

    [property: Category("Alignment")]
    [ObservableProperty]
    VerticalAlignment verticalAlignment;

    [ObservableProperty]
    Point location = new Point(11, 22);
}
```

Поиск данных

Панель поиска

Встроенная панель поиска контрола позволяет пользователям выполнять поиск свойств привязанного объекта(ов). Когда пользователь вводит текст в поле поиска, контрол PropertyGrid фильтрует строки, чтобы отображать только те, которые соответствуют тексту поиска.

Используйте свойство `ShowSearchPanel`, чтобы контролировать видимость окна поиска.

Редактирование данных

Встроенные редакторы Eremex, используемые по умолчанию

Если вы явно не указываете встроенные редакторы для строк, контрол PropertyGrid использует специальные встроенные редакторы Eremex для отображения и редактирования значений строк распространенных типов данных:

- Логические значения — `CheckEditor`
- Двойные значения — `SpinEditor`
- Значения перечисления — `ComboBoxEditor`
- Свойства с атрибутом `TypeConverter`, метод `TypeConverter.GetStandardValuesSupported` которого возвращает `true` — `ComboBoxEditor`
- Другие значения — `TextEditor`

Вы можете динамически получать доступ к экземплярам встроенных редакторов Eremex и изменять их, когда эти редакторы активированы. Смотрите раздел Доступ к активному встроенному редактору Eremex для получения более подробной информации.

Назначение встроенных редакторов Eremex

Вы можете явно назначить строкам встроенные редакторы Eremex, если вам нужно переопределить дефолтное назначение редактора или настроить редакторы строк в XAML или code-behind.

Используйте свойство `PropertyGridRow.EditorProperties` для этой цели следующим образом:

1. Создайте и настройте экземпляр вспомогательного класса `...EditorProperties`, который хранит свойства, специфичные для требуемого встроенного редактора. Все эти вспомогательные классы являются потомками `BaseEditorProperties`:
   - `ButtonEditorProperties` — Содержит свойства, специфичные для контрола `ButtonEditor`.
   - `CheckEditorProperties` — Содержит свойства, специфичные для контрола `CheckEditor`.
   - `ColorEditorProperties` — Содержит свойства, специфичные для контрола `ColorEditor`.
   - `ComboBoxEditorProperties` — Содержит свойства, специфичные для контрола `ComboBoxEditor`.

- `HyperlinkEditorProperties` — Содержит свойства, специфичные для контрола `HyperlinkEditor`.
- `PopupColorEditorProperties` — Содержит свойства, специфичные для контрола `PopupColorEditor`.
- `PopupEditorProperties` — Содержит свойства, специфичные для контрола `PopupEditor`.
- `SegmentedEditorProperties` — Содержит свойства, специфичные для контрола `SegmentedEditor`.
- `SpinEditorProperties` — Содержит свойства, специфичные для контрола `SpinEditor`.
- `TextEditorProperties` — Содержит свойства, специфичные для контрола `TextEditor`.

2. Установите для свойства `PropertyGridRow.EditorProperties` значение созданного экземпляра `...EditorProperties`.

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<mxpg:PropertyGridRow FieldName="OrderNo">
    <mxpg:PropertyGridRow.EditorProperties >
        <mxe:ButtonEditorProperties TextWrapping="Wrap"/>
    </mxpg:PropertyGridRow.EditorProperties>
</mxpg:PropertyGridRow>
```

Свойство `PropertyGridRow.CellTemplate` - это еще один способ назначить строке редактор Eremex. Убедитесь, что в редакторе Eremex для свойства `x:Name` установлено значение **"PART_Editor"**. В этом случае PropertyGrid автоматически привязывает свойство редактора `EditorValue` к привязанному полю строки. Кроме того, PropertyGrid начинает поддерживать свойства внешнего вида встроенного редактора (видимость границ и цвет отображаемого текста в активном и неактивном состояниях).

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
...
<mxpg:PropertyGridRow FieldName="Caption">
    <mxpg:PropertyGridRow.CellTemplate>
        <DataTemplate>
            <mxe:ButtonEditor x:Name="PART_Editor">
                <mxe:ButtonEditor.Buttons>
                    <mxe:ButtonSettings Content="..."/>
                </mxe:ButtonEditor.Buttons>
            </mxe:ButtonEditor>
        </DataTemplate>
    </mxpg:PropertyGridRow.CellTemplate>
</mxpg:PropertyGridRow>
```

Пользовательские редакторы

PropertyGrid позволяет вам указывать пользовательские редакторы для ячеек строк. Вы можете назначить редактор непосредственно определенной строке или динамически назначать редакторы строкам на основе типа данных нижележащего объекта строки. Смотрите следующий раздел для получения дополнительной информации: Пользовательские редакторы .

Доступ к активному встроенному редактору Eremex

Чтобы повысить производительность контрола, PropertyGrid использует облегченные версии встроенных редакторов Eremex при отображении значений ячеек в режиме отображения (когда редактирование текста не активно). Когда в ячейке начинается операция редактирования текста, встроенный редактор переключается со своей облегченной версии на полнофункциональную. В этом случае вы можете использовать свойство `ActiveEditor` для доступа к экземпляру редактора Eremex. Когда ячейка строки теряет фокус, редактор переключается обратно на свою облегченную версию, а свойство `ActiveEditor` возвращает значение `null`.

Обработайте событие `ShowingEditor`, чтобы получать уведомления при запуске операций редактирования в ячейках строк. Вы можете безопасно получить доступ к свойству `ActiveEditor` в обработчике событий `ShowingEditor`.

Как отключить активацию редактора

Чтобы предотвратить активацию редактора ячеек в определенных случаях, вы можете обработать событие `ShowingEditor` и установить для параметра обработчика событий `Cancel` значение `true`.

Пользовательские редакторы

`PropertyGrid` позволяет встраивать пользовательские редакторы в ячейки. Вы можете назначить редактор непосредственно определенной строке или динамически назначать редакторы строкам на основе типа данных нижележащего объекта строки.

Смотрите следующий раздел для получения информации о редакторах по умолчанию: Редактирование данных .

Назначение редактора непосредственно ячейке строки

Используйте свойство `PropertyGridRow.CellTemplate`, чтобы назначить редактор определенной строке. Для выполнения этой задачи выполните следующие действия:

- Создайте объект DataTemplate с редактором, определенным внутри шаблона.
- Присвойте DataTemplate свойству `PropertyGridRow.CellTemplate`.
- При необходимости явно привяжите редактор к привязанному полю строки.

В следующем примере показан код XAML, который инициализирует свойство `CellTemplate`:

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
...
<mxpg:PropertyGridControl x:Name="pGrid1" SelectedObject="{Binding}"
 Grid.Column="1">
    <mxpg:PropertyGridCategoryRow Caption="MyCategory1">
        <mxpg:PropertyGridRow FieldName="Number">
            <mxpg:PropertyGridRow.CellTemplate>
                <DataTemplate>
                    <TextBox Text="{Binding Value}"/>
                </DataTemplate>
            </mxpg:PropertyGridRow.CellTemplate>
        </mxpg:PropertyGridRow>
    </mxpg:PropertyGridCategoryRow>
</mxpg:PropertyGridControl>
```

Неявная привязка данных для редакторов Eremex

Если вы используете редактор Eremex внутри шаблона, вы можете опустить явную привязку данных к привязанному полю строки для редактора. Убедитесь, что в редакторе Eremex для свойства `x:Name` установлено значение **"PART_Editor"**. В этом случае PropertyGrid автоматически привязывает свойство редактора `EditorValue` к полю строки. Кроме того, PropertyGrid начинает поддерживать свойства внешнего вида встроенного редактора (видимость границ и цвет отображаемого текста в активном и неактивном состояниях).

```
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"
...
<mxpg:PropertyGridRow FieldName="Caption">
    <mxpg:PropertyGridRow.CellTemplate>
        <DataTemplate>
            <mxe:ButtonEditor x:Name="PART_Editor">
                <mxe:ButtonEditor.Buttons>
                    <mxe:ButtonSettings Content="..."/>
                </mxe:ButtonEditor.Buttons>
            </mxe:ButtonEditor>
        </DataTemplate>
    </mxpg:PropertyGridRow.CellTemplate>
</mxpg:PropertyGridRow>
```

Явная привязка данных

Явная привязка данных требуется в следующих случаях:

- Вы используете редактор, который не является редактором Eremex. Все редакторы Eremex являются производными от класса `Eremex.AvaloniaUI.Controls.Editors.BaseEditor`.
- Вам необходимо указать пользовательский конвертер значений в выражении привязки данных.

```
<DataTemplate DataType="pepa:Color">
    <mxe:PopupColorEditor x:Name="PART_Editor" EditorValue="{Binding Value,
     Converter={ecadpg:EcadColorToAvaloniaColorConverter}}"/>
</DataTemplate>
```

Динамическое назначение редакторов на основе типа данных строки

Контрол PropertyGrid может автоматически назначать шаблоны ячейкам строки на основе типа данных привязанного поля строки. Используйте свойство `PropertyGridControl.CellTemplate` для этой цели.

Следующий пример привязывает TextEditor (рисует значения зеленым цветом) к строкам, привязанным к целочисленным полям:

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:mxpg="clr-namespace:Eremex.AvaloniaUI.Controls.PropertyGrid;assembly=Eremex.Avalonia.Controls"
...
<mxpg:PropertyGridControl.CellTemplate>
    <DataTemplate DataType="sys:Int32">
        <mxe:TextEditor x:Name="PART_Editor1" EditorValue="{Binding Value}" Foreground="Green"/>
    </DataTemplate>
</mxpg:PropertyGridControl.CellTemplate>
```

Используйте следующий подход, если у вас есть список объектов DataTemplate, связанных с различными типами данных, и вы хотите назначить этот список контролу PropertyGrid.:

- Создайте пользовательский класс в коде, который находит DataTemplate на основе типа данных и создает контрол, связанный с этим типом данных, следующим образом:

```
using Avalonia.Collections;
using Avalonia.Controls.Templates;

namespace AvaloniaApplication1.Views;

public class CellTemplateLocator : AvaloniaList<IDataTemplate>, IDataTemplate
{
    public Control Build(object? param)
    {
        return this.First(x => x.Match(param)).Build(param);
    }

    public bool Match(object? data)
    {
        return this.Any(x => x.Match(data));
    }
}
```

> **СОВЕТ**
>
> Параметр *data* метода `Match` определяет значение целевого шаблона ячейки. Параметр *data* содержит значение привязанного свойства по умолчанию. Вы можете обработать событие `CustomCellTemplateData`, чтобы предоставить пользовательский объект в качестве значения шаблона ячейки. Указанный пользовательский объект будет передан методу `Match`.

- Инициализируйте свойство `PropertyGridControl.CellTemplate` объектом *CellTemplateLocator*.
- Заполните объект *CellTemplateLocator* объектами DataTemplate, связанными с вашими типами данных.

В следующем примере определены два объекта DataTemplate, связанные со строковым и целочисленным типами данных соответственно.

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:AvaloniaApplication1.Views"

<mxpg:PropertyGridControl.CellTemplate>
    <local:CellTemplateLocator>
        <DataTemplate DataType="sys:String">
            <mxe:TextEditor x:Name="PART_Editor" EditorValue="{Binding Value}" Foreground="Red"/>
        </DataTemplate>
        <DataTemplate DataType="sys:Int32">
            <mxe:TextEditor x:Name="PART_Editor1" EditorValue="{Binding Value}" Foreground="Green"/>
        </DataTemplate>
    </local:CellTemplateLocator>
</mxpg:PropertyGridControl.CellTemplate>
```

Часто используемый API

Свойства

Свойство	Описание
`ActiveEditor`	Возвращает активный редактор ячеек Eremex. Свойство возвращает `null`, если в данный момент ни один редактор не активен или когда в выделенной ячейке активен не-Eremex редактор.
`AllowEditing`	Возвращает или устанавливает, включены ли операции редактирования ячеек.
`AutoGenerateRows`	Получает или устанавливает, будет ли пустой контрол автоматически генерировать строки для всех публичных свойств, доступных в привязанных объектах, при инициализации контрола. Если контрол уже содержит строки, автоматическая генерация строк отключена. Чтобы вручную заполнить контрол строками, вы можете использовать метод `PopulateRows` или добавить строки в коллекцию `Rows`.
`CellTemplate`	Получает или задает шаблон для отображения редакторов ячеек во всех ячейках. Вы можете использовать свойство `PropertyGridRow.CellTemplate`, чтобы указать редактор ячеек для определенных строк.
`FocusedRow`	Получает или устанавливает сфокусированную строку. Это свойство позволяет вам сфокусировать определенную строку.
`Rows`	Возвращает строки контрола, отображаемые на корневом уровне. Для категориальных строк (объектов `PropertyGridCategoryRow`), хранящихся в этой коллекции, вы можете использовать свойство `PropertyGridCategoryRow.Rows` для доступа к их дочерним строкам.
`RowsDataTemplates`	Определяет шаблоны данных, используемые для отображения строк из источника строк (свойство `RowsSource`).

Свойство	Описание
`RowsSource`	Указывает источник данных для генерации строк с использованием шаблонов данных (`RowsDataTemplates`).
`SelectedObject`	Получает или задает объект, свойства которого отображаются в контроле.
`SelectedObjects`	Получает или задает список объектов, свойства которых отображаются в контроле.
`UseModernAppearance`	Получает или устанавливает, будет ли контрол отрисован с использованием стиля `Modern` или `Classic` .

Методы

Метод	Описание
`CloseEditor`	Скрывает активный редактор и сохраняет внесенные изменения.
`HideEditor`	Скрывает активный редактор и отменяет внесенные изменения.
`PopulateRows`	Генерирует строки для публичных свойств, предоставляемых привязанными объектами (свойствами) (`SelectedObject` и `SelectedObjects`). Существующие строки очищаются перед началом генерации строк.
`PostEditor`	Сохраняет изменения, внесенные в ячейку, в привязанный объект(ы).
`ShowEditor`	Активирует редактор выделенной строки.

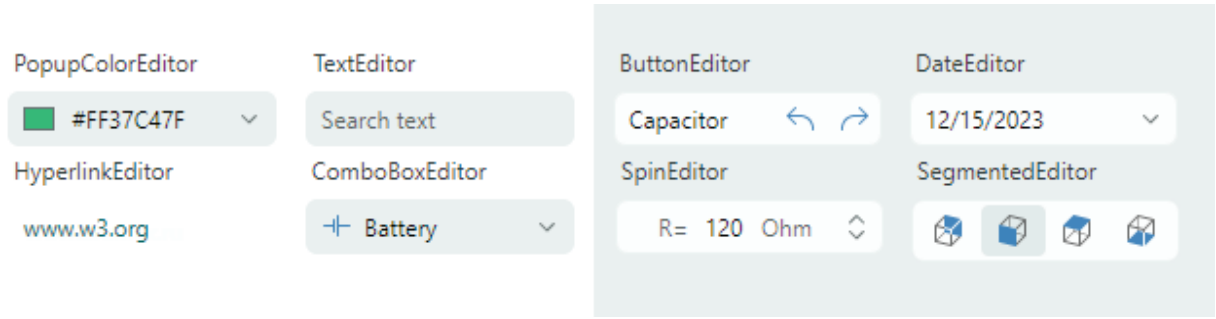
События

Событие	Описание
`CellValueChanged`	Срабатывает при изменении значения ячейки.
`CustomCellTemplateData`	Позволяет вам предоставить пользовательский объект в качестве значения шаблона ячейки. Смотрите следующий раздел для

Событие	Описание
	получения дополнительной информации: Пользовательские редакторы .
`ShowingEditor`	Срабатывает при активации редактора ячеек.

Редакторы

Библиотека контролов Eremex включает в себя несколько редакторов, которые предоставляют расширенные возможности редактирования данных. Редакторы позволяют отображать и редактировать данные различных типов (числовые, логические, дата-время, перечисления и т.д.). Они поддерживают механизм валидации данных для информирования пользователей об ошибках во время ввода данных.



Вы можете встроить редакторы данных Eremex в ячейки в контролах-контейнерах (DataGrid, TreeList, PropertyGrid и ToolbarManager) для представления и редактирования данных ячеек. Хотя вы можете встроить в ячейки любой пользовательский контрол, использование редакторов данных Eremex имеет много преимуществ с точки зрения производительности приложения.

- ComboBoxEditor — Позволяет пользователю выбрать элемент из списка элементов, отображаемого в связанном всплывающем окне.
    - Поддерживаемые источники объектов: список строк, список бизнес-объектов, а также тип-перечисление.
    - Поддержка шаблонов данных, используемых для отображения элементов произвольным способом.
    - Режимы выбора одного и нескольких элементов.
    - Встроенные флажки в режиме множественного выбора.
    - Функция автозавершения текста предсказывает выбор элемента, когда пользователь начинает вводить текст в поле редактирования в режиме одиночного выбора.

- SegmentedEditor — Отображает сегменты (элементы), один из которых может быть выбран пользователем.

    - Горизонтальное расположение сегментов.
    - Пользователь может щелкнуть по сегменту, чтобы выбрать его, и отменить выбор других сегментов.
    - Щелчок мышью с Ctrl по выбранному элементу очищает выделение.
    - Поддерживаемые источники объектов: список строк, список бизнес-объектов, а также тип-перечисление.
    - Используйте шаблоны данных для отображения элементов произвольным способом.

- ButtonEditor — Текстовый редактор со встроенными кнопками.
    - Обычные кнопки и переключатели.
    - Отображение текста и картинок на кнопках.
    - Выравнивание кнопок по левому и правому краям.
    - Всплывающие подсказки.
    - Встроенная кнопка "x" для удаления значения редактора.
    - Водяные знаки.

- CheckEditor — Отображает флажок, который переключается одним щелчком мыши.
    - Поддерживает два или три состояния нажатия (состояние нажатия, ненажатое состояние и неопределенное состояние).
    - Механизм валидации изменяет внешний вид контрола, чтобы информировать пользователей об ошибках.

- PopupColorEditor — Позволяет пользователю выбрать цвет во всплывающем окне.
    - Три цветовые палитры — по умолчанию, стандартная, пользовательская.
    - Дефолтная цветовая палитра может быть инициализирована в коде.
    - Стандартная цветовая палитра отображает предопределенные стандартные цвета.
    - Пользовательская цветовая палитра позволяет пользователям добавлять и изменять цвета с помощью встроенного диалога Выбора Цвета.
    - Возможность задания цветов в форматах RGB и HSB.

- DateEditor — Редактор со встроенным всплывающим календарем, который позволяет пользователям выбирать дату.
    - Встроенные кнопки "Сегодня" и "Очистить".
    - Поддержка нескольких форматов отображения даты.

    - Панель навигации в всплывающем календаре позволяет просматривать данные по месяцам и годам.
    - Три вида календаря: просмотр по месяцам, просмотр по годам и просмотр диапазона лет.
    - Опция для ограничения доступного диапазона дат.

- HyperlinkEditor — Отображает гиперссылку, по которой можно перейти.
    - Позволяет указать команду для обработки переходов по гиперссылке.

- `PopupEditor` — Текстовый редактор с связанным всплывающим окном.
    - Позволяет вам встроить любой контрол во всплывающее окно.

- SpinEditor — Позволяет редактировать числовые значения с помощью кнопок прокрутки.
    - Встроенные кнопки прокрутки позволяют пользователю увеличивать и уменьшать значение.
    - Ограничение доступного диапазона значений.
    - Пользовательское значение инкремента.
    - Отображение пользовательского префикса и суффикса в поле редактирования.

- `TextEditor` — текстовый редактор с базовыми функциями редактирования текста.
    - Предок всех текстовых редакторов Eremex.
    - Поддержка механизма валидации данных, используемого для показа ошибок пользователям.

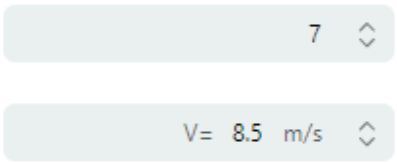
Общие функции

- Темы приложений Eremex
    - Темы приложений Eremex позволяют изменять внешний вид контролов Eremex.
    - Вы можете применить темы приложений Eremex к набору стандартных контролов Avalonia UI.
    - Редакторы Eremex поддерживают основные и дополнительные цветовые варианты для каждой темы. Эти варианты цвета помогут вам придать редакторам несколько иной цветовой акцент, изменив одно свойство.

- Валидация данных
    - Встроенный механизм валидации значений позволяет показывать ошибки пользователям во всех текстовых редакторах и CheckEditor.
    - Текстовые редакторы могут отображать ошибки валидации внутри полей редактирования или под ними.

SpinEditor

`SpinEditor` - это редактор числовых значений со встроенными кнопками прокрутки, используемыми для увеличения и уменьшения числа на определенное значение (инкремент). Пользователь может увеличивать и уменьшать число, нажимая эти кнопки или нажимая стрелки вверх и вниз на клавиатуре.



Основные функции контрола включают в себя:

- Встроенные кнопки прокрутки позволяют пользователю увеличивать и уменьшать значение.
- Ограничение доступного диапазона значений.
- Пользовательское значение инкремента.
- Отображение префикса и суффикса пользовательского значения в поле редактирования.

Получение и установка значения редактора

Используйте свойство `SpinEditor.Value`, чтобы вернуть и указать числовое значение редактора. Вы также можете использовать свойство `EditorValue` для той же цели. Эти свойства синхронизированы. Они различаются типом значения: свойство `Value` имеет десятичный тип, в то время как свойство `EditorValue` имеет тип `object`, как и во всех редакторах Eremex.

Смотрите также: Увеличение и уменьшение значения в коде .

Настройка

Следующие основные свойства помогут вам настроить контрол `SpinEditor`:

- `SpinEditor.Minimum` — Минимальное значение для редактора.
- `SpinEditor.Maximum` — Максимальное значение для редактора.
- `SpinEditor.Increment` — значение, на которое увеличивается или уменьшается значение редактора, когда пользователь нажимает кнопки прокрутки или клавиши со стрелками вверх и

вниз.

- `SpinEditor.ShowEditorButtons` — Указывает, отображать ли встроенные кнопки прокрутки.

Пример - Как создать SpinEditor

В следующем примере инициализируется контрол SpinEditor.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"

<mxe:SpinEditor x:Name="SpinEditor" Width="200"
    Value="7"
    Maximum="15"
    Minimum="1"
    Increment="1"/>
```

Префикс и суффикс отображаемого значения

Используйте свойства `SpinEditor.Prefix` и `SpinEditor.Suffix`, чтобы указать текст, отображаемый до и после числового значения. Пользователи не могут редактировать этот текст.

Пример - Как указать префикс и суффикс для значения SpinEditor

В следующем примере используются свойства `SpinEditor.Prefix` и `SpinEditor.Suffix` для отображения текста до и после значения редактора.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"

<mxe:SpinEditor x:Name="SpinEditor1" Width="200"
    Value="8.5"
    Maximum="35"
    Minimum="2"
    Increment="0.5"
    Prefix="V=" Suffix="m/s"/>
```

Увеличение и уменьшение значения в коде

Команды `SpinEditor.IncreaseCommand` и `SpinEditor.DecreaseCommand` увеличивают и уменьшают значение редактора на значение инкремента (`SpinEditor.Increment`).

Вы также можете изменить значение редактора напрямую, используя свойство `SpinEditor.Value` .

SegmentedEditor

`SegmentedEditor` представляет набор элементов (опций) в виде горизонтально расположенных сегментов. Пользователь может щелкнуть по одному из сегментов, чтобы выбрать соответствующую опцию, или нажать CTRL на выбранном сегменте, чтобы отменить выделение.

Основные функции контрола включают в себя:

- Заполнение сегментов из списка строк, списка бизнес-объектов или из тип-перечисления.
- Шаблоны элементов позволяют отображать сегменты произвольным способом.
- Использование контрола в качестве встроенного редактора в контролах-контейнерах (например, TreeList, TreeView и PropertyGrid).

Источник объектов

Используйте свойство `SegmentedEditor.ItemsSource`, чтобы указать источник объектов, используемый для создания сегментов контрола. Вы можете привязать редактор к списку строк, списку бизнес-объектов или к типу-перечислению.

Привязка к списку строк

Самый простой источник объектов - это список строк.

Пример - Как привязать к списку строк

В следующем примере контрол `SegmentedEditor` заполняется списком строк.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:col="using:System.Collections"

<mxe:SegmentedEditor>
    <mxe:SegmentedEditor.ItemsSource>
        <col:ArrayList>
            <sys:String>Montevideo</sys:String>
            <sys:String>Havana</sys:String>
            <sys:String>Santiago</sys:String>
            <sys:String>La Paz</sys:String>
        </col:ArrayList>
```

```
    </mxe:SegmentedEditor.ItemsSource>
</mxe:SegmentedEditor>
```

Привязка к списку бизнес-объектов

Вы можете привязать контрол `SegmentedEditor` к списку бизнес-объектов. В этом случае дефолтное поведение контрола является следующим:

- Метод `ToString` бизнес-объекта задает дефолтное представление текста элементов.
- Когда вы выбираете элемент, значение редактора (`SegmentedEditor.EditorValue`) устанавливается равным соответствующему бизнес-объекту.

Типичный бизнес-объект обладает множеством свойств. Вы можете указать, какие свойства бизнес-объекта предоставляют текст элемента для отображения и редактирования значений. Для этой цели используйте следующие элементы API:

- `SegmentedEditor.DisplayMember` - Получает или задает имя свойства бизнес-объекта, которое определяет текст элемента для отображения.
- `SegmentedEditor.ValueMember` - Получает или задает имя свойства бизнес-объекта, которое определяет значения элемента. Когда вы выбираете элемент, значение редактора (`SegmentedEditor.EditorValue`) устанавливается равным значению свойства `ValueMember` элемента.

Пример - Как привязать к списку бизнес-объектов

Следующий пример привязывает контрол SegmentedEditor к списку бизнес-объектов *Product*. Свойство *Product.ProductName* определяет текст элемента для отображения. Свойство *Product.ProductID* определяет значения элементов.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"

<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>

<mxe:SegmentedEditor
    Name="segmEditor1"
    ItemsSource="{Binding Products}"
    DisplayMember="ProductName"
    ValueMember="ProductID"/>
```

```
public MainViewModel()
{
```

```
    Products = new ObservableCollection<Product>();
    Products.Add(new Product(0, "Chai", "Beverages", 200));
    Products.Add(new Product(1, "Chang", "Beverages", 100))
    Products.Add(new Product(3, "Ikura", "Seafood", 500));
    Products.Add(new Product(5, "Tofu", "Produce", 430));
    //...
}

public partial class Product :ObservableObject
{
    public Product(int productID, string productName, string category, int productPrice)
    {
        ProductID = productID;
        ProductName = productName;
        Category = category;
        ProductPrice = productPrice;
    }

    [ObservableProperty]
    public int productID;

    [ObservableProperty]
    public string productName;

    [ObservableProperty]
    public string category;

    [ObservableProperty]
    public decimal productPrice;
}
```

Привязка к перечислению

`SegmentedEditor` может заполнять свои сегменты значениями с тип-перечислением.

Вспомогательный класс `Eremex.AvaloniaUI.Controls.Common.EnumItemsSource` облегчает привязку к перечислению. Его основные функции включают:

- Картинки для элементов перечисления. Примените атрибут `Eremex.AvaloniaUI.Controls.Common.ImageAttribute` к целевым элементам перечисления, чтобы указать картинки.
- Пользовательские отображаемые имена для элементов перечисления. Используйте атрибут `System.ComponentModel.DataAnnotations.DisplayAttribute` или пользовательский конвертер, чтобы изменить дефолтный отображаемый текст элемента.
- Всплывающие подсказки для элементов. Всплывающая подсказка содержит описание целевого элемента, которое вы можете предоставить с помощью атрибута

`System.ComponentModel.DataAnnotations.DisplayAttribute` или пользовательского конвертера.

Чтобы настроить привязку к типу-перечислению, используйте следующие свойства `EnumItemsSource`:

- `EnumItemsSource.EnumType` — Задает тип-перечисление, значения которого отображаются в контроле `SegmentedEditor`.
- `EnumItemsSource.ShowImages` — Указывает, следует ли отображать картинки для элементов перечисления. Вы можете предоставить картинки, используя атрибут `Eremex.AvaloniaUI.Controls.Common.ImageAttribute`.
- `EnumItemsSource.ShowNames` — Указывает, следует ли отображать текст элемента. Установите для `ShowNames` значение `false`, а для `ShowImages` значение `true`, чтобы отображать элементы перечисления с использованием картинок без текста.
- `EnumItemsSource.ImageSize` — Определяет размер отображения картинок, назначенных элементам перечисления.
- `EnumItemsSource.NameToDisplayTextConverter` — Позволяет назначить конвертер, который извлекает пользовательский текст для отображения для элементов перечисления.
- `EnumItemsSource.NameToDescriptionConverter` — Позволяет назначить конвертер, который извлекает описания элементов перечисления, которые отображаются в виде всплывающих подсказок, когда пользователь наводит курсор мыши на сегменты.

Пример - Как отобразить значения перечисления и использовать атрибуты для предоставления текста и картинок для элементов перечисления.

В следующем примере отображаются значения перечисления *ProductCategoryEnum* в `SegmentedEditor`. В нем используется класс `EnumItemsSource` для привязки данных.

Атрибуты `System.ComponentModel.DataAnnotations.DisplayAttribute` и `Eremex.AvaloniaUI.Controls.Common.ImageAttribute` определяют пользовательский текст для отображения, описания (всплывающие подсказки) и картинки для элементов перечисления.

```
xmlns:mxcom="clr-namespace:Eremex.AvaloniaUI.Controls.Common;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"

<mxe:SegmentedEditor Name="segmentedEditorEnum"
                ItemsSource="{mxcom:EnumItemsSource EnumType=local:ProductCategoryEnum,
                     ImageSize='16, 16', ShowImages=True, ShowNames=True}"
                >
</mxe:SegmentedEditor>
```

```
using Eremex.AvaloniaUI.Controls.Common;
using System.ComponentModel.DataAnnotations;
```

```
public enum ProductCategoryEnum
{
    // The images assigned to the enumeration values below are placed
    // in the EditorsSample/Images folder.
    // They have their "Build Action" properties set to "AvaloniaResource".
    [Image($"avares://EditorsSample/Images/Products/DairyProducts.svg")]
    [Display(Name = "Dairy Products", Description = "Products made from milk")]
    DairyProducts,

    [Image($"avares://EditorsSample/Images/Products/Beverages.svg")]
    [Display(Description = "Edible drinks")]
    Beverages,

    [Image($"avares://EditorsSample/Images/Products/Condiments.svg")]
    [Display(Description = "Flavor Enhancers")]
    Condiments,

    [Image($"avares://EditorsSample/Images/Products/Confections.svg")]
    [Display(Description = "Sweets")]
    Confections
}
```

Пример - Как отобразить значения перечисления и использовать пользовательские конвертеры для предоставления текста для элементов перечисления.

В следующем примере для отображения значений типа-перечисления в контроле `SegmentedEditor` используется класс `EnumItemsSource`. Объекты `EnumItemsSource.NameToDisplayTextConverter` и `EnumItemsSource.NameToDescriptionConverter` предоставляют пользовательский текст для отображения и описания (всплывающие подсказки) для элементов перечисления.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:mxcom="clr-namespace:Eremex.AvaloniaUI.Controls.Common;assembly=Eremex.Avalonia.Controls"
xmlns:local="clr-namespace:EditorsSample"

<mxe:SegmentedEditor Name="segmentedEditorEnumWithConverters"
                ItemsSource="{mxcom:EnumItemsSource EnumType=local:ProductCategoryEnum,
        ImageSize='16, 16', ShowImages=True, ShowNames=True,
        NameToDisplayTextConverter={local:EnumMemberNameToDisplayTextConverter},
        NameToDescriptionConverter={local:EnumMemberNameToDescriptionConverter}}"
                >
</mxe:SegmentedEditor>
```

```
using Eremex.AvaloniaUI.Controls.Common;

public enum ProductCategoryEnum
{
    // The images assigned to the enumeration values below are placed
    // in the EditorsSample/Images folder.
```

```
    // They have their "Build Action" properties set to "AvaloniaResource".
    [Image($"avares://EditorsSample/Images/Products/DairyProducts.svg")]
    DairyProducts,

    [Image($"avares://EditorsSample/Images/Products/Beverages.svg")]
    Beverages,

    [Image($"avares://EditorsSample/Images/Products/Condiments.svg")]
    Condiments,

    [Image($"avares://EditorsSample/Images/Products/Confections.svg")]
    Confections
}

public class EnumMemberNameToDisplayTextConverter : BaseEnumConverter
{
    protected override void PopulateDictionary()
    {
        TextValueDictionary.Add("ProductCategoryEnum_DairyProducts", "Dairy");
    }
}
public class EnumMemberNameToDescriptionConverter : BaseEnumConverter
{
    protected override void PopulateDictionary()
    {
        TextValueDictionary.Add("ProductCategoryEnum_DairyProducts", "Milk Products");
    }
}

public abstract class BaseEnumConverter : MarkupExtension, IValueConverter
{
    protected Dictionary<string, string> TextValueDictionary;

    public BaseEnumConverter()
    {
        TextValueDictionary = new Dictionary<string, string>();
        PopulateDictionary();
    }

    protected abstract void PopulateDictionary();

    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return this;
    }
    public object? Convert(object? value, Type targetType, object? parameter,
                           System.Globalization.CultureInfo culture)
    {
        var type = value?.GetType();
        var memberName = value?.ToString();
        if (type == null || !type.IsEnum || string.IsNullOrEmpty(memberName))
            return null;
        return EnumMemberToString(type.Name, memberName);
```

```
    }

    protected virtual string EnumMemberToString(string enumName, string enumMemberName)
    {
        string fullMemberName = enumName + "_" + enumMemberName;
        if (TextValueDictionary.ContainsKey(fullMemberName))
            return TextValueDictionary[fullMemberName];
        else
            return enumMemberName;
    }

    public object? ConvertBack(object? value, Type targetType, object? parameter,
                               System.Globalization.CultureInfo culture)
    {
        return null;
    }
}
```

Получение и установка выбранного элемента, а также выставление значения редактора

Когда пользователь выбирает сегмент или отменяет его выбор щелчком мыши с Ctrl, свойства `SegmentedEditor.SelectedItem` и `SegmentedEditor.EditorValue` изменяются соответствующим образом. Вы можете использовать любое из этих свойств, чтобы получить и установить выбранное значение редактора.

Свойство `SegmentedEditor.SelectedItem` определяет нижележащий объект данных выбранного сегмента.

Свойство `SegmentedEditor.EditorValue` имеет следующие значения:

- Если свойство `ValueMember` пусто, то свойства `EditorValue` и `SelectedItem` эквивалентны.
- В противном случае свойство `EditorValue` синхронизируется со значением свойства `ValueMember` выбранного нижележащего объекта данных.

Чтобы снять выделение, установите для свойства `SegmentedEditor.SelectedItem` значение `null`.

Пример - Как выбрать элемент, когда SegmentedEditor привязан к списку строк

В следующем примере показано, как использовать свойство `SelectedItem` или `EditorValue` для выбора элемента в контроле `SegmentedEditor`, который привязан к списку строк.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:col="using:System.Collections"
```

```
<mxe:SegmentedEditor Name="segmEditorStrings">
    <mxe:SegmentedEditor.ItemsSource>
        <col:ArrayList>
            <sys:String>Montevideo</sys:String>
            <sys:String>Havana</sys:String>
            <sys:String>Santiago</sys:String>
            <sys:String>La Paz</sys:String>
        </col:ArrayList>
    </mxe:SegmentedEditor.ItemsSource>
</mxe:SegmentedEditor>
```

```
// Use the SelectedItem property:
segmEditorStrings.SelectedItem = "Santiago";
//or the EditorValue property:
segmEditorStrings.EditorValue = "Santiago";
```

Пример - Как выбрать элемент, когда SegmentedEditor привязан к списку бизнес-объектов

В следующем примере контрол `SegmentedEditor` привязан к списку объектов *Product*. Свойство `ValueMember` ссылается на элемент *Product.ProductID*. Таким образом, идентификаторы продуктов служат значениями элементов. Когда пользователь выбирает сегмент, свойству `EditorValue` присваивается соответствующий идентификатор продукта. В примере используется свойство `EditorValue` для выбора элемента в коде.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:EditorsSample"

<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>

<mxe:SegmentedEditor
    Name="segmEditor1"
    ItemsSource="{Binding Products}"
    DisplayMember="ProductName"
    ValueMember="ProductID"/>
```

```
// Select an item by its product ID:
var itemSource = segmEditor1.ItemsSource as ObservableCollection<Product>;
if (itemSource != null)
    segmEditor1.EditorValue = itemSource[3].ProductID;
//The SelectedItem property will return the itemSource[3] object.

//...
[ObservableObject]
public partial class MainViewModel : ViewModelBase
```

```
{
    [ObservableProperty]
    public ObservableCollection<Product> products;

    public MainViewModel()
    {
        Products = new ObservableCollection<Product>();
        Products.Add(new Product(0, "Chai", "Beverages", 200));
        Products.Add(new Product(1, "Chang", "Beverages", 100))
        Products.Add(new Product(3, "Ikura", "Seafood", 500));
        Products.Add(new Product(5, "Tofu", "Produce", 430));
    }
}

public partial class Product :ObservableObject
{
    public Product(int productID, string productName, string category, int productPrice)
    {
        ProductID = productID;
        ProductName = productName;
        Category = category;
        ProductPrice = productPrice;
    }

    [ObservableProperty]
    public int productID;

    [ObservableProperty]
    public string productName;

    [ObservableProperty]
    public string category;

    [ObservableProperty]
    public decimal productPrice;
}
```

Шаблоны элементов

Когда `SegmentedEditor` привязан к списку строк или бизнес-объектов, сегменты контрола отображают дефолтное представление текста элементов. Дефолтный отображаемый текст элемента определяется следующим образом:

- Значение, возвращаемое методом `ToString` нижележащего объекта данных, если свойство `ValueMember` не задано.
- В противном случае используется текстовое представление значения, хранящегося в свойстве `ValueMember` объекта данных.

Шаблоны элементов дают вам возможность гибко указывать, что отображать в сегментах редактора. Они позволяют отображать картинки и значения нескольких свойств в сегментах. Используйте свойство `SegmentedEditor.ItemTemplate`, чтобы указать шаблон элемента.

Пример - Как отобразить картинку и текст для элементов SegmentedEditor

В следующем примере показано, как создать шаблон элемента, который отображает картинки и текст в сегментах контрола `SegmentedEditor`.

В примере контрол `SegmentedEditor` привязан к коллекции объектов *CapitalInfo*, которые хранят информацию о странах, столицах стран и национальных флагах. Созданный шаблон элемента (свойство `SegmentedEditor.ItemTemplate`) отображает флаг страны, за которым следует название столицы страны.

Свойство `SegmentedEditor.ValueMember` ссылается на свойство *CapitalInfo.Capital*. Когда пользователь выбирает сегмент, свойству редактора `EditorValue` присваивается название столицы соответствующей страны.

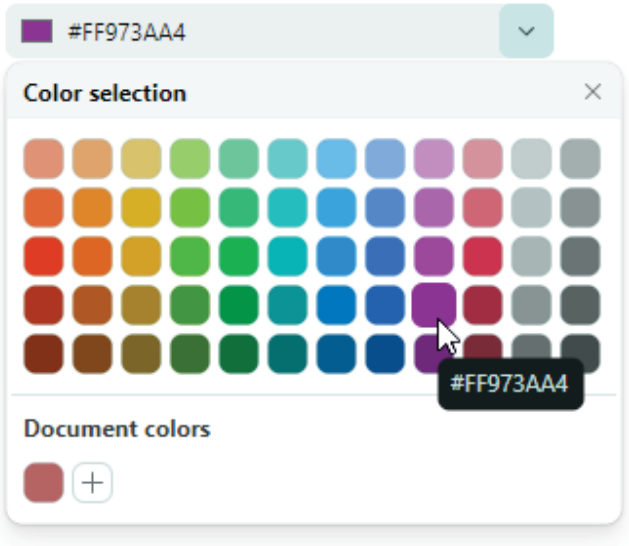
```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:local="clr-namespace:EditorsSample"

<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>

<Window.Resources>
    <DataTemplate x:Key="CapitalItemTemplate">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Image Width="16" Height="16" Source="{Binding Path=Flag}"/>
            <TextBlock VerticalAlignment="Center" Grid.Column="1" Margin="6,0,0,0"
             Text="{Binding Path=Capital}"/>
        </Grid>
    </DataTemplate>
</Window.Resources>

<mxe:SegmentedEditor
    Name="segmentedEditorCapitals"
    ItemsSource="{Binding Capitals}"
    ItemTemplate="{StaticResource CapitalItemTemplate}"
    ValueMember="Capital"
/>
```

```
using CommunityToolkit.Mvvm.ComponentModel;
using Avalonia.Media;
```

```
using Avalonia.Svg.Skia;

public partial class MainViewModel
{
    [ObservableProperty]
    public ObservableCollection<CapitalInfo> capitals;

    public MainViewModel()
    {
        var capitalDictionary = new List<(string capital, string country)>();
        capitalDictionary.Add(("Havana", "Cuba"));
        capitalDictionary.Add(("Santiago", "Chile"));
        capitalDictionary.Add(("La Paz", "Bolivia"));
        Capitals = new ObservableCollection<CapitalInfo>();
        foreach (var item in capitalDictionary)
        {
            string country = item.country.ToLower();
            // Load .SVG images that are stored in the Images/Flags folder as Avalonia Resources.
            IImage image = SvgImageExtension.ProvideValue(
                $"avares://EditorsSample/Images/Flags/{country}.svg",
                null!, null!);
            Capitals.Add(new CapitalInfo(item.capital, item.country, image));
        }
    }
}

public partial class CapitalInfo : ObservableObject
{
    public CapitalInfo(string capital, string country, IImage flag)
    {
        Capital = capital;
        Flag = flag;
        Country = country;
    }

    [ObservableProperty]
    public string capital;

    [ObservableProperty]
    public string country;

    [ObservableProperty]
    public IImage flag;
}
```

PopupColorEditor

Библиотека контролов Eremex включает в себя контрол `PopupColorEditor`, который позволяет отображать цвет в поле редактирования и выбирать цвет из связанного всплывающего окна.



Основные функции контрола включают в себя:

- Настраиваемая цветовая палитра по умолчанию.
- Стандартная цветовая палитра.
- Пользовательская цветовая палитра, которая может быть настроена пользователями.
- Встроенный диалог позволяет выбирать цвета с помощью диалога Выбора Цвета или путем указания отдельных цветовых компонентов в формате RGB или HSB.

Выбор цвета

Пользователь может выбрать цвет, используя цветовые палитры, отображаемые в всплывающем окне.

В коде вы можете указать цвет или прочитать выбранный в данный момент цвет с помощью свойства `PopupColorEditor.Color` или `PopupColorEditor.EditorValue`. Эти свойства синхронизированы. Они различаются типом значения: свойство `Color` имеет тип `Color` с возможностью обнуления, в то время как свойство `EditorValue` имеет тип `object`, как и во всех редакторах Eremex.

Палитры

`PopupColorEditor` поддерживает три палитры: по умолчанию, стандартную и пользовательскую. Используйте свойство `PopupColorEditor.ColorsShowMode`, чтобы настроить видимость отдельных палитр. Свойство `ColorsShowMode` определено как набор флагов.

Стандартная цветовая палитра

Включите флаг `StandardColors` в значении свойства `ColorsShowMode`, чтобы отобразить стандартную цветовую палитру.

Пользовательская цветовая палитра

Включите флаг `CustomColors` в значение свойства `ColorsShowMode`, чтобы отобразить пользовательскую палитру цветов.

Пользователи могут добавлять и настраивать цвета в пользовательской палитре во время выполнения программы. Нажмите кнопку "+", чтобы добавить цвет. Щелкните правой кнопкой мыши существующий цвет, чтобы получить доступ к командам контекстного меню, используемым для изменения и удаления цвета.

Вы можете заранее заполнить пользовательскую палитру цветов в коде, используя свойство `CustomColors`.

Пример - Как настроить пользовательскую палитру

Следующий код включает пользовательскую палитру цветов и заполняет ее из свойства *CustomColorCollection*, определенного в ViewModel.

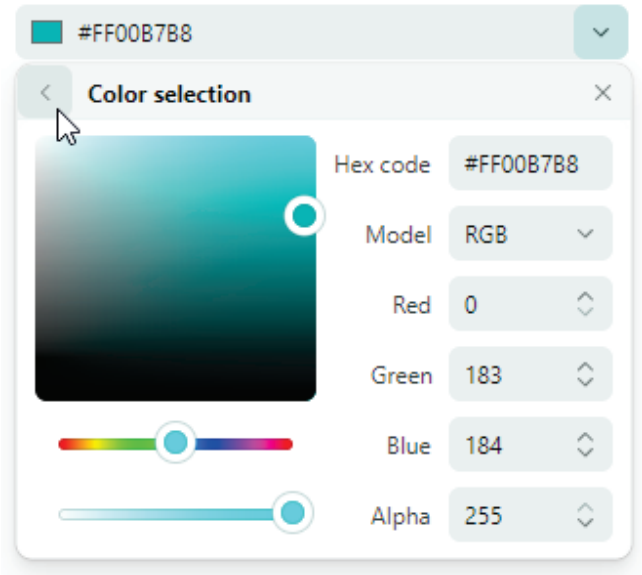
```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:ComboBoxTestSample"

<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>

<mxe:PopupColorEditor ColorsShowMode="CustomColors"
 CustomColors="{Binding CustomColorCollection}"/>
```

```
using Avalonia.Media;
using CommunityToolkit.Mvvm.ComponentModel;
using System.Collections.ObjectModel;
```

```
[ObservableObject]
public partial class MainViewModel
{
    public MainViewModel()
    {
        CustomColorCollection = new ObservableCollection<Color>()
        {
            Color.FromRgb(0x7d, 0xd7, 0xab), Color.FromRgb(0xc5, 0x94, 0x88),
            Color.FromRgb(0x47, 0xfe, 0xff), Color.FromRgb(0xe9, 0xbf, 0x3f),
        };
    }

    [ObservableProperty]
    ObservableCollection<Color> customColorCollection;
}
```

Диалог выбора цвета

Когда пользователь нажимает кнопку "+" или щелкает правой кнопкой мыши на существующем цветовом поле в пользовательской палитре, редактор активирует диалоговое окно "Диалог Выбора Цвета".



Пользовательский интерфейс диалогового окна содержит диалог Выбора Цвета и контролы для указания компонентов цвета в формате RGB или HSB.

Связанный API

- `ShowAlphaChannel` — Позволяет скрыть контролы, используемые для указания альфа-компонента цвета.
- `ShowConfirmationButtons` — Указывает, следует ли отображать кнопки "Применить" и "Отмена" в диалоговом окне выбора цвета. Если свойство включено, пользователю необходимо нажать кнопку Применить, чтобы подтвердить выбор цвета. Нажатие на кнопку

"Назад" или "Отмена" отменяет диалоговое окно. Если для свойства `ShowConfirmationButtons` установлено значение `false`, нажатие на кнопку "Назад" подтверждает выбранный цвет.

HyperlinkEditor

Контрол `HyperlinkEditor` отображает гиперссылку, по которой пользователь может щелкнуть. Редактор не запускает ссылку при нажатии на нее. Вместо этого он вызывает связанную команду, которую вы можете использовать для обработки переходов по ссылкам.

Текст редактора не может редактироваться пользователями.

Отображаемая гиперссылка

Используйте свойство `EditorValue` контрола, чтобы указать текст, который будет отображаться в редакторе. Редактор подчеркивает текст, имитируя гиперссылку.

Если редактору не назначена команда (см. ниже), щелчок по отображаемой ссылке не имеет эффекта.

Обработка переходов по гиперссылкам

Назначьте команду свойству редактора `Command` для обработки переходов по гиперссылке. Используйте `CommandParameter`, чтобы предоставить дополнительные данные команде.

Пример

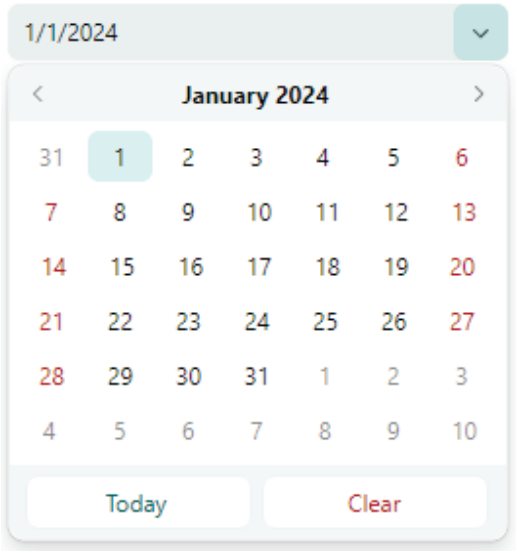
В следующем примере определяется `HyperlinkEditor`, который отображает ссылку на веб-страницу. Щелчок по ссылке вызывает команду *ShowWebPageCommand*. Адрес ссылки для вызова передается в качестве параметра команды.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"

<mxe:HyperlinkEditor EditorValue="https://www.w3.org" Command="{Binding ShowWebPageCommand}"
                     CommandParameter="https://www.w3.org"/>
```

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.Diagnostics;

public partial class HyperlinkEditorPageViewModel : ObservableObject
{
    [RelayCommand]
    public void ShowWebPage(string parameter)
    {
        try
        {
            Process.Start(new ProcessStartInfo(parameter) { UseShellExecute = true });
        }
        catch { };
    }
}
```

DateEditor

Контрол `DateEditor` содержит всплывающий календарь, который позволяет пользователям выбирать дату. Редактор поддерживает несколько форматов отображения значения даты, отображаемого в поле редактирования.



Всплывающий календарь содержит заголовок навигации, используемый для просмотра по месяцам и годам. Кнопки "Сегодня" и "Очистить" помогают пользователям быстро выбрать сегодняшнюю дату и очистить текущее значение соответственно.

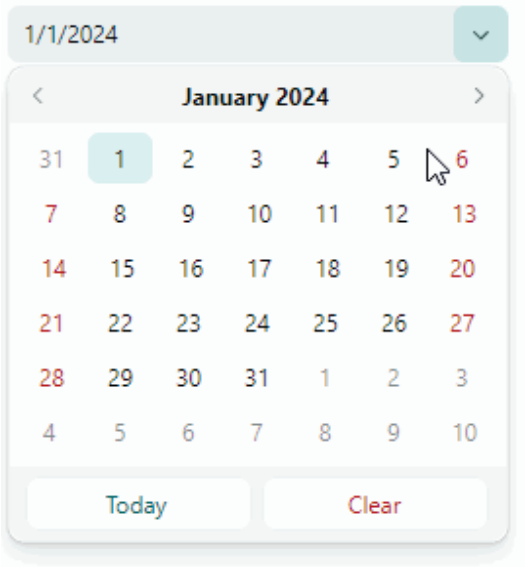
Основные функции контрола включают в себя:

- Выбор даты в всплывающем календаре с помощью мыши.
- Просматривайте месяцы и годы с помощью панели навигации.
- Три вида календаря: просмотр по месяцам, просмотр по годам и просмотр диапазона лет.
- Встроенные кнопки "Сегодня" и "Очистить".
- Ограничение доступного диапазона дат.
- Множество форматов отображения выбранного значения даты в поле редактирования.

Выбор даты

Пользователь может выбрать дату, открыв всплывающее окно и выбрав дату в появившемся календаре.

Навигационный заголовок всплывающего календаря позволяет пользователю просматривать месяцы и годы:



В коде вы можете указать дату или прочитать текущую выбранную дату с помощью свойства `DateEditor.DateTime` или `DateEditor.EditorValue`. Эти свойства синхронизированы. Они различаются типом значения: свойство `DateTime` имеет тип `System.DateTime` с возможностью обнуления, в то время как свойство `EditorValue` имеет тип `object`, как и во всех редакторах Eremex.

Настройка всплывающего календаря

Используйте следующие свойства для настройки календаря контрола `DateEditor`:

- `ShowToday` — Возвращает или устанавливает, следует ли выделять сегодняшнюю дату в календаре.
- `NullValueButtonPosition` — Возвращает или устанавливает, видна ли кнопка ('*x*') (очистить значение).
- `MinValue` — указывает минимально допустимую дату. Свойства `MinValue` и `MaxValue` позволяют указать диапазон значений, отображаемых в календаре.
- `MaxValue` — указывает максимально допустимую дату.

Пример - Как создать DateEditor

В следующем примере определяется DateEditor, устанавливается начальное значение и указывается допустимый диапазон дат.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"

<Window.DataContext>
```

```
    <local:MainViewModel/>
</Window.DataContext>

<mxe:DateEditor
    DateTime="{Binding SelectedDate, Mode=TwoWay}"
    MinValue="{Binding MinimumDate}"
    MaxValue="{Binding MaximumDate}"/>
```

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.Collections.ObjectModel;

[ObservableObject]
public partial class MainViewModel
{
    [ObservableProperty]
    DateTime selectedDate = DateTime.Now;
    [ObservableProperty]
    DateTime minimumDate = DateTime.Now.AddDays(-15);
    [ObservableProperty]
    DateTime maximumDate = DateTime.Now.AddDays(15);
}
```

Формат отображения значения

Используйте свойство `DisplayFormatString`, чтобы задать формат отображения значения даты/времени, отображаемого в поле редактирования.

ComboBoxEditor

Контрол `ComboBoxEditor` - это контрол, который отображает список элементов в своем всплывающем окне. Пользователь может выбрать один или несколько элементов одновременно в соответствии с режимом выбора контрола.

Основные функции контрола включают в себя:

- Отображает значения из списка строк, списка бизнес-объектов или тип-перечисления.
- Режимы выбора одного и нескольких элементов.
- Функция автозавершения.
- Редактирование текста в поле редактирования в режиме выбора одного элемента.
- Шаблоны элементов позволяют отображать элементы произвольным способом.
- Использование контрола в качестве встроенного редактора в контролах-контейнерах (например, TreeList, TreeView и PropertyGrid).

Источник объектов

Используйте свойство `ComboBoxEditor.ItemsSource`, чтобы указать список элементов, которые будут отображаться в всплывающем окне. Вы можете привязать редактор к списку строк, списку бизнес-объектов или к типу-перечислению.

Привязка к списку строк

Самый простой источник объектов - это список строк.

Пользователь может выбрать одно или несколько значений в соответствии с текущим режимом выбора (см. ниже). Если в поле редактирования включено редактирование текста (см. `IsTextEditable`), пользователь может ввести текст, который не соответствует ни одному элементу в всплывающем списке.

Пример - Как привязать к списку строк

В следующем примере `ComboBoxEditor` заполняется списком строк.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

```
xmlns:local="clr-namespace:ComboBoxTestSample"

<mxe:ComboBoxEditor x:Name="myComboBox1">
    <mxe:ComboBoxEditor.ItemsSource>
        <local:MyItemList>
            <sys:String>Moscow</sys:String>
            <sys:String>Kazan</sys:String>
            <sys:String>Tver</sys:String>
        </local:MyItemList>
    </mxe:ComboBoxEditor.ItemsSource>
</mxe:ComboBoxEditor>
```

```
public class MyItemList : List<string>
{
}
```

Привязка к списку бизнес-объектов

Вы можете привязать `ComboBoxEditor` к списку бизнес-объектов. В этом случае дефолтное поведение контрола является следующим:

- Метод `ToString` бизнес-объекта задает дефолтное представление текста элементов.
- Когда вы выбираете элемент, значение редактора (`ComboBoxEditor.EditorValue`) устанавливается равным соответствующему бизнес-объекту.

Типичный бизнес-объект обладает множеством свойств. Вы можете указать, какие свойства бизнес-объекта предоставляют текст элемента для отображения и редактирования значений. Для этой цели используйте следующие элементы API:

- `ComboBoxEditor.DisplayMember` - Получает или задает имя свойства бизнес-объекта, которое определяет текст элемента для отображения.
- `ComboBoxEditor.ValueMember` - Получает или задает имя свойства бизнес-объекта, которое определяет значения элемента. Когда вы выбираете элемент, значение редактора (`ComboBoxEditor.EditorValue`) устанавливается равным значению свойства `ValueMember` элемента.

Пример - Как привязать к списку бизнес-объектов

Следующий пример привязывает `ComboBoxEditor` к списку бизнес-объектов *Product*. Свойство *Product.ProductName* определяет текст элемента для отображения. Свойство *Product.ProductID* определяет значения элементов.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:local="clr-namespace:ComboBoxTestSample"
```

```
<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>

<mxe:ComboBoxEditor
    x:Name="myComboBox2"
    ItemsSource="{Binding Products}"
    DisplayMember="ProductName"
    ValueMember="ProductID"
    SelectionMode="Multiple"
/>
```

```
public MainViewModel()
{
    Products = new ObservableCollection<Product>();
    //...
}

public partial class Product :ObservableObject
{
    public Product(int productID, string productName, string category, int productPrice)
    {
        ProductID = productID;
        ProductName = productName;
        Category = category;
        ProductPrice = productPrice;
    }

    [ObservableProperty]
    public int productID;

    [ObservableProperty]
    public string productName;

    [ObservableProperty]
    public string category;

    [ObservableProperty]
    public decimal productPrice;
}
```

Привязка к перечислению

ComboBoxEditor может отображать значения типа-перечисления в всплывающем окне.

Вспомогательный класс `Eremex.AvaloniaUI.Controls.Common.EnumItemsSource` облегчает привязку к перечислению. Его основные функции включают:

- Картинки для элементов перечисления в всплывающем окне. Примените атрибут `Eremex.AvaloniaUI.Controls.Common.ImageAttribute` к целевым элементам перечисления, чтобы указать картинки.
- Пользовательские отображаемые имена для элементов перечисления. Используйте атрибут `System.ComponentModel.DataAnnotations.DisplayAttribute` или пользовательский конвертер, чтобы изменить дефолтный отображаемый текст элемента.
- Всплывающие подсказки для элементов всплывающего окна. Всплывающая подсказка содержит описание целевого элемента, которое вы можете предоставить с помощью атрибута `System.ComponentModel.DataAnnotations.DisplayAttribute` или пользовательского конвертера.

Чтобы настроить привязку к типу-перечислению, используйте следующие свойства `EnumItemsSource`:

- `EnumItemsSource.EnumType` — Задает тип-перечисление, значения которого отображаются в ComboBoxEditor.
- `EnumItemsSource.ShowImages` — Указывает, следует ли отображать картинки для элементов перечисления в всплывающем окне. Вы можете предоставить картинки, используя атрибут `Eremex.AvaloniaUI.Controls.Common.ImageAttribute`.
- `EnumItemsSource.ShowNames` — Указывает, следует ли отображать текст элемента. Установите для `ShowNames` значение `false`, а для `ShowImages` значение `true`, чтобы отображать элементы перечисления с использованием картинок без текста.
- `EnumItemsSource.ImageSize` — Определяет размер отображения картинок, назначенных элементам перечисления.
- `EnumItemsSource.NameToDisplayTextConverter` — Позволяет назначить конвертер, который извлекает пользовательский текст для отображения для элементов перечисления.
- `EnumItemsSource.NameToDescriptionConverter` — Позволяет назначить конвертер, который извлекает описания элементов перечисления, которые отображаются в виде всплывающих подсказок, когда пользователь наводит курсор мыши на элементы всплывающего окна.

Пример - Как отобразить значения перечисления и использовать атрибуты для предоставления текста и картинок для элементов перечисления.

В следующем примере отображаются значения перечисления *ProductCategoryEnum* в ComboBoxEditor. Он использует класс `EnumItemsSource` для привязки данных.

Атрибуты `System.ComponentModel.DataAnnotations.DisplayAttribute` и `Eremex.AvaloniaUI.Controls.Common.ImageAttribute` определяют пользовательский текст для отображения, описания (всплывающие подсказки) и картинки для элементов перечисления.

```
xmlns:mxcom="clr-namespace:Eremex.AvaloniaUI.Controls.Common;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
```

```
<mxe:ComboBoxEditor Name="comboBoxEditorEnum"
 ItemsSource="{mxcom:EnumItemsSource EnumType=local:ProductCategoryEnum,
  ImageSize='16, 16', ShowImages=True, ShowNames=True}"
 IsTextEditable="True"
 AutoComplete="True">
</mxe:ComboBoxEditor>
```

```
using Eremex.AvaloniaUI.Controls.Common;
using System.ComponentModel.DataAnnotations;

public enum ProductCategoryEnum
{
    // The images assigned to the enumeration values below are placed
    // in the ComboBoxTestSample/Images folder.
    // They have their "Build Action" properties set to "AvaloniaResource".
    [Image($"avares://ComboBoxTestSample/Images/Products/DairyProducts.svg")]
    [Display(Name = "Dairy Products", Description = "Products made from milk")]
    DairyProducts,

    [Image($"avares://ComboBoxTestSample/Images/Products/Beverages.svg")]
    [Display(Description = "Edible drinks")]
    Beverages,

    [Image($"avares://ComboBoxTestSample/Images/Products/Condiments.svg")]
    [Display(Description = "Flavor Enhancers")]
    Condiments,

    [Image($"avares://ComboBoxTestSample/Images/Products/Confections.svg")]
    [Display(Description = "Sweets")]
    Confections
}
```

Пример - Как отобразить значения перечисления и использовать пользовательские конвертеры для предоставления текста для элементов перечисления.

В следующем примере класс `EnumItemsSource` используется для отображения значений типа-перечисления в ComboBoxEditor. Объекты `EnumItemsSource.NameToDisplayTextConverter` и `EnumItemsSource.NameToDescriptionConverter` предоставляют пользовательский текст для отображения и описания (всплывающие подсказки) для элементов перечисления.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:mxcom="clr-namespace:Eremex.AvaloniaUI.Controls.Common;assembly=Eremex.Avalonia.Controls"
xmlns:local="clr-namespace:ComboBoxTestSample"

<mxe:ComboBoxEditor Name="comboBoxEditorEnumWithConverters"
 ItemsSource="{mxcom:EnumItemsSource EnumType=local:ProductCategoryEnum,
  ImageSize='16, 16', ShowImages=True, ShowNames=True,
  NameToDisplayTextConverter={local:EnumMemberNameToDisplayTextConverter},
  NameToDescriptionConverter={local:EnumMemberNameToDescriptionConverter}}"
```

```
 IsTextEditable="True"
 AutoComplete="True">
</mxe:ComboBoxEditor>
```

```
using Eremex.AvaloniaUI.Controls.Common;

public enum ProductCategoryEnum
{
    // The images assigned to the enumeration values below are placed
    // in the ComboBoxTestSample/Images folder.
    // They have their "Build Action" properties set to "AvaloniaResource".
    [Image($"avares://ComboBoxTestSample/Images/Products/DairyProducts.svg")]
    DairyProducts,

    [Image($"avares://ComboBoxTestSample/Images/Products/Beverages.svg")]
    Beverages,

    [Image($"avares://ComboBoxTestSample/Images/Products/Condiments.svg")]
    Condiments,

    [Image($"avares://ComboBoxTestSample/Images/Products/Confections.svg")]
    Confections
}

public class EnumMemberNameToDisplayTextConverter : BaseEnumConverter
{
    protected override void PopulateDictionary()
    {
        TextValueDictionary.Add("ProductCategoryEnum_DairyProducts", "Dairy");
    }
}
public class EnumMemberNameToDescriptionConverter : BaseEnumConverter
{
    protected override void PopulateDictionary()
    {
        TextValueDictionary.Add("ProductCategoryEnum_DairyProducts", "Milk Products");
    }
}

public abstract class BaseEnumConverter : MarkupExtension, IValueConverter
{
    protected Dictionary<string, string> TextValueDictionary;

    public BaseEnumConverter()
    {
        TextValueDictionary = new Dictionary<string, string>();
        PopulateDictionary();
    }

    protected abstract void PopulateDictionary();

    public override object ProvideValue(IServiceProvider serviceProvider)
```

```
    {
        return this;
    }
    public object? Convert(object? value, Type targetType, object? parameter,
                           System.Globalization.CultureInfo culture)
    {
        var type = value?.GetType();
        var memberName = value?.ToString();
        if (type == null || !type.IsEnum || string.IsNullOrEmpty(memberName))
            return null;
        return EnumMemberToString(type.Name, memberName);
    }

    protected virtual string EnumMemberToString(string enumName, string enumMemberName)
    {
        string fullMemberName = enumName + "_" + enumMemberName;
        if (TextValueDictionary.ContainsKey(fullMemberName))
            return TextValueDictionary[fullMemberName];
        else
            return enumMemberName;
    }

    public object? ConvertBack(object? value, Type targetType, object? parameter,
                               System.Globalization.CultureInfo culture)
    {
        return null;
    }
}
```

Режим выбора элемента

Свойство `SelectionMode` позволяет вам выбирать между режимами выбора одного или нескольких элементов.

Режим выбора одного элемента установлен по умолчанию. Если включено редактирование текста (см. `IsTextEditable`), можно указать значение редактирования (или текст в поле редактирования), которое не соответствует ни одному элементу в всплывающем списке.

Установите для свойства `SelectionMode` значение `ItemSelectionMode.Multiple`, чтобы включить множественный выбор элемента. В режиме множественного выбора редактор отображает флажки перед каждым элементом в всплывающем списке. Пользователь может переключать эти флажки, чтобы добавлять и удалять элементы в выборку или из нее. Редактирование текста отключено в поле редактирования в режиме множественного выбора.

Получение и установка выбранного элемента(ов)

Свойство `SelectedItem` позволяет вам выбирать и извлекать выбранный элемент в режиме одиночного выбора. Свойство возвращает значение `null`, если ни один элемент не выбран.

Свойство `SelectedItems` определяет список (объект `IList`), который содержит выбранные элементы в режиме множественного выбора. Свойство возвращает пустой список, если ни один элемент не выбран.

> **ЗАМЕЧАНИЕ**
>
> Свойства `SelectedItem` и `SelectedItems` синхронизированы способом, описанным ниже.
>
> В режиме одиночного выбора свойство `SelectedItems` определяет список, содержащий один элемент – выбранный элемент (значение свойства `SelectedItem`). Свойство возвращает пустой список, если ни один элемент не выбран.
>
> В режиме множественного выбора свойство `SelectedItem` определяет первый выбранный элемент (первый элемент списка `SelectedItems`). Свойство возвращает значение `null`, если ни один элемент не выбран.

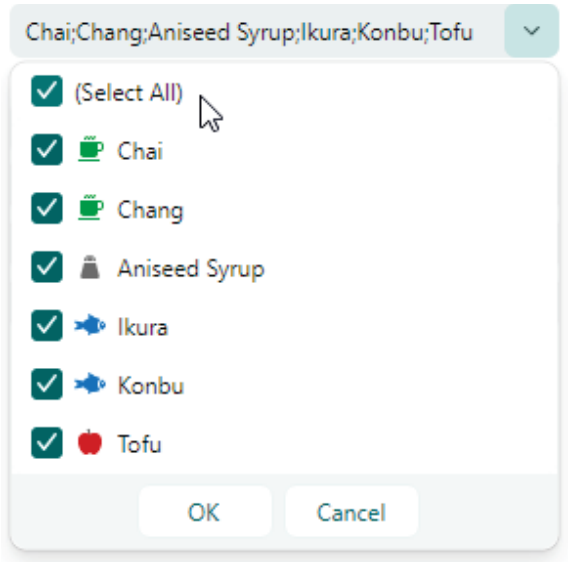
Пример - Как выбрать элементы

Следующий код выбирает элементы в ComboBoxEditor, которые работают в режимах одиночного и множественного выбора.

```
// Select an item in single selection mode
var itemSource1 = myComboBox1.ItemsSource as MyItemList;
if(itemSource1 != null)
    myComboBox1.SelectedItem = itemSource1[0];

// Select items in multi-select mode
var itemSource2 = myComboBox2.ItemsSource as ObservableCollection<Product>;
if (itemSource2 != null)
    myComboBox2.SelectedItems = new List<Product>() { itemSource2[0], itemSource2[1] };
```

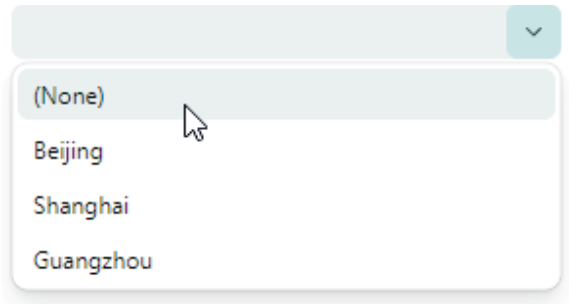
Элемент "(Выбрать все)" в режиме множественного выбора

В режиме множественного выбора редактор отображает предопределенный флажок *(Select All)* в верхней части всплывающего окна. Этот флажок позволяет пользователю выбирать/отменять выбор всех элементов. Чтобы скрыть флажок *(Select All)*, установите для свойства `ComboBoxEditor.ShowPredefinedSelectItem` значение `false`.



Элемент '*(None)*' в режиме одиночного выбора

В режиме одиночного выбора вы можете установить для свойства `ComboBoxEditor.ShowPredefinedSelectItem` значение `true`, чтобы отобразить предопределенный элемент *(None)* в всплывающем окне. Элемент *(None)* устанавливает значение редактора в `null` и, таким образом, очищает выделение.



Значение редактора

Когда пользователь вводит текст, выбирает элемент в поле редактирования с помощью функции автозаполнения или выбирает элемент в всплывающем окне, редактор изменяет его значение редактирования (значение свойства `EditorValue`).

Как правило, значение редактирования совпадает со значением свойства `SelectedItem` (в режиме одиночного выбора) или свойства `SelectedItems` (в режиме множественного выбора), за исключением случаев, описанных в этом разделе.

Значение редактирования зависит от режима выбора элемента. В режиме выбора одного элемента значением редактирования является одно значение, в то время как в режиме выбора нескольких элементов значением редактирования является объект `IList`, который определяет список значений.

Свойства `EditorValue` и `SelectedItem/SelectedItems` не синхронизированы в следующих случаях:

- ComboBoxEditor привязан к списку строк, и текст, указанный в поле редактирования, не соответствует ни одному элементу в всплывающем окне. В этом случае свойство `SelectedItem` возвращает значение `null`, в то время как свойство `EditorValue` определяет текст, отображаемый в поле редактирования.

  > **СОВЕТ**
  >
  > Включите опцию `ComboBoxEditor.IsTextEditable`, чтобы разрешить пользователю редактировать текст в поле редактирования.

Пример - Как установить значение редактора, когда ComboBox привязан к списку строк

В следующем примере задается значение редактирования для ComboBoxEditor, привязанного к списку строк.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:ComboBoxTestSample"

<mxe:ComboBoxEditor x:Name="myComboBox1">
    <mxe:ComboBoxEditor.ItemsSource>
        <local:MyItemList>
            <sys:String>Moscow</sys:String>
            <sys:String>Kazan</sys:String>
            <sys:String>Tver</sys:String>
        </local:MyItemList>
    </mxe:ComboBoxEditor.ItemsSource>
</mxe:ComboBoxEditor>
```

```
myComboBox1.EditorValue = "Moscow";
//The SelectedItem property will return "Moscow" as well.

myComboBox1.EditorValue = "Rostov";
// The SelectedItem property will return `null`,
// since the edit value does not match any item in the bound list.

//...
public class MyItemList : List<string>
{
}
```

- A ComboBoxEditor is bound to a list of business objects, and the `ValueMember` property is set. The `ValueMember` property specifies the name of the business object's property that supplies item values. When you select an item(s), the `SelectedItem/SelectedItems` property contains the selected object(s), while the `EditorValue` property specifies the selected objects' value (values).

Example - How to select an item when ComboBoxEditor is bound to a business object list

In the following example, a ComboBoxEditor is bound to a list of *Product* objects in multi-select mode. The `ValueMember` property refers to the *Product.ProductID* member. Thus, product IDs serve as item values. When a user selects items, the `EditorValue` property is set to a list that contains correponding product IDs. The example uses the `EditorValue` property to select items in code.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:ComboBoxTestSample"

<mxe:ComboBoxEditor
    x:Name="myComboBox2"
    ItemsSource="{Binding Products}"
    DisplayMember="ProductName"
    ValueMember="ProductID"
    SelectionMode="Multiple"/>
```

```
// Select items by their product IDs:
var itemSource2 = myComboBox2.ItemsSource as ObservableCollection<Product>;
if(itemSource2 != null)
    myComboBox2.EditorValue = new List<int>()
    { itemSource2[3].ProductID, itemSource2[5].ProductID };
//The SelectedItems property will return a list that contains
// two Product objects (itemSource2[3] and itemSource2[5]).

//...
public partial class Product :ObservableObject
{
    public Product(int productID, string productName, string category, int productPrice)
    {
        ProductID = productID;
        ProductName = productName;
        Category = category;
        ProductPrice = productPrice;
    }

    [ObservableProperty]
    public int productID;

    [ObservableProperty]
    public string productName;
```

```
    [ObservableProperty]
    public string category;

    [ObservableProperty]
    public decimal productPrice;
}
```

Шаблоны элементов

ComboBoxEditor отображает каждый элемент в всплывающем списке, используя по умолчанию текст элемента для отображения. Методом `ToString` для элементов задается дефолтный отображаемый текст элемента. Когда редактор привязан к списку бизнес-объектов, вы можете использовать свойство `DisplayMember`, чтобы указать свойство, предоставляющее отображаемый текст элемента.

Вы можете использовать свойство `ItemTemplate` для назначения шаблона данных, который представляет элементы в всплывающем окне произвольным способом. Например, шаблон данных поможет вам отображать картинки для элементов, как показано в примере ниже.

Включите опцию `ComboBoxEditor.ApplyItemTemplateToEditBox`, чтобы применить указанный шаблон элемента (свойство `ItemTemplate`) к полю редактирования. Это свойство не действует, если включено редактирование текста (для опции `IsTextEditable` установлено значение `true`).

Пример - Как отобразить картинки для элементов ComboBox с помощью Data Template

В следующем примере свойство `ItemTemplate` используется для указания шаблона данных, который отображает картинки для элементов ComboBox в всплывающем списке.

ComboBoxEditor привязан к списку, в котором хранятся объекты *Product*. Объект *Product* содержит свойство *Category*, которое определяет категорию продукта (напитки, приправы, морепродукты или другие продукты).

Предполагается, что проект хранит SVG-картинки для категорий товаров в папке "*ComboBoxTestSample/Images/Products*". Картинки имеют следующие названия: "*Beverages.svg*", "*Condiments.svg*", "*Seafood.svg*" и "*Produce.svg*", и они помечены флагом "*AvaloniaResource*".

Созданный шаблон данных элемента отображает название продукта и картинку, соответствующую категории продукта.

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:local="clr-namespace:ComboBoxTestSample"

<Window.DataContext>
```

```
    <local:MainViewModel/>
</Window.DataContext>
<Window.Resources>
    <local:NameToSvgConverter x:Key="NameToSvgConverter"/>
    <DataTemplate x:Key="ProductItemTemplate">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Image Width="16" Height="16"
             Source="{Binding Path=Category, Converter={StaticResource NameToSvgConverter}}"/>
            <TextBlock VerticalAlignment="Center" Grid.Column="1" Margin="6,0,0,0"
             Text="{Binding Path=ProductName}"/>
        </Grid>
    </DataTemplate>
</Window.Resources>

<mxe:ComboBoxEditor
    x:Name="myComboBox2"
    ItemsSource="{Binding Products}"
    DisplayMember="ProductName"
    ValueMember="ProductID"
    SelectionMode="Multiple"
    ItemTemplate="{StaticResource ProductItemTemplate}"
/>
```

```
using Avalonia.Svg.Skia;

public partial class MainViewModel : ViewModelBase
{
    [ObservableProperty]
    public ObservableCollection<Product> products;

    public MainViewModel()
    {
        Products = new ObservableCollection<Product>();
        Products.Add(new Product(0, "Chai", "Beverages", 200));
        Products.Add(new Product(1, "Chang", "Beverages", 100));
        Products.Add(new Product(2, "Aniseed Syrup", "Condiments", 150));
        Products.Add(new Product(3, "Ikura", "Seafood", 500));
        Products.Add(new Product(4, "Konbu", "Seafood", 390));
        Products.Add(new Product(5, "Tofu", "Produce", 430));
    }
}

public partial class Product :ObservableObject
{
    public Product(int productID, string productName, string category, int productPrice)
    {
        ProductID = productID;
        ProductName = productName;
```

```
        Category = category;
        ProductPrice = productPrice;
    }

    [ObservableProperty]
    public int productID;

    [ObservableProperty]
    public string productName;

    [ObservableProperty]
    public string category;

    [ObservableProperty]
    public decimal productPrice;
}

public class NameToSvgConverter : MarkupExtension, IValueConverter
{
    public object? Convert(object? value, Type targetType, object? parameter,
    CultureInfo culture)
    {
        if(value == null)
            return null;
        string name = value.ToString();

        return SvgImageExtension.ProvideValue(
            $"avares://ComboBoxTestSample/Images/Products/{name}.svg",
            null!, null!);
    }

    public object? ConvertBack(object? value, Type targetType, object? parameter,
    CultureInfo culture)
    {
        throw new NotImplementedException();
    }

    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return this;
    }
}
```

Добавление пользовательских кнопок

ComboBoxEditor является потомком контрола ButtonEditor. Таким образом, вы можете добавить пользовательские кнопки в поле редактирования рядом с кнопкой всплывающего окна по умолчанию. Используйте коллекцию `ComboBoxEditor.Buttons` для добавления пользовательских кнопок.

Пример - Как добавить пользовательские кнопки

В следующем примере в ComboBoxEditor добавляются обычная кнопка и кнопка-переключатель для проверки. Редактор привязан к перечислению *ProductCategoryEnum* с использованием вспомогательного класса `EnumItemsSource`.

Первая кнопка является обычной (ее свойству `ButtonKind` присвоено значение `Simple`). Щелчок по этой кнопке вызывает команду *ResetValue*, которая устанавливает значение редактора в значение по умолчанию (первый элемент связанного типа-перечисления).

Вторая кнопка - это переключающая кнопка (ее свойству `ButtonKind` присвоено значение `Toggle`). Нажатие на эту кнопку переключает значение свойства редактора `IsTextEditable`. Конвертер *LockedStateToSvgNameConverter* присваивает картинке "*locked.svg*" или "*unlocked.svg*" картинку кнопки в соответствии с состоянием нажатия кнопки. Эти картинки хранятся в папке *ComboBoxTestSample/Images* и помечены флагом "*AvaloniaResource*".

```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:mxcom="clr-namespace:Eremex.AvaloniaUI.Controls.Common;assembly=Eremex.Avalonia.Controls"
xmlns:local="clr-namespace:ComboBoxTestSample"

<mxe:ComboBoxEditor Name="comboBoxEditorEnum"
 ItemsSource="{mxcom:EnumItemsSource EnumType=local:ProductCategoryEnum,
               ImageSize='16, 16', ShowImages=True, ShowNames=True}"
 IsTextEditable="True"
 AutoComplete="True">
    <mxe:ComboBoxEditor.Buttons>
        <mxe:ButtonSettings ButtonKind="Simple"
         Glyph="{SvgImage 'avares://ComboBoxTestSample/Images/square_dot_icon.svg'}"
         Command="{Binding ResetValueCommand}"
         CommandParameter="{Binding #comboBoxEditorEnum}">
        </mxe:ButtonSettings>
        <mxe:ButtonSettings ButtonKind="Toggle"
         Glyph="{Binding $self.IsChecked, Converter={local:LockedStateToSvgNameConverter}}"
         IsChecked="{Binding !$parent.IsTextEditable}">
        </mxe:ButtonSettings>
    </mxe:ComboBoxEditor.Buttons>
</mxe:ComboBoxEditor>
```

```
using CommunityToolkit.Mvvm.Input;
using Eremex.AvaloniaUI.Controls.Editors;
using Avalonia.Svg.Skia;

public partial class MainViewModel : ViewModelBase
{
    // Sets the editor's value to the first element.
    [RelayCommand]
    void ResetValue(ComboBoxEditor editor)
    {
```

```
        // Get the first item in the ComboBoxEditor's bound list.
        var enumerator = editor.ItemsSource.GetEnumerator();
        enumerator.MoveNext();
        object firstItem = enumerator.Current;
        // When the ComboBoxEditor is bound to an EnumItemsSource,
        // the editor's items are EnumMemberInfo objects.
        EnumMemberInfo mInfo = firstItem as EnumMemberInfo;
        if (mInfo != null)
            editor.EditorValue = mInfo.Id;
    }
}

public class LockedStateToSvgNameConverter : MarkupExtension, IValueConverter
{
    public object? Convert(object? value, Type targetType, object? parameter,
    CultureInfo culture)
    {
        if (value == null)
            return null;
        bool isLocked = (bool)value;
        string lockedState = isLocked ? "locked" : "unlocked";

        if (isLocked)
            lockedState = "locked";

        return SvgImageExtension.ProvideValue(
            $"avares://ComboBoxTestSample/Images/{lockedState}.svg", null!, null!);
    }

    public object? ConvertBack(object? value, Type targetType, object? parameter,
    CultureInfo culture)
    {
        throw new NotImplementedException();
    }

    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return this;
    }
}

public enum ProductCategoryEnum
{
    [Image($"avares://ComboBoxTestSample/Images/Products/DairyProducts.svg")]
    [Display(Name = "Dairy Products", Description = "Products made from milk")]
    DairyProducts,

    [Image($"avares://ComboBoxTestSample/Images/Products/Beverages.svg")]
    [Display(Description = "Edible drinks")]
    Beverages,

    [Image($"avares://ComboBoxTestSample/Images/Products/Condiments.svg")]
    [Display(Description = "Flavor Enhancers")]
```

```
    Condiments,

    [Image($"avares://ComboBoxTestSample/Images/Products/Confections.svg")]
    [Display(Description = "Sweets")]
    Confections
}
```

Автозавершение текста

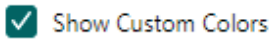
Установите для опции `AutoComplete` значение `true`, чтобы включить автоматическое заполнение текста. Эта функция автоматически дополняет текст, введенный пользователями, если он соответствует какому-либо элементу в всплывающем списке.

ComboBoxEditor не поддерживает автозавершение текста в следующих случаях:

- Редактирование текста отключено (свойству `IsTextEditable` присвоено значение `false`).
- Используется режим выбора нескольких элементов (свойству `SelectionMode` присвоено значение `Multiple`).

CheckEditor

Контрол `CheckEditor` отображает нажатый флажок, который пользователь может переключать с помощью мыши и клавиатуры. В дефолтном режиме редактор поддерживает два состояния нажатия (нажато и ненажато). При необходимости вы можете включить неопределенное состояние.



Основные функции контрола включают в себя:

- Вы можете выбрать между двумя или тремя состояниями нажатия.
- Отображение метки рядом с флажком.
- Пользователь может переключать контрол с помощью мыши и клавиатуры.
- Механизм валидации изменяет внешний вид контрола, чтобы информировать пользователей об ошибках.

Состояние нажатия

Пользователь может переключать CheckEditor с помощью мыши. Если редактор находится в фокусе, пользователь может нажать клавишу пробела или Enter на клавиатуре, чтобы изменить состояние нажатия.

В коде вы можете указать состояние нажатия или прочитать текущее состояние нажатия с помощью свойства `CheckEditor.IsChecked` или `CheckEditor.EditorValue`. Эти свойства синхронизированы. Они различаются типом значения: свойство `IsChecked` имеет тип `bool` с возможностью обнуления, в то время как свойство `EditorValue` имеет тип `object`, как и во всех редакторах Eremex.

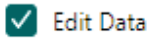
Отображаемый текст

Контрол `CheckEditor` может отображать текст справа от флажка. Используйте свойство `Content` для присвоения текста.

Пример - Как создать CheckEditor

В следующем примере определяется объект `CheckEditor`, который контролирует включенное состояние контрола TreeList. Пользователь не может редактировать данные TreeList, если нажатый

флажок ненажат.




```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:mxtl="using:Eremex.AvaloniaUI.Controls.TreeList"

<mxe:CheckEditor IsChecked="{Binding IsEnabled, ElementName=treeList1}" Content="Edit Data"/>

<mxtl:TreeListControl Name="treeList1">...</mxtl:TreeListControl>
```

Включение трех состояний

Включите свойство `IsThreeState`, чтобы разрешить редактору иметь три состояния. Последовательные переключения редактора переключают между нажатым, неопределенным и ненажатым состояниями.

В неопределенном состоянии свойства `CheckEditor.IsChecked` и `CheckEditor.EditorValue` возвращают значение `null`.

ButtonEditor

`ButtonEditor` - это текстовый редактор, который может отображать множество встроенных обычных кнопок и переключателей.

Основные функции контрола включают в себя:

- Обычные кнопки и переключатели.
- Отображение текста и картинок на кнопках.
- Выравнивание правой и левой кнопок.
- Всплывающие подсказки.
- Встроенная кнопка "x" для удаления значения редактора.
- Водяные знаки.

Встроенные кнопки

Чтобы добавить встроенную кнопку, добавьте объект `ButtonSettings` в коллекцию `ButtonEditor.Buttons`. Объект `ButtonSettings` содержит опции отображения кнопки и поведения.

Основные свойства и события

- `ButtonKind` — Указывает, является ли кнопка обычной или кнопкой переключения.
- `Content` — Текст кнопки или пользовательские данные. Используйте свойство `ContentTemplate`, чтобы указать шаблон для отображения пользовательских данных.
- `Glyph` — Картинка кнопки.
- `GlyphSize` — Размер картинки кнопки.
- `IsLeft` — Указывает, выровнена ли кнопка по левому или правому краю поля редактирования (по умолчанию).
- `Click` — Событие, которое позволяет вам реагировать на нажатия кнопок.
- `Command` — Указывает команду, вызываемую при нажатии на кнопку.
- `CommandParameter` — указывает параметр команды.
- `IsChecked` — Возвращает или устанавливает, кнопка нажата ли для переключения.

Пример - Как добавить обычные кнопки и переключатели

В следующем примере определяется контрол `ButtonEditor` с двумя кнопками:

- Обычная кнопка, связанная с командой *ResetValue*, которая устанавливает значение редактора равным '0'.
- Кнопка-переключатель, которая переключает свойства родительского редактора `IsTextEditable`. Редактирование текста отключено, когда кнопка нажата.

Предполагается, что проект содержит картинки '*dot.svg*', '*locked.svg*' и '*unlocked.svg*' в папке '*Images*'. Обычная кнопка отображает картинку "*dot.svg*". На кнопке-переключателе отображается картинка "*locked.svg*" или "*unlocked.svg*" в зависимости от состояния нажатия.

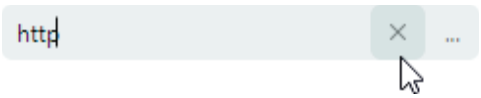
```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"
xmlns:local="clr-namespace:EditorsSample"

<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>

<mxe:ButtonEditor Name="btnEditor1"
                  EditorValue="{Binding Quantity, Mode=TwoWay}"
                  IsTextEditable="True">
    <mxe:ButtonEditor.Buttons>
        <mxe:ButtonSettings ButtonKind="Simple"
         Glyph="{SvgImage 'avares://EditorsSample/Images/dot.svg'}"
         Command="{Binding ResetValueCommand}"
         CommandParameter="{Binding #btnEditor1}"
         ToolTip.Tip = "Reset value">
        </mxe:ButtonSettings>
        <mxe:ButtonSettings ButtonKind="Toggle"
         Glyph="{Binding $self.IsChecked,
          Converter={local:LockedStateToSvgNameConverter}}"
         IsChecked="{Binding !$parent.IsTextEditable}"
         ToolTip.Tip = "Toggle text editing mode">
        </mxe:ButtonSettings>
    </mxe:ButtonEditor.Buttons>
</mxe:ButtonEditor>
```

```
namespace EditorsSample;

[ObservableObject]
public partial class MainViewModel
{
    [ObservableProperty]
    decimal quantity = 0;

    // Sets the editor's value to '0'.
    [RelayCommand]
```

```
    void ResetValue(TextEditor editor)
    {
        // You can access the current editor from the command parameter,
        // and change its value.
        editor.EditorValue = 0;
    }
}

public class LockedStateToSvgNameConverter : MarkupExtension, IValueConverter
{
    public object? Convert(object? value, Type targetType, object? parameter,
    CultureInfo culture)
    {
        if (value == null)
            return null;
        bool isLocked = (bool)value;
        string lockedState = isLocked ? "locked" : "unlocked";
        if (isLocked)
            lockedState = "locked";
        return SvgImageExtension.ProvideValue(
            $"avares://EditorsSample/Images/{lockedState}.svg", null!, null!);
    }

    public object? ConvertBack(object? value, Type targetType, object? parameter,
    CultureInfo culture)
    {
        throw new NotImplementedException();
    }

    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return this;
    }
}
```

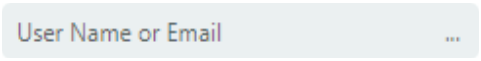
Кнопка Очистить Значение ('x')

Установите для свойства `ButtonEditor.NullValueButtonPosition` значение `ComponentPlacement.EditorBox`, чтобы включить встроенную кнопку "x", которая позволяет пользователю установить текущее значение равным null.



Водяной знак

Как и все потомки `TextEditor`, контрол `ButtonEditor` поддерживает водяной знак (подсказка, выделенная серым цветом, отображается, когда значение редактора пустое или null).

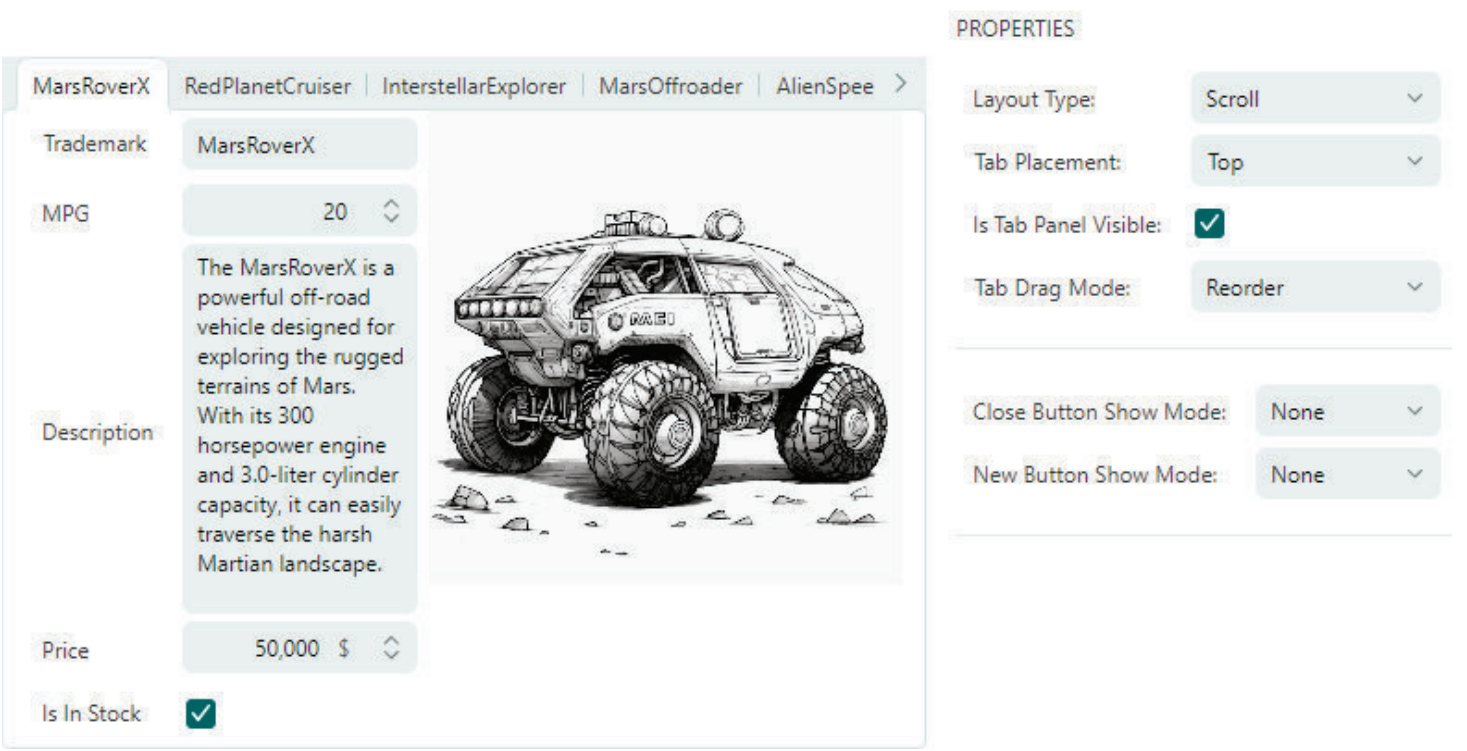


Используйте унаследованное свойство `TextEditor.Watermark`, чтобы указать водяной знак.

```
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<mxe:ButtonEditor Name="buttonEditor1" Watermark="User Name or Email" Margin="5">
    ...
</mxe:ButtonEditor>
```

Вспомогательные контролы

Множество вспомогательных контролов, поставляемых вместе с библиотекой контролов Eremex, помогут вам создать функциональный и привлекательный пользовательский интерфейс.

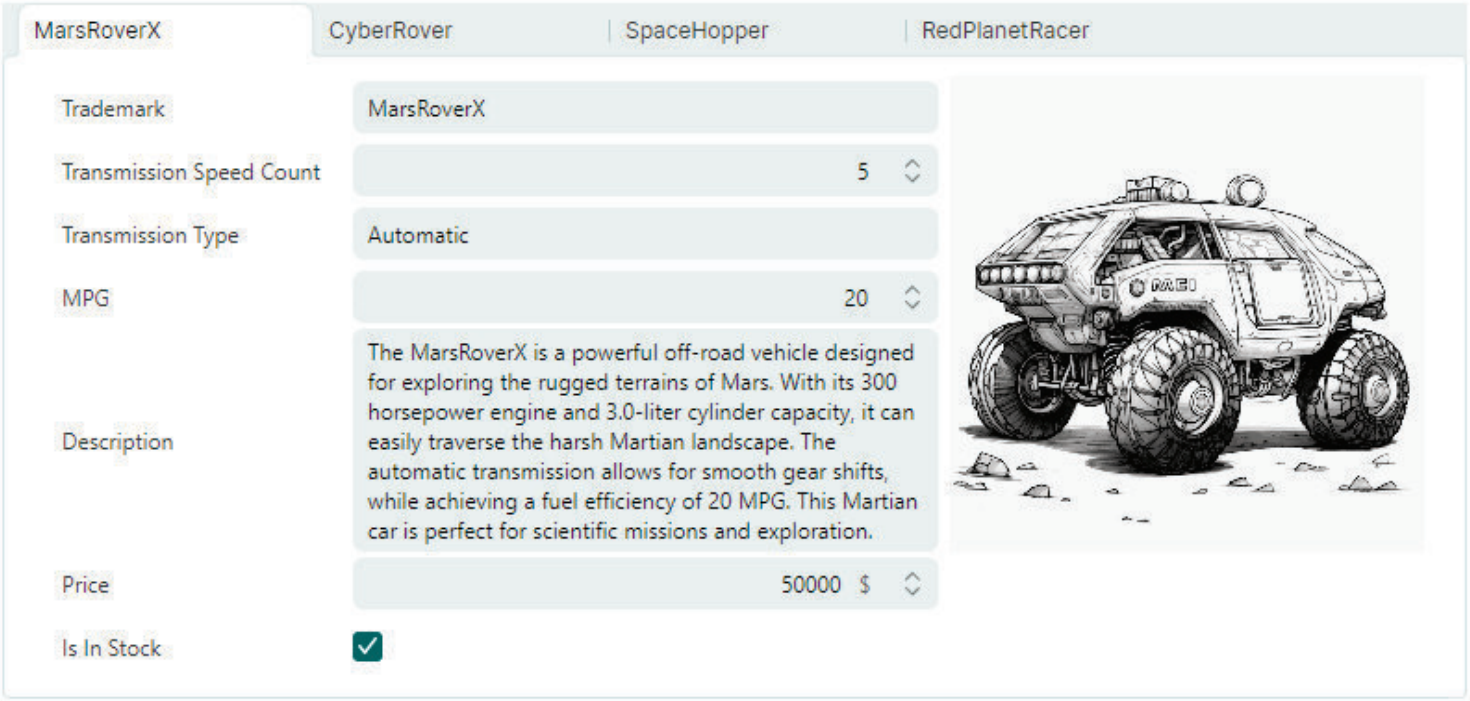


- TabControl — контрол, который позволяет объединять панели в пользовательский интерфейс с вкладками.
    - Неограниченное количество вкладок.
    - Заполнение вкладок и содержимого вкладок из источника объектов.
    - Переупорядочивание вкладок с помощью перетаскивания.
    - Режимы размещения вкладок: растягивание, прокрутка и многострочная линия.
    - Кнопки "Закрыть вкладку".
    - кнопка "Новая вкладка".
    - Пользовательские контролы в области заголовка вкладки.
- SplitContainerControl — Разделяет две панели с помощью разделителя.
    - Пользователь может перетащить разделитель, чтобы изменить размер панелей.
    - Вертикальное или горизонтальное расположение панелей.
    - Возможность сворачивать/разворачивать одну из панелей.

    - Опция для скрытия разделителя.
- `ColorEditor` — автономный контрол, который позволяет пользователю выбирать цвет.
    - ColorEditor используется как часть контрола PopupColorEditor, а также может использоваться как автономный контрол.
    - Три цветовые палитры — по умолчанию, стандартная, пользовательская.
    - Дефолтная цветовая палитра может быть инициализирована в коде.
    - Стандартная цветовая палитра отображает предопределенные стандартные цвета.
    - Пользовательская цветовая палитра позволяет пользователям добавлять и изменять цвета с помощью встроенного диалога Выбора Цвета.
    - Возможность задания цветов в форматах RGB и HSB.
- CalendarControl — автономный контрол, который позволяет пользователю выбирать дату.
    - CalendarControl используется как часть контрола DateEditor. Вы также можете использовать его как автономный контрол.
    - Выбор даты в календаре с помощью мыши и клавиатуры.
    - Панель навигации позволяет просматривать данные за месяцы и годы.
    - Три вида календаря: просмотр по месяцам, просмотр по годам и просмотр диапазона лет.
    - Опция для ограничения доступного диапазона дат.
- GroupBox — Панель с заголовком.
    - Заголовок контрола позволяет отображать пользовательский текст.
    - Контрол отображает линию под клиентской областью.

TabControl

Контрол `MxTabControl` позволяет вам организовывать панели в пользовательский интерфейс с вкладками. Пользователи могут активировать определенную панель щелчком мыши на соответствующей вкладке.



`MxTabControl` является потомком стандартного контрола `Avalonia.Controls.TabControl`. Он наследует базовую функциональность `TabControl` и расширяет ее дополнительными функциями:

- Неограниченное количество вкладок.
- Совокупность вкладок и содержимого вкладок из источника объектов.
- Переупорядочивание вкладок с помощью перетаскивания.
- Режимы размещения вкладок: Растягивание, прокрутка и многострочная линия.
- Кнопки "Закрыть вкладку".
- кнопка "Новая вкладка".
- Пользовательские контролы в области заголовка вкладки.

Создание вкладок

`MxTabControl` наследует функциональность создания вкладок от базового класса `Avalonia.Controls.TabControl`. Контролы поддерживают два подхода к добавлению вкладок:

- Добавляйте элементы вкладок вручную
- Заполняйте вкладки и содержимое вкладок из источника объектов.

Добавление элементов вкладок вручную

Элементы вкладки в `MxTabControl` контрол инкапсулируются объектами `MxTabItem`. Они предоставляют свойства для указания текста заголовка вкладки, картинки заголовка вкладки (картинки-глифа), команды "Закрыть" (команда активируется при нажатии кнопки "Закрыть" на вкладке) и т.д.

Пример

В следующем примере создаются две вкладки с предопределенным содержимым (редакторы и метки).


```
xmlns:mx="clr-namespace:Eremex.AvaloniaUI.Controls;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<mx:MxTabControl Margin="5" Height="160" Name="tabControl1" Width="300">
    <mx:MxTabItem Header="Features">
        <Grid RowDefinitions="Auto Auto Auto" ColumnDefinitions="Auto *" Margin="5">
            <Label Classes="LayoutItem" Grid.Column="0" Grid.Row="0" Content="Trademark"/>
            <mxe:TextEditor EditorValue="{Binding SelectedCar.Trademark}" Margin="5"
             Height="20" Grid.Row="0" Grid.Column="1"/>
            <Label Classes="LayoutItem" Grid.Column="0" Grid.Row="1" Content="Liters"/>
                        <mxe:SpinEditor EditorValue="{Binding SelectedCar.Liter}" Margin="5" Height="20"
             Grid.Row="1" Grid.Column="1"/>
            <Label Classes="LayoutItem" Grid.Column="0" Grid.Row="2" Content="MPG"/>
            <mxe:SpinEditor EditorValue="{Binding SelectedCar.MPG}" Margin="5" Height="20"
             Grid.Row="2" Grid.Column="1"/>
        </Grid>
    </mx:MxTabItem>
    <mx:MxTabItem Header="More Info" >
        <mxe:TextEditor Margin="5" EditorValue="{Binding SelectedCar.Description}"
         TextWrapping="Wrap" VerticalAlignment="Stretch"/>
    </mx:MxTabItem>
</mx:MxTabControl>
```

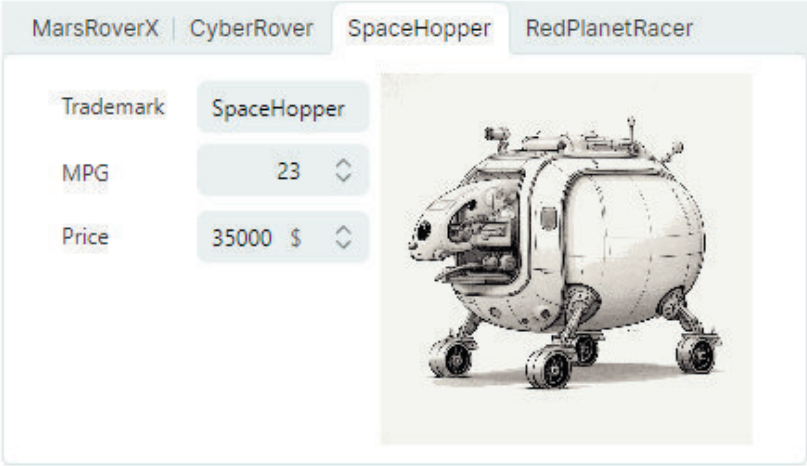
Заполнение вкладок из источника объектов

Одним из предков `MxTabControl` является класс `Avalonia.Controls.ItemsControl`. Это позволяет вам заполнить TabControl элементами из источника объектов. Используйте для этой цели следующие основные свойства:

- `ItemsSource` — Указывает источник, содержащий элементы, которые будут отображаться в виде вкладок.
- `ItemTemplate` — Указывает шаблон данных, используемый для отображения заголовков вкладок.
- `ContentTemplate` — Указывает шаблон данных, используемый для отображения содержимого вкладки.

Пример

Следующий пример привязывает контрол `MxTabControl` к источнику объектов и определяет DataTemplates, которые отображают заголовки и содержимое вкладок.




```
xmlns:mx="clr-namespace:Eremex.AvaloniaUI.Controls;assembly=Eremex.Avalonia.Controls"
xmlns:mxe="clr-namespace:Eremex.AvaloniaUI.Controls.Editors;assembly=Eremex.Avalonia.Controls"

<mx:MxTabControl ItemsSource="{Binding Cars}" SelectedItem="{Binding SelectedCar}">
    <mx:MxTabControl.ItemTemplate>
        <DataTemplate DataType="demoData:CarInfo">
            <TextBlock Text="{Binding Trademark}"></TextBlock>
        </DataTemplate>
    </mx:MxTabControl.ItemTemplate>

    <mx:MxTabControl.ContentTemplate>
        <DataTemplate DataType="demoData:CarInfo">
            <StackPanel Orientation="Horizontal" Margin="10">
                <Grid RowDefinitions="Auto Auto Auto" ColumnDefinitions="Auto *">
                    <Label Classes="LayoutItem" Grid.Column="0" Grid.Row="0" Content="Trademark"/>
                    <mxe:TextEditor Classes="LayoutItem"  Grid.Column="1" Grid.Row="0"
                     EditorValue="{Binding Trademark}"/>

                    <Label Classes="LayoutItem" Grid.Column="0" Grid.Row="1" Content="MPG"/>
                    <mxe:SpinEditor Classes="LayoutItem" Grid.Column="1" Grid.Row="1"
                     EditorValue="{Binding  MPG}" Minimum="0"/>

                    <Label Classes="LayoutItem" Grid.Column="0" Grid.Row="2" Content="Price"/>
                    <mxe:SpinEditor Classes="LayoutItem" Grid.Column="1" Grid.Row="2"
                     EditorValue="{Binding  Price}" Suffix="{Binding Currency}" Minimum="0"/>
                </Grid>
                <Image Grid.Column="1" Width="200" Source="{Binding Image}" VerticalAlignment="Top"/>
```

```
            </StackPanel>
        </DataTemplate>
    </mx:MxTabControl.ContentTemplate>
</mx:MxTabControl>
```

Выбор вкладки

Используйте следующие свойства, чтобы выбрать вкладки и получить выбранные вкладки:

- `SelectedIndex` — Указывает индекс, начинающийся с нуля, выбранной вкладки среди всех вкладок. Вы можете использовать свойство `MxTabControl.Items` для доступа к коллекции вкладок.
- `SelectedItem` — Указывает элемент выбранной вкладки. Если вкладки добавляются вручную в TabControl с использованием объектов `MxTabItem`, свойство `SelectedItem` указывает выбранный объект `MxTabItem`. Если TabControl заполняется из источника объектов, то свойство `SelectedItem` определяет объект данных выбранной вкладки из связанного источника объектов.

Свойство `SelectedValue` определяет значение выбранной вкладки. Это свойство возвращает следующие значения:

- `null`, если вкладки вручную добавлены в TabControl с использованием объектов `MxTabItem`.
- Нижележащий объект данных вкладки, если TabControl заполнен из источника объектов.

Следующие свойства позволяют вам вернуть содержимое выбранной вкладки:

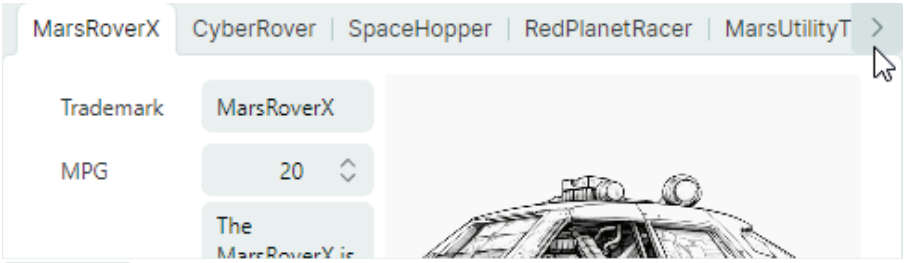
- `SelectedContent` — Указывает содержимое выбранной вкладки.
- `SelectedContentTemplate` — Указывает шаблон содержимого выбранной вкладки.

Расположение вкладок

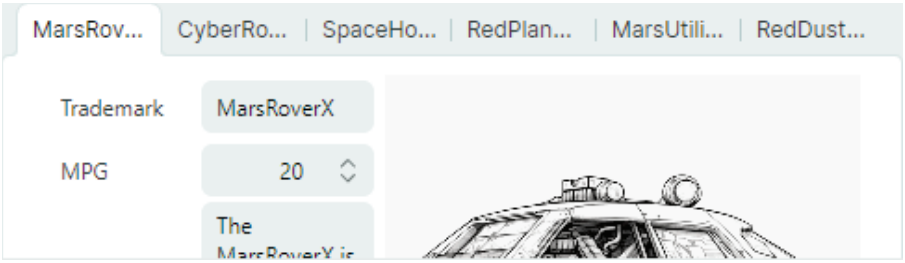
Заголовки вкладок отображаются вдоль верхнего края TabControl по умолчанию. Вы можете использовать свойство `TabStripPlacement`, чтобы указать позицию заголовков вкладок.

TabControl поддерживает три режима размещения заголовков вкладок. Используйте свойство `MxTabControl.TabStripLayoutType`, чтобы указать тип расположения заголовка вкладки:

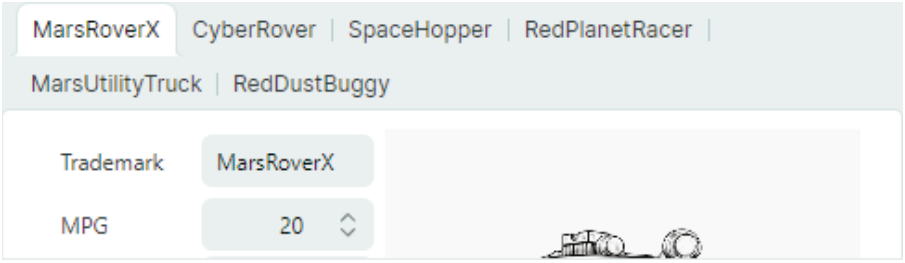
- `Scroll` — Заголовки вкладок достаточно широки для отображения содержимого заголовков вкладок. Кнопки прокрутки отображаются в области заголовка вкладки, если недостаточно места для отображения всех заголовков вкладок целиком.



- `Stretch` — Заголовки всех вкладок размещены в виде линии, растягивания, подгоняемого под ширину контрола. Они имеют одинаковую ширину или высоту в зависимости от позиции полосы вкладок (смотрите свойство `TabStripPlacement`).



- `MultiLine` — Заголовки вкладок размещены в несколько линий, если недостаточно места для их отображения в одной линии.



Перемещение вкладок

Пользователь может изменить размещение вкладок с помощью перетаскивания, если для свойства `MxTabControl.TabDragMode` установлено значение `Reorder`.

Обработайте события `TabItemStartDragging` и `TabItemCompleteDragging`, чтобы отменить операции перетаскивания или выполнить дополнительные действия при перетаскивании вкладок.

Кнопки "Закрыть"

TabControl поддерживает встроенные кнопки "Закрыть вкладку" ("x"). Они могут отображаться на вкладках и/или в области заголовка вкладки, как указано в свойстве `MxTabControl.CloseButtonShowMode`.

Кнопки "Закрыть вкладку" по умолчанию не выполняют никаких действий. Вам необходимо реализовать действия с кнопкой "Закрыть", используя один из следующих подходов:

- Обработайте событие `MxTabControl.CloseButtonClick`.

- Укажите команды с помощью элемента `MxTabItem.CloseCommand`.

Кнопка "Новая вкладка"

Установите для свойства `MxTabControl.NewButtonShowMode` значение `InHeaderPanel`, чтобы отобразить кнопку "Новая вкладка" ("+"). Нажатие на эту кнопку по умолчанию не приводит к каким-либо действиям. Вы можете указать действие со следующими элементами:

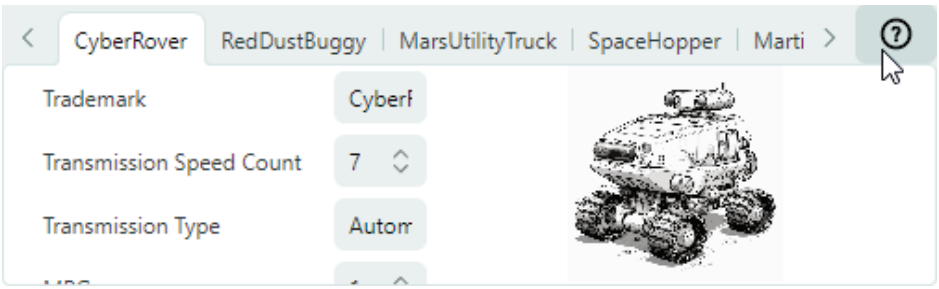
- Событие `MxTabControl.NewButtonClick`.
- Команда `MxTabControl.NewCommand`.

Пользовательские контролы в области заголовка вкладки

Вы можете использовать свойства `ControlBoxContent` и `ControlBoxContentTemplate`, чтобы добавить пользовательские контролы в область заголовка вкладки.

Пример

В следующем примере кнопка "?" добавлена в область заголовка вкладки. Нажатие на кнопку вызывает команду *HelpCommand*. Элемент текущей выбранной вкладки передается в качестве параметра команды.




```
xmlns:mx="clr-namespace:Eremex.AvaloniaUI.Controls;assembly=Eremex.Avalonia.Controls"
x:Class="SampleTabControl.TabControlPageView"
xmlns:vm="using:SampleTabControl"
x:DataType="vm:TabControlPageViewModel"

<mx:MxTabControl ItemsSource="{Binding Cars}"
                 SelectedItem="{Binding SelectedCar}"
                 x:Name="TabControl"
    >
    <mx:MxTabControl.ControlBoxContent>
        <Button Command="{Binding HelpCommand}"
         CommandParameter="{Binding $parent.SelectedItem}">
            <Image Width="20" Height="20"
             Source="{SvgImage 'avares://SampleTabControl/Images/help-icon.svg'}"/>
        </Button>
    </mx:MxTabControl.ControlBoxContent>
```

```
    ...
</mx:MxTabControl>
```

```
public partial class TabControlPageViewModel : PageViewModelBase
{
    //...
    [RelayCommand]
    private void Help(object parameter)
    {
        //...
    }
}
```

SplitContainerControl

`SplitContainerControl` - это составной контрол, который отображает две панели, разделенные подвижным разделителем. Пользователи могут перетаскивать разделитель, чтобы изменить размер панелей. Они также могут щелкнуть разделитель, чтобы свернуть выбранную панель, а затем снова щелкнуть разделитель, чтобы восстановить панель.

Основные функции контрола включают в себя:

- Пользователи могут перетаскивать разделитель, чтобы изменить размер панелей.
- Настройка размера свойств панелей в коде.
- Вертикальное или горизонтальное расположение панелей.
- Возможность сворачивать/разворачивать одну из панелей.
- Опция для скрытия разделителя.

Содержимое для панелей

Используйте свойства `SplitContainerControl.Panel1` и `SplitContainerControl.Panel2` для размещения содержимого на панелях контрола. Поддерживаются два сценария использования:

- Инициализируйте эти свойства с помощью контролов, которые будут отображаться на панелях.
- Инициализируйте эти свойства пользовательскими объектами. В этом случае используйте свойства `SplitContainerControl.Panel1Template` и `SplitContainerControl.Panel2Template`, чтобы указать DataTemplates, которые будут отображать пользовательские объекты.

Настройка размера и направления панели

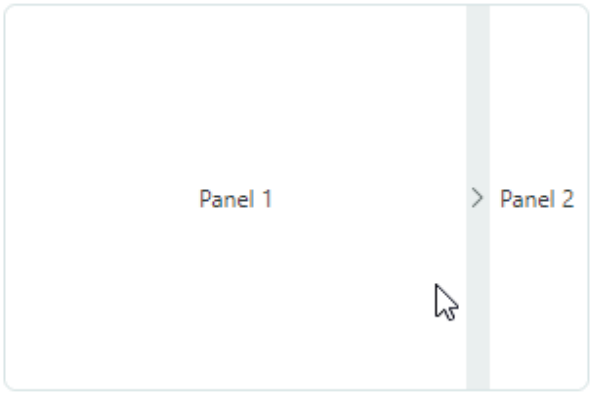
Используйте свойство `SplitContainerControl.Orientation`, чтобы выбрать между горизонтальным (по умолчанию) и вертикальным расположением панелей.

Если панели размещены горизонтально, вы можете задать ширину одной из панелей, используя свойство `SplitContainerControl.Panel1Length` или `SplitContainerControl.Panel2Length`. Если вы установите свойство `Panel1Length`, вторая панель заполнит оставшееся пространство. И наоборот, если вы установите свойство `Panel2Length`, первая панель заполнит оставшееся пространство. Не устанавливайте оба свойства `Panel1Length` и `Panel2Length` одновременно, так как второе свойство в этом случае игнорируется.

Если панели размещены вертикально, вы можете использовать свойство `Panel1Length` или `Panel2Length`, чтобы задать высоту одной из панелей.

Сворачивание и восстановление панели

Значок стрелки, отображаемый в разделителе, указывает на то, что панель будет свернута, когда пользователь щелкнет по разделителю. В свернутом состоянии значок стрелки разделителя меняет направление. Пользователь может щелкнуть по разделителю еще раз, чтобы восстановить панель.



Свойство `SplitContainerControl.CollapsePanel` позволяет указать сворачиваемую панель. Значение свойства по умолчанию - *Panel2*.

Чтобы свернуть и восстановить панель в коде, используйте свойство `IsCollapsed`.

Отключение сворачивания панели

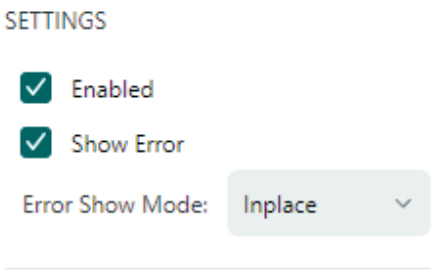
Установите для свойства `SplitContainerControl.CollapsePanel` значение `None`, чтобы отключить функцию сворачивания панели. В этом режиме разделитель не отображает значки со стрелками.

Скрытие разделителя

Используйте свойство `SplitContainerControl.IsSplitterVisible` для `false`, чтобы скрыть разделитель в определенных случаях. Это не позволяет пользователю выполнять операции изменения размера и сворачивания/восстановления панелей.

GroupBox

`GroupBox` - это панель с заголовком и линией внизу, которая визуально отделяет `GroupBox` от других контролов.



`GroupBox` является потомком `Avalonia.Controls.Primitives.HeaderedContentControl`.

Заголовок

Используйте свойство `Header`, чтобы задать заголовок `GroupBox`. Свойство `HeaderTemplate` позволяет вам настроить шаблон для отображения заголовка контрола произвольным способом.

Связанный API

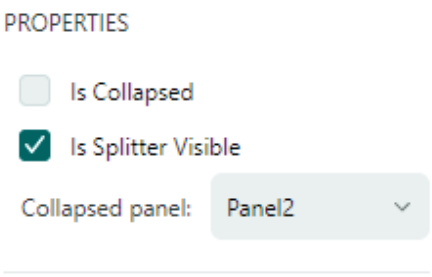
- `ShowHeader` — Получает или устанавливает, является ли заголовок видимым.
- `HeaderHorizontalAlignment` — Задает горизонтальное выравнивание заголовка.
- `HeaderVerticalAlignment` — Задает выравнивание заголовка по вертикали.

Определение содержимого

Вы можете указать содержимое `GroupBox` в XAML между начальным и конечным тегами `<GroupBox>`. В коде для этой цели используйте унаследованное свойство `Content`. Вы можете установить свойство `ContentTemplate`, чтобы указать шаблон, используемый для отображения содержимого контрола.

Пример

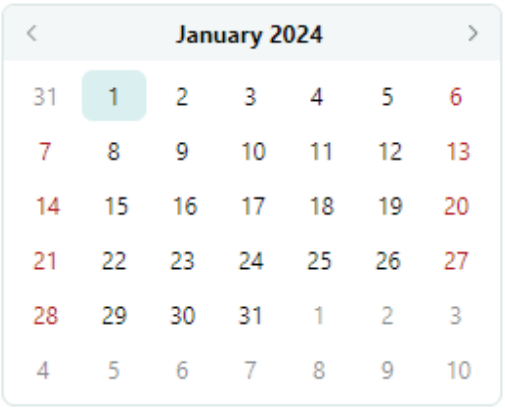
В следующем примере определяется `GroupBox`, который отображает StackPanel с контролами.




```
xmlns:mxe="using:Eremex.AvaloniaUI.Controls.Editors"

<mxe:GroupBox Header="PROPERTIES">
    <StackPanel>
        <mxe:CheckEditor x:Name="IsCollapsedSelector" Content="Is Collapsed" Classes="LayoutItem"/>
        <mxe:CheckEditor x:Name="IsSplitterVisibleSelector"
         Content="Is Splitter Visible" IsChecked="True" Classes="LayoutItem"/>
        <DockPanel>
            <Label Content="Collapsed panel:" Classes="LayoutItem"/>
            <mxe:ComboBoxEditor EditorValue="{Binding CollapsedPanel, Mode=TwoWay}"
             ItemsSource="{mxc:EnumItemsSource EnumType=mxe:SplitContainerControlCollapsePanel}"/>
        </DockPanel>
    </StackPanel>
</mxe:GroupBox>
```

CalendarControl

`CalendarControl` отображает календарь, который позволяет пользователю выбрать дату. В заголовке навигации контрола отображаются кнопки для просмотра по месяцам и годам.



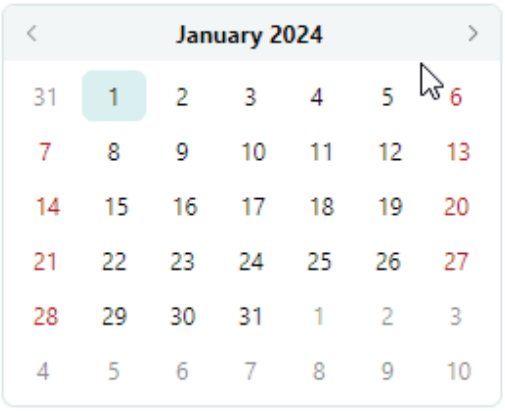
Основные функции контрола включают в себя:

- Выбор даты в календаре с помощью мыши и клавиатуры.
- Панель навигации позволяет просматривать данные за месяцы и годы.
- Три вида календаря: просмотр по месяцам, просмотр по годам и просмотр диапазона лет.
- Опция для ограничения доступного диапазона дат.

Выбор даты

Пользователь может выбрать дату с помощью клавиш со стрелками мыши и клавиатуры.

Навигационный заголовок календаря позволяет пользователю просматривать месяцы и годы. Пользователь может щелкнуть по тексту заголовка, чтобы уменьшить масштаб текущего представления. При просмотре месяца щелчок по тексту заголовка переключает календарь на просмотр за год. В представлении "Год" щелчок по тексту заголовка переключает календарь в представление "диапазон лет".



Используйте свойство `CalendarControl.SelectedDate`, чтобы выбрать дату или прочитать текущую выбранную дату.

Настройка календаря

Используйте следующие свойства для настройки календаря:

- `FirstDayOfWeek` — Возвращает или устанавливает день недели, который стоит первым в представлении месяца календаря.
- `IsTodayHighlighted` — Возвращает или устанавливает, следует ли выделять сегодняшнюю дату в календаре.
- `DisplayDateStart` — указывает минимально допустимую дату. Свойства `DisplayDateStart` и `DisplayDateEnd` позволяют указать диапазон значений, отображаемых в календаре.
- `DisplayDateEnd` — указывает максимально допустимую дату.