HubView Controls

Guide

June 2019

Revision 1.0

Intel Confidential

**Legal Disclaimers and Copyrights**

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, Intel® Core™ vPro™ processor family are trademarks of Intel Corporation in the United States and/or other countries.

\*Other names and brands may be claimed as the property of others

Contents

[1 HubView Controls 6](#_Toc22291921)

[2 Code 7](#_Toc22291922)

[2.1 HubView Control : View 7](#_Toc22291923)

[2.1.1 DataContext 7](#_Toc22291924)

[2.2 HubView Control : ViewModel 9](#_Toc22291925)

[2.2.1 Setup For Messaging 10](#_Toc22291926)

[2.3 EventArgs 12](#_Toc22291927)

Code

[Code 1 – View XAML 7](#_Toc22291907)

[Code 2 – View XAML.cs (Using SingletonViewModelFactory) 7](#_Toc22291908)

[Code 3 – SingleviewModelFactgory Method 8](#_Toc22291909)

[Code 4 – DataContext Defined In XAML 9](#_Toc22291910)

[Code 5 – ViewModel for a HubView Control 9](#_Toc22291911)

[Code 6 – EventArgs for a HubView Control 12](#_Toc22291912)

[Code 7 – Model/EventArguments/EvenArguments.cs 12](#_Toc22291913)

Revision History

|  |  |  |
| --- | --- | --- |
| Revision | Description | Date |
| 1.0 | Initial release | October 2019 |

# HubView Controls

A HubView Control is a clickable UI object that can send a custom message to the Unite Core.

Each WPF HubView Control is comprised of:

* View
* ViewModel
* An EventArg uniquely named for each Control that also serves as a steering mechanism for the message transport.

# Code

## HubView Control : View

The View is a WPF Button with binding to the Click Handler (Code 1).

Code 1 – View XAML

|  |
| --- |
| <UserControl x:Class="UnitePlugin.View.AuthViewControlView"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:local="clr-namespace:UnitePlugin.View"  mc:Ignorable="d"  d:DesignHeight="40" d:DesignWidth="40">  <Grid>  <Button Command="{Binding ShowAuthViewButton\_ClickCommand}" CommandParameter="{Binding}" >  <TextBlock FontSize="6" TextWrapping="Wrap" TextAlignment="Center">Auth View</TextBlock>  </Button>  </Grid>  </UserControl> |

### DataContext

#### As A Singleton (Static)

The HubView Controls setup DataContext in the XAML.cs file (Code 2). All other code is located in the ViewModel file.

SingletonViewModelFactory<> is an optional method used to ensure that all instances of a HubView Control (AuthView, for example) uses the same instance of the ViewModel. When implemented as such, clicking on one HubView Control results in all of the same type responding in concert.

Code 2 – View XAML.cs (Using SingletonViewModelFactory)

|  |
| --- |
| using System.Windows.Controls;  using UnitePlugin.ViewModel.Controls;  using UnitePlugin.ViewModel.Factory;  namespace UnitePlugin.View.Controls  {  /// <summary>  /// Interaction logic for AuthViewControlView.xaml  /// </summary>  public partial class AuthViewControlView : UserControl  {  public AuthViewControlView()  {  InitializeComponent();  DataContext = SingletonViewModelFactory<AuthViewControlViewModel>.GetInstance;  }  }  } |

Code 3 – SingleviewModelFactgory Method

|  |
| --- |
| namespace UnitePlugin.ViewModel.Factory  {  public static class SingletonViewModelFactory<T> where T : new()  {  private static T \_instance;  public static T GetInstance  {  get  {  if(\_instance == null)  \_instance = Getobject();  return \_instance;  }  }  private static T Getobject()  {  return new T();  }  }  } |

#### Non-Singleton (Non-Static)

If SingletonViewModelFactory is not implemented, DataContext needs to be defined another way, preferably in the XAML file (Code 4).

In this case, each HubView Control of the same type that is clicked will not affect behavior of any other of the same type.

Code 4 – DataContext Defined In XAML

|  |
| --- |
| <UserControl x.Class….  -  -.  xmlns:viewModel="clr-namespace:UnitePlugin.ViewModel" >  <UserControl.DataContext>  <viewModel:AuthViewViewModel />  </UserControl.DataContext> |

## HubView Control : ViewModel

The ViewModel is used instead of Code-Behind that would typically reside in the XAML file (Code 5).

Code 5 – ViewModel for a HubView Control

|  |
| --- |
| using System;  using System.Windows.Input;  using Appccelerate.EventBroker;  using Appccelerate.EventBroker.Handlers;  using UnitePlugin.Model.Command;  using UnitePlugin.Model.EventArguments;  using UnitePlugin.Static;  using UnitePlugin.Utility;  namespace UnitePlugin.ViewModel.Controls  {  [Serializable]  public class AuthViewControlViewModel : HubViewModel  {  #region Fields  [field: NonSerialized]  private EventHandler<ShowAuthViewEventArgs> \_showAuthView;  private ICommand \_showAuthViewButton\_ClickCommand;  #endregion  #region Properties  public event EventHandler<ShowAuthViewEventArgs> ShowAuthView  {  add => \_showAuthView += value;  remove => \_showAuthView -= value;  }  public ICommand ShowAuthViewButton\_ClickCommand  {  get  {  return \_showAuthViewButton\_ClickCommand ?? (\_showAuthViewButton\_ClickCommand = new RelayCommand<AuthViewControlViewModel>(  x =>  {  ShowAuthViewButton\_SendMsgAndClick(this, new ShowAuthViewEventArgs  {  viewModel = x,  SenderControlIdentifier = ControlIdentifier,  HubViewType = UI.HubView.Type.AuthImage,  HubViewMethod = "Allocate",  IsOnAllDisplays = true,  });  }));  }  }  #endregion  public AuthViewControlViewModel()  {  MessagingEventBroker.GlobalEventBroker.Register(this);  \_showAuthView += UnitePluginConfig.HubViewManager.EventCommandInvoker;  }  #region Methods  public void ShowAuthViewButton\_SendMsgAndClick(object sender, ShowAuthViewEventArgs eventArgs)  {  UnitePluginConfig.RuntimeContext.MessageSender.TrySendMessage(  new CommandWraper<ShowAuthViewEventArgs>(eventArgs).ToMessage());  ShowAuthViewButton\_Click(sender, eventArgs);  }  [EventSubscription("topic://" + "ShowAuthViewEventArgs", typeof(OnUserInterface))]  public void ShowAuthViewButton\_Click(object sender, ShowAuthViewEventArgs eventArgs)  {  \_showAuthView?.Invoke(this, eventArgs);  }  #endregion  }  } |

### Setup For Messaging

#### EventArgs Payload

In Code 5 – ViewModel for a Hubview Control, the EventArg ShowAuthViewEventArgs is populated with the essential fields to enable messaging:

* ViewModel
* SenderControlIdentifier,
* HubViewType (AuthImage – Factory will use enum to consume, in this case, related AuthImage.cs.
* HubViewMethod (“Allocate” – Tells HubView to display the control.)
* IsOnAllDisplays (true – Control will be visible on all displays.)

#### Message Transport

The UnitePluginConfig.RuntimeContext.MessageSender.TrySenbMessaage() method sends the payload directory to the Unite Core with the purpose to message other Plugins.

## EventArgs

Messaging uses an EventArgs that is uniquely named: AuthViewEventArgs, for example (Code 4). It must reside in the Model/EventArguments folder and its name added to the Model/EventArguments/EventArgumentTypes.cs file (Code 4).

Code 6 – EventArgs for a HubView Control

|  |
| --- |
| using System;  using UnitePlugin.ViewModel.Controls;  namespace UnitePlugin.Model.EventArguments  {  ///Create in UnitePlugin.Model.EventArguments  ///and add to enum EventArgumentTypes  [Serializable]  public class ShowAuthViewEventArgs : HubViewEventArgs  {  public AuthViewControlViewModel viewModel { get; set; }  }  } |

Code 7 – Model/EventArguments/EvenArguments.cs

|  |
| --- |
| namespace UnitePlugin.Model.EventArguments  {  /// <summary>  /// ENUM that holds all Event types for subscription based messages  /// </summary>  public enum EventArgumentTypes  {  HubViewEventArgs = 300,  ShowAuthViewEventArgs,  ShowPartialBackgroundViewEventArgs,  ShowPresentationViewEventArgs,  ShowRibbonViewEventArgs,  ShowStatusImageEventArgs,  // ADD HubView Control EventArg Type here. //  }  } |