Intel Unite® Plugin Software Development Kit (SDK)

Guide

June 2019

Revision 2.0

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 Introduction 7](#_Toc22199475)

[Audience 7](#_Toc22199476)

[Terminology 7](#_Toc22199477)

[2 Intel Unite® Software Overview 8](#_Toc22199478)

[2.1 The Hub Application 8](#_Toc22199479)

[2.2 Icon View 10](#_Toc22199480)

[2.3 Background Layer 14](#_Toc22199481)

[2.4 Presentation Layer 16](#_Toc22199482)

[The Client(s) - Intel® Unite™ Client Application 19](#_Toc22199483)

[2.5 Plugin Icon Image 20](#_Toc22199484)

[2.6 Browser Emulator 21](#_Toc22199485)

[The (PIN) Intel Unite® Server 22](#_Toc22199486)

[2.7 Admin Portal 22](#_Toc22199487)

[3 Intel Unite® Software Plugin Module 24](#_Toc22199488)

[Requirements 24](#_Toc22199489)

[Runtime Context 24](#_Toc22199490)

[Events 25](#_Toc22199491)

[HTML Plugin 25](#_Toc22199492)

[3.1 HTML Events 26](#_Toc22199493)

[3.2 HTML Functions 26](#_Toc22199494)

[4 Code Examples 28](#_Toc22199495)

[4.1 Minimum Plugin Project Setup 28](#_Toc22199496)

[4.2 Implement the HubFeatureModuleBase Abstract Class 34](#_Toc22199497)

[4.3 Implement ModuleInfo 36](#_Toc22199498)

[4.4 Implement ModuleManifest 37](#_Toc22199499)

[4.5 Add Post-build Events to Project to Create a *.cab* File and Sign It 38](#_Toc22199500)

[5 Add Plugin UI Capability To Enable Load 40](#_Toc22199501)

[5.1 Create a QuickAccessAppIconView View 40](#_Toc22199502)

[5.2 Implement the *Load Method* 43](#_Toc22199503)

[5.3 Package and Sign *Loadable* Plugin 45](#_Toc22199504)

[6 Deploy a .cab File 46](#_Toc22199505)

[Deploying the Plugin 46](#_Toc22199506)

[7 Tests, Troubleshooting, Notes… 49](#_Toc22199507)

[7.1 Developer Notes 49](#_Toc22199508)

[7.2 Logs 49](#_Toc22199509)

[7.3 Commands 50](#_Toc22199510)

[7.4 Manually Create a .cab File and Deployment 52](#_Toc22199511)

[7.5 Manually Creating the Plugin .cab File 52](#_Toc22199512)

[7.6 Tests 52](#_Toc22199513)

[Troubleshooting 58](#_Toc22199514)

[7.7 Where is the Intel.Unite.Common.dll? 58](#_Toc22199515)

[8 Support 59](#_Toc22199516)

## Figures

[Figure 1 - Authentication Layer 9](#_Toc20733132)

[Figure 2 - Apps View 10](#_Toc20733133)

[Figure 3 - Icon View 11](#_Toc20733134)

[Figure 4 - Quick Access Control View 12](#_Toc20733135)

[Figure 5 - StatusAuth View 12](#_Toc20733136)

[Figure 6 - Status View 13](#_Toc20733137)

[Figure 7 - Auth View 13](#_Toc20733138)

[Figure 8 - Toast Message View 14](#_Toc20733139)

[Figure 9 - Background View 15](#_Toc20733140)

[Figure 10 - Partial Background View 16](#_Toc20733141)

[Figure 11 - Presentation Layer 17](#_Toc20733142)

[Figure 12 - Presentation View 18](#_Toc20733143)

[Figure 13 - Ribbon Control View 19](#_Toc20733144)

[Figure 14 - Client UI 20](#_Toc20733145)

[Figure 15 -Client Icon 21](#_Toc20733146)

[Figure 16 - Browser Emulator 22](#_Toc20733147)

[Figure 17 - Admin Portal Home 23](#_Toc20733148)

[Figure 18 - Project Setup : Step 1 29](#_Toc20733149)

[Figure 19 - Project Setup : Step 2 30](#_Toc20733150)

[Figure 20 - Project Setup : Step 3 31](#_Toc20733151)

[Figure 21 - Project Setup : Step 4 32](#_Toc20733152)

[Figure 22 - Project Setup : Step 5 33](#_Toc20733153)

[Figure 23 - Project Setup : Step 6 34](#_Toc20733154)

[Figure 24 - Implement Abstract Class : Steos 1-3 Completed 35](#_Toc20733155)

[Figure 25 - Build Event Editors 39](#_Toc20733156)

[Figure 26 - Device Management Drop Down 47](#_Toc20733157)

[Figure 27 - Hub Feature Tab 48](#_Toc20733158)

[Figure 28 - Hub Feature Edit 48](#_Toc20733159)

[Figure 29 - Log Level Enum Options 49](#_Toc20733160)

[Figure 30 – Commands Ladder Diagram 50](#_Toc20733161)

[Figure 31 - Class Diagram 51](#_Toc20733162)

## Code

[**Code 1 - Implenent ModuleInfo** 36](#_Toc22199517)

[**Code 2 - Implement ModuleManifest** 37](#_Toc22199518)

[**Code 3 - Post-Build Event Command Line** 39](#_Toc22199519)

[**Code 4 - QuickAccessAppIconView.xaml After All Edits** 41](#_Toc22199520)

[**Code 5 - ViewModel.cs Afgter All Edits** 41](#_Toc22199521)

[**Code 6 - Implement QuickAccessView(s)** 44](#_Toc22199522)

[**Code 7 - Manifest Generator Tests** 54](#_Toc22199523)

[**Code 8 - HubViewModel Tests** 55](#_Toc22199524)

[**Code 9 - Load Module Tests** 57](#_Toc22199525)

# Revision History

|  |  |  |
| --- | --- | --- |
| Revision | Description | Date |
| 1.0 | Initial release | September 2018 |
| 1.1 | Revised Software Overview and Software Plugin Module | May 2019 |
| 1.2 | Added Minimum Plugin Project Setup | May 2019 |
| 1.3 | Added Load Module | June 2019 |
| 1.4 | Modified all sections for clarity and error corrections. | September 2019 |
| 1.5 | Minor changes | January 2019 |

1. Introduction

Welcome to the *Intel Unite® Plugin Software Development Kit (SDK) Application Programming Interface (API) Guide*. Intel Unite® software offers APIs that enables implementation of additional business needs and functionality into Enterprise deployments of the Intel Unite® Solution.

## Audience

Software developers who want to develop a plugin for the Intel Unite® application.

For information about deploying Intel Unite®, refer to the *Intel Unite® Deployment Guide*.

## Terminology

Table 1. Terminology

|  |  |
| --- | --- |
| **Term** | **Description** |
| UI | User Interface |
| WPF | Windows\* Presentation Foundation |
| Hub | Any Intel® Core™ vPro™ processor family-based Device running the Intel Unite® Hub Application |
| Client(s) | Any Device which uses the Intel Unite® Client Application to connect to a Hub. There may be many Clients connected to the hub. |
| Server | Also known as the PIN Server, is a computer that manages access and configuration of the Hub and all Client Devices. A Server may reside On Premises (On Prem), or on the Cloud (not sharing a LAN with a Unite Hub). |
| Admin Portal | The website hosted by the Server used to configure the Intel Unite® solution |

1. Intel Unite® Software Overview

The Intel Unite® solution allows for fast, secure, wireless content-sharing and collaboration. Connect displays, employees, and mixed technology environments using the Intel Unite® solution enabling rich communication seamlessly, wherever they are.

For Developers, Intel Unite® solution exposes APIs through Intel.Unite.Common.dll which with minimal effort, any WPF program can be adapted to integrate (plugin) to the Intel Unite® application.

The three main Components of the Intel Unite® application are the Hub, the Client(s), and the (PIN) Server. Following is a drill down of the applications that run on the Hub and Client(s). The PIN Server is beyond the scope of this guide, and therefore no drill-down will be provided other than a brief overview in the Server Section of this document.

## Intel Unite® Software Hub Application

The Hub Application running the Intel Unite® solution which includes the UI and Hub Plugins through which peripheral communication is enabled. User interaction with the Hub Application is provided via three layers: Authentication, Background, and Presentation.

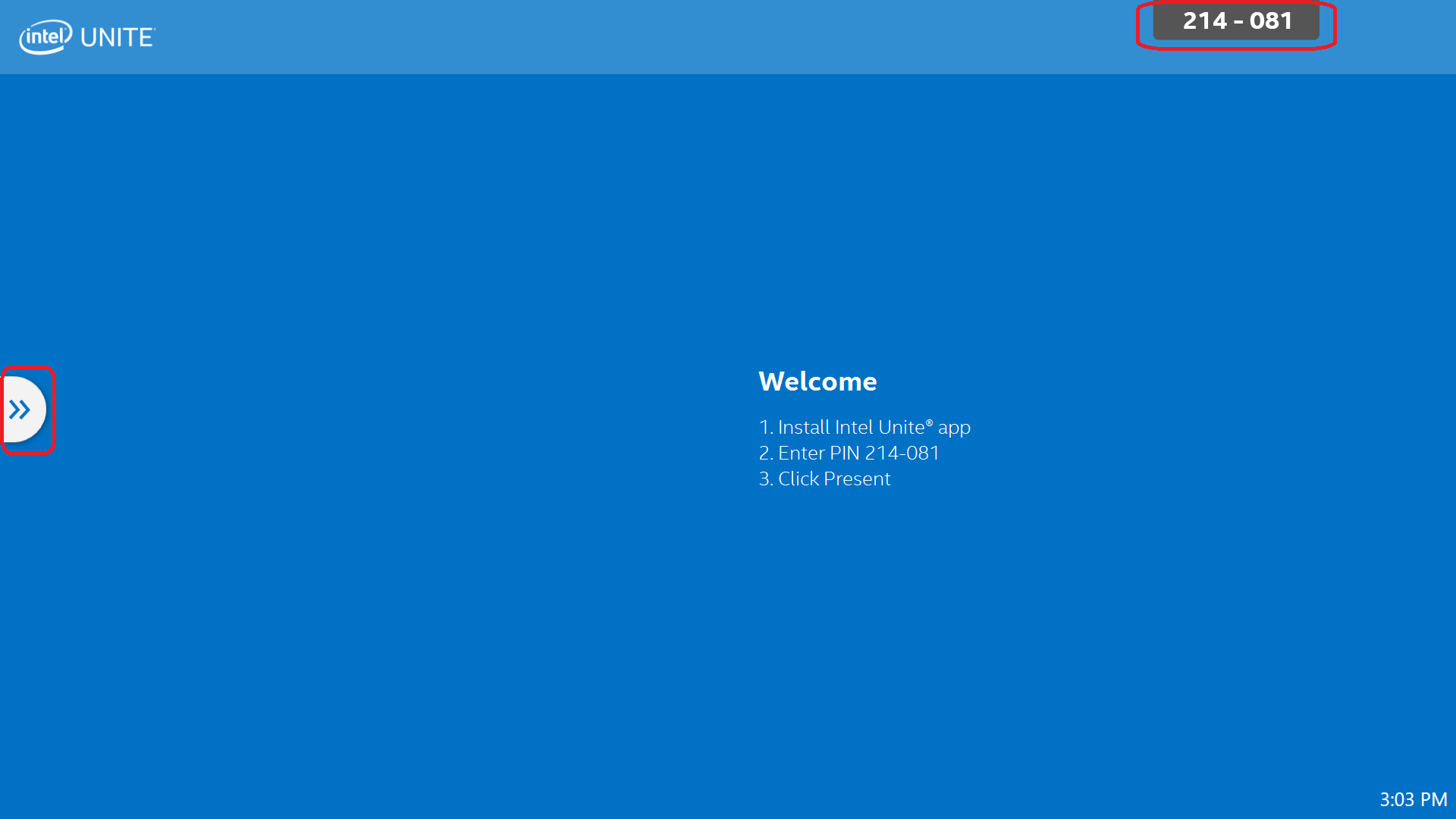
### Authentication Layer

The Authentication Layer contains three sub-views that overlay all other views including any that a Plugin may render. They are the:

* Quick View
* StatusAuth View
* Toast Message View

The Authentication Layer always shows the StatusAuth view which shows the current session PIN and other status information, and a chevron button which enables navigation to an and from the Quick View, and the Toast Message View which is momentarily visible when a message is available.

Figure 1 - Authentication Layer



### Quick View Layer

The Quick View Layer has two views, **App View** and **Participants View**, that provide easy access to important session information and are revealed by clicking: **>>**, or toggled to hide, by clicking: **<<**.

**App View**: shows all Apps (Plugins) available for the current session and automatically manages their display states, full screen or minimized, for example.

**Participants View**: displays an icon representation of each Participant attending the current session.

#### Participant View

The **Participant View** is not modified by Plugins directly, therefore it is out of scope of this document.

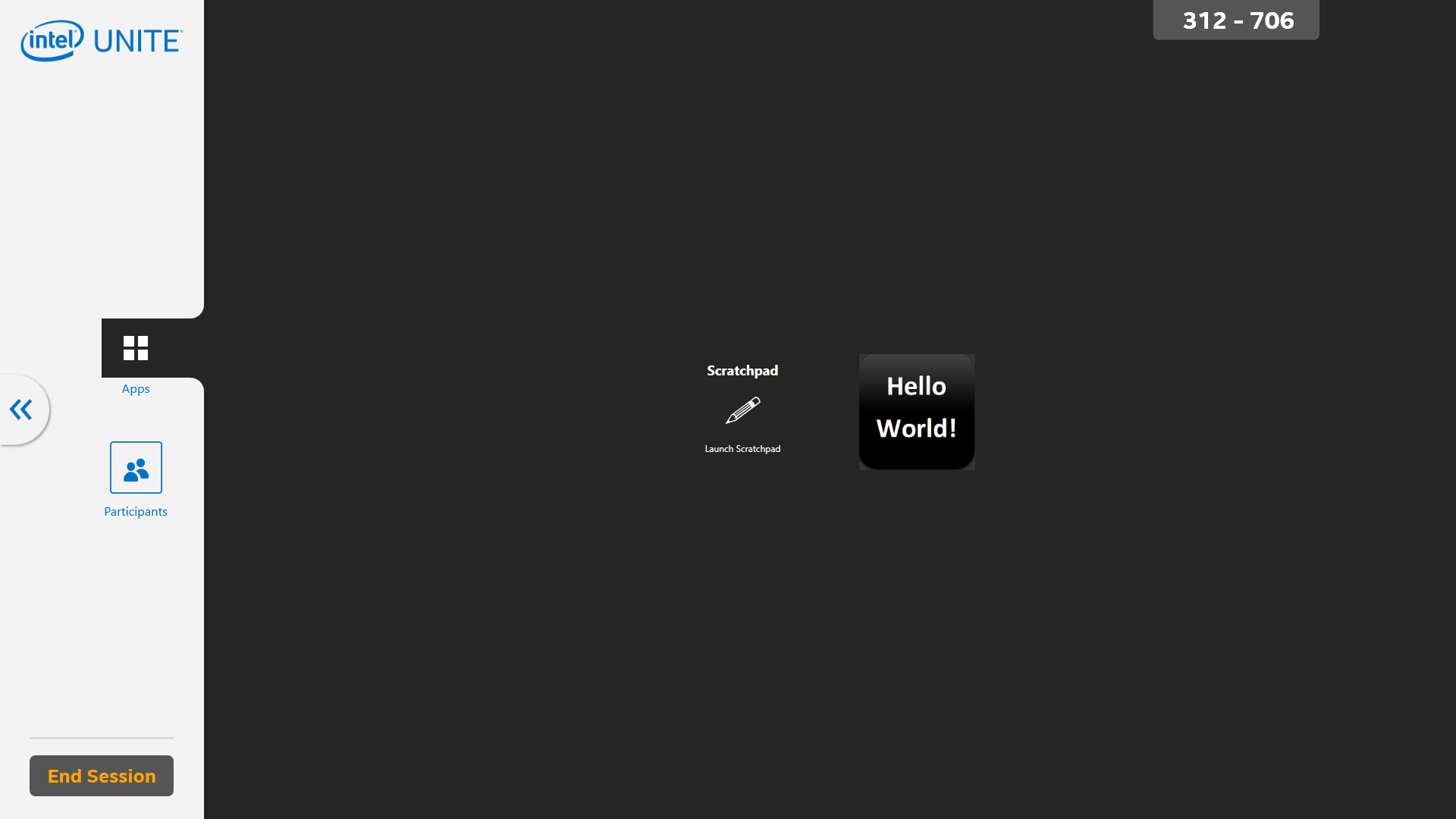
#### Apps View

The apps view shows a list of Plugin Icons and when selected a specific plugin Control View. There can only be one Quick Access Control View displayed at a time. When a Quick Access View is shown, the icons are minimized.

The App View Layer contains:

* Icon View
* Quick Access Control View

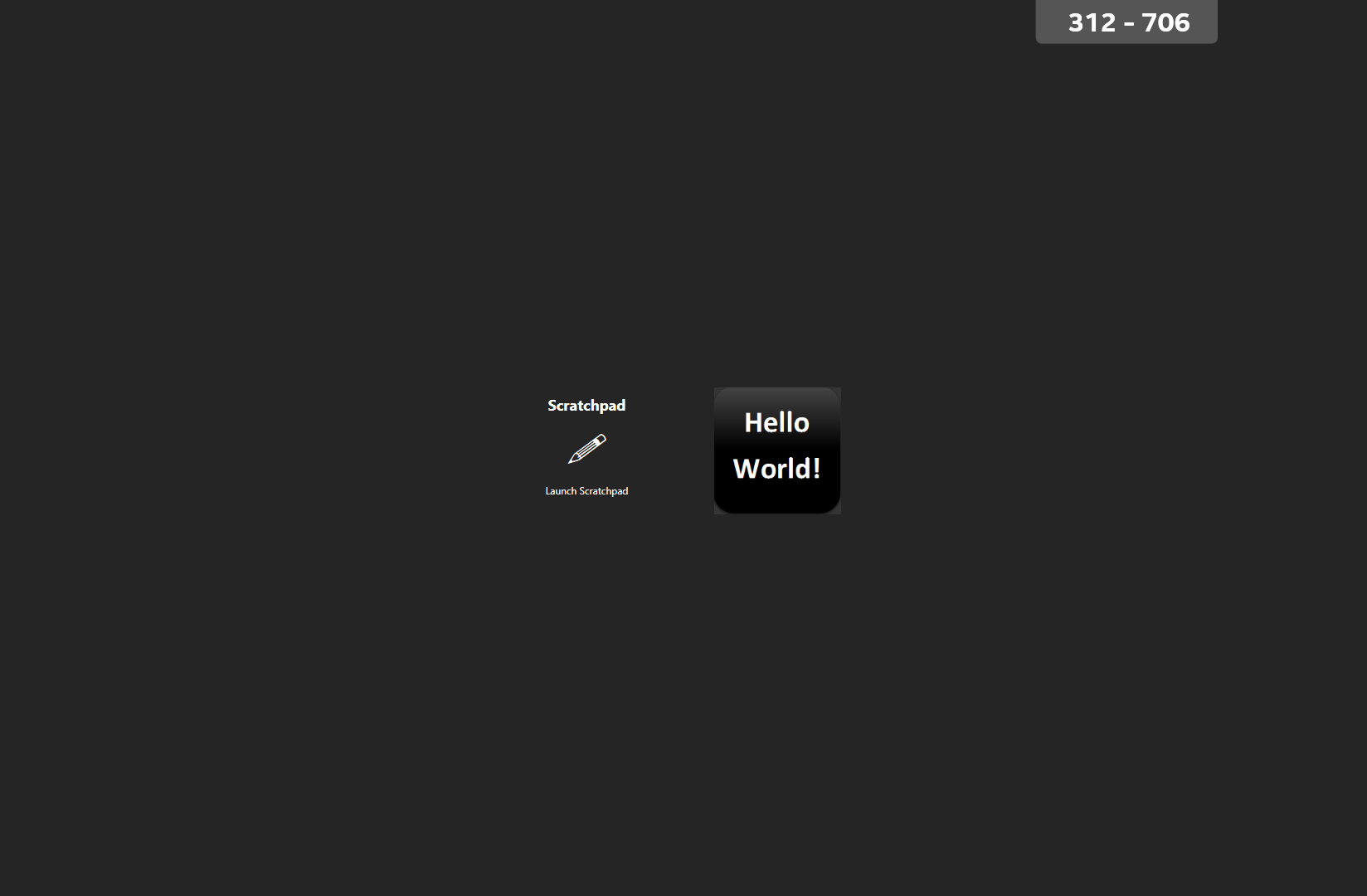
Figure 2 - Apps View



### Icon View

The **Icon View** displays the Plugin icon and is intended to manage the running of the Plugin with a Click event.

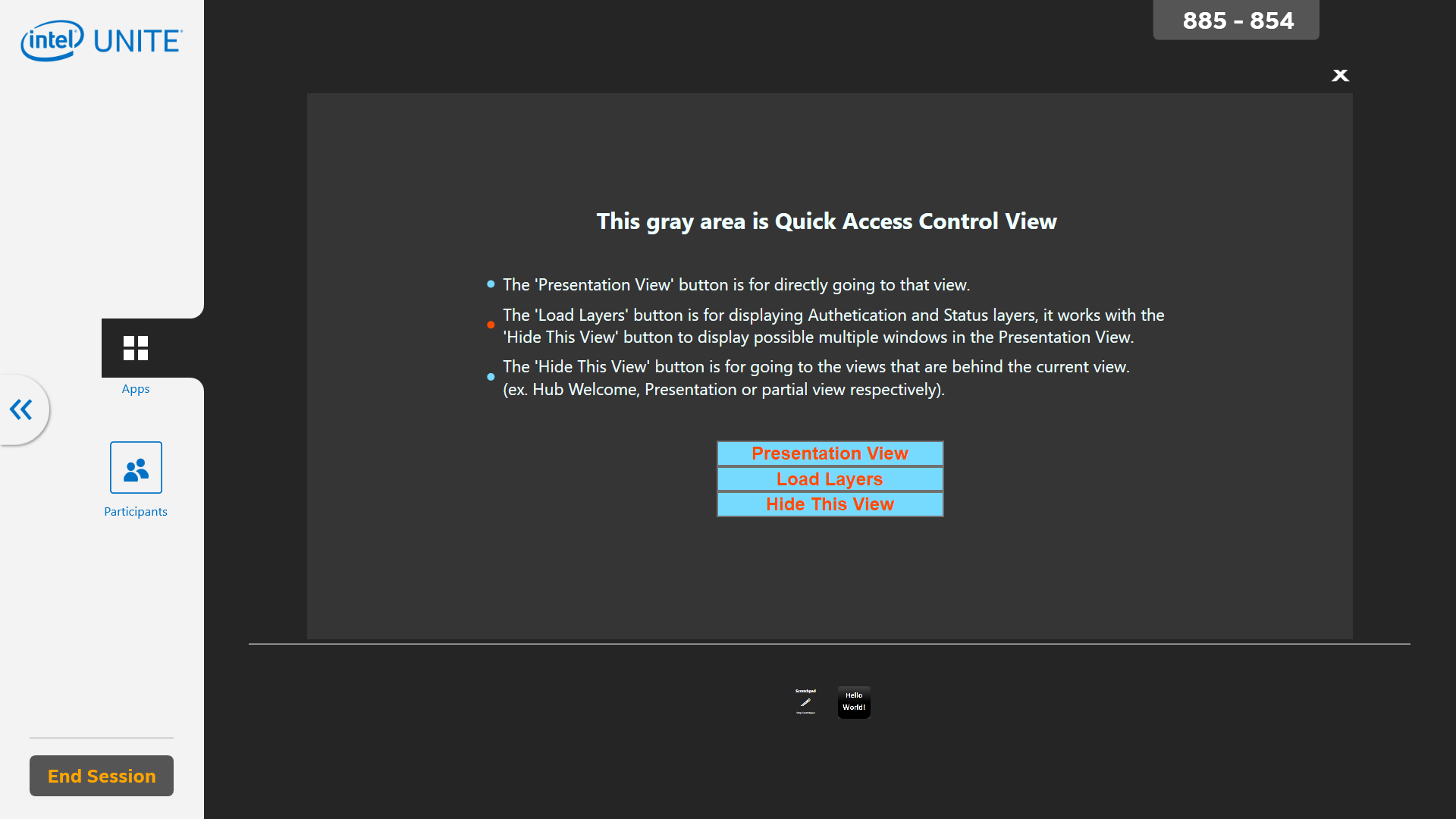
Figure 3 - Icon View



#### Quick Access Control View

This view contains Plugin Controls and information, and persists until another Quick Access Control View is displayed or the X is clicked.

Figure 4 - Quick Access Control View



### StatusAuth View

The StatusAuth View (Figure 5) holds the Status and Auth Views. The view defaults to the upper-right corner of the Authentication View and is always visible.

Figure 5 - StatusAuth View



#### Status View

A grid of Images. Plugins can allocate any number of images to the Status View.

Figure 6 - Status View



#### Auth View

The Auth View contains a single image followed by the PIN number. Only one Image can reside in the Auth View at any one time. If another Image is allocated it replaces the previous image.

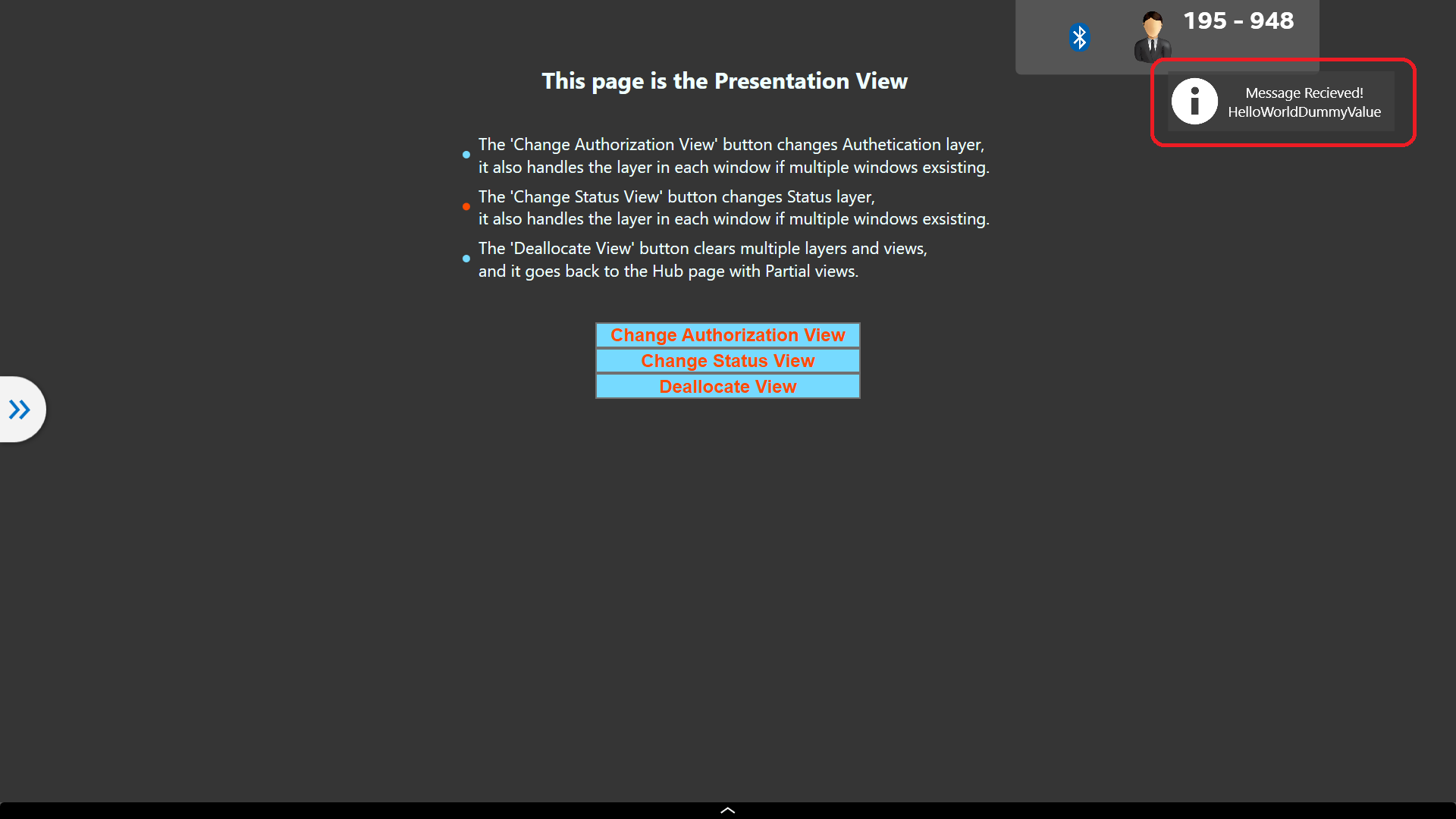
Figure 7 - Auth View



### Toast Message View

The Toast Message View (Figure 8) provides a way for Unite to communicate messages of interest. Messages will display for a brief period before auto-hiding.

Figure 8 - Toast Message View



### Background Layer

The Background Layer is displayed behind all other layers. The Background Layer can be hidden programmatically by a Plugin or by the Desktop button exposed when “Show Toggle Desktop Button” is set to True on the Admin Portal. If the Background Layer is hidden, the Background View and all Partial Background Views are also hidden.

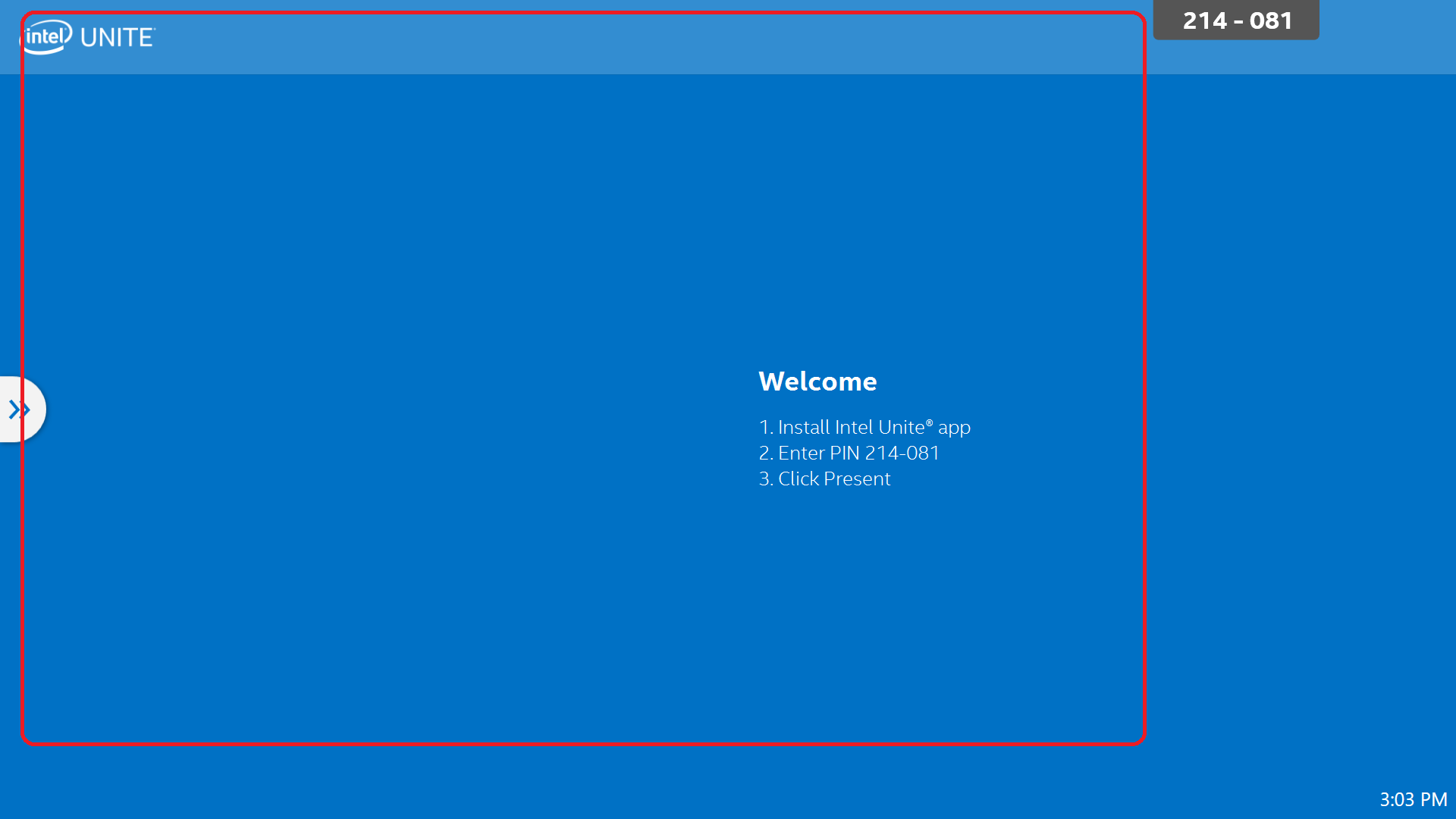
Views Contained on the Background layer:

* Background
* Partial Background

#### Background View

This View (Figure 9) is shown after Unite is loaded.

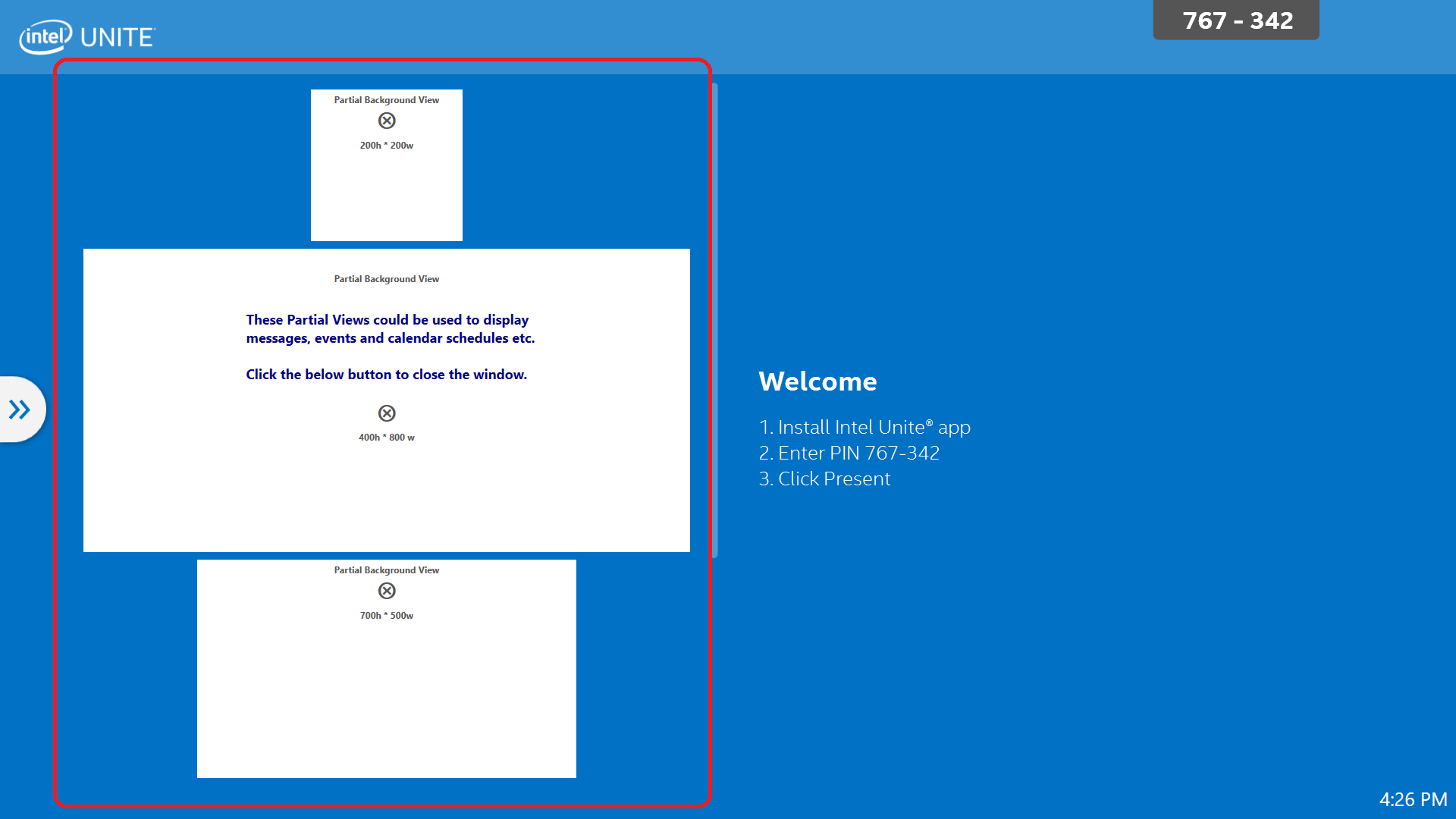
Figure 9 - Background View



#### Partial Background View

Partial Background View are views added on top of the Background View. There is no limit to the number of Partial Background Views that can be added. When added, each Partial Background View is wrapped in order of its allocation. If the background is hidden all partial background views are also hidden.

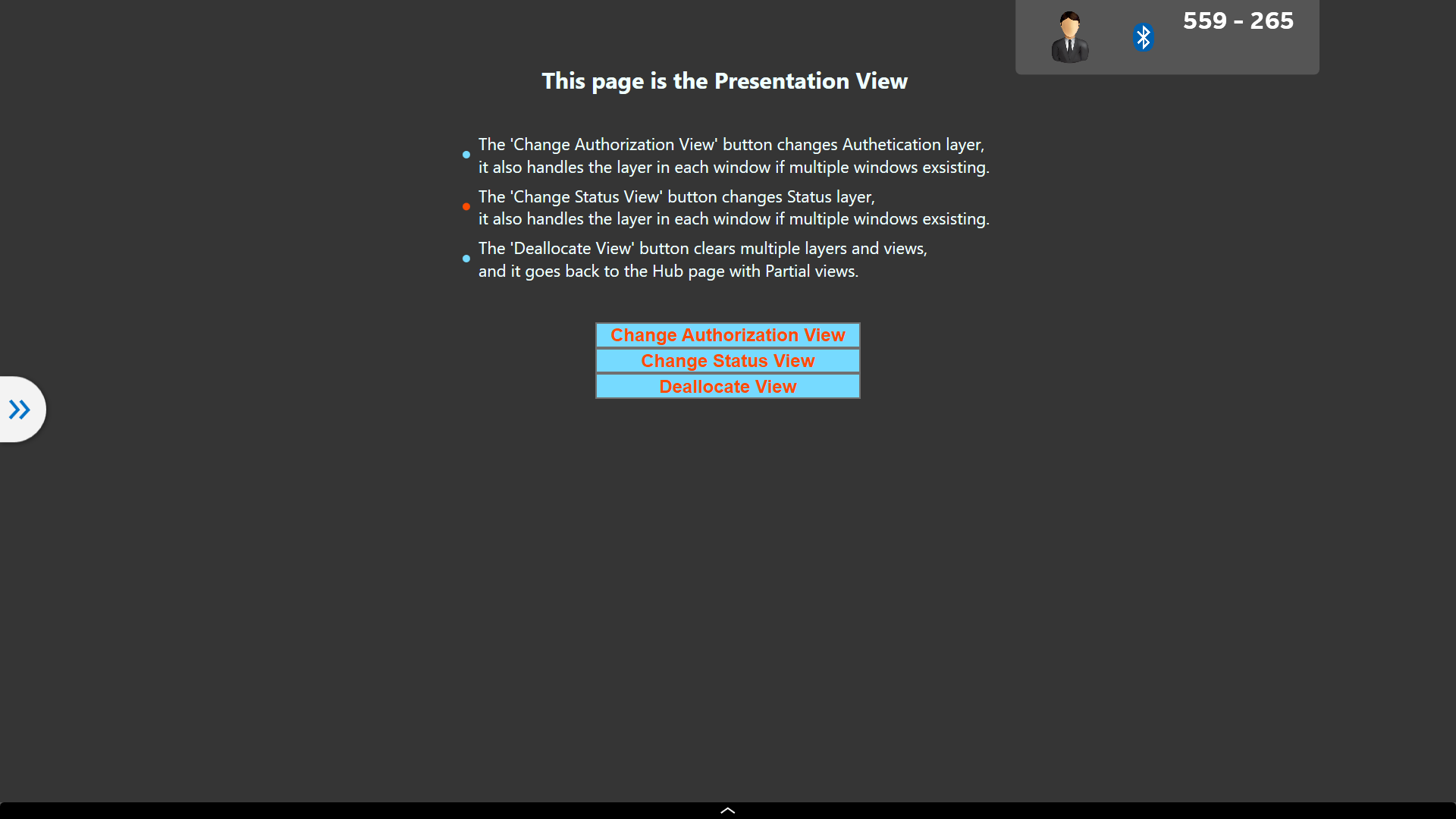
Figure 10 - Partial Background View



### Presentation Layer

The Presentation Layer is the layer presenting to the screen. The layer can contain up to four Presentation Views and their associated Ribbon View. This Layer is shared/shown to all Unite Clients. When a Client presents a new Presentation Layer is allocated.

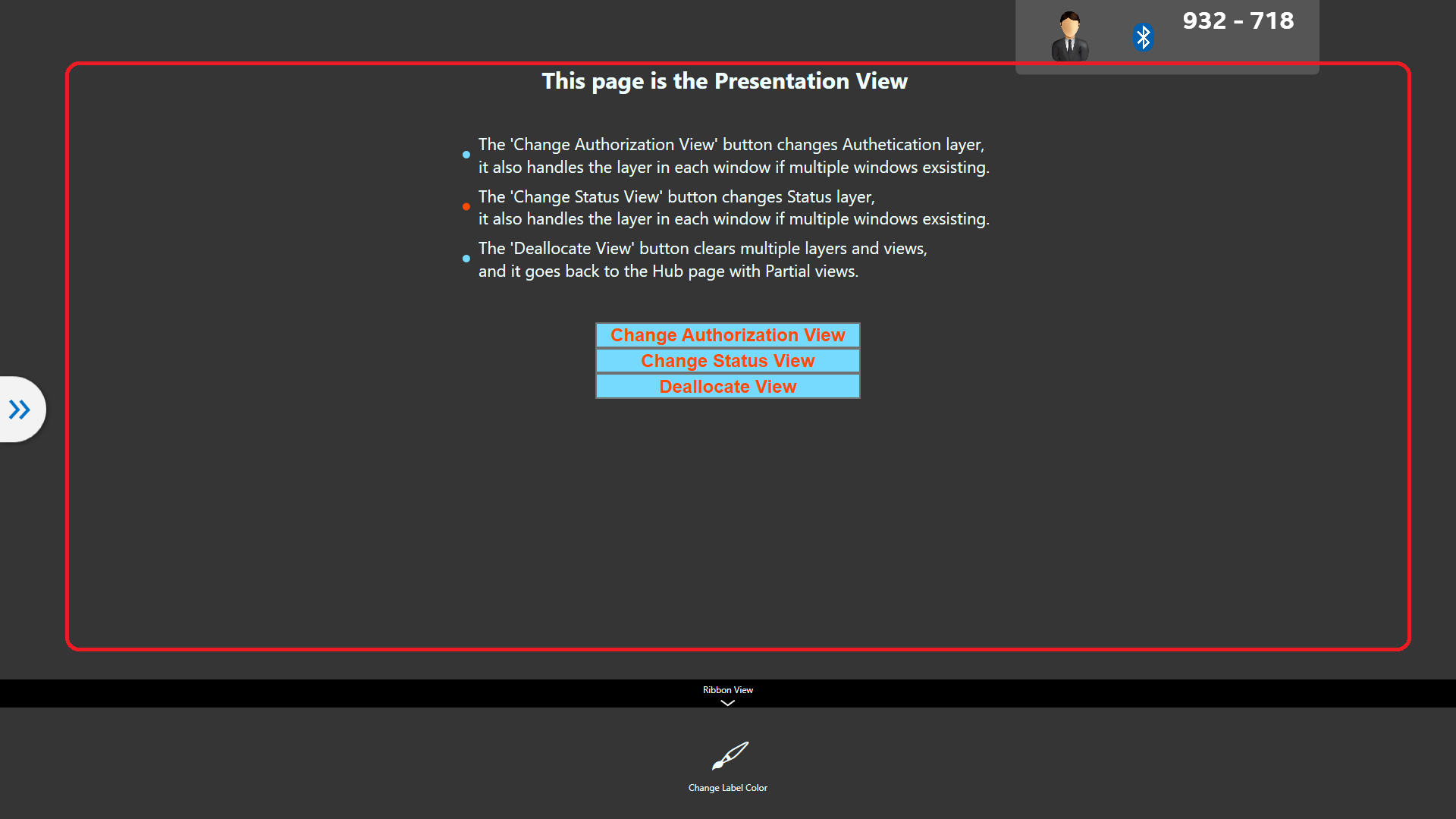
Figure 11 - Presentation Layer



#### Presentation View

The Presentation View is a User Control allocated above the Background layer as shown in Figure 12.

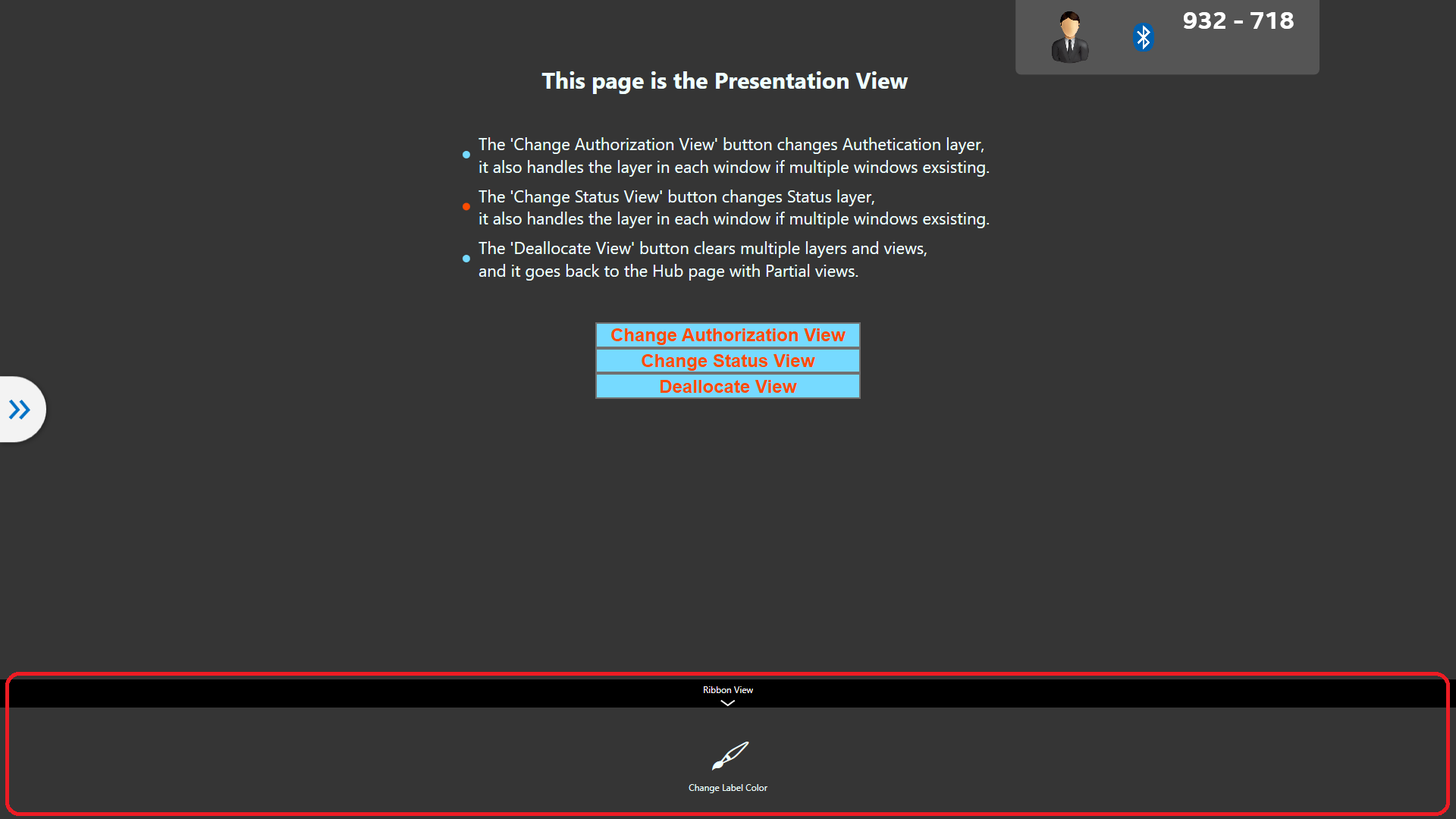
Figure 12 - Presentation View



#### Ribbon View

The ribbon view is an optional control associated with a Presentation View (Figure 13). A Ribbon View will be visible only when the associated Presentation View has focus.

Figure 13 - Ribbon Control View

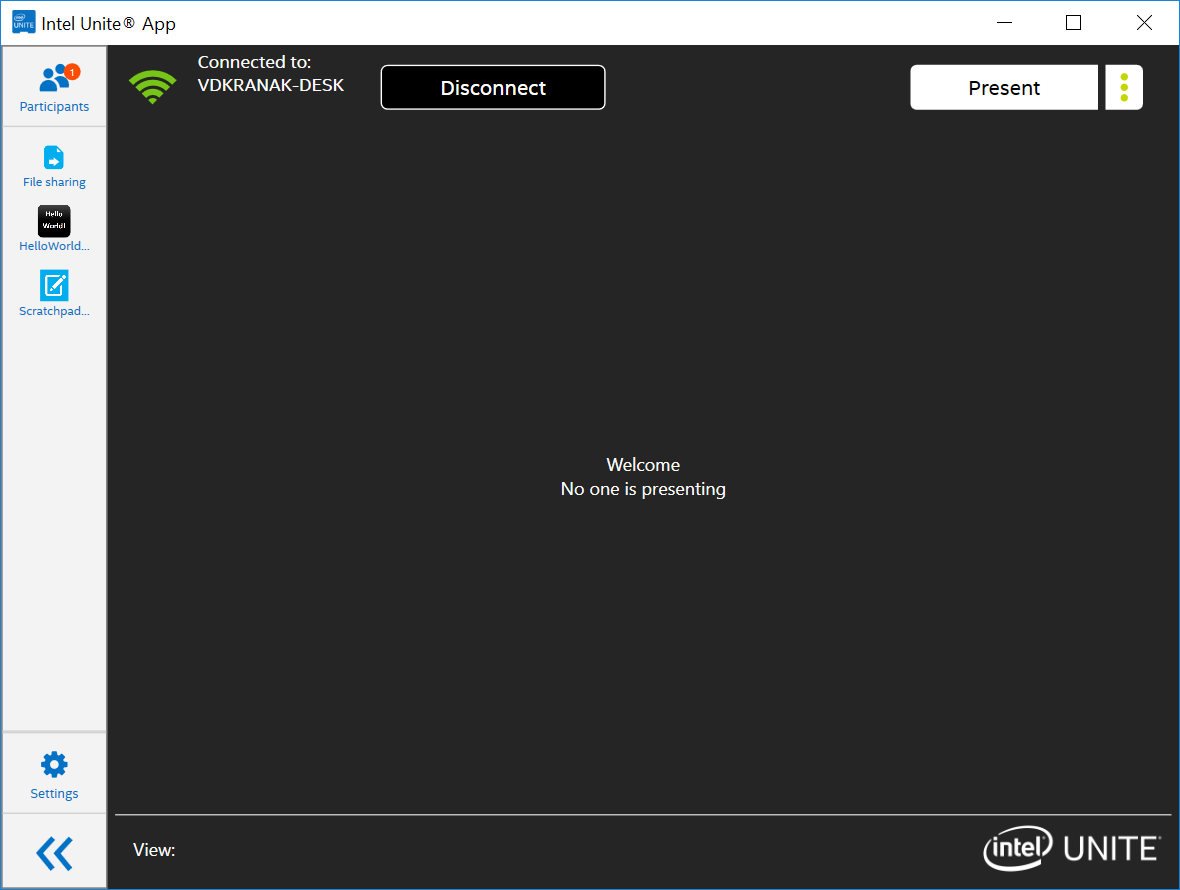


## Intel Unite® Software Client Application

The User Interface on the devices connecting to the hub. For the purpose of a plugin there are two main parts of the client UI:

* The Plugin Icon
* The Browser Emulator

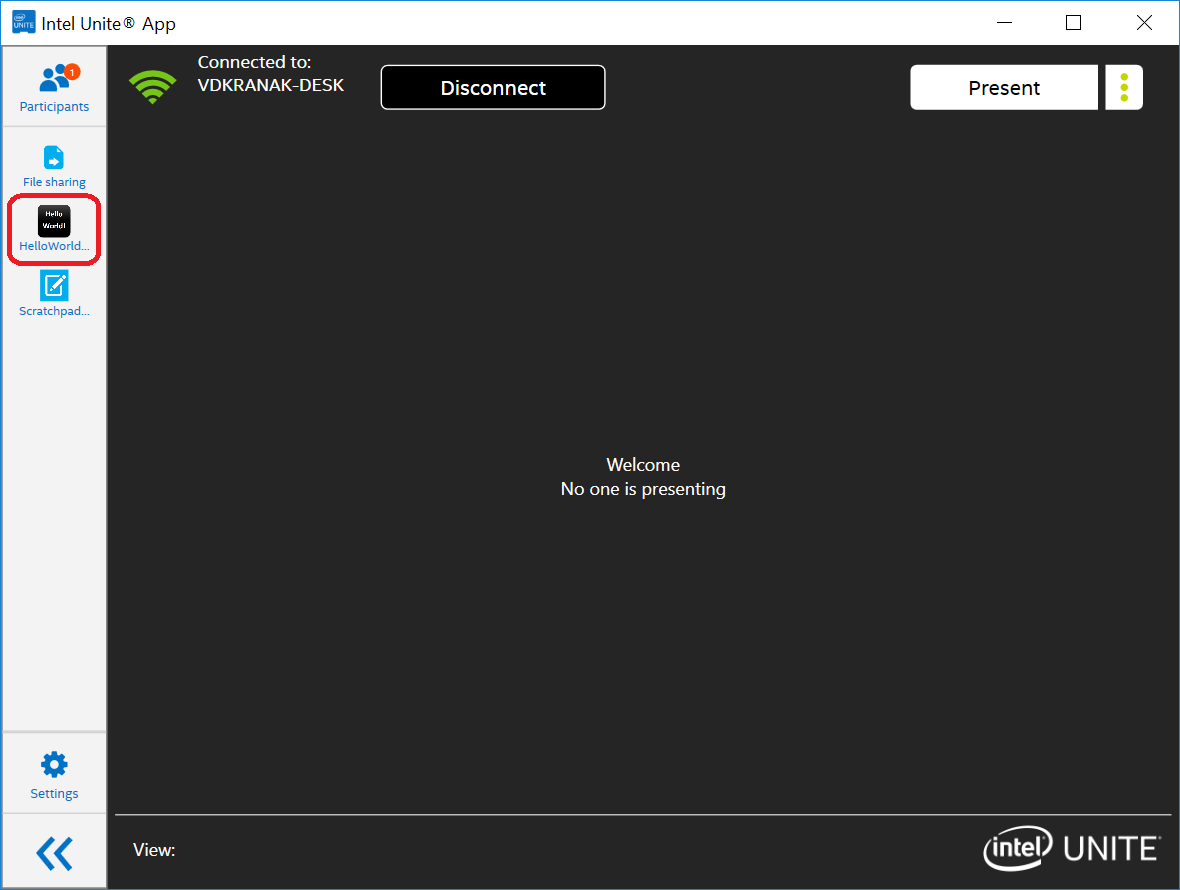
Figure 14 - Client UI



### Plugin Icon Image

The Icon Image that launches a browser emulator for the Plugin.

Figure 15 -Client Icon

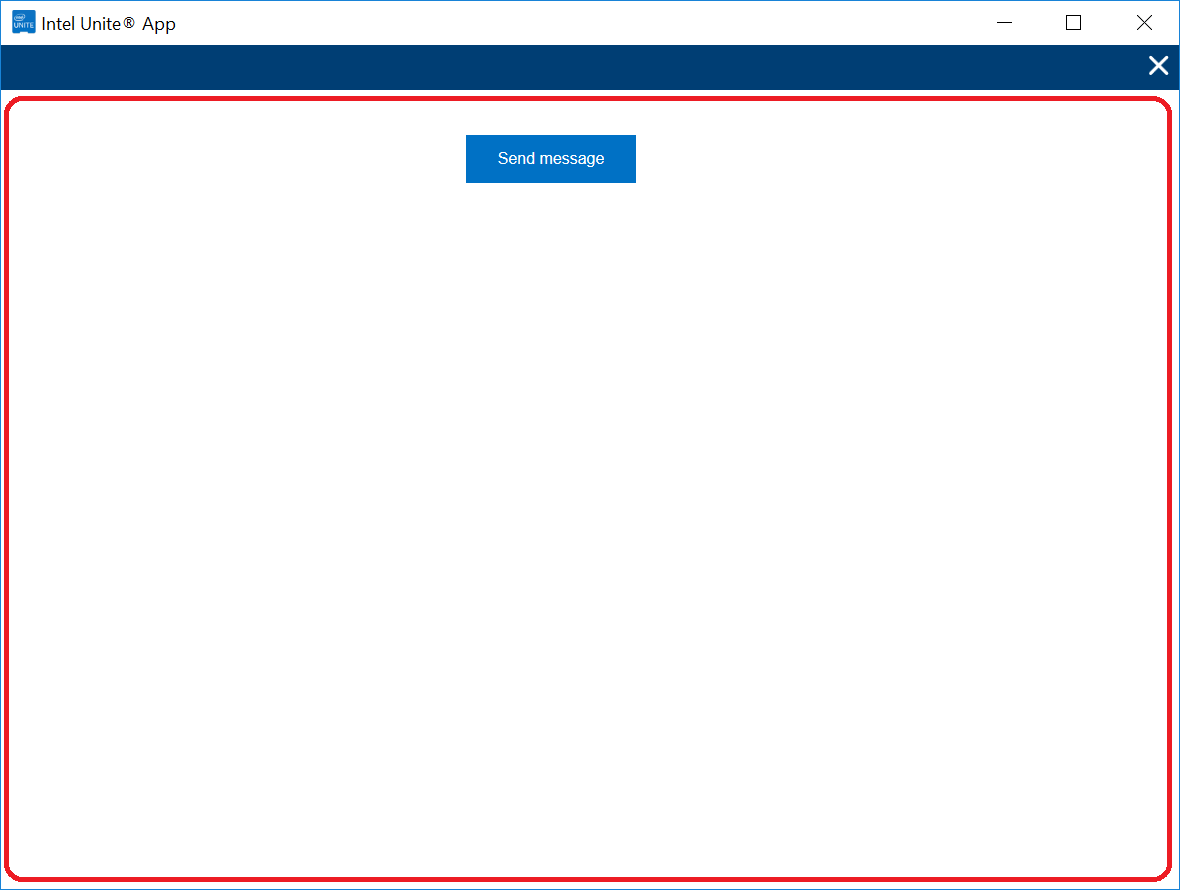


### Browser Emulator

HMTL from the Plugin that interacts with the Hub through exposed events and functions.

Note: Full browser functionality is limited due to security constraints.

Figure 16 - Browser Emulator



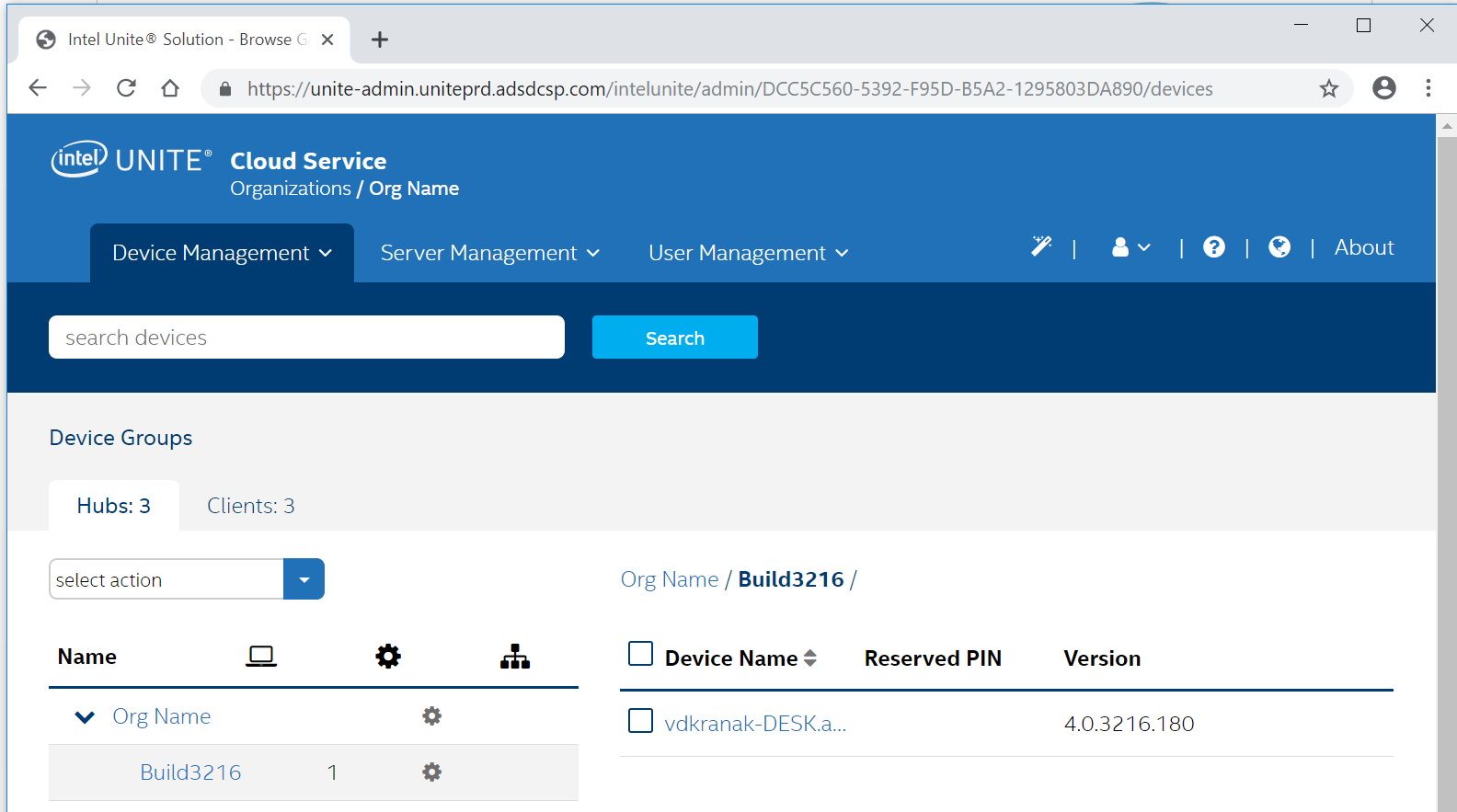
## Intel Unite® Software Pin Server

The Device used to administrate Intel Unite. This can be a Cloud Service or an on-premises Server. The Server hosts an Admin Portal used to configure Unite.

### Admin Portal

The web page that is used to access the features of Intel Unite® Server

Figure 17 - Admin Portal Home



1. Intel Unite® Software Plugin Module

## Requirements

The Intel Unite® solution is developed using the **Microsoft\* .NET\* Framework** and is built as a combination of distinct components that can be invoked individually.

The following requirements must be met for Intel Unite® software Plugin modules:

* A Plugin module must be packaged as a .cab file using the Intel Unite® Manifest Generator Tool
* Cab files must be signed
* The Project must Include a reference to Intel.Unite.Common.dll
* A class must be created and implemented that inherits from the abstract class Intel.Unite.Common.Module.Feature.Hub.HubFeatureModuleBase
* The class must have a constructor implementing the base class which takes no arguments
* The class must have a constructor implementing the base class which takes an IModuleRuntimeContext variable.
* The Plugin module must implement constants in the properties ModuleInfo and ModuleManifest

## Runtime Context

The Runtime Context lets the Unite Core know which Managers that the Plugin can use. The Managers are:

* **CalendarManager** - Exposes information about the Calendar
* **CommandManager** - Enables the Plugin to interact with the command’s functionality - the command refers to a message that is process by the Hub
* **ConfigurationManager**- the Plugin to obtain configuration properties
* **LafManager (Look and Feel Manager** - Exposes common UI properties, such as primary color, primary font, and so forth
* **LogManager** - Enables the Plugin to log messages
* **MessageSender**— - the Plugin to send messages, using TLS, from Client to Hub
* **NotificationManager** - Enables the functionality for using native notifications
* **TelemetryManager** - Enables the Plugin to log telemetry events

## Events

Several events notify Plugins. The events are fired when a User performs a specific action, as indicated:

* **UserConnected** - User connects
* **UserInfoChanged** - User information changes
* **UserDisconnected** - User disconnects
* **HubConnected** - Hub connects
* **HubInfoChanged** - Hub information changes
* **HubDisconnected** - Hub disconnects
* **IncomingMessage** - New message available for the Plugin module
* **SessionKeyChanged** - Session key changes, such as LockStatus, ModerationMode, and so forth

1. HTML Plugin

The HTML Plugin is a Hub Plugin that exposes an HTML interface in the Client UIs. This Plugin is multiplatform. The same Client implementation can be used by any platform that supports HTML. The HTML Plugin runs on a web browser emulator, which means the browser functionality is not fully supported due to security reasons. For example, browser developer tools are not allowed. In Windows, the web browser component supports versions IE7 through IE 11.

This interface allows interaction to occur between a Client and the Hub. Several events and functions are exposed to the Plugin (refer to the code examples, later in this document). The events and functions are described in the next sections.

## HTML Events

HTML events should be defined by developers in the HTML Content property. Events fire according to specific actions, as follows:

* **onUserConnected** - User connects
* **onUserInfoChanged** - User information changes
* **onUserDisconnected** - User disconnects
* **onMessage** - A new message is available for the Plugin
* **onModerationModeUpdated** - Moderation mode changes
* **onLockSession** - Session is locked
* **onUnlockSession** - Session is unlocked
* **sessionKeyChanged**- key changes (for example, LockStatus, ModerationMode, and so forth)
* **showScreenSharingCallback** - Result of showScreenSharing function indicating a call is ready

## HTML Functions

Functions are defined in the IntelUnite.Js library. This library is injected by the Hub in the HTML Content property. Some function examples include the following:

* **getContext** - Exposes context information to the Plugin (for example, a list of Users/Hubs connected with properties like name, email, ID, lock status, and so forth)
* **getTheme** - Exposes common UI properties, like primary color, primary font, and so forth
* **logMessage** - Enables Plugins to log messages
* **logTelemetryEvent** - Enables Plugins to log telemetry events
* **sendMessage** - Enables Plugins to send messages
* **setNumberNotification** - Enables Plugins to update the notification number shown in the menu button
* **showScreenSharing** - Enables Plugins to show the common screen sharing dialog
* **updateToolMenu** - Enables Plugins to update information of the tool menu button associated with the Plugin

File manager functions are used to handle file creation, save a screen shot, save files, and so forth. File manager functions include the following:

* **closeFile** - Closes the stream and saves the file
* **createFile** - Initializes the stream for file creation
* **deleteFile** - Deletes the file that was created with the identifier generated by createFile
* **getFileTransferConfiguration** - Gets the file transfer configuration from the client
* **openFolder** - Opens the file’s folder location o
* **writeToFile**- bytes to the stream in the client (the stream is initialize when createFile is executed)

1. Code Examples

## Minimum Plugin Project Setup

The basic setup necessary to upload an Intel Unite® Plugin to the Admin portal.

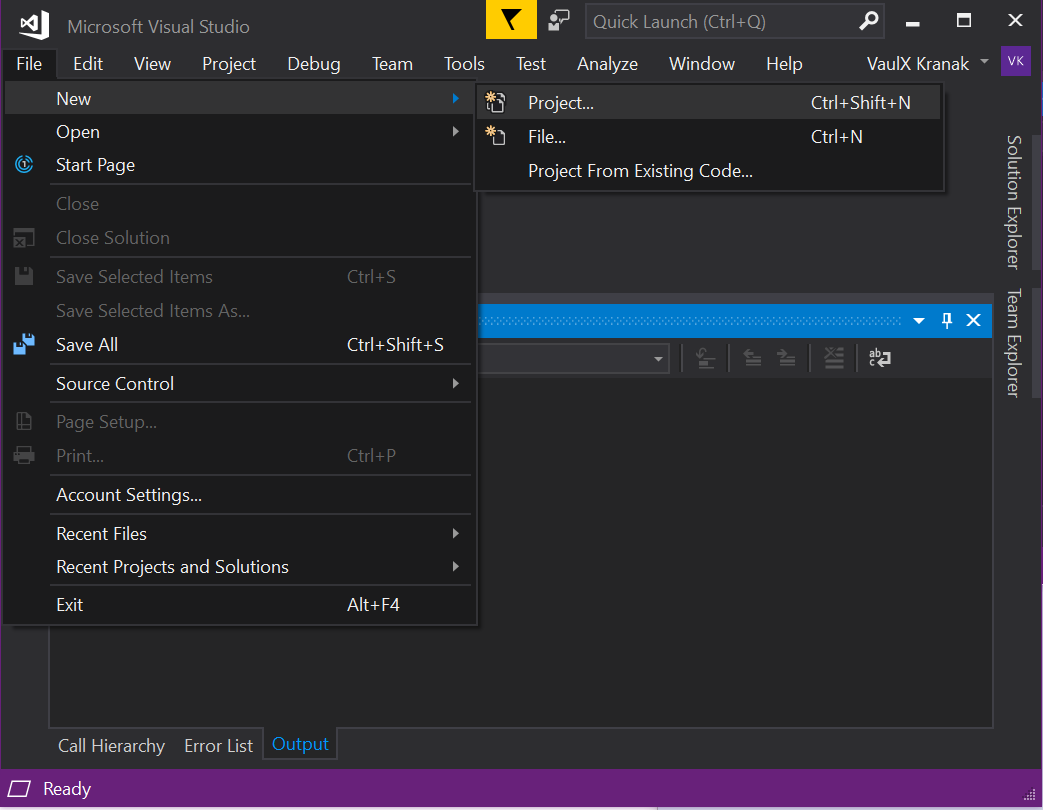
Requirements:

* Visual Studio
* Windows 10
* Windows 10 SDK
* Intel Unite® Manifest Generator

### Create and Configure a New Unite Plugin Solution in Visual Studio

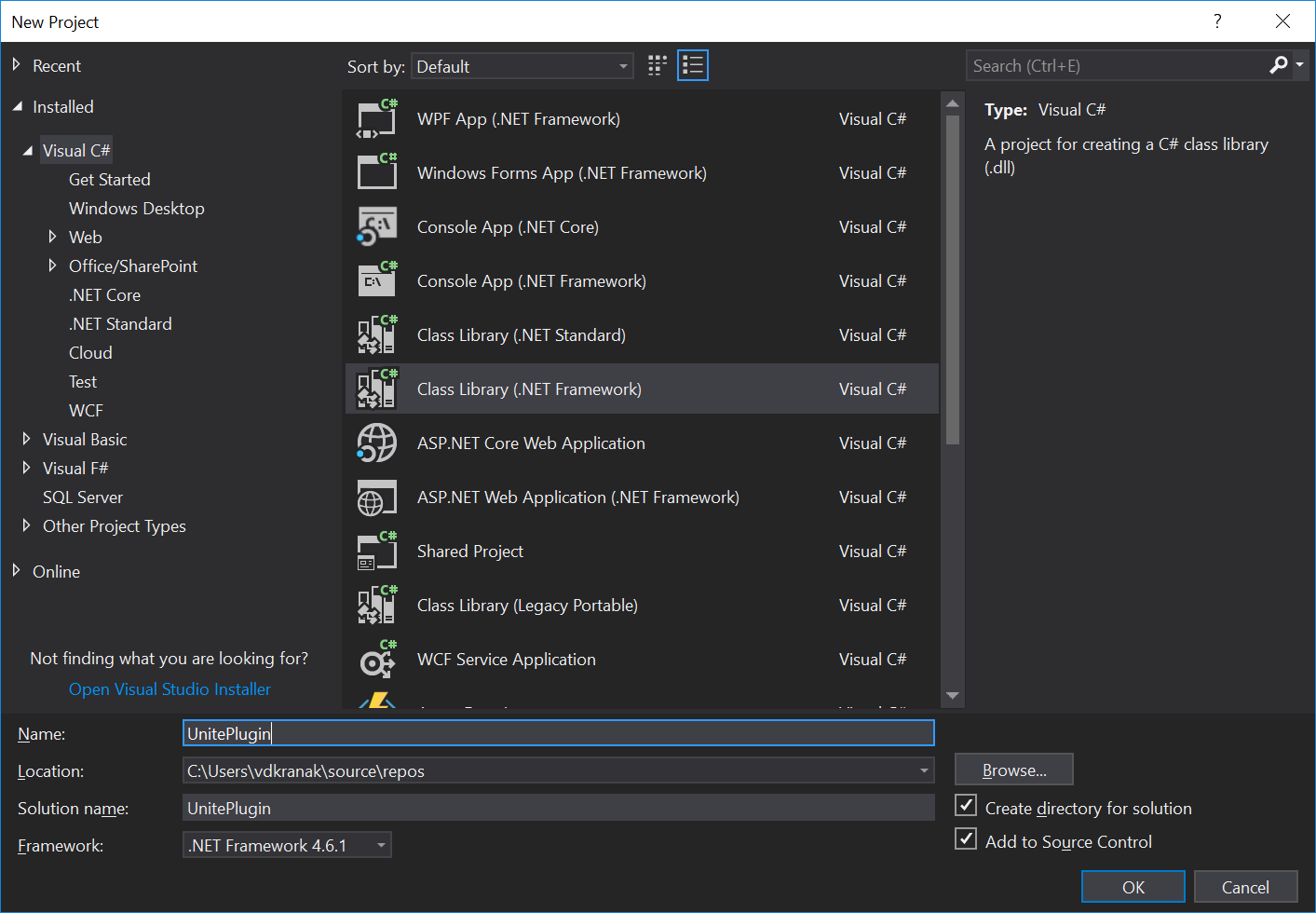
1. **In Visual Studio (Figure 18 – Project Setup Step 1): Select File -> New -> Project**

Figure 18 – New Solution



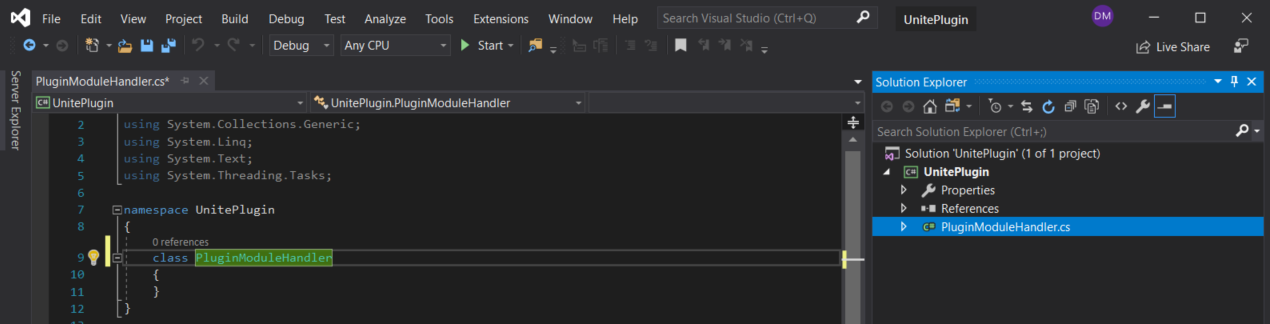
1. **Choose framework (Figure 19 – Project Setup Step 2): Select Class Library (.Net Framework - Visual C#) -> Name: UnitePlugin -> OK**

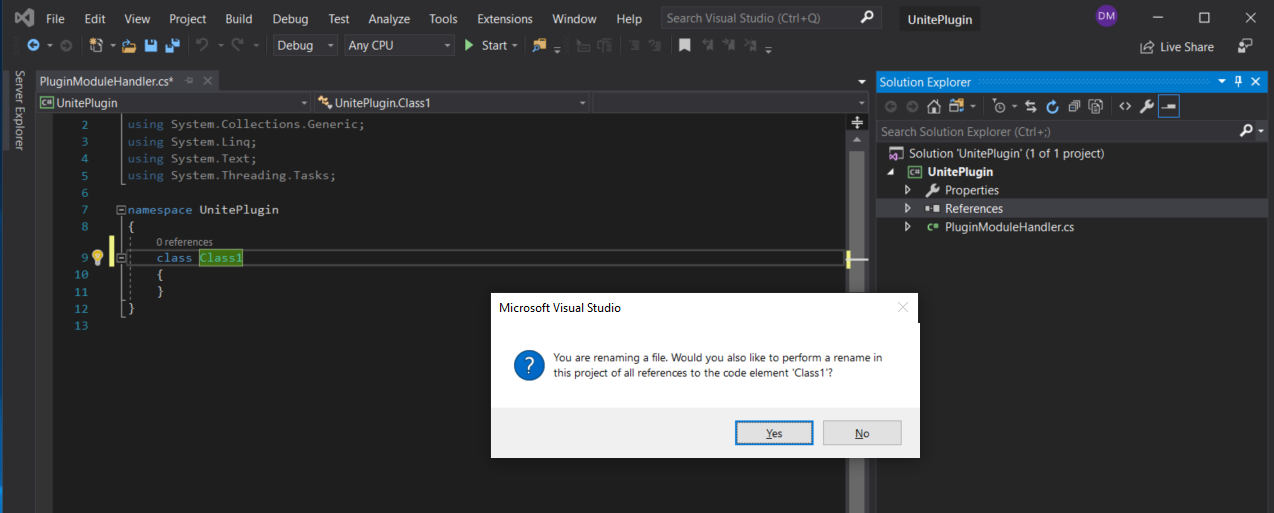
Figure 19 – Project Type



1. **Rename Class1.cs (Figure 20 – Project Setup Step 3): In the Solution Explorer: Select Class1.cs -> F2 -> type: PluginModuleHandler.cs -> Enter -> YES (on popup dialog) - > Enter.**

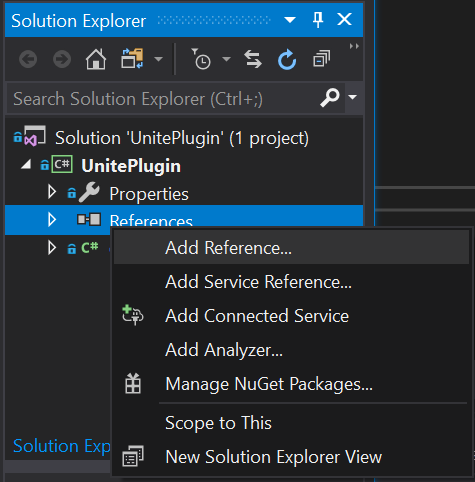
Figure 20 – Rename Default Class





1. **Add References (Figure 21 – Project Setup Step 4): In Solution Explorer right-click on References, then from the drop-down menu select: Add Reference… . The Reference Manager is then displayed.**

Figure 21 – Add References



1. From the Reference Manager: Click **Assemblies** -> Framework, then ensure the following are checked:

* PresentationCore.
* PresentationFramework
* System.AddIn
* System.AddIn.Contract
* System.Xml
* WindowsBase

Figure 22 – Add References

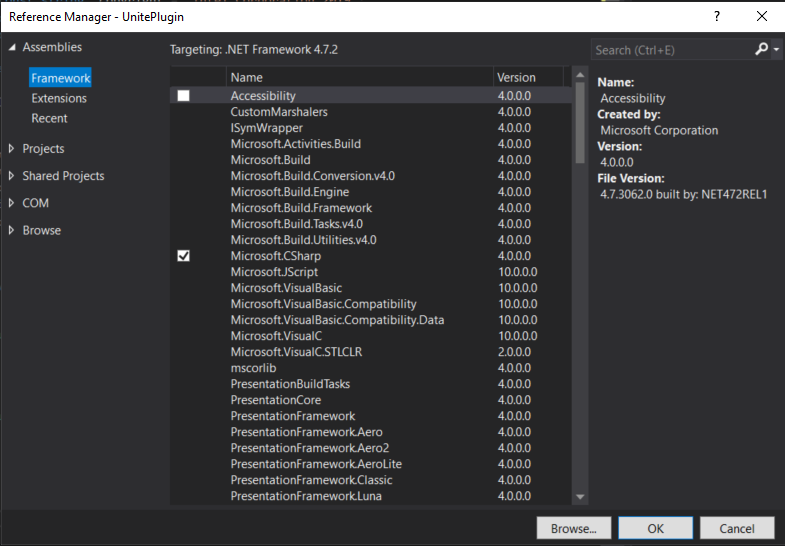
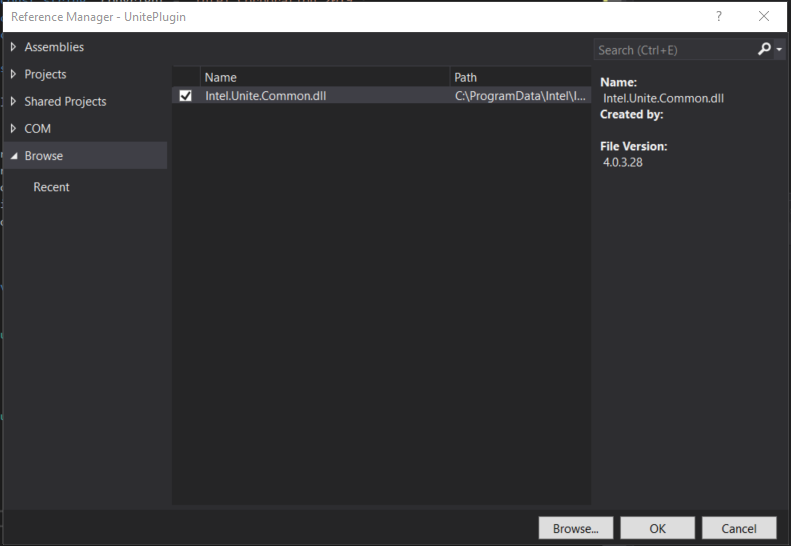
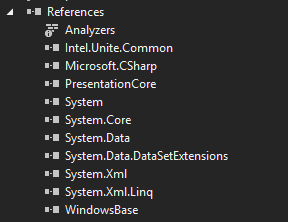


Figure 24 - Project Setup : Step 6



1. Click OK.

Figure 23 - Default References

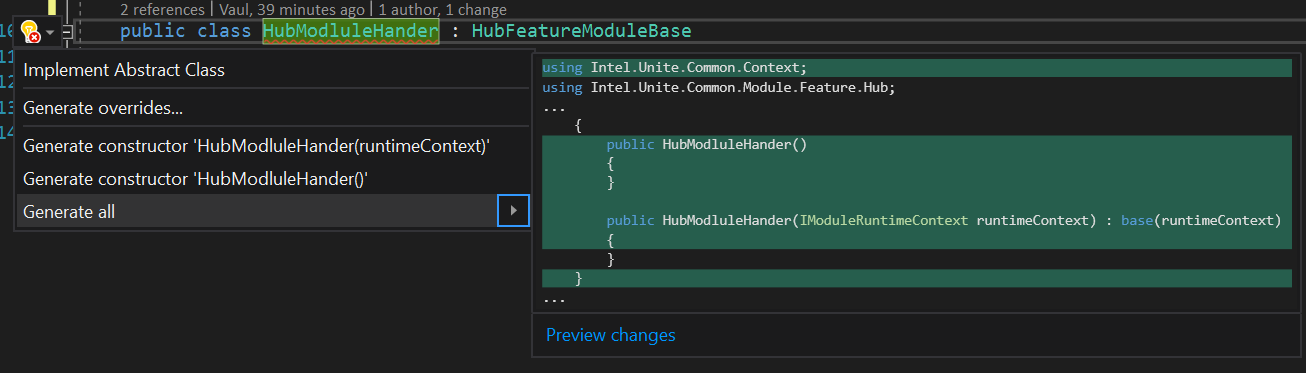


### Implement the HubFeatureModuleBase Abstract Class

All pugins must implement the HubFeatureModuleBase class

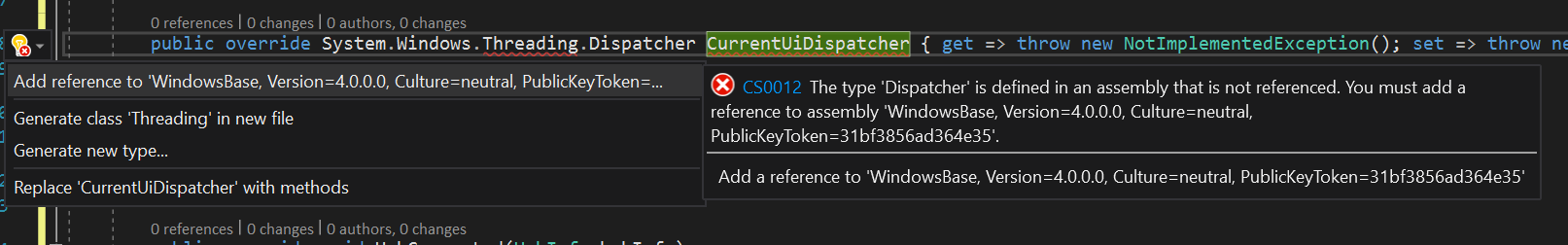
1. Create a Class that Implement the abstract class HubFeatureModuleBase

Figure 128. Implement Abstract Class



1. Add WindowsBase reference

Figure 129. Add Reference to WindowsBase



1. Create Constructors

Code 1. Create Module Constructors

|  |
| --- |
| public PluginModuleHandler() : base()  {  }  public PluginModuleHandler(IModuleRuntimeContext runtimeContext) : base(runtimeContext)  {  } |

### Implement ModuleInfo

Module Info is used by Intel Unite® to distinguish one Plugin from others. Each time updates to the Project occurs, the *Version Number* should be changed because the Admin Portal will not allow two Plugins with the same GUID and the same Version Number to be uploaded at the same time. Many versions of the same Plugin may exist on the Admin Portal if the Version Number is different. The values in the ModuleInfo will show up on the Admin Portal.

1. Add the following code to the PluginModuleHandler class:

Code 2. Implement ModuleInfo

|  |
| --- |
| private const string \_guid = "a9bbad72-eeb3-47cc-b147-345cc48738cf";  private const string \_name = "Unite Plugin Example";  private const string \_description = "Unite Plugin Example";  private const string \_copyright = "Intel Corporation 2019";  private const string \_vendor = "Intel Corporation";  private const string \_version = "1.0.0.3";  private static readonly ModuleInfo \_moduleInfo = new ModuleInfo  {  ModuleType = ModuleType.Feature,  Id = Guid.Parse(\_guid),  Name = \_name,  Description = \_description,  Copyright = \_copyright,  Vendor = \_vendor,  Version = Version.Parse(\_version),  SupportedPlatforms = ModuleSupportedPlatform.Mac | ModuleSupportedPlatform.Windows,  };  public override ModuleInfo ModuleInfo => \_moduleInfo; |

### Implement ModuleManifest

Add information for the manifest tool to create the cab file. All necessary DLLs should be added as part of the collection of ManifestFiles otherwise they will not be included in creating the cab file or uploaded to the admin portal.

1. Add the following code to the module

Code 3. Implement ModuleManifest

|  |
| --- |
| private const string \_minimumUniteVersion = "4.0.0.0";  private const string \_entryPoint = "UnitePlugin.dll";  private static readonly ManifestOsSet \_files = new ManifestOsSet  {  Windows = new Collection<ManifestFile>  {  new ManifestFile()  {  SourcePath = \_entryPoint,  TargetPath = \_entryPoint,  }  }  };  private static readonly ModuleManifest \_moduleManifest = new ModuleManifest  {  Owner = UniteModuleOwner.Hub,  ModuleId = \_moduleInfo.Id,  Name = new MultiLanguageString(\_moduleInfo.Name),  Description = new MultiLanguageString(\_moduleInfo.Description),  ModuleVersion = \_moduleInfo.Version,  MinimumUniteVersion = Version.Parse(\_minimumUniteVersion),  Settings = new Collection<ConfigurationSetting>(),  Files = \_files,  Installers = new Collection<ManifestInstaller>(),  EntryPoint = \_entryPoint,  ModuleType = \_moduleInfo.ModuleType,  };  public override ModuleManifest ModuleManifest => \_moduleManifest; |

### Add Post-build Events to Project to Create a *.cab* File and Sign It

Inte Unite® Solution requires a plugin to be released in a cab file. A way to streamline the process of creating a cab file is to add a scitpt to the post build events.

#### Install BuildTools

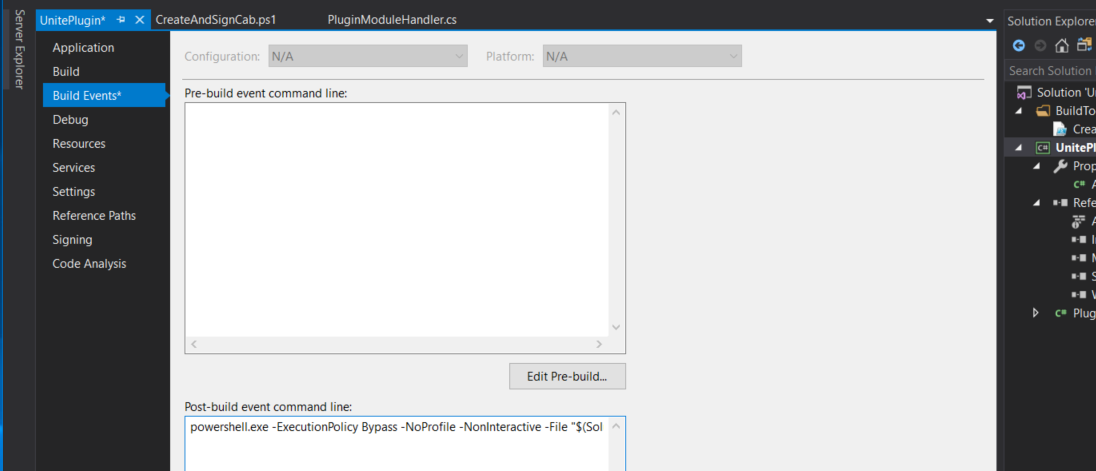
The **CreateAndSignCab.ps1** PowerShell script is part of the SDK and will work on any Windows 10 Version with the Windows 10 SDK installed.

1. Copy the BuildTools directory from the downloaded and installed SDK kit. Note: The SDK kit resides in a location selected by the User when the SDK Installer was executed.
2. Paste it into the UnitePlugin *Solution* directory. It *must not* reside in any specific Project directory, such as UnitePlugin, or UnitePluginTest.

#### Create the Post Build Event

1. Navigate to the Properties of the Project:
   1. Right-click on *UnitePlugin* in the Solutions Explorer -> Properties
2. Select *Build Events* (Figure 131) from the left column. Event editors are displayed showing two areas - *Pre-build event command line*, and *Post-build event command line*.

Figure 26 - Build Event Editors



1. Add text from the Post Buid Event Comandline
   1. Adjust the path location of the attributes for your local environment,
   2. This will be executed as a single line - make sure that no newline characters are included.

Code 4. Post Build Event ComandLine

|  |
| --- |
| powershell.exe -ExecutionPolicy Bypass -NoProfile -NonInteractive -File "$(SolutionDir)\BuildTools\CreateAndSignCab.ps1" -SolutionDir "$(ProjectDir).." -SolutionName "$(SolutionName)" -TargetPath "$(TargetPath)" -ManifestGeneratorDir "$(SolutionDir)\BuildTools\ManifestGenerator" |

## Load Module

Integrate a plugin into Intel Unite® solution.

Requirements:

* Ensure *Section 4.1 – Minimum Plugin Project Setup* builds without errors and the *.cab* file is generated.

**NOTE: All ViewModels must be *Serializable*. UIControls and other**

Essential steps necessary to Load a Module:

1. Create a Quick Access App Icon View (with icons)
2. Create ViewModel and its code-behind
3. HubViewModel and its code-behind

### Create a QuickAccessAppIconView View

The QuickAccessAppIconView View implements the WPF *UserControl* component to create a custom, clickable Plugin icon that, when clicked, launches Plugin-specific Controls.

1. **Create View folder**: From Solution Explorer -> Right-click on UnitePlugin -> Add -> New Folder -> type: **View** to name it.
2. **Create the QuickAccessAppIconView View**:
   1. **Add View xaml**: Right-click on the View folder -> Add -> New Item -> WPF -> User Control (WPF) -> Name type: QuickAccessAppIconView.xaml -> Add.
   2. **Add StackPanel xaml element:** Copy *StackPanel* element from *Code 4* text block and paste it into QuickAccessAppIconView.xaml inside the *Grid* element.
   3. **Add ToolTip to StackPanel element**
3. **Source and Set Property for the Hello World image**:
4. **Create Images folder**: From Solution Explorer -> Right-click on UnitePlugin -> Add  
   -> New Folder -> type: **Images** to name it.
   1. **Install Menu icon source image**: Right-click on the Images folder -> Add -> Existing Item -> select: All Files (\*.\*) -> browse to <path to SDK>\Images -> select menu-icon.png -> Add.
   2. **Set Menu image Properties**: Right-click on menu-icon.png -> Properties -> *Build Action* -> select: **Resource** -> *Copy to Output Directory* -> select: **Copy if newer** -> X.

Code 7. QuickAccesIconView xaml

|  |
| --- |
| <UserControl x:Class="UnitePlugin.View.QuickAccessIconView"  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">  <Grid>  <StackPanel Background="#FF423E4F">  <Button x:Name="QuickAccessButton">  <Image x:Name="QuickAccessIconImage" Source="/UnitePlugin;component/Images/menu-icon.png"/>  </Button>  </StackPanel>  </Grid>  </UserControl> |

1. **Create the Hub View**:
   1. **Create ViewModel folder**: From Solution Explorer - > Right-click on UnitePlugin -> Add -> New Folder -> type: **ViewModel** to name it.
   2. **Create the HubViewModel class**: Right-click on the ViewModel folder -> Add -> Class… -> Code -> Class -> Name -> type: HubViewModel.cs -> Add.
   3. **Add Serialization to HubViewModel class**: See Code 5 text block.
   4. **Implement INotifyProperyChanged interface**: See Code 5 text block.  
      **IMPORTANT**: The *PropertyChanged* field cannot be Serialized.
   5. **Add Display Manager-specific fields**

Code 8. Serialize ViewModel

|  |
| --- |
| namespace UnitePlugin.ViewModel  {  [Serializable]  class HubViewModel  {  }  } |

Code 9. Impllement INotifyPropertyChanged

|  |
| --- |
| using System;  using System.ComponentModel;  using System.Runtime.CompilerServices;  namespace UnitePlugin.ViewModel  {  [Serializable]  class HubViewModel : INotifyPropertyChanged  {  [field: NonSerialized]  public event PropertyChangedEventHandler PropertyChanged;  private void NotifyPropertyChanged([CallerMemberName] String propertyName = "")  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }  }  } |

Code 10. DisplayManger Specific Fields

|  |
| --- |
| public HubAllocationInfo HubAllocationInfo { get; set; }  public Guid ControlIdentifier { get; private set; }  public bool IsAllocated { get; private set; } |

Code 11. HubViewModel Constructor

|  |
| --- |
| public HubViewModel()  {  IsAllocated = false;  } |

Code 12. DisplayManger Specific Methods

|  |
| --- |
| public void AllocatedCallBack(HubAllocationResult hubAllocationResult)  {  if (hubAllocationResult.Success)  {  ControlIdentifier = hubAllocationResult.AllocatedView.Id;  IsAllocated = true;  }  else  {  throw new Exception(hubAllocationResult.ResultType.ToString());  }  }  public void DeallocatedCallBack(HubAllocationResult hubAllocationResult)  {  if (hubAllocationResult.Success)  {  IsAllocated = false;  }  else  {  throw new Exception(hubAllocationResult.ResultType.ToString());  }  } |

### Implement the *Load Method*

The Load method is called during Intel Unite launch process when a module is added to a configuration. This method is used to add a module specific Icon UserControl to the QuickAccessView and ready the module for use.

Important*:* Views must be allocated for each display otherwise they will only display on one monitor

To the PluginModuleHandler class:

1. **Ensure the PresentationFramework.dll** assembly referenced using Solution Explorer.
2. Add two *using* declarations:
   1. using System.Windows;
   2. using Systems.Collections.Generic;
3. Add *views* field:
   1. private readonly List<FrameworkElement> views = new List<FrameworkElement>();**Add** *AddQuickAccessIconToViews*: The following code block:
   2. Allocates QuickAccessIconView(s) through DisplayManager,
   3. Allocates QuickAccessIconViews to Hub.

Code 16. Add Views Field

|  |
| --- |
| private readonly List<FrameworkElement> views = new List<FrameworkElement>(); |

Code 17. Add using for views field

|  |
| --- |
| using System.Collections.Generic;  using System.Windows; |

Code 18. AddQuickAccessIconToViews

|  |
| --- |
| public override void Load()  {  AddQuickAccessIconToViews();  }  private void AddQuickAccessIconToViews()  {  CurrentUiDispatcher.Invoke(delegate  {  RuntimeContext.DisplayManager.AvailableDisplays.ToList().ForEach(d => views.Add(GetNewQuickAccessIconView(d)));  });  }  private QuickAccessIconView GetNewQuickAccessIconView(PhysicalDisplay display)  {  return new QuickAccessIconView  {  DataContext = new HubViewModel  {  HubAllocationInfo = new HubAllocationInfo  {  FriendlyName = "QuickAccessIcon",  ModuleOwnerId = ModuleInfo.Id,  PhysicalDisplay = display,  ViewType = HubDisplayViewType.QuickAccessAppIconView,  }  }  };  } |

Code 19. AllocateQuickAccessIconViewsToHub

|  |
| --- |
| public override void Load()  {  AddQuickAccessIconToViews();  AllocateQuickAccessIconViewsToHub();  }  private void AllocateQuickAccessIconViewsToHub()  {  List<FrameworkElement> quickAccessIconViews = views.Where(view => (view.DataContext as HubViewModel).HubAllocationInfo.ViewType == HubDisplayViewType.QuickAccessAppIconView).ToList();  quickAccessIconViews.ForEach(view => AllocateView(view));  }  private void AllocateView(FrameworkElement view)  {  HubViewModel hubViewModel = null;  CurrentUiDispatcher.Invoke(delegate  {  hubViewModel = (HubViewModel)view.DataContext;  });  RuntimeContext.DisplayManager.AllocateUiInHubDisplayAsync(  CreateContract(view),  hubViewModel.HubAllocationInfo,  hubViewModel.AllocatedCallBack  );  } |

## Add Sensor Data for other plugins

Intel Unite® solution provides a way for plugins to access sensor information. Mock data will be used for this illustration on how to accomplish Plugin-to-Sensor communication.

Requirements:

* Minimum Plugin Project Setup
* Sensor Data (In this example the data is mocked)

### Understanding the Sensor Class

The Sensor class is used to store information about the sensor as well as the data it collects.

#### FriendlyName

A name which is easy to understand.

#### UniqueName

A Unite Unique Name, for example "UnitePlugin\_Temp\_Probe\_1".

#### Id

A unique GUID.

#### ModuleId

The GUID of the plugin the sensor was generated from.

#### Type

Type of sensor [0-127 are reserved types, 128-255 are custom]. The reserved types are defined in Intel.Unite.Common.Sensor.UniteSensorType.

#### KeyValueProperties

This is where the data of the sensor is held. A sensor can have multiple readings. A weather station sensor might have air pressure, temperature, and humidity readings.

#### Expiration

The moment when the data is no longer valid. Data will be kept for a maximum of 24 hours. Intel Unite® solution assumes that if the sensor is active, then data will be regularly updated. If the sensor doesn’t update, the data will expire and be removed.

### Create a Plugin SensorManager

Used to create a method to attach to a plugin specific event which will route to SensorManager.Set.

Code 1 - PluginSensorManager

|  |
| --- |
| public class PluginSensorManager  {  public ISensorManager SensorManager { get; }  public PluginSensorManager(ISensorManager sensorManager)  {  SensorManager = sensorManager;  }  public void UpdateSensorData(object sender, SensorArgs args)  {  SensorManager.Set(args.Sensor);  }  } |

### Create a Sensor

A sensor must regularly update, otherwise its data will expire and other plugins will not have access to that data. The sensor created must generate the Sensor class for consumption by the SensorManager.

Code 2 - MockSensor

|  |
| --- |
| public class MockSensor  {  private const int \_maxChange = 3;  private static readonly Random \_random = new Random();  private static readonly Timer \_aTimer = new Timer(2000);    private static int Temp { get; set; } = 72;  public static event EventHandler<SensorArgs> UpdateSensorData;  public const string UniqueName = "UnitePlugin\_Temp\_Probe\_1";    public static void Start()  {  \_aTimer.Elapsed += OnTimedEvent;  \_aTimer.AutoReset = true;  \_aTimer.Start();  }  public static void Stop()  {  \_aTimer.Elapsed -= OnTimedEvent;  \_aTimer.AutoReset = false;  \_aTimer.Stop();  }  private static void OnTimedEvent(object source, ElapsedEventArgs e)  {  Temp += \_random.Next(\_maxChange) - \_random.Next(\_maxChange);  UpdateSensorData?.Invoke(source, new SensorArgs(GetTempSensor(Temp)));  }  private static Sensor GetTempSensor(int temp)  {  return new Sensor  {  FriendlyName = "Unite Plugin Temperature",  Expiration = DateTime.Now.AddHours(24),  Id = Guid.NewGuid(),  KeyValueProperties = new List<SensorKeyValue> {new SensorKeyValue{Key = "Value", Value = temp.ToString(), ValueType = SensorValueType.Int}},  ModuleId = Guid.NewGuid(),  Type = (int)UniteSensorType.Temperature,  UniqueName = UniqueName  };  }  } |

### Configure the Sensor

Setup the links between the Sensor, Plugin SensorManager, and the Common SensorManager

Code 3 - SensorConfig

|  |
| --- |
| public static class SensorConfig  {  public static void Setup()  {  SetupPluginSensorManager();  SetupLocalSensorUpdates();  }  private static void SetupLocalSensorUpdates()  {  MockSensor.UpdateSensorData += UnitePluginConfig.PluginSensorManager.UpdateSensorData;  }  private static void SetupPluginSensorManager()  {  UnitePluginConfig.PluginSensorManager = new PluginSensorManager(UnitePluginConfig.RuntimeContext.SensorManager);  }  } |

### Add sensor methods to Load

The load method is called when a plugin is loaded by the Unite Core and is the best place to call configuration methods.

Code - 4 Load Sensor

|  |
| --- |
| public override void Load()  {  UnitePluginConfig.RuntimeContext = RuntimeContext;  SensorConfig.Setup();  MockSensor.Start();  } |

## Get Sensor Data from other plugins

Intel Unite® solution provides a way for plugins to communicate sensor information from one plugin to another through the SensorManager Interface.

Requirements:

* Minimum Plugin Project Setup
* A Sensor is added by a plugin

## Displaying Images within Unite

Images can be worth a 1000 words but getting them displayed correctly can be a chore. This primer will cover a few basics when wanting to use Images with Unite.

Unite has and uses a unique image type called a UniteImage – this object has some differences when attempting to create. In this primer we will be using PNG extensions of our images, but UniteImage supports other image extensions such as JPG.

Status and Auth Views – require using an UniteImage. Attempting to use a normal Image regardless of its extension will result in no image being displayed in these views.

Clients – Inside Unite’s abstract class: Module Base is a UniteImage named ModuleImage and this is the Image used by Intel Unite Clients. If this UniteImage is not set during the Load method in your plugin’s ModuleHandler, then it is possible that a Client could have an empty Image in their application.

### Image as an Embedded Resource

When setting Images to be an Embedded Resource, it is important to pay attention to the physical directory structure inside Visual Studio of the images because the path uses a period (.) as a separator instead of slashes (/).

Looking at the target path, you can see that we are referring to the Project Name. Then “Folder Name.Name” of resource as PNG.

Code 1 - Using as an Embedded Resource

|  |
| --- |
| /// <summary>  /// creates a UniteImage from and Embedded Resouce  /// </summary>  /// <param name="resoucePath"></param>  /// <param name="uniteImageType" cref="UniteImageType"></param>  /// <returns cref="UniteImage"></returns>  /// <example>  /// <code>  /// UniteImage image = GetUniteImageFromEmbeddedResource("ProjectName.FolderName.recording-icon.png", UniteImageType.Png);  /// </code>  /// </example>  protected UniteImage GetUniteImageFromEmbeddedResource(string resoucePath, UniteImageType uniteImageType)  {  return Intel.Unite.Common.Utils.BytesHelper.SetImageFromResource(Guid.NewGuid(),  uniteImageType, resoucePath, Assembly.GetExecutingAssembly());  } |

### Image as a Resource

When setting Images to be a Resource, it is also important to pay attention to the directory structure of the images because the path uses a slash (/) as one would expect.

Code 2 - Using as a Resource

|  |
| --- |
| /// <summary>  /// creates a UniteImage from and Resource  /// </summary>  /// <param name="resoucePath"></param>  /// <param name="uniteImageType" cref="UniteImageType"></param>  /// <returns cref="UniteImage"></returns>  /// <example>  /// <code>  /// var Image = GetUniteImageFromResouce("/Namespace;component/Images/recording-icon.png", UniteImageType.Png);  /// </code>  /// </example>  protected UniteImage GetUniteImageFromResouce(string resourcePath, UniteImageType uniteImageType)  {  var resourceLocater = new Uri(resourcePath, System.UriKind.Relative);  var resourceInfo = Application.GetResourceStream(resourceLocater);  using (var memoryStream = new MemoryStream())  {  resourceInfo.Stream.CopyTo(memoryStream);  var imgObj = System.Drawing.Image.FromStream(memoryStream);  var size = new UniteDisplayRect { Height = imgObj.Size.Height, Width = imgObj.Size.Width };  return new UniteImage  {  Id = Guid.NewGuid(),  Data = memoryStream.ToArray(),  DataType = uniteImageType,  Size = size,  };  }  } |

### ImageViewBase

Unite Hub requires the Status and Auth Views to be Allocated differently in that the UniteImage is to be used as a parameter. When using a StatusView or AuthView ensure that UI components inherit from the ImageViewBase class.

1. In the **UI** folder, create a file name ImageViewBase.cs

Code 3 - ImageViewBase.cs contents

|  |
| --- |
| public class ImageViewBase : ViewBase  {  [field: NonSerialized]  private UniteImage \_image;  protected UniteImage Image { get => \_image; set => \_image = value; }  protected ImageViewBase(IHubModuleRuntimeContext runtimeContext, PhysicalDisplay display, Dispatcher currentUiDispatcher, Func<FrameworkElement, MarshalNativeHandleContract> createContract) :  base(runtimeContext, display, currentUiDispatcher, createContract)  { }  public override void Allocate()  {  RuntimeContext.DisplayManager.AllocateUiInHubDisplayAsync(  this.Image,  HubAllocationInfo,  AllocatedCallBack  );  }  public override void DeAllocate()  {  RuntimeContext.DisplayManager.DeallocateUiFromHubDisplayAsync(  DisplayView,  DeallocateCallBack  );  }  public override bool Show()  {  if (!IsAllocated) Allocate();  CurrentUiDispatcher.BeginInvoke(DispatcherPriority.ApplicationIdle, new Action(() => { })).Wait();  return IsAllocated && RuntimeContext.DisplayManager.ShowAllocatedUi(DisplayView);  }  public override void AllocatedCallBack(HubAllocationResult hubAllocationResult)  {  if (hubAllocationResult.Success)  {  AllocatedSuccess(hubAllocationResult.AllocatedView);  }  else  {  AllocatedFail();  throw new Exception(hubAllocationResult.ResultType.ToString());  }  }  public override bool AllocatedFail()  {  lock (this)  {  }  return true;  }  public override void AllocatedSuccess(DisplayView allocatedView)  {  lock (this)  {  DeepCopy.CopyDisplayView(DisplayView, allocatedView);  }  }  public override void DeallocateCallBack(HubAllocationResult hubAllocationResult)  {  if (hubAllocationResult.Success)  {  lock (this)  {  SetDeallocate();  }  }  else  {  throw new Exception(hubAllocationResult.ResultType.ToString());  }  }  private void SetDeallocate()  {  DisplayView.Id = new Guid();  }    // Removed the two GetUniteImage helper methods from above to save space  } |

Note that the second parameter of the Allocate method is the class UniteImage, instead of a DisplayView.

Code 4 - Allocate method

|  |
| --- |
| public override void Allocate()  {  RuntimeContext.DisplayManager.AllocateUiInHubDisplayAsync(  this.Image,  HubAllocationInfo,  AllocatedCallBack  );  } |

### Process Sensor Data

How the plugin will handle sensor data updates. In this example a toast message is created every time the temperature is updated.

Code 5 SensorHandler

|  |
| --- |
| public static class MockSensorHandler  {  public const int VisibilityTime = 10;  public static void ProcessData(object sender, SensorArgs e)  {  if (e.Sensor.UniqueName == MockSensor.UniqueName)  {  UnitePluginConfig.RuntimeContext.DisplayManager.TryShowToastMessage(e.Sensor.KeyValueProperties  .FirstOrDefault(x => x.Key == "Value")?.Value, VisibilityTime);  }  }  } |

### Subscribe to Sensor Updates

Link the sensor handler to the sensor added event. The sensor added event will trigger every time SensorManager.Set is called.

Code 6 Setup Sensor Updates

|  |
| --- |
| public static class SensorConfig  {  public static void Setup()  {  SetupCoreUpdates();  }  private static void SetupCoreUpdates()  {  UnitePluginConfig.RuntimeContext.SensorManager.SensorAdded += MockSensorHandler.ProcessData;  }  } |

## Displaying Images within Unite

Images can be worth a 1000 words but getting them displayed correctly can be a chore. This primer will cover a few basics when wanting to use Images with Unite.

Unite has and uses a unique image type called a UniteImage – this object has some differences when attempting to create. In this primer we will be using PNG extensions of our images, but UniteImage supports other image extensions such as JPG.

Status and Auth Views – require using an UniteImage. Attempting to use a normal Image regardless of its extension will result in no image being displayed in these views.

Clients – Inside Unite’s abstract class: Module Base is a UniteImage named ModuleImage and this is the Image used by Intel Unite Clients. If this UniteImage is not set during the Load method in your plugin’s ModuleHandler, then it is possible that a Client could have an empty Image in their application.

### Image as an Embedded Resource

When setting Images to be an Embedded Resource, it is important to pay attention to the physical directory structure inside Visual Studio of the images because the path uses a period (.) as a separator instead of slashes (/).

Looking at the target path, you can see that we are referring to the Project Name. Then “Folder Name.Name” of resource as PNG.

Code 1 - Using as an Embedded Resource

|  |
| --- |
| /// <summary>  /// creates a UniteImage from and Embedded Resouce  /// </summary>  /// <param name="resoucePath"></param>  /// <param name="uniteImageType" cref="UniteImageType"></param>  /// <returns cref="UniteImage"></returns>  /// <example>  /// <code>  /// UniteImage image = GetUniteImageFromEmbeddedResource("ProjectName.FolderName.recording-icon.png", UniteImageType.Png);  /// </code>  /// </example>  protected UniteImage GetUniteImageFromEmbeddedResource(string resoucePath, UniteImageType uniteImageType)  {  return Intel.Unite.Common.Utils.BytesHelper.SetImageFromResource(Guid.NewGuid(),  uniteImageType, resoucePath, Assembly.GetExecutingAssembly());  } |

### Image as a Resource

When setting Images to be a Resource, it is also important to pay attention to the directory structure of the images because the path uses a slash (/) as one would expect.

Code 2 - Using as a Resource

|  |
| --- |
| /// <summary>  /// creates a UniteImage from and Resource  /// </summary>  /// <param name="resoucePath"></param>  /// <param name="uniteImageType" cref="UniteImageType"></param>  /// <returns cref="UniteImage"></returns>  /// <example>  /// <code>  /// var Image = GetUniteImageFromResouce("/Namespace;component/Images/recording-icon.png", UniteImageType.Png);  /// </code>  /// </example>  protected UniteImage GetUniteImageFromResouce(string resourcePath, UniteImageType uniteImageType)  {  var resourceLocater = new Uri(resourcePath, System.UriKind.Relative);  var resourceInfo = Application.GetResourceStream(resourceLocater);  using (var memoryStream = new MemoryStream())  {  resourceInfo.Stream.CopyTo(memoryStream);  var imgObj = System.Drawing.Image.FromStream(memoryStream);  var size = new UniteDisplayRect { Height = imgObj.Size.Height, Width = imgObj.Size.Width };  return new UniteImage  {  Id = Guid.NewGuid(),  Data = memoryStream.ToArray(),  DataType = uniteImageType,  Size = size,  };  }  } |

### ImageViewBase

Unite Hub requires the Status and Auth Views to be Allocated differently in that the UniteImage is to be used as a parameter. When using a StatusView or AuthView ensure that UI components inherit from the ImageViewBase class.

1. In the **UI** folder, create a file name ImageViewBase.cs

Code 3 - ImageViewBase.cs contents

|  |
| --- |
| public class ImageViewBase : ViewBase  {  [field: NonSerialized]  private UniteImage \_image;  protected UniteImage Image { get => \_image; set => \_image = value; }  protected ImageViewBase(IHubModuleRuntimeContext runtimeContext, PhysicalDisplay display, Dispatcher currentUiDispatcher, Func<FrameworkElement, MarshalNativeHandleContract> createContract) :  base(runtimeContext, display, currentUiDispatcher, createContract)  { }  public override void Allocate()  {  RuntimeContext.DisplayManager.AllocateUiInHubDisplayAsync(  this.Image,  HubAllocationInfo,  AllocatedCallBack  );  }  public override void DeAllocate()  {  RuntimeContext.DisplayManager.DeallocateUiFromHubDisplayAsync(  DisplayView,  DeallocateCallBack  );  }  public override bool Show()  {  if (!IsAllocated) Allocate();  CurrentUiDispatcher.BeginInvoke(DispatcherPriority.ApplicationIdle, new Action(() => { })).Wait();  return IsAllocated && RuntimeContext.DisplayManager.ShowAllocatedUi(DisplayView);  }  public override void AllocatedCallBack(HubAllocationResult hubAllocationResult)  {  if (hubAllocationResult.Success)  {  AllocatedSuccess(hubAllocationResult.AllocatedView);  }  else  {  AllocatedFail();  throw new Exception(hubAllocationResult.ResultType.ToString());  }  }  public override bool AllocatedFail()  {  lock (this)  {  }  return true;  }  public override void AllocatedSuccess(DisplayView allocatedView)  {  lock (this)  {  DeepCopy.CopyDisplayView(DisplayView, allocatedView);  }  }  public override void DeallocateCallBack(HubAllocationResult hubAllocationResult)  {  if (hubAllocationResult.Success)  {  lock (this)  {  SetDeallocate();  }  }  else  {  throw new Exception(hubAllocationResult.ResultType.ToString());  }  }  private void SetDeallocate()  {  DisplayView.Id = new Guid();  }    // Removed the two GetUniteImage helper methods from above to save space  } |

Note that the second parameter of the Allocate method is the class UniteImage, instead of a DisplayView.

Code 4 - Allocate method

|  |
| --- |
| public override void Allocate()  {  RuntimeContext.DisplayManager.AllocateUiInHubDisplayAsync(  this.Image,  HubAllocationInfo,  AllocatedCallBack  );  } |

1. Package Creation and Deployment

## Package Creation

The Intel Unite® Plugin Software Development Kit (Plugin SDK) includes a command line Manifest Tool used to create a *.cab* file. The Tool is located in \BuildTools\ManifestGenerator.

To generate a *.cab* file from the command line:

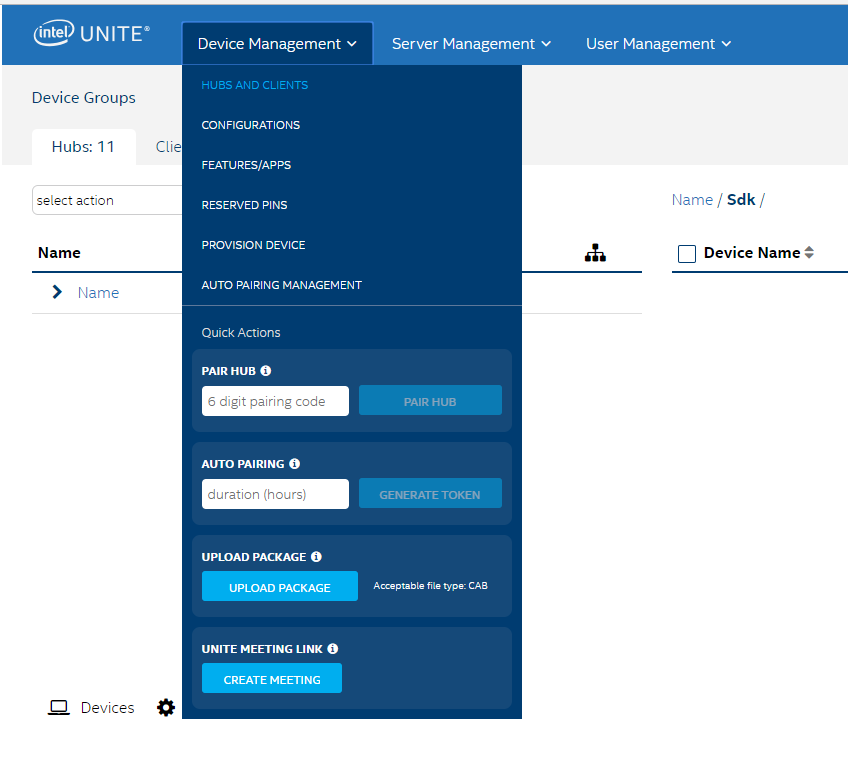
1. Execute: <Path to SDK>\BuildTools\ManifestGenerator\”Intel Unite Manifest Tool.exe” -m <Path to Project>\UnitePlugin.dll -o <Path to Project>UnitePlugin.cab".
2. Output: UnitePlugin.cab
3. Sign the .cab file. (This process is out of scope for this guide.)

## Plugin Deployment

Upload the *.cab* file using the Intel Unite Admin Portal as follows:

1. **Open Intel Unite® Admin Portal:** Navigate to **admin.unitecloud.intel.com/intelunite/admin -**> Click **Manage Organization** -> Click **Device Management** tab -> Device Management Drop Down is revealed (see Device Management Drop Down) -> Click the **UPLOAD PACKAGE** button.

Figure 27 - Device Management Drop Down



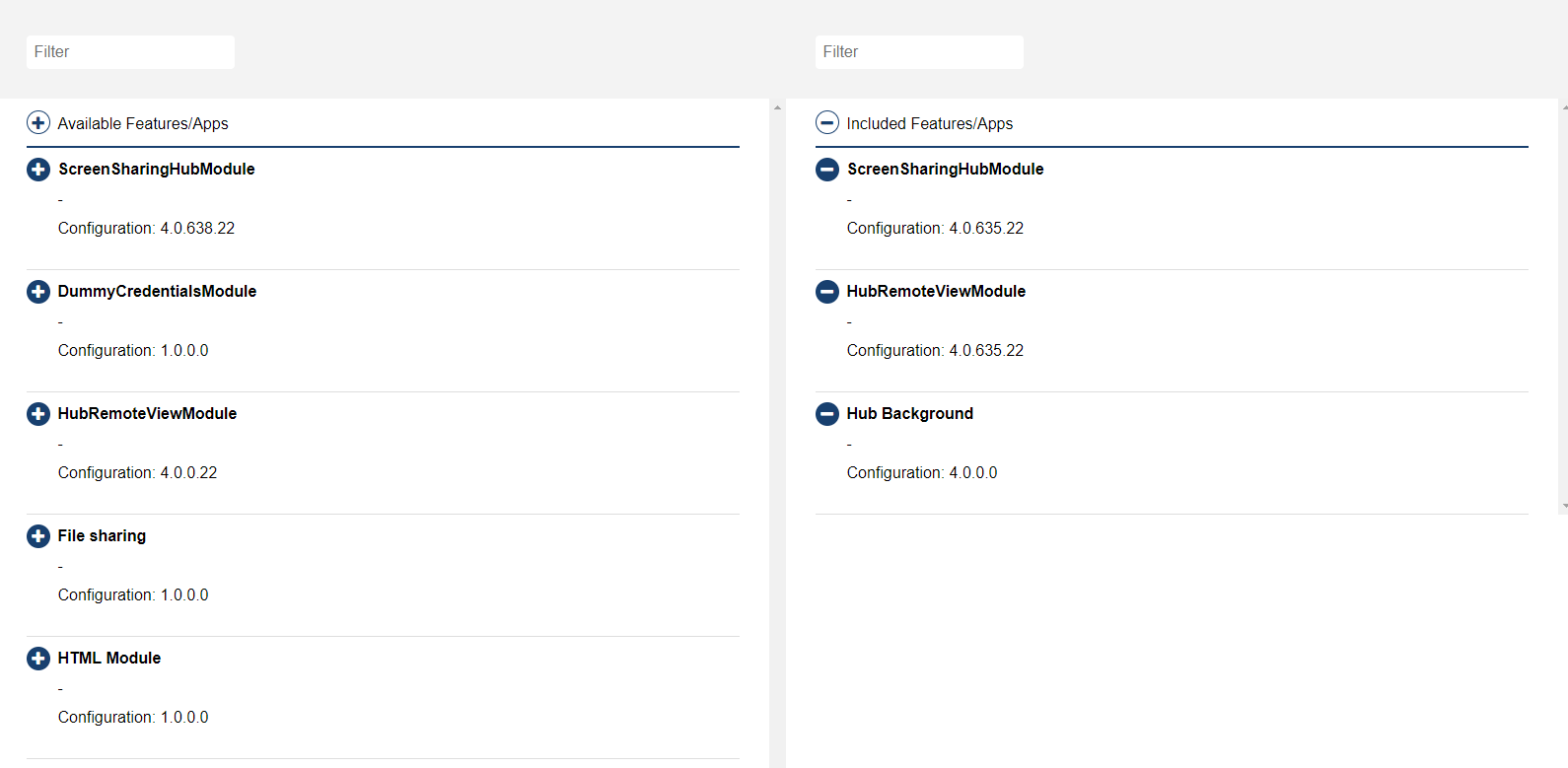
1. **Select Plugin of interest:** From Windows Explorer -> **select the .cab file** -> click **Open**
2. **Success:** Intel Unite® Admin Portal Toast message displays **Uploading Package…** -> **Success**
3. **Approve Upload:** If not already on the Features/App page (see Module Features) , click **Device Management -> Features/Apps -> Package Approval tab ->** click **Approve** button -> Message Toast displays **Success** -> Approved Plugin is displayed in **Hub Features** tab.

Figure 28 - Hub Feature Tab



1. **Assign Plugin** (refer to **Hub Feature List**) **: Device** Management -> **Configurations** -> click **Hub Configurations** tab -> click **Edit** button -> click **+** on **Unite Plugin Example** -> **Unite Plugin Example** appears in right column.

Figure 29 - Hub Feature Edit



# Troubleshooting, Notes…

## Developer Notes

* Events must be abstract and non-blocking. If a long running operation occurs, a thread should be used to ensure that the UI remains responsive.
* A plugin must handle its own exceptions.
* When a user control has to maintain the UI changes through the deallocation and allocation process, the HubAllocationInfoObject must be configured with the property ReuseControl set to True and specified with a unique ID. If the ID is set up with an empty GUID, the control will not respect the configuration. This can cause and allocation error.

Also, if the control is allocated and the developer performs a deallocation process to reuse the same control, the contract of the control must be sent as null in the method RuntimeContext.DisplayManager.AllocateUiInHubDisplayAsync, as explained earlier in this document.

## Logs

The developer can use RuntimeContext.LogManagerfor logging proposes. The interface contains three methods, as shown in Figure 144. Figure 145 shows the enum LogLevel used to classify the log.

The logs are written to the Unite.sql file. This file can be found on the %Temp% path, as shown in Figure 146. The file can be opened with the open source tool SQLLiteBrowser\*. Figure 147 shows the logging table written by Intel Unite.

Figure 144. IModuleLoggingManager Methods

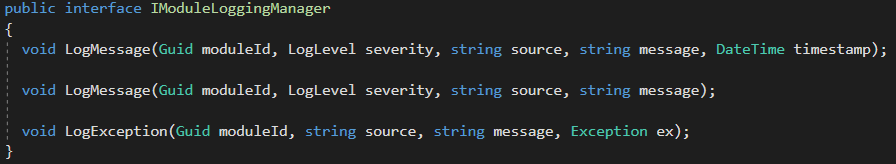


Figure 145. LogLevel enum Options

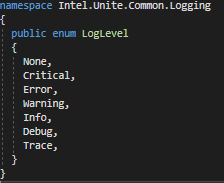


Figure 146. Unite Logs Path

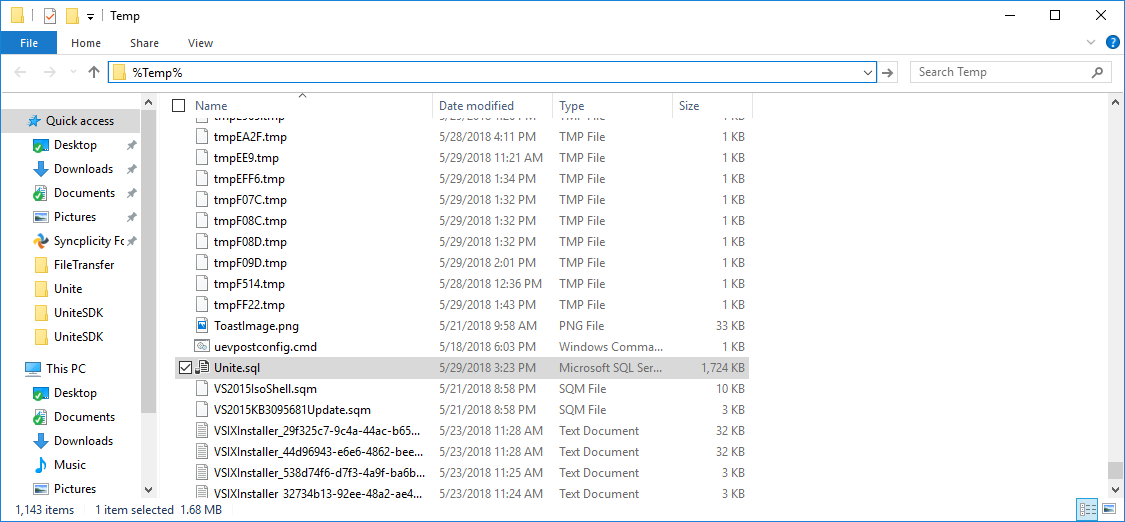
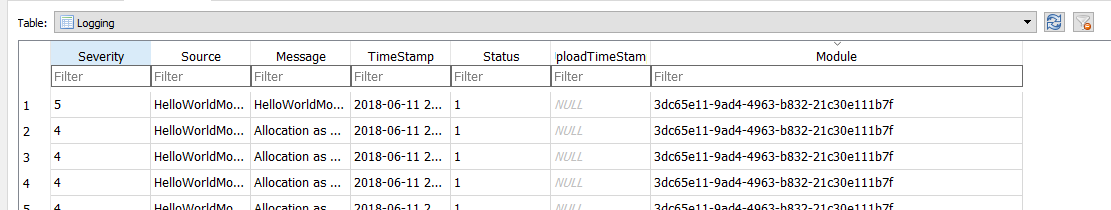


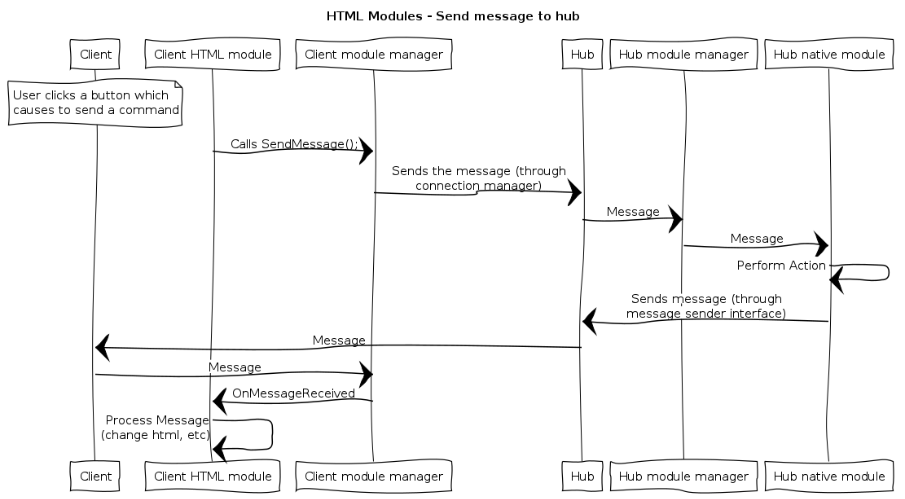
Figure 147. Logging Table



## Commands

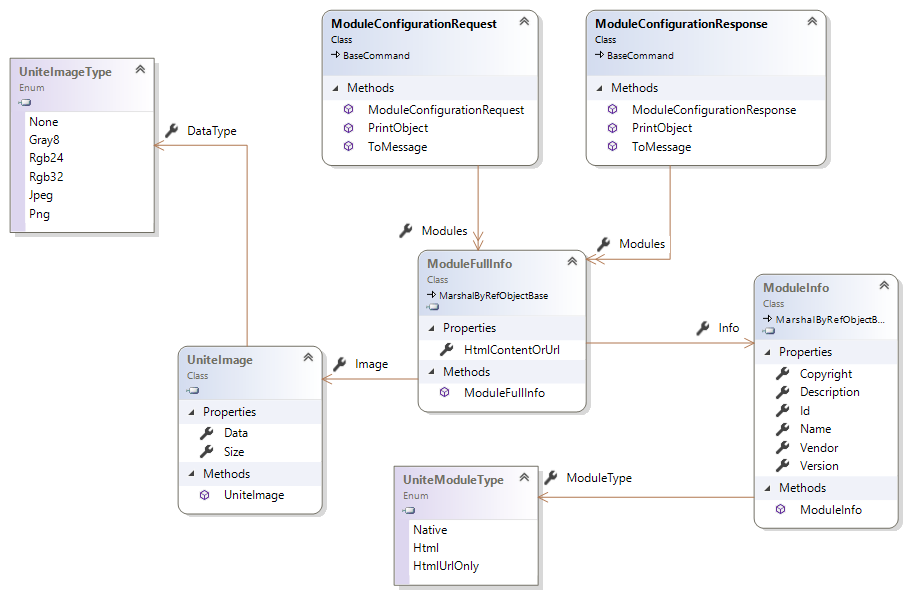
Commands are used to share information between the hub and client. *Figure 148* illustrates the communication. In most cases, a client sends a request to the hub, and the hub returns a response. This is possible using commands.

Figure 31 – Commands Ladder Diagram



Most of the time, the hub requires a response for each request, but this depend of the logic and module configuration commands (ModuleConfigurationRequest and ModuleConfigurationResponse). Figure 149 shows the class diagram on the module structure.

Figure 32 - Class Diagram



Target IDs are used to identify the destination of a command, but sometimes Target IDs are incorrectly defined. Therefore, keep the following in mind:

1. Target ID is used to identify the destination of the command. Most of the time, the constant MessageConstants.TargetBroadcastId is used. This means the message is sent to all clients in the session.
2. A message can be sent to a specific client. Clients connected to the hub contain a GUID identifier. To get the identifier, consult the session context with the line RuntimeContext.SessionContext.Users. This returns all the users connected in the session.
3. If a message is not arriving to the hub, RuntimeContext.MessageSender.TrySendMessage is returning false. This situation can occur when the size of a message is greater than the maximum allowed permitted in Intel Unite. The default value is 65535 bytes. The size of the message can be reduced in the Admin Portal, or the command can be split into chunk messages to respect the size limitations.

## Troubleshooting

### Where is the Intel.Unite.Common.dll?

If you installed and ran the Intel Unite® software Hub Application, the dll can be found here: %ProgramData%\Intel\Intel Unite\Hub\current. The dll is also included in the SDK itself and can be found included in the extracted files.

1. Support

Intel provides no official support for the Intel Unite® Plugin SDK, but best efforts are made by the team to address the needs of the development community.

For developers with an established relationship with Intel Corporation, contact your respective Business Development Manager, Application Engineer, or other Intel team member to request assistance with specific needs using this SDK and supporting documentation.

For all others, contact Intel Customer Support or participate in the Intel Developer Zone at www.intel.com to receive additional attention and assistance with this SDK.

As Intel continues development the Intel Unite application and features, new updates are posted to the SDK. Please check back for additional updates as they become available in the future.