Drag and drop

Article • 07/12/2024

Drag and drop is an intuitive way to transfer data within an application or between applications on the Windows desktop. Drag and drop lets the user transfer data between applications or within an application using a standard gesture (press-hold-and-pan with the finger or press-and-pan with a mouse or a stylus).

> **Important APIs**: CanDrag property, AllowDrop property

The drag source, which is the application or area where the drag gesture is triggered, provides the data to be transferred by filling a data package object that can contain standard data formats, including text, RTF, HTML, bitmaps, storage items or custom data formats. The source also indicates the kind of operations it supports: copy, move or link. When the pointer is released, drop occurs. The drop target, which is the application or area underneath the pointer, processes the data package and returns the type of operation it performed.

During drag and drop, the drag UI provides a visual indication of the type of drag-and-drop operation that's taking place. This visual feedback is initially provided by the source but can be changed by the targets as the pointer moves over them.

Modern drag and drop is available on all devices that support UWP. It allows data transfer between or within any kind of application, including Classic Windows apps, although this article focuses on the XAML API for modern drag and drop. Once implemented, drag and drop works seamlessly in all directions, including app-to-app, app-to-desktop, and desktop-to app.

Here's an overview of what you need to do to enable drag and drop in your app:

1. Enable dragging on an element by setting its **CanDrag** property to true.
2. Build the data package. The system handles images and text automatically, but for other content, you'll need to handle the **DragStarting** and **DropCompleted** events and use them to construct your own data package.
3. Enable dropping by setting the **AllowDrop** property to **true** on all the elements that can receive dropped content.
4. Handle the **DragOver** event to let the system know what type of drag operations the element can receive.
5. Process the **Drop** event to receive the dropped content.

Enable dragging

To enable dragging on an element, set its **CanDrag** property to **true**. This make the element—and the elements it contains, in the case of collections like ListView—draggable.

Be specific about what's draggable. Users won't want to drag everything in your app, only certain items, such as images or text.

Here's how to set **CanDrag**.

XML

```
<Image x:Name="Image" CanDrag="True" Margin="10,292,10,0" Height="338"></Im-
age>
```

You don't need to do any other work to allow dragging, unless you want to customize the UI (which is covered later in this article). Dropping requires a few more steps.

Construct a data package

In most cases, the system will construct a data package for you. The system automatically handles:

- Images
- Text

For other content, you'll need to handle the **DragStarting** and **DropCompleted** events and use them to construct your own DataPackage.

Enable dropping

The following markup shows how the **AllowDrop** property can be used to specify that an area of the app is a valid drop target for a dragged item (the specified area must not have a null background, it must be able to receive pointer input, and the item cannot be dropped anywhere other than the specified area).

> ⓘ **Note**
>
> Typically, a UI element has a null background by default. If you want users to be able to drop an item anywhere within your app, the app background cannot be null (set `Background="Transparent"` if the background should not be visible).

XML

```
<Grid AllowDrop="True" DragOver="Grid_DragOver" Drop="Grid_Drop"
      Background="LightBlue" Margin="10,10,10,353">
    <TextBlock>Drop anywhere in the blue area</TextBlock>
</Grid>
```

Handle the DragOver event

The **DragOver** event fires when a user has dragged an item over your app, but not yet dropped it. In this handler, you need to specify what kind of operations your app supports by using the **AcceptedOperation** property. Copy is the most common.

C#

```
private void Grid_DragOver(object sender, DragEventArgs e)
{
    e.AcceptedOperation = DataPackageOperation.Copy;
}
```

Process the Drop event

The **Drop** event occurs when the user releases items in a valid drop area. Process them by using the **DataView** property.

For simplicity in the example below, we'll assume the user dropped a single photo and access it directly. In reality, users can drop multiple items of varying formats simultaneously. Your app should handle this possibility by checking what types of files were dropped and how many there are, and process each accordingly. You should also consider notifying the user if they're trying to do something your app doesn't support.

C#

```
private async void Grid_Drop(object sender, DragEventArgs e)
{
    if (e.DataView.Contains(StandardDataFormats.StorageItems))
    {
        var items = await e.DataView.GetStorageItemsAsync();
        if (items.Count > 0)
        {
            var storageFile = items[0] as StorageFile;
            var bitmapImage = new BitmapImage();
            bitmapImage.SetSource(await storageFile.OpenAsync(FileAccessMod-
e.Read));
            // Set the image on the main page to the dropped image
            Image.Source = bitmapImage;
        }
    }
}
```

Customize the UI

The system provides a default UI for dragging and dropping. However, you can also choose to customize various parts of the UI by setting custom captions and glyphs, or by opting not to

show a UI at all. To customize the UI, use the DragEventArgs.DragUIOverride property.

```
C#
```

```
private void Grid_DragOverCustomized(object sender, DragEventArgs e)
{
    e.AcceptedOperation = DataPackageOperation.Copy;
    e.DragUIOverride.Caption = "Custom text here"; // Sets custom UI text
    // Sets a custom glyph
    e.DragUIOverride.SetContentFromBitmapImage(
        new BitmapImage(
            new Uri("ms-appx:///Assets/CustomImage.png", UriKind.RelativeOr-
Absolute)));
    e.DragUIOverride.IsCaptionVisible = true; // Sets if the caption is vis-
ible
    e.DragUIOverride.IsContentVisible = true; // Sets if the dragged content
is visible
    e.DragUIOverride.IsGlyphVisible = true; // Sets if the glyph is visibile
}
```

Open a context menu on an item you can drag with touch

When using touch, dragging a UIElement and opening its context menu share similar touch gestures; each begins with a press and hold. Here's how the system disambiguates between the two actions for elements in your app that support both:

- If a user presses and holds an item and begins dragging it within 500 milliseconds, the item is dragged and the context menu is not shown.
- If the user presses and holds but does not drag within 500 milliseconds, the context menu is opened.
- After the context menu is open, if the user tries to drag the item (without lifting their finger), the context menu is dismissed and the drag will start.

Designate an item in a ListView or GridView as a folder

You can specify a ListViewItem or GridViewItem as a folder. This is particularly useful for TreeView and File Explorer scenarios. To do so, explicitly set the AllowDrop property to **True** on that item.

The system will automatically show the appropriate animations for dropping into a folder versus a non-folder item. Your app code must continue to handle the Drop event on the folder item (as well as on the non-folder item) in order to update the data source and add the dropped item to the target folder.

Enable drag and drop reordering within ListViews

ListViews support drag-based reordering out of the box, using an API very similar to the **CanDrop** API described in this article. At minimum, you add the **AllowDrop** and **CanReorderItems** properties.

See **ListViewBase.CanReorderItems** for more information.

Implementing custom drag and drop

The UIElement class does most of the work of implementing drag-and-drop for you. But if you want, you can implement your own version by using the APIs below.

Expand table

Functionality	WinAppSDK API Microsoft.UI.Input.DragDrop namespace	UWP API Windows.Applicationmodel.DataTransfer.DragDrop.Core namespace
DragPrimitive	DragOperation	CoreDragOperation
Create a data package	DataPackage	same
Hand off drag to the shell	DragOperation.StartAsync	CoreDragOperation.StartAsync
Receive drop from the shell	DragDropManager.TargetRequested ICoreDropOperationTarget	CoreDragDropManager.TargetRequested ICoreDropOperationTarget

See also

- App-to-app communication
- AllowDrop
- CanDrag
- DragOver
- AcceptedOperation
- DataView
- DragUIOverride
- Drop
- IsDragSource

- DragStarting
- DropCompleted