ComponentOne

MultiSelect for WPF

Table of Contents

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

MultiSelect for WPF

MultiSelect for WPF (Beta) is a control that provides the ease of selecting multiple objects from a list or a collection of selected items. It comprises two elements, C1TagEditor and C1CheckList, which can be used as stand-alone controls as well. Flexible C1TagEditor gives you an option to display the selected items either as strings or as tags so that you can easily give your application an Office365 Outlook-like interface. Not just this, this control is smart enough to display the summarized text or tag instead of all selected items if the count goes beyond a specified limit. Moreover, C1CheckList element of the MultiSelect control allows you to highlight your selection as a checklist or as a simple list of items. Above all, the control also supports data binding with TagDataSource and DataSource apart from the unbound mode.

Help with WPF Edition

For information on installing **ComponentOne Studio WPF Edition**, licensing, theming, technical support, namespaces and creating a project with the control, please visit Getting Started with WPF Edition.

Key Features

- **Effortless Selection**
  MultiSelect allows you to select specific or all items from a defined list of items, which are displayed in the control header post selection. It provides you control over selection of items through Single, Multiple, Extended modes which define how items in the list can be selected. MultiSelect also allows you to access selected items from the list which further can be used as data source for any other control to display items as per the requirement.

- **Add/Remove Items**
  MultiSelect allows you to add and remove items from the list through code or directly from the control header.

- **AutoSuggest**
  MultiSelect offers the functionality of providing suggestions depending on the text you enter in the control header. As soon as you enter text, the dropdown opens and all the items matching the alphabet(s) or word you enter are displayed in the list.

- **Smart Header**
  MultiSelect allows you to control the number of items to be displayed in header. The control header displays selected items if the number of selected items is less than or equal to the value set for the MaxHeaderItems property and if number of selected items is greater than MaxHeaderItems the header displays the count of selected items.

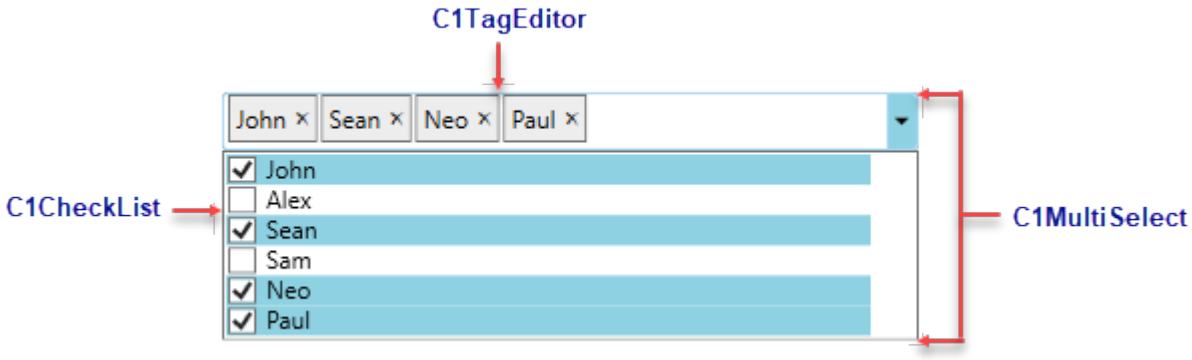
- **Edit Mode**
  MultiSelect supports text input which makes the selected tags editable in both the appearances – comma separated strings and tags. To change an item, you can simply double click on it to edit.

- **Bind to DataSource**
  MultiSelect can be bound to two different data sources, TagsDataSource and DataSource. TagsDataSource can be used if the application author wants to propagate end-user selection to some other data source, probably with other data structure than the first one. On the other hand, DataSource is used to fill the list of all available items that are shown in dropdown checklist.

Elements

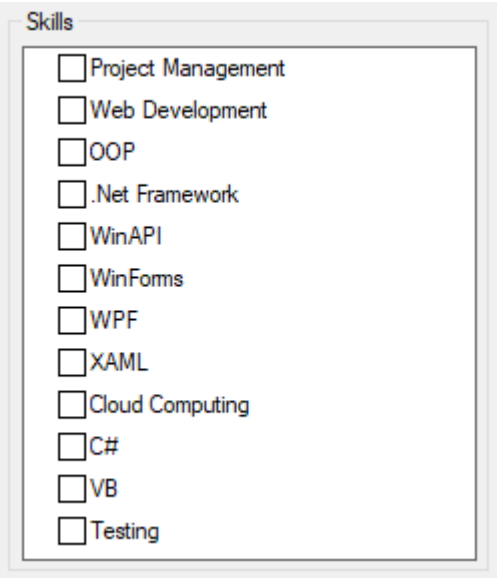
The MultiSelect control mainly consists of two elements which are available in **C1.WPF.Input.dll** and can be used as stand-alone controls as well.

- **C1CheckList**
- **C1TagEditor**



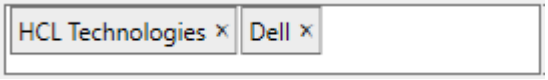
C1CheckList

The C1CheckList control, instantiated using C1CheckList class, displays a collection of items in a static list, and allows user to select desired items from a defined list. For example, C1CheckList can be used to display a list of skills in a CV form on a job portal as shown in the following image.



C1TagEditor

The C1TagEditor control, instantiated using C1TagEditor class, provides the user with a textbox area where each tag behaves as an individual entry which can be inserted, edited, and removed individually. For example, C1TagEditor can be used to enter the name of all the previous workplaces in a CV form on a job portal. These names are added in the control as individual tags which can be edited or removed later.



To understand the implementation of the controls, refer the MultiSelect sample available at the default installation

folder.
Documents\ComponentOne Sample\WPF\C1.WPF.Input\CS\InputSamples

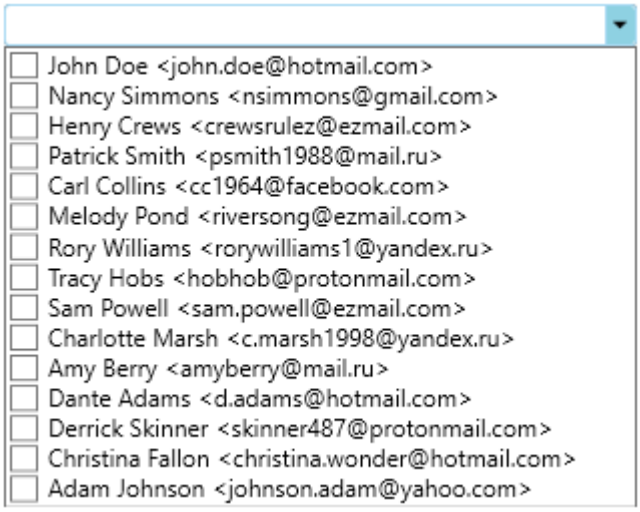
Quick Start

This quick start will guide you through the steps of adding C1MultiSelect to a project and binding the control to a data source.

Complete the steps given below to see how the MultiSelect control appears after data binding.

1. **Adding MultiSelect control to the Application**
2. **Binding MultiSelect to a list**

The following image shows how the MultiSelect control appears after data binding.



Step 1: Adding MultiSelect control to the Application

1. Create a new **WPF App** in **Visual Studio**.
2. Drag and Drop the **C1MultiSelect** control from the toolbox onto the form. The following references automatically get added to the **References**.
   - C1.WPF.Input.4.dll
   - C1.WPF.4.dll
3. Open **MainWindow.xaml** and replace the existing XAML with the following code.

C#

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
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:local="clr-namespace:MultiSelectQS_WPF"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="MultiSelectQS_WPF.QuickStart" mc:Ignorable="d" Title="QuickStart"
Height="386.701" Width="451.918" Loaded="Window_Loaded">
  <Grid>
        <c1:C1MultiSelect x:Name="mselect" HorizontalAlignment="Left"
Height="21" VerticalAlignment="Top" Width="316" Margin="40,79,0,0"/>
  </Grid>
</Window>
```

Step 2: Binding MultiSelect to a list

1. Open **MainWindow.xaml.cs** and add following code to **Window_Loaded** event.

C#

```
IList<string> addressBook = new List<string>()
{
        "John Doe <john.doe@hotmail.com>",
        "Nancy Simmons <nsimmons@gmail.com>",
        "Henry Crews <crewsrulez@ezmail.com>",
        "Patrick Smith <psmith1988@mail.ru>",
        "Carl Collins <cc1964@facebook.com>",
        "Melody Pond <riversong@ezmail.com>",
        "Rory Williams <rorywilliams1@yandex.ru>",
        "Tracy Hobs <hobhob@protonmail.com>",
        "Sam Powell <sam.powell@ezmail.com>",
        "Charlotte Marsh <c.marsh1998@yandex.ru>",
        "Amy Berry <amyberry@mail.ru>",
        "Dante Adams <d.adams@hotmail.com>",
        "Derrick Skinner <skinner487@protonmail.com>",
        "Christina Fallon <christina.wonder@hotmail.com>",
        "Adam Johnson <johnson.adam@yahoo.com>"
};

mselect.ItemsSource = addressBook;
```

2. Build and run the application.

Use MultiSelect Control

MultiSelect allows you to add, remove, and access specific items with minimal code. It also lets you display or hide the check boxes and dropdown button appearing in the control. Learn how they can be implemented.

- **Add an item**
- **Edit an item**
- **Remove an Item**
- **Access specific item**
- **Show/Hide check boxes**
- **Show/Hide dropdown button**

Add an item

To add items to the MultiSelect control, use **Add** method of ItemCollection class as shown in the following code. For example, the following code adds “Edward” in dropdown list of the MultiSelect control:

C#

```
mselect.Items.Add("Edward");
```

MultiSelect also allows you to add an item at a specific position using **Insert** method of ItemCollection class and specify an index value for the new item in it. For example, the following code inserts a name, Robert, to the second position, adjusting the position of the other items in the dropdown list:

C#

```
mselect.Items.Insert(1, "Robert");
```

Back to Top

Edit an item

MultiSelect provides you an option to allow or restrict a user to edit tags in the control header through IsTagEditable property.

C#

```
mselect.IsTagEditable = false;
```

Back to Top

Remove an item

MultiSelect lets you delete an item from the list using **RemoveAt** method of ItemCollection class as shown in the following code. The method takes one argument, index, which specifies the item to remove. For example, the following code deletes sixth entry from the list.

C#

```
mselect.Items.RemoveAt(5);
```

MultiSelect also allows you to delete a selected item from the MultiSelect control using **Remove** method as shown in the following code:

C#

```
mselect.Items.Remove(mselect.SelectedItem);
```

To remove all the entries from the list, use **Clear** method as shown in the following code:

C#

```
mselect.Items.Clear();
```

Back to Top

Access specific item

To access and change a specific item from the MultiSelect control dropdown list, use Items property of C1MultiSelect class and specify the index of that item. For example, to access and change the value of second item from the list, write the following line of code:

C#

```
mselect.Items[1] = "Jake";
```

Back to Top

Show/Hide check boxes

By default, list of items in the MultiSelect control are displayed with check boxes. However, you can disable the default style by setting ShowCheckBoxes property to **false**.

To hide the check boxes from the list, use the following code:

C#

```
mselect.ShowCheckBoxes = false;
```

Back to Top

Show/Hide dropdown button

MultiSelect displays the dropdown button to show the list of available items. However, you can hide the dropdown button in the control by setting ShowDropDownButton property to **false**.

To hide the drop down button, use the following code:

C#

```
mselect.ShowDropDownButton = false;
```

Back to Top

Data Binding

MultiSelect provides data binding support that lets you populate data in the control. It allows you to bind the control to complex objects and data sources. To bind MultiSelect to a data source, you need to access the **DisplayMemberPath** property of ItemsControl class. This property contains information about path to a value on the source object to serve as the visual representation of the object. There are a few more properties which are frequently required for binding. These properties are CheckedMemberPath, DisabledMemberPath, DisplayMemberPath and ItemsSource.

You can bind MultiSelect using any of the following ways:

Bind MultiSelect to a data source
: Learn how to bind MultiSelect to a data source in code.

Bind MultiSelect to object collection
: Learn how to bind MultiSelect to object collection in code.

MultiSelect in unbound mode
: Learn how to implement MultiSelect in unbound mode through code.

Bind MultiSelect to a Data Source

To bind MultiSelect to a data source, follow these steps:

1. Create a connection string and fetch data from a database to a data set.

C#

```
static string GetConnectionString()
{
      string conn = @"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Users\GPCTAdmin\Documents\ComponentOne Samples\Common\C1NWind.mdb;";
      return string.Format(conn, path);
}
DataTable GetDataSource(string connectionString)
{
       // set up connection string
       string conn = GetConnectionString();

      // set up SQL statement
      string rs = connectionString;

      // retrieve data into DataSet
      OleDbDataAdapter da = new OleDbDataAdapter(rs, conn);
      DataSet ds = new DataSet();
      da.Fill(ds);

      // return data table
      return ds.Tables[0];
}
```

2. Set the ItemsSource and DisplayMemberPath properties for the MultiSelect control.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
```

```
      mselect.ItemsSource = GetDataSource("Select * from Employees
").AsDataView();
      mselect.BindingInfo.DisplayMemberPath = "FirstName";
}
```

Bind MultiSelect to Object Collection

You can bind MultiSelect to a list of complex data objects which can have multiple properties. To bind the control to a list of data objects, follow these steps:

1. Create a class named Customer using the following code.

C#

```
public class Customer
    {
        string name;
        string customerID;
        string mobile;
        string email;
        public Customer(string _name, string _custId, string _mobile, string
_email)
        {
            this.name = _name;
            this.customerID = _custId;
            this.mobile = _mobile;
            this.email = _email;
        }

        public string Name
        {
            get
            {
                return name;
            }
        }
        public string CustomerID
        {
            get
            {
                return customerID;
            }
        }
        public string Mobile
        {
            get
            {
                return mobile;
            }
        }
        public string Email
        {
```

```
            get
            {
                return email;
            }
        }
    }
```

2. Add objects of Customer class to a BindingList and set C1MultiSelect's ItemsSource and DisplayMemberPath properties at Window_Loaded event.

C#

```
public BindingList<Customer> customers;
private void Window_Loaded(object sender, RoutedEventArgs e)
{
     customers = new BindingList<Customer>();
     customers.Add(new Customer("John", "C001", "8888832141",
"john@gmail.com"));
     customers.Add(new Customer("Alex", "C002", "8888832142", "@gmail.com"));
     customers.Add(new Customer("Shawn", "C003", "8888832143",
"shawn@gmail.com"));
     customers.Add(new Customer("Sam", "C004", "8888832144", "sam@gmail.com"));
     customers.Add(new Customer("Neo", "C005", "8888832145", "neo@gmail.com"));
     customers.Add(new Customer("Paul", "C006", "8888832146",
"paul@gmail.com"));
     this.mselect.DataContext = customers;

}
```

MultiSelect in Unbound Mode

In unbound mode, you can populate the control with data by generating its content. For generating content, you need to add items either at design time, or at run time.

The following code populates C1MultiSelect control with customer names when the application loads at run time.

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    mselect.Items.Insert(1, "John");
    mselect.Items.Add("Alex");
    mselect.Items.Add("Sean");
}
```

Features

This section comprises all the features available in MultiSelect control.

Appearance
: Learn how to customize the appearance of the control elements through code.

Auto Suggest Mode
: Learn how to filter list items based on the text you enter.

Display mode
: Learn how to change the display modes through code.

Multi-Selection mode
: Learn how to change the selection modes through code.

Selection
: Learn how to implement selection related features through code.

Tag Appearance
: Learn how to display tags in different ways through code.

Tag Wrap
: Learn how to wrap the header items through code.

Appearance

MultiSelect allows you to customize the appearance of all the individual elements of the control and manage its overall appearance.

- **Apply Styles to C1MultiSelect**
- **Apply Styles to Tags**
- **Apply Styles to CheckList Items**

Apply Styles to C1MultiSelect

The following image shows styles applied to C1MultiSelect.

To apply style to MultiSelect using System.Windows.Style class, use the following code.

In code

C#

```
Style s = new Style(typeof(C1MultiSelect));
s.Setters.Add(new Setter(ItemsControl.BackgroundProperty,
              new SolidColorBrush(Color.FromRgb(255, 235, 205))));
mselect.Style = s;
```

In XAML

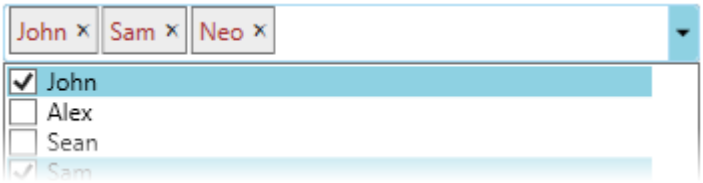
XAML

```
<c1:C1MultiSelect x:Name="mselect" ShowDropDownButton="true"
HorizontalAlignment="Left"
        Margin="9,35,0,0" VerticalAlignment="Top" Width="270" Style="{StaticResource
styleMultiSelect}">
    <c1:C1MultiSelect.Resources>
        <Style TargetType="c1:C1MultiSelect" >
              <Setter Property="Background" Value="#FFEBCD"/>
        </Style>
     </c1:C1MultiSelect.Resources>
</c1:C1MultiSelect>
```

Back to Top

Apply Styles to Tags

The following image shows styles applied to the tags in MultiSelect.



To customize the appearance of tags of C1MultiSelect, use the following code. Tags styles in C1MultiSelect can be accessed via the TagStyle property.

In code

C#

```
Style ts = new Style(typeof(C1Tag));
ts.Setters.Add(new Setter(ItemsControl.ForegroundProperty,
               new SolidColorBrush(Color.FromRgb(165, 42, 42))));
mselect.TagStyle = ts;
```

In XAML

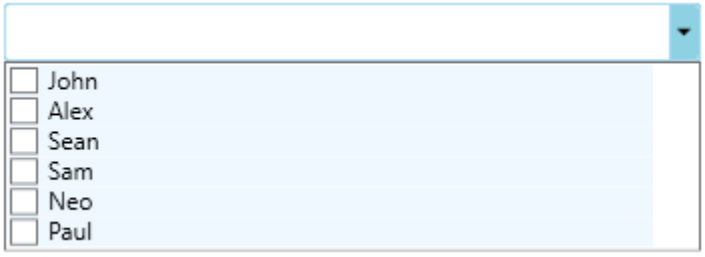
XAML

```
<Window.Resources>
     <Style x:Key="StyleForTag" TargetType="c1:C1Tag">
          <Setter Property="Foreground" Value="#A52A2A" />
     </Style>
</Window.Resources>
<Grid>
    <c1:C1MultiSelect x:Name="mselect" ShowDropDownButton="true"
HorizontalAlignment="Left"
        Margin="9,35,0,0" VerticalAlignment="Top" Width="270"
TagStyle="{StaticResource StyleForTag}" />
</Grid>
```

Back to Top

Apply Styles to CheckList Items

The following image shows styles applied to the CheckList items in MultiSelect.



To customize the appearance of checklist items in C1MultiSelect, use the following code. Styles an be applied to the Checklist items in C1MultiSelect using the ItemContainerStyle property.

In code

C#

```
Style cs = new Style(typeof(C1CheckListItem));
cs.Setters.Add(new Setter(ItemsControl.BackgroundProperty,
               new SolidColorBrush(Color.FromRgb(240, 248, 255))));
mselect.ItemContainerStyle = cs;
```

In XAML

XAML

```
<Window.Resources>
     <Style x:Key="StyleForItems">
          <Setter Property="Foreground" Value="#F0F8FF" />
     </Style>
</Window.Resources>
<Grid>
    <c1:C1MultiSelect x:Name="mselect" ShowDropDownButton="true"
HorizontalAlignment="Left"
        Margin="9,35,0,0" VerticalAlignment="Top" Width="270"
TagStyle="{StaticResource StyleForTag}" />
</Grid>
```

Back to Top

Auto Suggest Mode

MultiSelect provides suggestions as soon as you enter text in the control header. It allows you to filter and configure whether filtering starts from the beginning of the word or after entering the entire word. The list of items gets filtered based on the text you enter in the control header. This functionality can be achieved using AutoSuggestMode property of C1MultiSelect class that allows you to set the condition of filtering through SuggestMode enumeration. This enumeration defines whether to filter the items containing input text or filter the items starting with input text.

The following GIF shows the suggested items which contains the text entered in the header.

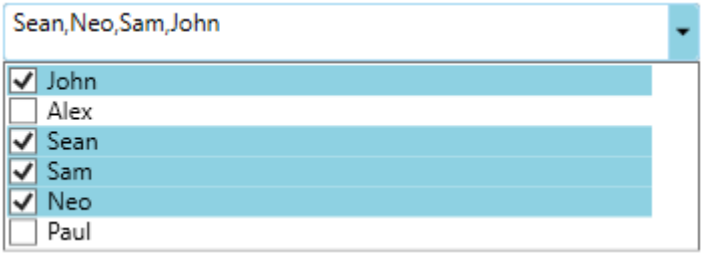
To show a list of suggested items which contains the text entered in the header, use the following code:

```
C#
mselect.AutoSuggestMode = SuggestMode.Contains;
```

Display Mode

MultiSelect allows you to choose the appearance of items in header between text or tags. In text mode, the text is displayed as strings separated by a separator character. On the other hand, in tag mode, the tags appear like labels separated by a space. You can choose how the items appear in the TagEditor element using DisplayMode property that accepts the values from DisplayMode enumeration.

The following image shows the selected items displayed as text in the header.



To display the selected items as text in the header, use the following code:

```
C#
mselect.DisplayMode = C1.WPF.Input.DisplayMode.Text;
```

By default, the MultiSelect control uses comma (,) as separator character to separate the items in header. However, you can specify a separator character of your choice using Separator property.

```
C#
mselect.Separator = "/";
```

Multi-Selection Mode

MultiSelect provides SelectionMode property to determine whether you can select one or more than one item from the header. This property lets you choose between the following selection modes through SelectionMode enumeration:

- **Single**: Allows you to select only one item at a time.
- **Multiple**: Allows you to select multiple items without holding down a modifier key.
- **Extended**: Allows you to select multiple consecutive items while holding down the corresponding modifier key.

To set the selection mode to single, use the following code:

```
C#
mselect.SelectionMode = C1.WPF.Input.SelectionMode.Single;
```

Selection

MultiSelect provides you with an option to select/deselect all items in the list with single selection. To enable this option, you need to set the ShowSelectAll property to true. On setting this property to true, the Select All check box appears on top of the list of items. When you deselect the Select All check box, all items in the list are deselected.

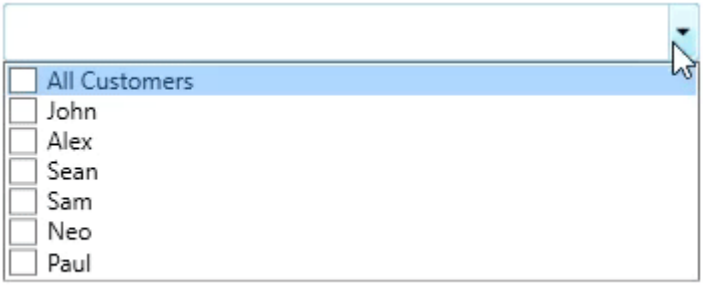
The following GIF shows the Select All and Unselect All check box appear in the MultiSelect control.

To enables Select All option in the MultiSelect control, use the following code:

```
C#
mselect.ShowSelectAll = true;
```

In addition, MultiSelect also allows you to change the caption of Select All option using SelectAllCaption property and caption of Unselect All option using UnSelectAllCaption property of C1MultiSelect class.

The following GIF shows the changed caption of Select All and Unselect All options in the MultiSelect control.



To change the caption, use the following code:

```
C#
mselect.SelectAllCaption = "All Customers";
mselect.UnSelectAllCaption = "Uncheck All";
```

Tag Appearance

MultiSelect has customizable header which, by default, displays all the items selected from the list. However, you can control this default behavior to adjust the maximum number of selected items to display in the header using MaxHeaderItems property of the C1MultiSelect class. For example, when you set the value of **MaxHeaderItems** property to 3, as soon as you select the fourth item, the header starts showing the total count of selected items as "4 items selected".

The following GIF shows how the items in the header appear on setting the MaxHeaderItems property.

To display maximum three items in the header, use the following code:

C#

```
mselect.MaxHeaderItems = 3;
```

Tag Wrap

MultiSelect provides a way to wrap the items within the control header. This can be achieved using TagWrapping property of C1TagEditor class.

The following GIF shows how the items in the header get wrapped on setting the TagWrap property.

To wrap items in header, use the following code:

C#

```
C1TagEditor te = new C1TagEditor();
te.TagWrapping = true;
```