https://bhrnjica.net/2014/04/28/drag-and-drop-item-outside-wpf-application/

In WFP applications drag and drop functionality is provided by subscribing several events. First of all you need LeftMouseButtonClick and MouseMove events in order to start Drag and Drop.

This is accomplished by the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | private void Window\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)  {      this.\_startPoint = e.GetPosition(null);  }    private void Window\_MouseMove(object sender, MouseEventArgs e)  {      //drag is heppen      //Prepare for Drag and Drop      Point mpos = e.GetPosition(null);      Vector diff = this.\_startPoint - mpos;        if (e.LeftButton == MouseButtonState.Pressed &&          (Math.Abs(diff.X) > SystemParameters.MinimumHorizontalDragDistance ||          Math.Abs(diff.Y) > SystemParameters.MinimumVerticalDragDistance))      {            //hooking on Mouse Up          InterceptMouse.m\_hookID = InterceptMouse.SetHook(InterceptMouse.m\_proc);            //ataching the event for hadling drop          this.QueryContinueDrag += queryhandler;          //begin drag and drop          DataObject dataObj = new DataObject(this.text1);          DragDrop.DoDragDrop(this.text1, dataObj, DragDropEffects.Move);        }  } |

As we can see from the code above, first with LeftMouseButtonClick event, we captured the starting point, then by MouseMove event, calculated distance between starting point and the current mouse position. If the distance is big enough we start DragDrop.

When the Drag starts it is necessary to prepare Drag functionality by providing the object to be dragged. In our case (code sample above) we are going to drag TextBox. In order to drag TextBox first we create DragData object by specifying TextBox object in the Constructor, and call DragDrop static method by passing objects we mentioned.  
Beside preparing data to be dragged, we need to subscribe to QueryContinueDrag event in order to track dragging status.

This is all we need to prepare Drag and Drop in our application. If we want to drag object out of WPF aplication, we have no enough information to accomplish drop. As soon as the mouse is outside the app, mousemove event is not firing any more. One of the solution of the problem could be capturing the mouse move position outside the WFP application, and when the left mouse button is up, start with dropping item functionality.

In order to track mouse position outside the WPF application we need to hook and subscribe to the messages of whole Windows OS and filter only we are interesting in. Great blog post about how to capture  mouse messages regardless of the  mouse position can be found [here](http://blogs.msdn.com/b/toub/archive/2006/05/03/589468.aspx).

Implementation of Low Level Mouse Hook in C# blog post can be modified quickly in order to adopt to our case.

First we need a property to indicate when the mouse is outside the application.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static bool IsMouseOutsideApp  {      get;      set;  } |

Then we need to modify HookCallback method so that when the Left mouse button is up, and set the property (IsMouseOutsideApp ) to true if the mouse outside the application.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | internal static IntPtr HookCallback( int nCode, IntPtr wParam, IntPtr lParam)  {      if (nCode >= 0 && MouseMessages.WM\_LBUTTONUP == (MouseMessages)wParam)      {          MSLLHOOKSTRUCT hookStruct = (MSLLHOOKSTRUCT)Marshal.PtrToStructure(lParam, typeof(MSLLHOOKSTRUCT));            //check if POint in main window          Point pt = new Point(hookStruct.pt.x, hookStruct.pt.y);          var ptw = App.Current.MainWindow.PointFromScreen(pt);          var w = App.Current.MainWindow.Width;          var h = App.Current.MainWindow.Height;          //if point is outside MainWindow          if (ptw.X < 0 || ptw.Y < 0 || ptw.X > w || ptw.Y > h)              IsMouseOutsideApp = true;          else              IsMouseOutsideApp = false;      }      return CallNextHookEx(m\_hookID, nCode, wParam, lParam);  } |

The Last thing we need to implement is DragSourceQueryContinueDrag event. The implementation is straightforward:  
1. check the keystate value None – this is the case when the mouse is released.  
2. unhook from the mouse intercepting messages  
3. show the message text based on the result of drag and dropping.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | /// <summary>  /// Continuosly tracking Dragging mouse position  /// </summary>  ///  ///  private void DragSourceQueryContinueDrag(object sender, QueryContinueDragEventArgs e)  {      //when keystate is non, draop is heppen      if (e.KeyStates == DragDropKeyStates.None)      {          //unsubscribe event          this.QueryContinueDrag -= queryhandler;          e.Handled = true;          //Unhooking on Mouse Up          InterceptMouse.UnhookWindowsHookEx(InterceptMouse.m\_hookID);            //notifiy user about drop result          Task.Run(              () =>;              {                  //Drop hepend outside Instantly app                  if (InterceptMouse.IsMouseOutsideApp)                      MessageBox.Show("Dragged outside app");                  else                      MessageBox.Show("Dragged inside app");              }              );      }  } |

Complete dem