Intel Unite®

Unified Communication

Reference Design for

ViewManager

Guide

August 2019

Revision 1.0

Intel Confidential

Revision History

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

**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 Terminology 4](#_Toc17464133)

[2 UC Requirements 5](#_Toc17464134)

[2.1 Load and Allocate 5](#_Toc17464135)

[2.1.1 PluginModuleHandler Load() method 5](#_Toc17464136)

[2.2 IHubViewManager 7](#_Toc17464137)

[2.3 HubViewManager 8](#_Toc17464137)

[2.3.1 HubViewManager implementation details 12](#_Toc17464138)

Figures

**No table of figures entries found.**

Code

[Code 1 Create HubView object 5](#_Toc17464139)

[Code 2 Allocate HubView 6](#_Toc17464140)

[Code 3 Create IHubViewManager 7](#_Toc17464141)

[Code 4 Create HubViewManager 8](#_Toc17464141)

[Code 5 HubViewManager Implemented 12](#_Toc17464142)

# Terminology

Table 1. Terminology

| Term | Description |
| --- | --- |
| UI | User Interface |
| WPF | Windows\* Presentation Foundation |
| Hub | Intel® Core™ vPro™ processor family-based PC running the Intel Unite Hub application |
| Client | Device used to connect to the hub |
| Server | Device that manages access and configuration of Hub and Client Devices |
| UC | Unified Communications |
| UC User | Unified Communication User |

# UC Requirements

This UC Reference Design targets the following feature:

* As a user, I want an easier way to manage views in the plugin, so users have a better experience during development.

## Load and Allocate

To successfully add and start a new plug-in, we need to be aware of a few things before we integrate with Intel Unite.

Minimum calls to be implemented by a plug-in:

* Calling Load() – creates and prepares a view for Allocate()
* Calling Allocate() – tells Unite to connect the specified view
* Calling Deallocate() – tells Unite to disconnect the view and cleanup internal resources \*\*NOTE\*\* this is not done for the QuickAccessControl object.

Requirements:

* Implementation of the HubView Factory Strategy

### PluginModuleHandler Load() method

For a plug-in to do anything, we must first Load the plugin and create a view, with the following code.

This is added to the PluginModuleHandlers.cs Load method

Code Create HubView object

|  |
| --- |
| public override void Load()  {  var hubview = new HubView().ExecuteCreation(HubView.Type.QuickAccessIcon,  RuntimeContext, CreateContract,  new PhysicalDisplay{Id = new Guid { },IsPrimary = true,},  CurrentUiDispatcher, null);  } |

It is important to stress that Creating a view is not enough to get a plug-in functioning.

Calling Allocate() is now required – Allocate() tells Intel Unites core to display the given view on the desired display.

Code Allocate HubView

|  |
| --- |
| public override void Load()  {  hubview = new HubView().ExecuteCreation(HubView.Type.QuickAccessIcon,  RuntimeContext, CreateContract,  new PhysicalDisplay{Id = new Guid { },IsPrimary = true,},  CurrentUiDispatcher, null);    hubview.Allocate();  } |

## IHubViewManager

A helper class can also be created to make this process easier. The interface for this class is detailed below.

To create the IHubViewManager interface,

1. Open Solution Explorer and navigate to the Interfaces folder
2. Right click and Add NewItem -> C# Class and name it IHubViewManager.cs
3. Paste the code from Code 3 IHubViewManager example
4. Save

Code 3 Create IHubViewManager

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Collections.ObjectModel;  using System.Windows;  using System.Windows.Threading;  using Intel.Unite.Common.Context.Hub;  using Intel.Unite.Common.Display;  using Intel.Unite.Common.Module.Common;  using UnitePlugin.Model.EventArguments;  using UnitePlugin.UI;  namespace UnitePlugin.Interfaces  {  public interface IHubViewManager  {  ReadOnlyCollection<IHubView> HubViews { get; }  void LoadHubView(HubView.Type hubViewType, IHubModuleRuntimeContext runtimeContext, Func<FrameworkElement, MarshalNativeHandleContract> createContract, PhysicalDisplay display, Dispatcher currentUiDispatcher);  void Allocate(UI.HubView.Type type);  bool Show(HubView.Type type);  void DeAllocate(UI.HubView.Type type);  void Allocate(Guid viewGuid);  bool Show(Guid viewGuid);  void DeAllocate(Guid viewGuid);  IHubView GetSpecificView(Guid viewGuid);  IHubView GetSpecificView(UI.HubView.Type type, PhysicalDisplay display);  List<IHubView> GetSpecificViews(UI.HubView.Type type);  void EventCommandInvoker(object sender, HubViewEventArgs e);  void LoadandAllocateForAllDisplays(UI.HubView.Type hubViewType);  void LoadForAllDisplays(UI.HubView.Type hubViewType);  }  } |

## HubViewManager

A helper class can also be created to make this process easier like our HubViewManger class detailed below.

To use the HubViewManager class,

1. Open Solution Explorer and navigate to the Utility folder
2. Right click and Add NewItem -> C# Class and name it HubViewManager.cs
3. Paste the code from Code 3 HubViewManager example
4. Save

Code 4 Create HubViewManager

|  |
| --- |
| using Intel.Unite.Common.Context.Hub;  using Intel.Unite.Common.Display;  using Intel.Unite.Common.Module.Common;  using System;  using System.Collections.Generic;  using System.Windows;  using System.Linq;  using System.Windows.Threading;  using System.Collections.ObjectModel;  using UnitePlugin.Interfaces;  using UnitePlugin.UI;  using UnitePlugin.Model.EventArguments;  namespace UnitePlugin.Utility  {  [Serializable]  public class HubViewManager : IHubViewManager  {  private readonly List<IHubView> \_hubViews = new List<IHubView>();  [field: NonSerialized]  private readonly IHubModuleRuntimeContext \_runtimeContext;  [field: NonSerialized]  private readonly Dispatcher \_currentUiDispatcher;  [field: NonSerialized]  private readonly Func<FrameworkElement, MarshalNativeHandleContract> \_createContract;  public ReadOnlyCollection<IHubView> HubViews => \_hubViews.AsReadOnly();  public HubViewManager(IHubModuleRuntimeContext runtimeContext, Dispatcher currentUiDispatcher, Func<FrameworkElement, MarshalNativeHandleContract> createContract)  {  lock (this)  {  \_runtimeContext = runtimeContext;  \_currentUiDispatcher = currentUiDispatcher;  \_createContract = createContract;  }  }  public void LoadHubView(HubView.Type hubViewType, IHubModuleRuntimeContext runtimeContext, Func<FrameworkElement, MarshalNativeHandleContract> createContract, PhysicalDisplay display, Dispatcher currentUiDispatcher)  {  lock (this)  {  \_hubViews.Add(new UI.HubView().ExecuteCreation(hubViewType, runtimeContext, createContract, display, currentUiDispatcher, EventCommandInvoker));  }  }  public void Allocate(UI.HubView.Type type)  {  GetSpecificViews(type).ForEach(hubView => hubView.Allocate());  }  // ReSharper disable once UnusedMember.Global  public bool Show(HubView.Type type)  {  var result = true;  GetSpecificViews(type).ForEach(hubView => result = hubView.Show() && result);  return result;  }  public void DeAllocate(UI.HubView.Type type)  {  GetSpecificViews(type).ForEach(hubView => hubView.DeAllocate());  }  // ReSharper disable once UnusedMember.Global  public void Allocate(Guid viewGuid)  {  GetSpecificView(viewGuid).Allocate();  }  public bool Show(Guid viewGuid)  {  return GetSpecificView(viewGuid).Show();  }  // ReSharper disable once UnusedMember.Global  public void DeAllocate(Guid viewGuid)  {  GetSpecificView(viewGuid).DeAllocate();  }  // ReSharper disable once MemberCanBePrivate.Global  public IHubView GetSpecificView(Guid viewGuid)  {  return \_hubViews.FirstOrDefault(hubView => hubView.ViewGuid == viewGuid);  }  // ReSharper disable once MemberCanBePrivate.Global  public IHubView GetSpecificView(UI.HubView.Type type, PhysicalDisplay display)  {  return \_hubViews.FirstOrDefault(hubView =>  hubView.GetType() == Type.GetType("UnitePlugin.UI." + Enum.GetName(typeof(UI.HubView.Type), type)) &&  hubView.HubAllocationInfo.PhysicalDisplay == display  );  }  // ReSharper disable once MemberCanBePrivate.Global  public List<IHubView> GetSpecificViews(UI.HubView.Type type)  {  return \_hubViews.Where(hubView => hubView.GetType() == Type.GetType("UnitePlugin.UI." + Enum.GetName(typeof(UI.HubView.Type), type))).ToList();  }  public void EventCommandInvoker(object sender, HubViewEventArgs e)  {  if (e.IsOnAllDisplays) EventCommandInvokerAllDisplay(sender, e);  else EventCommandInvokerSingleDisplay(sender, e);  }  // ReSharper disable once MemberCanBePrivate.Global  public void EventCommandInvokerSingleDisplay(object sender, HubViewEventArgs e)  {  var senderView = GetSpecificView(e.SenderControlIdentifier);  var targetView = GetViewAndCreateIfNull(e.HubViewType, senderView.HubAllocationInfo.PhysicalDisplay);  typeof(IHubView).GetMethod(e.HubViewMethod)?.Invoke(targetView, new object[] { });  }  // ReSharper disable once MemberCanBePrivate.Global  public void EventCommandInvokerAllDisplay(object sender, HubViewEventArgs e)  {  List<IHubView> targetViews = new List<IHubView>();  \_runtimeContext.DisplayManager.AvailableDisplays.ToList().ForEach(display => targetViews.Add(GetViewAndCreateIfNull(e.HubViewType, display)));  targetViews.ForEach(targetView => typeof(IHubView).GetMethod(e.HubViewMethod)?.Invoke(targetView, new object[] { }));  }  private IHubView GetViewAndCreateIfNull(UI.HubView.Type hubViewType, PhysicalDisplay display)  {  return GetSpecificView(hubViewType, display) ?? CreateView(hubViewType, display);  }  private IHubView CreateView(HubView.Type hubViewType, PhysicalDisplay display)  {  LoadHubView(hubViewType, \_runtimeContext, \_createContract, display, \_currentUiDispatcher);  return GetSpecificView(hubViewType, display);  }  public void LoadandAllocateForAllDisplays(UI.HubView.Type hubViewType)  {  LoadForAllDisplays(hubViewType);  Allocate(hubViewType);  }  // ReSharper disable once MemberCanBePrivate.Global  public void LoadForAllDisplays(UI.HubView.Type hubViewType)  {  \_runtimeContext.DisplayManager.AvailableDisplays.ToList().ForEach(display =>  LoadHubView(hubViewType, \_runtimeContext, \_createContract, display, \_currentUiDispatcher)  );  }  }  } |

### HubViewManager implementation details

Using the HubViewManager class, we will change the file PluginModuleHandler.cs Load method now uses the HubViewManager instance to Load and Allocate on the displays.

Code 5 HubViewManager Implemented

|  |
| --- |
| using Intel.Unite.Common.Command;  using Intel.Unite.Common.Context;  using Intel.Unite.Common.Core;  using Intel.Unite.Common.Manifest;  using Intel.Unite.Common.Module.Common;  using Intel.Unite.Common.Module.Feature.Hub;  using System;  using System.Linq;  using System.Windows.Threading;  using UnitePlugin.Constants;  using UnitePlugin.Utility;  using UnitePlugin.ViewFactory;  namespace UnitePlugin  {  [Serializable]  public class PluginModuleHandler : HubFeatureModuleBase  {  private HubViewManager \_hubViewManager;  public PluginModuleHandler() : base()  { }  public PluginModuleHandler(IModuleRuntimeContext runtimeContext) : base(runtimeContext)  { }  public override string HtmlUrlOrContent => throw new NotImplementedException();  public override ModuleManifest ModuleManifest => ModuleConstants.ModuleManifest;  public override ModuleInfo ModuleInfo => ModuleConstants.ModuleInfo;  public override Dispatcher CurrentUiDispatcher { get; set; }  public override void HubConnected(HubInfo hubInfo)  {  throw new NotImplementedException();  }  public override void HubDisconnected(HubInfo hubInfo)  {  throw new NotImplementedException();  }  public override void HubInfoChanged(HubInfo hubInfo)  {  throw new NotImplementedException();  }  public override void IncomingMessage(Message message)  {  throw new NotImplementedException();  }  public override void Load()  {  \_HubViewManager = new HubViewManager(RuntimeContext, CurrentUiDispatcher, CreateContract);  \_HubViewManager.LoadandAllocateForAllDisplays(HubView.Type.QuickAccessIcon);  }  public override bool OkToSleepDisplay()  {  throw new NotImplementedException();  }  public override void SessionKeyChanged(KeyValuePair sessionKey)  {  throw new NotImplementedException();  }  public override void Unload()  {  throw new NotImplementedException();  }  public override void UserConnected(UserInfo userInfo)  {  throw new NotImplementedException();  }  public override void UserDisconnected(UserInfo userInfo)  {  throw new NotImplementedException();  }  public override void UserInfoChanged(UserInfo userInfo)  {  throw new NotImplementedException();  }  }  } |