Intel Unite®

Unified Communication

Reference Design for Messaging

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](#_Toc19702271)

[2 UC Requirements 5](#_Toc19702272)

[2.1 Display WCs 5](#_Toc19702273)

[2.1.1 MVVM files 5](#_Toc19702274)

[2.1.2 Implement Factory Strategy 8](#_Toc19702275)

[2.1.4 Allocate HubView.Type 9](#_Toc19702276)

Figures

[Figure 2 Add View 6](#_Toc19702277)

[Figure 3 View Design Example 7](#_Toc19702278)

Code

[Code 1 WCsDisplay View Model 6](#_Toc19702279)

[Code 2 WCsView xaml 6](#_Toc19702280)

[Code 3 WCsDisplay.cs 7](#_Toc19702281)

[Code 4 enum HubView.Type 8](#_Toc19702282)

[Code 5 WCsDisplayFactory 9](#_Toc19702283)

[Code 6 Allocate HubView Example 9](#_Toc19702284)

# 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 |
| Partial Background View | Area on the background layer where applications can reside |
| WC | Web Conference |
| UC | Unified Communications |
| UC User | Unified Communication User |

Plugins can communicate with Unite and other plugins via a unique Messaging system - this messaging system consists of 2 use cases – Processing Internal and External messages.

## Internal Messaging

### Requirements

We will be using a NuGet package to handle sending messages in the plugin under development called Appccelerate.EventBroker and Appccelerate.Fundamentals.

1. Open the NuGet package manager, enter Appccelerate, select these two items and install in all projects.

\*\* NOTE \*\* -

### MessgingEventBroker and EventArgumentTypes

The messaging system allows us to define the data which we wish to consume and to make this easier we will create a static class to facilitate this called MessagingEventBroker. The EventArgumentTypes houses an ENUM of all the event arguments used in your plugin allowing us to an easy method of accessing.

Code 1 MessagingEventBroker.cs

|  |
| --- |
| public static class MessagingEventBroker  {  public static IEventBroker GlobalEventBroker { get; } = new EventBroker();  public static void Process(Message message)  {  GetMessenger((EventArgumentTypes)message.DataType). InvokeSubscriptions (message);  }  private static IMessenger GetMessenger(EventArgumentTypes eventArgumentTypes)  {  Type genericType = typeof(Messenger<>);  Type[] genericTypeArgs = { System.Type.GetType("UnitePlugin.Model.EventArguments." + eventArgumentTypes) };  var argMessenger = genericType.MakeGenericType(genericTypeArgs);  if (argMessenger == null) throw new Exception("Valid Messenger not seen.");    return (IMessenger)Activator.CreateInstance( argMessenger);  }  } |

Inside the class constructor were message event(s) are expected to fire must register.

Code 1 Register example

|  |
| --- |
| public QuickAccessIconViewModel() : base()  {  MessagingEventBroker.GlobalEventBroker.Register(this);  } |

When sending messages to allocate, deallocate or show Unite specific views you should send a unique EventArg that will be called HubViewEventArgs.

Code 2 Argument Type ENUM example

|  |
| --- |
| public enum EventArgumentTypes  {  HubViewEventArgs = 300,  } |

HubViewEventArgs should contain the properties below.

Table 2 HubViewEventArgs.cs example

|  |
| --- |
| [Serializable]  public class HubViewEventArgs : EventArgs  {  public bool IsOnAllDisplays { get; set; }  public Guid SenderControlIdentifier { get; set; }  public UI.HubView.Type HubViewType { get; set; }  public String HubViewMethod { get; set; }  } |

### Appccelerate

Appccelerate allows sending events instead of creating layers of code creating/giving access to click methods or public methods. One important aspect to adding messaging is consuming the events when they are fired, and this happens by setting the Event Subscription attribute above the method.

It is important to ensure that the attribute and the eventArg seen below match

|  |
| --- |
| [EventSubscription("topic://" + " HubViewEventArgs", typeof(OnUserInterface))]  public void JoinWCEventReceived(object sender, HubViewEventArgs eventArgs)  {  ...  } |

### Interface for Messenger

The IMessenger interface has the InvokeSubscriptions method which initiates the Message being sent to its subscribers.

Code 6 IMessenger.cs example

|  |
| --- |
| public interface IMessenger  {  void InvokeSubscriptions(Message message);  } |

### Messenger

The Messenger class is responsible for sending the Message to the Event Broker.

Code 3 Messenger.cs

|  |
| --- |
| public class Messenger<T> : IMessenger  {  public void InvokeSubscriptions(Message message)  {  OnReceivedMessage(ConvertMessage<T>.Deserialize(message));  }  private void OnReceivedMessage(object o)  {  MessagingEventBroker.GlobalEventBroker.Fire("topic://" + typeof(T).Name, this, HandlerRestriction.None, this, (EventArgs)o);  }  } |

This completes the base setup for the Messaging system and any plug-in specific setup will be covered in those reference designs.

## External Messaging

External messages between Intel Unite Hubs and Clients and plugins require some thought and consideration when attempting to broadcast messages externally and when consuming messages sent from external sources.

Requirements:

* Implementation of HubFeatureModuleBase abstract class and its pre-requisites.

### IncomingMessage

At the heart of receiving external messages is the IncomingMessage method. This method provides an entry point to grab and process incoming messages. Plugins can listen for any number of messages as long at the C# style ENUM types have unique integer-based values. Typically setting the first value to be in multiples of 100 is a great starting point – allowing users the space needed to expand when necessary.

Code 4 IncomingMessage() example

|  |
| --- |
| public override void IncomingMessage(Message message)  {  if (message.TargetModuleId == ModuleInfo.Id)  {  if (typeof(EventArgumentTypes).IsEnumDefined(message.DataType))  {  MessagingEventBroker.Process(message);  }  }  } |

### BaseCommand Implementation

The BaseCommand class is a generic class to facilitate converting Commands to a Message for internal use.

Code 7 BaseCommand.cs example

|  |
| --- |
| public class BaseCommand<T> : CommandBase  {  public Guid ModuleId;  public T EventArgs;  public string EventArgsAsString = string.Empty;  public BaseCommand(ICommandSerializer commandSerializer, T t, Guid moduleId) : base(commandSerializer)  {  EventArgs = t;  EventArgsAsString = typeof(T).Name;  ModuleId = moduleId;  }  public override string PrintObject()  {  return GetType().Name;  }  public override Message ToMessage()  {  Message message = new Message  {  Priority = MessagePriority.High,  Data = CommandSerializer.Serialize(EventArgs),  DataType = (int)Enum.Parse(typeof(EventArgumentTypes), EventArgsAsString),  SourceModuleId = ModuleId,  TargetId = MessageConstants.TargetLocalhostId,  TargetModuleId = MessageConstants.TargetModuleBroadcastId  };  return message;  }  } |

A critically important bit of information is in the ToMessage method. When you are attempting to send messages to the hub and/or clients –the TargetId and TargetModuleId values are critical.

Below you can see how the settings of the Message affects the delivery and receiving of messages.

Table 3 Message Routing Tables

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Message Contents | | | |  | Message Received | | | | | | | | |
|  | Plugin 1 | | | Plugin 2 | | | Plugin 3 | | |
| Source Id | Source Module Id | Target Id | Target Module Id |  | Hub | Client A | Client B | Hub | Client A | Client B | Hub | Client A | Client B |
| Hub | Plugin 1 | Local | Plugin 1 |  | x |  |  |  |  |  |  |  |  |
| Hub | Plugin 1 | Local | Plugin 2 |  |  |  |  | x |  |  |  |  |  |
| Hub | Plugin 1 | Local | ModuleBroadcastId |  | x |  |  | x |  |  | x |  |  |
| Hub | Plugin 1 | Broadcast | Plugin 1 |  |  | x | x |  |  |  |  |  |  |
| Hub | Plugin 1 | Broadcast | Plugin 2 |  |  |  |  |  | x | x |  |  |  |
| Hub | Plugin 1 | Broadcast | ModuleBroadcastId |  |  | x | x |  | x | x |  | x | x |
| Client A | Plugin 1 | Local | Plugin 1 |  |  | x |  |  |  |  |  |  |  |
| Client A | Plugin 1 | Local | Plugin 2 |  |  |  |  |  | x |  |  |  |  |
| Client A | Plugin 1 | Local | ModuleBroadcastId |  |  | x |  |  | x |  |  | x |  |
| Client A | Plugin 1 | Broadcast | Plugin 1 |  | x |  |  |  |  |  |  |  |  |
| Client A | Plugin 1 | Broadcast | Plugin 2 |  |  |  |  | x |  |  |  |  |  |
| Client A | Plugin 1 | Broadcast | ModuleBroadcastId |  | x |  |  | x |  |  | x |  |  |