Intel Unite®

Unified Communication

For Client UIs

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 |

Client UI

## Client UI

Connecting clients have the unique ability to have a browser like experience inside the Intel Unite Client application.

### Requirements

For an Intel Unite Client to communicate with a Hub or other plugins, a clear understanding of the massaging system should be reviewed and understood.

1. Pre-requisites: Messaging section of the Unite SDK ensuring that the message(s) sent to the plugin are correctly processed.

### HtmlContent.html

Looking at the client html page, we have 4 buttons that will be used to toggle the state of the Status, Auth, Presentation and Partial Background views. As expected, the onclick events point to unique methods to toggle the target view.

1. If HtmlContent.html does not exist, create it in the folder **ClientUI/Source**.

|  |
| --- |
| <!DOCTYPE html>  <html lang="en" xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta charset="utf-8" />  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <title>Dummy HTML module</title>    <!-- our custom, minified CSS -->  <style type="text/css">  {sitecss}  </style>    <!-- our custom, minified JS -->  <!-- -->  <script type="text/javascript">  {HelloWorldControlJS}  </script>  </head>  <body>  <p>  <button id="send-message-buttonPartial" onclick="sendMessagePartial()">Partial Background</button>  <button id="send-message-buttonPresentation" onclick="sendMessagePresentation()">Presentation</button>  <button id="send-message-buttonStatus" onclick="sendMessageStatus()">Status</button>  <button id="send-message-buttonAuth" onclick="sendMessageAuth()">Auth</button>  <button id="fromHub" class="hideB"></button>  </body>  </html> |

### Site.min.css

The dynamic web page cascading style sheet should also be added.

1. If site.min.js does not exist, create it in the folder **ClientUI/Source/css**.

|  |
| --- |
| body {  font-family: Arial, "Helvetica Neue", Helvetica, sans-serif  }  @font-face {  font-family: 'intelclear\_whans\_rg';  src: url('../fonts/intelclear\_whans\_rg.ttf') format('truetype'), url('../fonts/intelclear\_whans\_rg.eot?') format('embedded-opentype')  }  #debugoutput {  width: 800px;  height: 300px;  border: 1px solid #00f;  color: #fff;  background-color: #000;  font-weight: bold;  margin: 0 auto  }  button {  background-color: #0071c5;  border: none;  color: #fff;  padding: 15px 32px;  text-align: center;  text-decoration: none;  display: inline-block;  font-size: 16px;  margin: 4px 2px;  cursor: pointer  }  .hideB {  visibility: display;  background-color: #fff;  border: none;  color: #0071c5;  weight: bold;  padding: 15px 32px;  text-align: center;  text-decoration: none;  display: inline-block;  font-size: 16px;  margin: 4px 2px;  cursor: pointer  }  .clear {  clear: both  }  .divWrapper {  margin: 0 auto;  text-align: center;  width: 500px  }  .divTable {  display: table;  margin: 0 auto  }  .divTableRow {  display: table-row  }  .divTableHeading {  display: table-header-group  }  .divTableCell,  .divTableHead {  display: table-cell;  padding: 3px 10px;  text-align: center  }  .divTableHeading {  display: table-header-group;  font-weight: bold;  height: 50px  }  .divTableFoot {  display: table-footer-group;  font-weight: bold  }  .divTableBody {  display: table-row-group  }  .left {  width: 75%;  border: #d3d3d3 solid 1px  }  .right {  border: #d3d3d3 solid 1px;  width: 25%  }  p {  display: block;  color: #127bca;  font-size: 1.5em;  text-align: center  }  .middlepad {  height: 96px  }  .lower-control {  width: 500px;  display: table-cell;  text-align: center  }  .camera-label {  margin-top: 7px;  border: 1px solid #d3d3d3;  text-align: left;  padding-left: 25px  }  .camera-label p {  margin-left: 50px;  vertical-align: middle;  border: solid 1px #00f  }  input,  select {  display: inline-block;  background-color: #fff;  border: solid 1px #127bca;  margin: 0;  padding: 5px;  margin-bottom: 3px;  -moz-border-radius: 2px;  -webkit-border-radius: 2px;  border-radius: 2px  }  #camera-loading {  display: block  }  #select-camera {  margin-right: 15px  }  #flip-camera {  float: right;  margin: 15px;  width: 140px  }  .disabled {  opacity: .6;  cursor: not-allowed  }  .small {  font-size: 1em  }  input[type=checkbox] {  -ms-transform: scale(2);  -moz-transform: scale(2);  -webkit-transform: scale(2);  -o-transform: scale(2);  padding: 10px  } |

### HelloWorldControl.main.js

The base JavaScript file has methods for sending a message to the target plugin and other JavaScript methods. When interacting with the plugin one should tailor these messages accordingly and in this example the methods attached to the buttons on the client, are toggling the Status, Auth, Presentation and Partial Background views.

1. If HelloWorldControl.main.js does not exist, create it in the folder **ClientUI/Source/js**.

As you can see below, the button methods are basically the same, the important piece and difference between these methods is the DataType. DataType targets the EventArgumentsTypes.cs in this plugin. In this example, we are not using the Data field which is the variable **n** and calls JSON.stringify. We could add any string, int or class properties to the area between { } for example { TargetInteger: “1087” }.

|  |
| --- |
| function sendMessagePartial()  {  var n = JSON.stringify({ }),  t = btoa(n),  i = {  DataType: 309,  Base64Data: t,  Priority: 2  };  window.IntelUnite.sendMessage(i)  }  function sendMessagePresentation()  {  var n = JSON.stringify({ }),  t = btoa(n),  i = {  DataType: 308,  Base64Data: t,  Priority: 2  };  window.IntelUnite.sendMessage(i)  }  function sendMessageStatus()  {  var n = JSON.stringify({ }),  t = btoa(n),  i = {  DataType: 307,  Base64Data: t,  Priority: 2  };  window.IntelUnite.sendMessage(i)  }  function sendMessageAuth()  {  var n = JSON.stringify({ }),  t = btoa(n),  i = {  DataType: 306,  Base64Data: t,  Priority: 2  };  window.IntelUnite.sendMessage(i)  } |

### PluginModuleHandler.cs

In the PluginModuleHandler.cs file, we have some code that is required to create the HTML for clients.

|  |
| --- |
| private string \_html = @"<!DOCTYPE html><html><head><title>Error</title><script type='text/javascript'>window.onload=function(){alert();}</script></head><body onclick='alert()'><div>If you're reading this, something went wrong.</div></body></html>";  public PluginModuleHandler(IModuleRuntimeContext runtimeContext) : base(runtimeContext)  {  ConfigureModuleForClient();  }  private void ConfigureModuleForClient()  {  FeatureModuleType = FeatureModuleType.Html;  ModuleImage = UniteImageHelper.GetUniteImageFromResource("/UnitePlugin;component/Images/menu-icon.png", UniteImageType.Png);  \_html = ClientUiSetup.getHtml();  } |

### ClientUISetup.cs

ClientUISetup.cs contains the setup and configuration of the Client UI HTML – the HTML is dynamically built. Unite clients can have web-based pages that appear as icons and can interact with plugin(s), the Unite Hub or interact with 3rd party tools or APIs’.

1. If ConfigUISetup.cs does not exist, create it in the folder **ClientUI**.

|  |
| --- |
| public static string getHtml()  {  var task = ReadHtmlContentAsync();  task.Wait();  return task.Result;  }  public static async Task<string> ReadHtmlContentAsync()  {  // Make sure the BuildAction for these files is set to Embedded Resource  var sitecss = await GetFileContentAsStringAsync("UnitePlugin.ClientUI.Source.css.site.min.css");  // minified JS to be injected  var helloWorldControlJs = await GetFileContentAsStringAsync("UnitePlugin.ClientUI.Source.js.HelloWorldControl.min.js");  //\*\*\* dynamic html built and put into string \*\*\*  // load html page  var html = await GetFileContentAsStringAsync("UnitePlugin.ClientUI.Source.HtmlContent.html");    // string replace to inject external CSS and JS  return html.Replace("{sitecss}", sitecss).Replace("{HelloWorldControlJS}", helloWorldControlJs);  }  public static async Task<string> GetFileContentAsStringAsync(string resource)  {  using (var sr = new StreamReader(Assembly.GetExecutingAssembly().GetManifestResourceStream(resource) ?? throw new InvalidOperationException()))  {  return await sr.ReadToEndAsync();  }  } |