UXDataFilter Class

Remarks

The **UXDataFilter** control provides a configurable user interface for filtering through a data collection. Similar to other ClientUI data controls, **UXDataFilter** supports both server side and client side operation.

Client Data Operation

Client data operation means that the data operation in this case data filtering is executed in client side against the data source provided to **UXDataFilter**. To use this mode, you set the QueryOperation property to **Client**.

You need to wrap your collection in a PagedCollectionView class to provide data filtering functionality to the **IEnumerable** collection. The PagedCollectionView provides consistent handling for data operation in other data controls as well such as UXGridView and UXDataPager.

To learn how to implement data filter using **UXDataFilter**, see How-to: Implement Data Filtering using UXDataFilter.

Server Data Operation

Server data operation means that the data operation, in this case data filtering, is processed in the server. This means that **UXDataFilter** does not handle the filtering operation by its own. Instead, **UXDataFilter** provides the query information allowing you to process it further.

To use this mode, you set the QueryOperation property to **Server**. When this mode is selected, **UXDataFilter** will not attempt to perform the data operation on the given data source. Instead, it will store and distribute the query information to FilterDescriptors property. When the collection of the property change, the QueryChanged event of the associated QueryDescriptor will be raised. This allows you to streamline the query processing in a centralized function, which is one of the strong benefits of QueryDescriptor. For more information about QueryDescriptor, see QueryDescriptor Overview.

To learn how to filter data using **FilterDescriptors** and **UXDataFilter**, see How-to: Implement Data Filtering using FilterDescriptors and UXDataFilter.

Working with IsBatchFilter Property and ValueMemberPath Property

By default, the filtering process takes place immediately when an item is checked or unchecked. If you prefer to process the filtering in batch, you can set the **IsBatchFilter** property **true**. In this mode, **UXDataFilter** will provide you with **Apply** and **Cancel** button which you can use to apply or cancel the changes you made in the **UXDataFilter**.

To populate the **UXDataFilter** from a collection, you can assign the collection to **ItemsSource** property and set the **DisplayMemberPath** for the displayed text and ValueMemberPath for the filter expression. If the ValueMemberPath is not specified, **UXDataFilter** will use the member path specified in **DisplayMemberPath**.

XAML

```
<Intersoft:UXDataFilter FilterDescriptors="{Binding QueryDescriptor.FilterDescriptors,
Mode=TwoWay}"
 ItemsSource="{Binding Categories}" QueryOperation="Server" IsBatchFilter="True"
 Header="By Category:" DisplayMemberPath="CategoryName" ValueMemberPath="CategoryID"
Margin="8,0">
</Intersoft:UXDataFilter>
```

Working with SearchBoxVisibility Property

In certain cases, you might end up with relatively large number of filter items in the **UXDataFilter**, which makes it difficult for users to find the items to include or exclude in the data filtering process. To address this challenge, you can enable search functionality feature by setting the **SearchBoxVisibility** property of the **UXDataFilter** to **Visible**.

As the results, a search box element will appear in the top of the check boxes which allow users to search the particular items to include or exclude during the data filtering process. The search box element is using UXSearchBox control, therefore it inherits all the rich user experience features already available in the control.

XAML

```
<Intersoft:UXDataFilter FilterDescriptors="{Binding QueryDescriptor.FilterDescriptors,
Mode=TwoWay}"
 ItemsSource="{Binding Categories}" QueryOperation="Server"
SearchBoxVisibility="Visible"
 Header="By Category:" DisplayMemberPath="CategoryName" ValueMemberPath="CategoryID"
Margin="8,0">
</Intersoft:UXDataFilter>
```

Definition

public class UXDataFilter : ISHeaderedItemsControl

Summary

The following table summarizes the members exposed in this class.

Public Constructors

UXDataFilter Constructor()	Initializes a new instance of the UXDataFilter class.

Public Properties

CollectionView	Gets or sets the data collection that the UXDataFilter controls filtering for.
FilterDescriptors	Gets or sets the descriptor object that encapsulates the filtering related information.
FilterMemberPath	Gets or sets the id member path that will be used to generate the filter descriptors.
GroupBoxStyle	Gets or sets the style that will be used for the group box element.
HeaderVisibility	Gets or sets group box's header element visibility
HorizontalScrollBarVisibility	
IsBatchFilter	Gets or sets a value that indicates whether batch filtering is enabled.
QueryOperation	Gets or sets a value that determines whether the filtering operation should be performed in client-side or server-side.
RootElement	Gets the root element.
ScrollViewerStyle	Gets or sets the style that will be used for the scroll viewer element.
SearchBoxVisibility	Gets or sets the search box element visibility.
ValueMemberPath	Gets or sets the value member path that will be used to generate the filter descriptors.
VerticalScrollBarVisibility	

Protected Properties

Product	Product Info.

Fields

CollectionViewProperty	Identifies the CollectionView dependency property.
FilterDescriptorsProperty	Identifies the FilterDescriptors dependency property.
FilteringEvent	Identifies the FilteringEvent routed event.
FilterMemberPathProperty	Identifies the FilterMemberPath dependency property.
GroupBoxStyleProperty	Identifies the GroupBoxStyle dependency property.
HeaderVisibilityProperty	Identifies the HeaderVisibility dependency property.
HorizontalScrollBarVisibilityProperty	Identifies the HorizontalScrollBarVisibility dependency property.
IsBatchFilterProperty	Identifies the IsBatchFilter dependency property.
QueryOperationProperty	Identifies the QueryOperation dependency property.
ScrollViewerStyleProperty	Identifies the ScrollViewerStyle dependency property.
SearchBoxVisibilityProperty	Identifies the SearchBoxVisibility dependency property.
ValueMemberPathProperty	Identifies the ValueMemberPath dependency property.
VerticalScrollBarVisibilityProperty	Identifies the VerticalScrollBarVisibility dependency property.

Public Methods

AttachEventHandlers()	Attach built-in event handlers to control templates. Call this method if necessary.
DetachedEventHandlers()	Detach built-in event handlers from control templates. Call this method if necessary.
InitializeTemplates()	Initializes control templates.
OnApplyTemplate()	Builds the visual tree for the UXDataFilter when a new template is applied.

Protected Methods

GetContainerForItemOverride()	Creates or identifies the element that is used to display the given item.
IsItemItsOwnContainerOverride(object)	Determines if the specified item is (or is eligible to be) its own container.
PrepareContainerForItemOverride(DependencyObject, object)	Prepares the specified element to display the specified item.

Events

Filtering	Occurs when the process filtering is ended.

Public Constructors

public UXDataFilter()

Initializes a new instance of the UXDataFilter class.

Public Properties

public PagedCollectionView CollectionView { get; set; }

Gets or sets the data collection that the UXDataFilter controls filtering for.

public CompositeFilterDescriptorCollection FilterDescriptors { get; set; }

Gets or sets the descriptor object that encapsulates the filtering related information.
Remarks

The UXDataFilter control provides a configurable user interface for filtering through a data collection. Similar to other ClientUI data controls, UXDataFilter supports both server side and client side operation.

Client Data Operation

Client data operation means that the data operation in this case data filtering is executed in client side against the data source provided to UXDataFilter. To use this mode, you set the UXDataPager property to Client. You need to wrap your collection in a PagedCollectionView class to provide data filtering functionality to the IEnumerable collection. The PagedCollectionView provides consistent handling for data operation in other data controls as well such as UXGridView and UXDataPager. To learn how to implement data filter using UXDataFilter, see How-to: Implement Data Filtering using UXDataFilter.

Server Data Operation

Server data operation means that the data operation, in this case data filtering, is processed in the server. This means that UXDataFilter does not handle the filtering operation by its own. Instead, UXDataFilter provides the query information allowing you to process it further. To use this mode, you set the UXDataPager property to Server. When this mode is selected, UXDataFilter will not attempt to perform the data operation on the given data source. Instead, it will store and distribute the query information to FilterDescriptors property. When the collection of the property change, the QueryChanged event of the associated QueryDescriptor will be raised. This allows you to streamline the query processing in a centralized function, which is one of the strong benefits of QueryDescriptor. For more information about QueryDescriptor, see QueryDescriptor Overview. To learn how to filter data using FilterDescriptors and UXDataFilter, see How-to: Implement Data Filtering using FilterDescriptors and UXDataFilter.

public string FilterMemberPath { get; set; }

Gets or sets the id member path that will be used to generate the filter descriptors.

public Style GroupBoxStyle { get; set; }

Gets or sets the style that will be used for the group box element.

public Visibility HeaderVisibility { get; set; }

Gets or sets group box's header element visibility

public ScrollBarVisibility HorizontalScrollBarVisibility { get; set; }

public bool IsBatchFilter { get; set; }

Gets or sets a value that indicates whether batch filtering is enabled.
Remarks

By default, the filtering process takes place immediately when an item is checked or unchecked. If you prefer to process the filtering in batch, you can set the IsBatchFilter property true. In this mode, UXDataFilter will provide you with Apply and Cancel button which you can use to apply or cancel the changes you made in the UXDataFilter.

public QueryOperation QueryOperation { get; set; }

Gets or sets a value that determines whether the filtering operation should be performed in client-side or server-side.
Remarks

The UXDataFilter control provides a configurable user interface for filtering through a data collection. Similar to other ClientUI data controls, UXDataFilter supports both server side and client side operation.

Client Data Operation

Client data operation means that the data operation in this case data filtering is executed in client side against the data source provided to UXDataFilter. To use this mode, you set the QueryOperation property to Client. You need to wrap your collection in a PagedCollectionView class to provide data filtering functionality to the IEnumerable collection. The PagedCollectionView provides consistent handling for data operation in other data controls as well such as UXGridView and UXDataPager. To learn how to implement data filter using UXDataFilter, see How-to: Implement Data Filtering using UXDataFilter.

Server Data Operation

Server data operation means that the data operation, in this case data filtering, is processed in the server. This means that UXDataFilter does not handle the filtering operation by its own. Instead, UXDataFilter provides the query information allowing you to process it further. To use this mode, you set the QueryOperation property to Server. When this mode is selected, UXDataFilter will not attempt to perform the data operation on the given data source. Instead, it will store and distribute the query information to UXDataFilter property. When the collection of the property change, the QueryChanged event of the associated QueryDescriptor will be raised. This allows you to streamline the query processing in a centralized function, which is one of the strong benefits of QueryDescriptor. For more information about QueryDescriptor, see QueryDescriptor Overview. To learn how to filter data using FilterDescriptors and UXDataFilter, see How-to: Implement Data Filtering using FilterDescriptors and UXDataFilter.

public Grid RootElement { get; }

Gets the root element.

public Style ScrollViewerStyle { get; set; }

Gets or sets the style that will be used for the scroll viewer element.

public Visibility SearchBoxVisibility { get; set; }

Gets or sets the search box element visibility.
Remarks

In certain cases, you might end up with relatively large number of filter items in the UXDataFilter, which makes it difficult for users to find the items to include or exclude in the data filtering process. To address this challenge, you can enable search functionality feature by setting the SearchBoxVisibility property of the UXDataFilter to Visible. As the results, a search box element will appear in the top of the check boxes which allow users to search the particular items to include or exclude during the data filtering process. The search box element is using UXSearchBox control, therefore it inherits all the rich user experience features already available in the control.

public string ValueMemberPath { get; set; }

Gets or sets the value member path that will be used to generate the filter descriptors.
Remarks

To populate the UXDataFilter from a collection, you can assign the collection to ItemsSource property and set the DisplayMemberPath for the displayed text and UXDataFilter for the filter expression. If the UXDataFilter is not specified, UXDataFilter will use the member path specified in DisplayMemberPath.

public ScrollBarVisibility VerticalScrollBarVisibility { get; set; }

Protected Properties

protected ProductInfo Product { get; }

Product Info.

Fields

public static readonly DependencyProperty CollectionViewProperty

Identifies the CollectionView dependency property.

public static readonly DependencyProperty FilterDescriptorsProperty

Identifies the FilterDescriptors dependency property.

public static readonly RoutedEvent FilteringEvent

Identifies the FilteringEvent routed event.

public static readonly DependencyProperty FilterMemberPathProperty

Identifies the FilterMemberPath dependency property.

public static readonly DependencyProperty GroupBoxStyleProperty

Identifies the GroupBoxStyle dependency property.

public static readonly DependencyProperty HeaderVisibilityProperty

Identifies the HeaderVisibility dependency property.

public static readonly DependencyProperty HorizontalScrollBarVisibilityProperty

Identifies the HorizontalScrollBarVisibility dependency property.

public static readonly DependencyProperty IsBatchFilterProperty

Identifies the IsBatchFilter dependency property.

public static readonly DependencyProperty QueryOperationProperty

Identifies the QueryOperation dependency property.

public static readonly DependencyProperty ScrollViewerStyleProperty

Identifies the ScrollViewerStyle dependency property.

public static readonly DependencyProperty SearchBoxVisibilityProperty

Identifies the SearchBoxVisibility dependency property.

public static readonly DependencyProperty ValueMemberPathProperty

Identifies the ValueMemberPath dependency property.

public static readonly DependencyProperty VerticalScrollBarVisibilityProperty

Identifies the VerticalScrollBarVisibility dependency property.

Public Methods

public void AttachEventHandlers()

Attach built-in event handlers to control templates. Call this method if necessary.

public void DetachedEventHandlers()

Detach built-in event handlers from control templates. Call this method if necessary.

public void InitializeTemplates()

Initializes control templates.

public void OnApplyTemplate()

Builds the visual tree for the UXDataFilter when a new template is applied.

Protected Methods

protected DependencyObject GetContainerForItemOverride()

Creates or identifies the element that is used to display the given item.
Return Types

The element that is used to display the given item.

protected bool IsItemItsOwnContainerOverride(object item)

Determines if the specified item is (or is eligible to be) its own container.
Parameters

item	The item to check.

Return Types

true if the item is (or is eligible to be) its own container; otherwise, false.

protected void PrepareContainerForItemOverride(DependencyObject element, object item)

Prepares the specified element to display the specified item.
Parameters

element	The container element used to display the specified item.
item	The item to display.

Events

public event ISRoutedEventHandler Filtering

Occurs when the process filtering is ended.

Namespace: Intersoft.Client.UI.Data

Assembly Information: Intersoft.Client.UI.Data, Version 3.0.5000.1

Target Frameworks: Silverlight 5, WPF 4